

# Aerospace Blockset™

## User's Guide



# MATLAB® & SIMULINK®

R2020a



## How to Contact MathWorks



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



Phone: 508-647-7000



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

*Aerospace Blockset™ User's Guide*

© COPYRIGHT 2002–2020 by The MathWorks, Inc.

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

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

### Trademarks

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

### Patents

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

## Revision History

|                |                 |  |
|----------------|-----------------|--|
| July 2002      | Online only     | New for Version 1.0 (Release 13)                 |
| July 2003      | Online only     | Revised for Version 1.5 (Release 13SP1)          |
| June 2004      | Online only     | Revised for Version 1.6 (Release 14)             |
| October 2004   | Online only     | Revised for Version 1.6.1 (Release 14SP1)        |
| March 2005     | Online only     | Revised for Version 1.6.2 (Release 14SP2)        |
| May 2005       | Online only     | Revised for Version 2.0 (Release 14SP2+)         |
| September 2005 | First printing  | Revised for Version 2.0.1 (Release 14SP3)        |
| March 2006     | Online only     | Revised for Version 2.1 (Release 2006a)          |
| September 2006 | Online only     | Revised for Version 2.2 (Release 2006b)          |
| March 2007     | Online only     | Revised for Version 2.3 (Release 2007a)          |
| September 2007 | Second printing | Revised for Version 3.0 (Release 2007b)          |
| March 2008     | Online only     | Revised for Version 3.1 (Release 2008a)          |
| October 2008   | Online only     | Revised for Version 3.2 (Release 2008b)          |
| March 2009     | Online only     | Revised for Version 3.3 (Release 2009a)          |
| September 2009 | Online only     | Revised for Version 3.4 (Release 2009b)          |
| March 2010     | Online only     | Revised for Version 3.5 (Release 2010a)          |
| September 2010 | Online only     | Revised for Version 3.6 (Release 2010b)          |
| April 2011     | Online only     | Revised for Version 3.7 (Release 2011a)          |
| September 2011 | Online only     | Revised for Version 3.8 (Release 2011b)          |
| March 2012     | Online only     | Revised for Version 3.9 (Release 2012a)          |
| September 2012 | Online only     | Revised for Version 3.10 (Release 2012b)         |
| March 2013     | Online only     | Revised for Version 3.11 (Release 2013a)         |
| September 2013 | Online only     | Revised for Version 3.12 (Release 2013b)         |
| March 2014     | Online only     | Revised for Version 3.13 (Release 2014a)         |
| October 2014   | Online only     | Revised for Version 3.14 (Release 2014b)         |
| March 2015     | Online only     | Revised for Version 3.15 (Release 2015a)         |
| September 2015 | Online only     | Revised for Version 3.16 (Release 2015b)         |
| October 2015   | Online only     | Rereleased for Version 3.15.1 (Release 2015aSP1) |
| March 2016     | Online only     | Revised for Version 3.17 (Release 2016a)         |
| September 2016 | Online only     | Revised for Version 3.18 (Release 2016b)         |
| March 2017     | Online only     | Revised for Version 3.19 (Release 2017a)         |
| September 2017 | Online only     | Revised for Version 3.20 (Release 2017b)         |
| March 2018     | Online only     | Revised for Version 3.21 (Release 2018a)         |
| September 2018 | Online only     | Revised for Version 4.0 (Release 2018b)          |
| March 2019     | Online only     | Revised for Version 4.1 (Release 2019a)          |
| September 2019 | Online only     | Revised for Version 4.2 (Release 2019b)          |
| March 2020     | Online only     | Revised for Version 4.3 (Release 2020a)          |



## Getting Started

### 1

|   |             |
|---|-------------|
| <b>Aerospace Blockset Product Description</b> .....             | <b>1-2</b>  |
| Key Features .....  | <b>1-2</b>  |
| <b>Code Generation Support</b> .....                            | <b>1-3</b>  |
| <b>Support for Aerospace Toolbox Quaternion Functions</b> ..... | <b>1-4</b>  |
| <b>Explore the NASA HL-20 Model</b> .....                       | <b>1-5</b>  |
| Introduction .....  | <b>1-5</b>  |
| What This Example Illustrates .....                             | <b>1-5</b>  |
| Open the Model .....  | <b>1-5</b>  |
| Key Subsystems .....  | <b>1-7</b>  |
| NASA HL-20 Example .....  | <b>1-7</b>  |
| Modify the Model .....  | <b>1-10</b> |
| <b>Open Aerospace Examples</b> .....                            | <b>1-12</b> |

## Aerospace Blockset Software

### 2

|   |             |
|---|-------------|
| <b>Create Aerospace Models</b> .....                | <b>2-2</b>  |
| Basic Steps .....                                   | <b>2-2</b>  |
| <b>Build a Simple Actuator System</b> .....         | <b>2-4</b>  |
| Build the Model .....                               | <b>2-4</b>  |
| Run the Simulation .....                            | <b>2-7</b>  |
| <b>About Aerospace Coordinate Systems</b> .....     | <b>2-8</b>  |
| Fundamental Coordinate System Concepts .....        | <b>2-8</b>  |
| Coordinate Systems for Modeling .....               | <b>2-9</b>  |
| Body Coordinates .....                              | <b>2-9</b>  |
| Wind Coordinates .....                              | <b>2-10</b> |
| Coordinate Systems for Navigation .....             | <b>2-11</b> |
| Coordinate Systems for Display .....                | <b>2-13</b> |
| <b>Flight Simulator Interface</b> .....             | <b>2-16</b> |
| About the FlightGear Interface .....                | <b>2-16</b> |
| Supported FlightGear Versions .....                 | <b>2-16</b> |
| Obtain FlightGear .....                             | <b>2-17</b> |
| Configure Your Computer for FlightGear .....        | <b>2-17</b> |
| FlightGear and Video Cards in Windows Systems ..... | <b>2-17</b> |

|  |             |
|--|-------------|
| Install and Start FlightGear . . . . .   | 2-18        |
| Install Additional FlightGear Scenery . . . . .  | 2-18        |
| <b>Work with the Flight Simulator Interface . . . . .</b>  | <b>2-20</b> |
| Introduction . . . . .   | 2-20        |
| About Aircraft Geometry Models . . . . .   | 2-20        |
| Work with Aircraft Geometry Models . . . . .   | 2-22        |
| Run FlightGear with Simulink Models . . . . .  | 2-24        |
| Run the HL-20 Example with FlightGear . . . . .  | 2-29        |
| Send and Receive Data . . . . .  | 2-31        |
| <b>Projects Template for Flight Simulation Applications . . . . .</b>  | <b>2-34</b> |
| Flight Simulation Applications . . . . .   | 2-34        |
| <b>Flight Instrument Gauges . . . . .</b>  | <b>2-38</b> |
| <b>Display Measurements with Cockpit Instruments . . . . .</b>   | <b>2-39</b> |
| <b>Programmatically Interact with Gauge Band Colors . . . . .</b>  | <b>2-41</b> |
| <b>Calculate UT1 to UTC Values . . . . .</b>   | <b>2-43</b> |
| Use the Delta UT1 Block to Create Difference Values for the Direction<br>Cosine Matrix ECI to ECEF Block . . . . . | 2-43        |
| <b>Analyze Dynamic Response and Flying Qualities of Aerospace Vehicles<br/>. . . . .</b>                           | <b>2-45</b> |
| Flight Control Analysis Live Scripts . . . . .   | 2-45        |
| Modify Flight Analysis Templates . . . . .   | 2-45        |
| Explore Flight Control Analysis Functions . . . . .  | 2-48        |
| <b>Getting Started with CubeSats . . . . .</b>   | <b>2-49</b> |
| <b>Model and Simulate CubeSats . . . . .</b>   | <b>2-51</b> |
| CubeSat Orbit Propagation Model Template . . . . .   | 2-51        |
| CubeSat Simulation Project . . . . .   | 2-53        |
| Utility Functions . . . . .  | 2-56        |
| <b>Analyze CubeSat Mission Geometry . . . . .</b>  | <b>2-58</b> |
| Utility Function . . . . .   | 2-59        |

## Case Studies

### 3

|  |            |
|--|------------|
| <b>Ideal Airspeed Correction . . . . .</b> | <b>3-2</b> |
| Introduction . . . . .                     | 3-2        |
| Airspeed Correction Models . . . . .       | 3-2        |
| Measure Airspeed . . . . .                 | 3-2        |
| Model Airspeed Correction . . . . .        | 3-3        |
| Simulate Airspeed Correction . . . . .     | 3-5        |
| <b>1903 Wright Flyer . . . . .</b>         | <b>3-7</b> |
| Introduction . . . . .                     | 3-7        |

|   |             |
|---|-------------|
| Wright Flyer Model .....                      | 3-7         |
| Airframe Subsystem .....                      | 3-8         |
| Environment Subsystem .....                   | 3-10        |
| Pilot Subsystem .....                         | 3-11        |
| Run the Simulation .....                      | 3-11        |
| References .....                              | 3-12        |
| <b>NASA HL-20 Lifting Body Airframe .....</b> | <b>3-14</b> |
| Introduction .....                            | 3-14        |
| NASA HL-20 Lifting Body .....                 | 3-14        |
| The HL-20 Airframe and Controller Model ..... | 3-15        |
| References .....                              | 3-21        |

## Blocks

**4**

## Functions

**5**

## Examples

**6**

|   |             |
|---|-------------|
| <b>1903 Wright Flyer and Pilot with Scopes for Data Visualization .....</b>       | <b>6-2</b>  |
| <b>1903 Wright Flyer and Pilot with Simulink® 3D Animation™ .....</b>             | <b>6-4</b>  |
| <b>Fly the De Havilland Beaver .....</b>  | <b>6-7</b>  |
| <b>Lightweight Airplane Design .....</b>  | <b>6-9</b>  |
| <b>Multiple Aircraft with Collaborative Control .....</b>                         | <b>6-25</b> |
| <b>HL-20 with Flight Instrumentation Blocks .....</b>                             | <b>6-27</b> |
| <b>HL-20 with Simulink® 3D Animation™ and Flight Instrumentation Blocks .....</b> | <b>6-32</b> |
| <b>HL-20 Project with Optional FlightGear Interface .....</b>                     | <b>6-37</b> |
| <b>Quaternion Estimate from Measured Rates .....</b>                              | <b>6-39</b> |
| <b>Indicated Airspeed from True Airspeed Calculation .....</b>                    | <b>6-40</b> |
| <b>Six Degree of Freedom Motion Platform .....</b>                                | <b>6-42</b> |

|   |             |
|---|-------------|
| <b>Gravity Models with Precessing Reference Frame</b> .....                 | <b>6-45</b> |
| <b>True Airspeed from Indicated Airspeed Calculation</b> .....              | <b>6-48</b> |
| <b>Airframe Trim and Linearize with Simulink® Control Design™</b> .....     | <b>6-50</b> |
| <b>Airframe Trim and Linearize with Control System Toolbox™</b> .....       | <b>6-53</b> |
| <b>Self-conditioned Controller Comparison</b> .....                         | <b>6-57</b> |
| <b>Quadcopter Project</b> .....   | <b>6-59</b> |
| <b>Electrical Component Analysis for Hybrid and Electric Aircraft</b> ..... | <b>6-66</b> |

## **Aerospace Units Appendix**

### **A**

|                              |            |
|------------------------------|------------|
| <b>Aerospace Units</b> ..... | <b>A-2</b> |
|------------------------------|------------|



# Getting Started

---

- “Aerospace Blockset Product Description” on page 1-2
- “Code Generation Support” on page 1-3
- “Support for Aerospace Toolbox Quaternion Functions” on page 1-4
- “Explore the NASA HL-20 Model” on page 1-5
- “Open Aerospace Examples” on page 1-12

## **Aerospace Blockset Product Description**

### **Model, simulate, and analyze aerospace vehicle dynamics**

Aerospace Blockset provides Simulink® blocks for modeling, simulating, and analyzing aerospace vehicles. You can incorporate vehicle dynamics, validated models of the flight environment, and pilot behavior, and then connect your model to the FlightGear Flight Simulator to visualize simulation results.

With Aerospace Blockset you can use aerodynamic coefficients or Data Compendium (Datcom) derivatives to model fixed-wing, rotary-wing, and multirotor vehicles. Prebuilt component libraries let you design GNC algorithms and model actuator dynamics and the propulsion subsystem. Built-in aerospace math operations and coordinate system and spatial transformations let you describe the behavior of three-degrees-of-freedom (3DOF) and six-degrees-of-freedom (6DOF) bodies.

The blockset includes validated environment models for atmosphere, gravity, wind, geoid height, and magnetic field to represent flight conditions and increase simulation fidelity. Flight control analysis tools let you analyze the dynamic response and flying qualities of aerospace vehicles. To complete your analysis, you can visualize the vehicle in flight directly from Simulink with standard cockpit instruments and using the prebuilt FlightGear Flight Simulator interface.

### **Key Features**

- Equations of motion with fixed or variable mass, including 3DOF and 6DOF
- Built-in axis transformations for body, wind, and geodetic coordinates, including earth-centered / earth-fixed (ECEF) and earth-centered inertial (ECI)
- Validated environment models, including gravity, atmosphere, wind, geoid height, magnetic field, and planetary ephemerides
- Prebuilt aerospace vehicle components, including sensor and controller designs for GNC and propulsion, actuators, and pilot models
- Flight control analysis tools for studying a vehicle's dynamic response and flying qualities
- 3D animations and flight instruments for visualizing navigation variables
- Data Compendium (Datcom) derivatives for aerodynamic forces and moments

## **Code Generation Support**

Use the Aerospace Blockset software with the Simulink Coder software to automatically generate code for real-time execution in rapid prototyping and for hardware-in-the-loop systems.

## Support for Aerospace Toolbox Quaternion Functions

The Aerospace Blockset product supports the following Aerospace Toolbox quaternion functions in the MATLAB Function block:

quatconj  
quatinv  
quatmod  
quatmultiply  
quatdivide  
quatnorm  
quatnormalize

For further information on using the MATLAB Function block, see:

- “Implementing MATLAB Functions Using Blocks” (Simulink)
- `asbQuatEML` example, which illustrates quaternions and models the equations

# Explore the NASA HL-20 Model

## In this section...

“Introduction” on page 1-5  
“What This Example Illustrates” on page 1-5  
“Open the Model” on page 1-5  
“Key Subsystems” on page 1-7  
“NASA HL-20 Example” on page 1-7  
“Modify the Model” on page 1-10

## Introduction

This section introduces a NASA HL-20 lifting body airframe model that uses blocks from the Aerospace Blockset software to simulate the airframe of a NASA HL-20 lifting body, in conjunction with other Simulink blocks.

The model simulates the NASA HL-20 lifting body airframe approach and landing flight phases using an automatic-landing controller.

For more information on this model, see “NASA HL-20 Lifting Body Airframe” on page 3-14.

## What This Example Illustrates

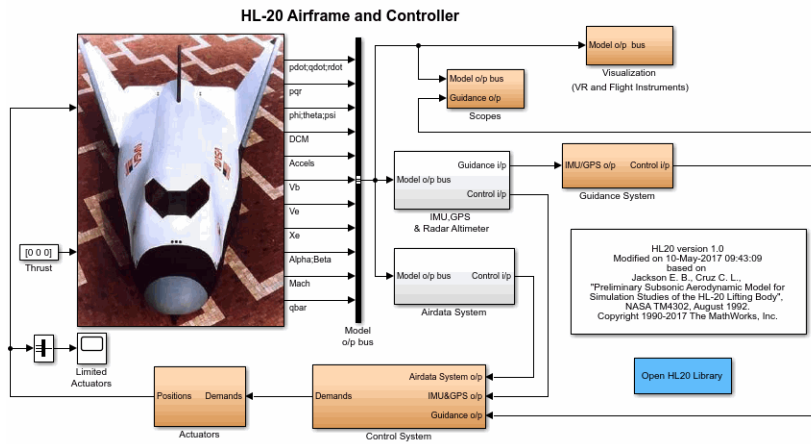
The NASA HL-20 lifting body airframe example illustrates the following features of the blockset:

- Representing bodies and their degrees of freedom with the Equations of Motion library blocks
- Using the Aerospace Blockset blocks with other Simulink blocks
- Feeding Simulink signals to and from Aerospace Blockset blocks with Actuator and Sensor blocks
- Encapsulating groups of blocks into subsystems
- Visualizing an aircraft with Simulink 3D Animation™ and Aerospace Blockset Flight Instrument library blocks.

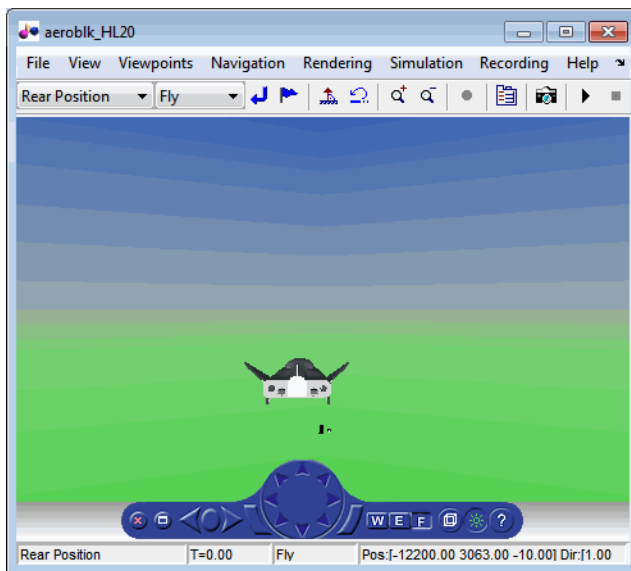
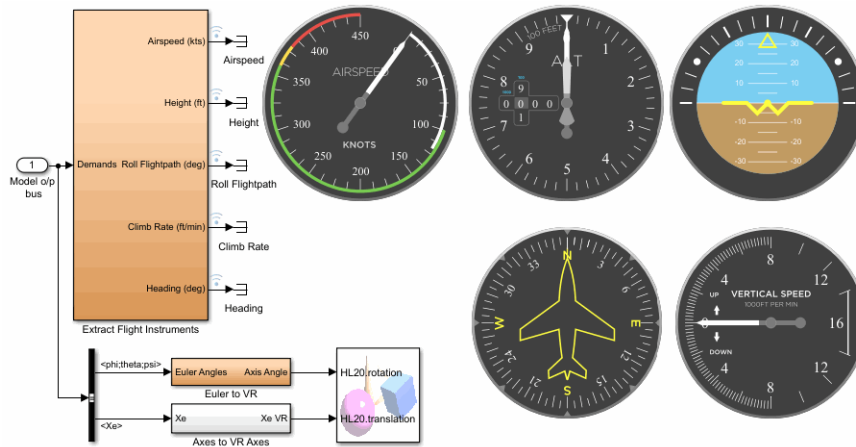
## Open the Model

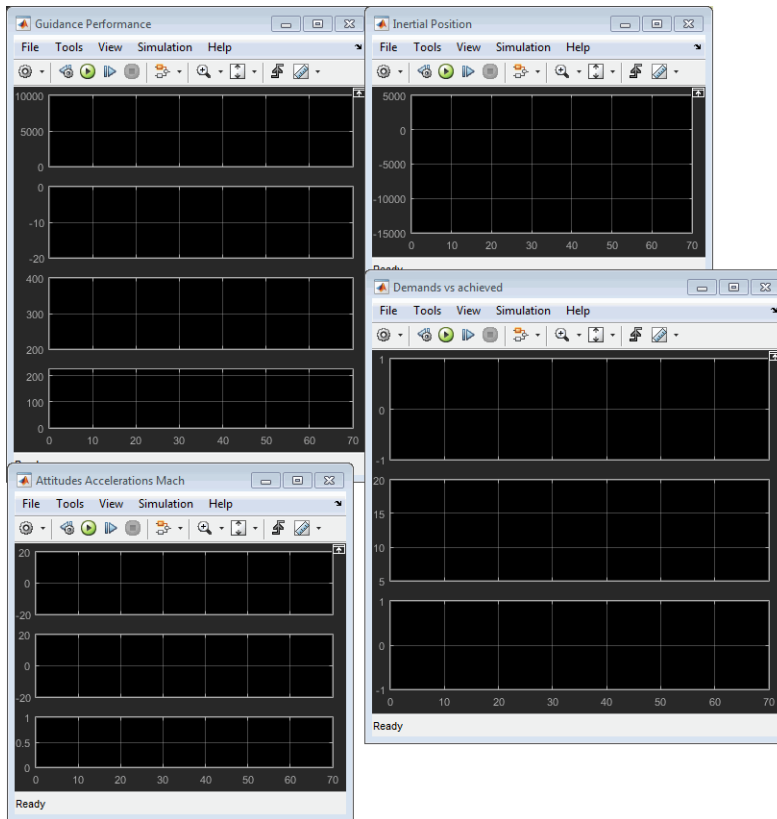
To open the NASA HL-20 airframe example, type the example name, `aeroblk_HL20`, at the MATLAB® command line. The model opens.

# 1 Getting Started



The visualization subsystem, four scopes, and a Simulink 3D Animation viewer for the airframe might also appear.





## Key Subsystems

The model implements the airframe using the following subsystems:

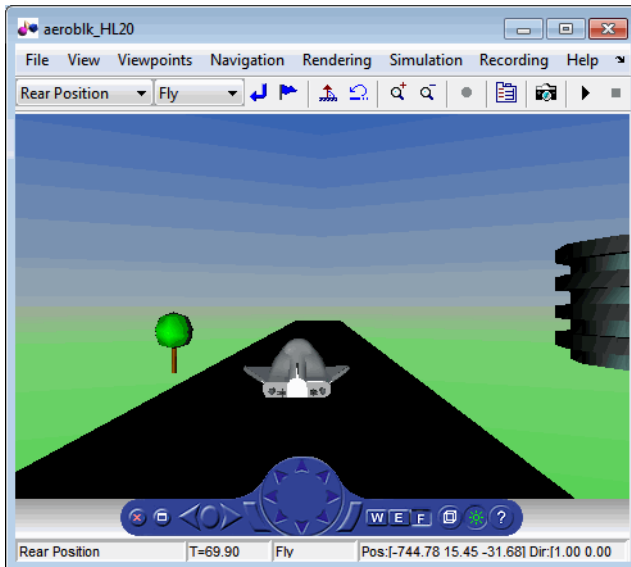
- The 6DOF (Euler Angles) subsystem implements the 6DOF (Euler Angles) block along with other Simulink blocks.
- The Environment Models subsystem implements the WGS84 Gravity Model and COESA Atmosphere Model blocks. It also contains a Wind Models subsystem that implements a number of wind blocks.
- The Alpha, Beta, Mach subsystem implements the Incidence, Sideslip & Airspeed, Mach Number, and Dynamic Pressure blocks. These blocks calculate aerodynamic coefficient values and lookup functionality.
- The Forces and Moments subsystem implements the Aerodynamic Forces and Moments block. This subsystem calculates body forces and body moments.
- The Aerodynamic Coefficients subsystem implements several subsystems to calculate six aerodynamic coefficients.

## NASA HL-20 Example

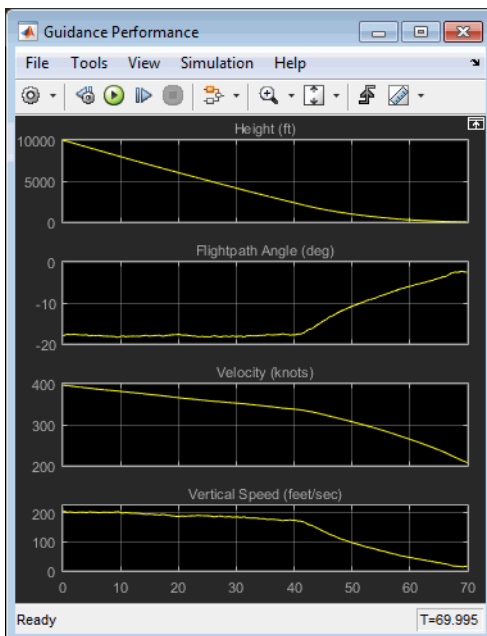
Running an example lets you observe the model simulation in real time. After you run the example, you can examine the resulting data in plots, graphs, and other visualization tools. To run this model, follow these steps:

- 1 If it is not already open, open the `aeroblk_HL20` example.
- 2 From the **Simulation** menu, select **Start**. On Microsoft® Windows® systems, you can also click the **Start** button in the model window toolbar.

The simulation proceeds until the aircraft lands:

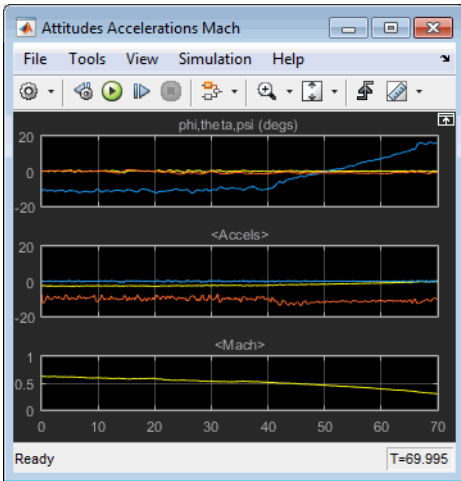


## View of the landed airframe

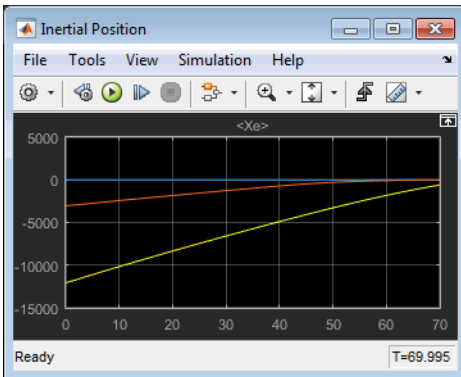


## Plot that Measures Guidance Performance

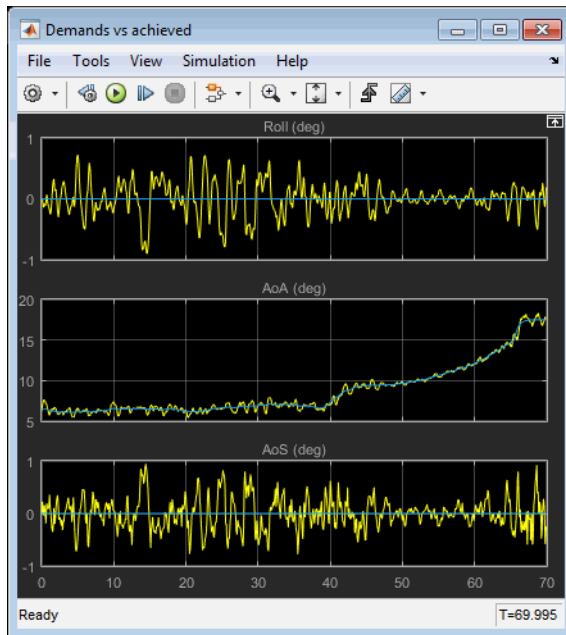




**Plot that Measures Altitude Accelerations Mach**



**Plot that Measures Inertial Position**



**Plot that Measures Demand Data Against Achieved Data**

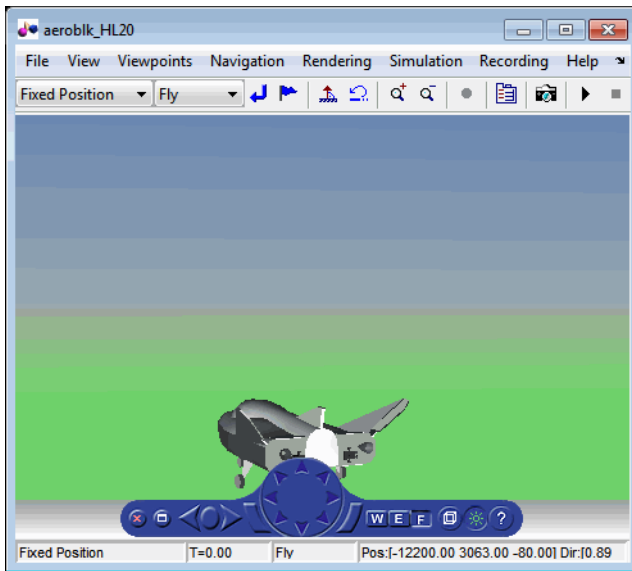
## Modify the Model

You can adjust the airframe model settings and examine the effects on simulation performance. Here is one modification that you can try. It changes the camera point of view for the landing animation.

### Change the Animation Point of View

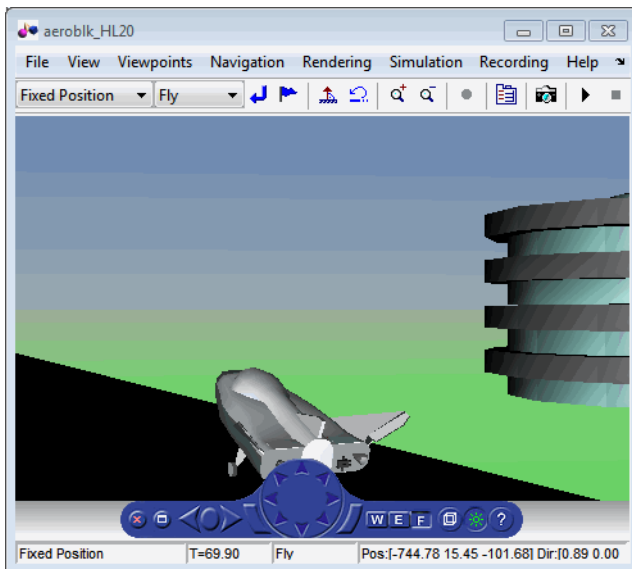
By default, the airframe animation viewpoint is `Rear position`, which means the view tracks with the airframe flight path from the rear. You can change the animation point of view by selecting another viewpoint from the Simulink 3D Animation viewer:

- 1 Open the `aeroblk_HL20` model, and click the Simulink 3D Animation viewer.
- 2 From the list of existing viewpoints, change the viewpoint to `Fixed Position`.



The airframe view changes to a fixed position.

- 3 Start the model again. Notice the different airframe viewpoint when the airframe lands.



You can experiment with different viewpoints to watch the animation from different perspectives.

## Open Aerospace Examples

To open an Aerospace Blockset example from the Help Browser:

- 1** Open the MATLAB Command Window.
- 2** Click the question mark.
- 3** Navigate to Aerospace Blockset and click the **Examples** tab.

For in-depth case studies of the following examples, see:

- “Ideal Airspeed Correction” on page 3-2
- “1903 Wright Flyer” on page 3-7
- “NASA HL-20 Lifting Body Airframe” on page 3-14

# Aerospace Blockset Software

---

- “Create Aerospace Models” on page 2-2
- “Build a Simple Actuator System” on page 2-4
- “About Aerospace Coordinate Systems” on page 2-8
- “Flight Simulator Interface” on page 2-16
- “Work with the Flight Simulator Interface” on page 2-20
- “Projects Template for Flight Simulation Applications” on page 2-34
- “Flight Instrument Gauges” on page 2-38
- “Display Measurements with Cockpit Instruments” on page 2-39
- “Programmatically Interact with Gauge Band Colors” on page 2-41
- “Calculate UT1 to UTC Values” on page 2-43
- “Analyze Dynamic Response and Flying Qualities of Aerospace Vehicles” on page 2-45
- “Getting Started with CubeSats” on page 2-49
- “Model and Simulate CubeSats” on page 2-51
- “Analyze CubeSat Mission Geometry” on page 2-58

## Create Aerospace Models

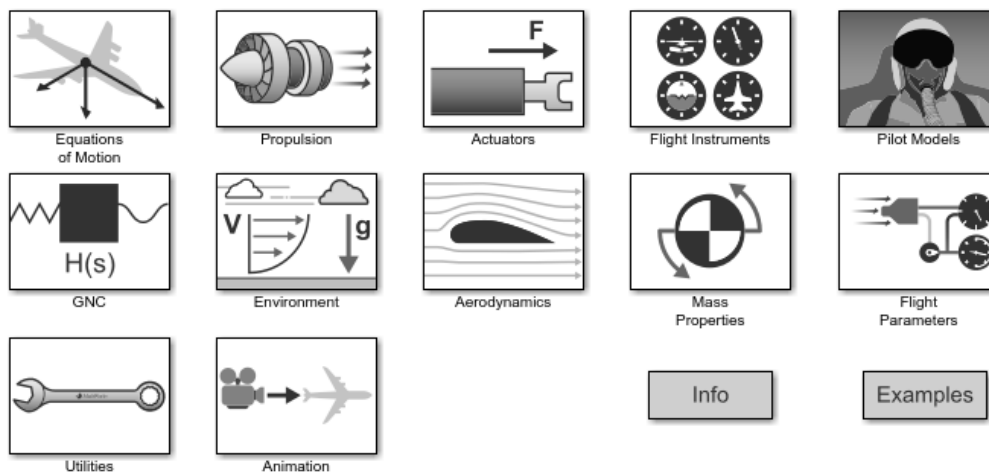
### Basic Steps

Regardless of the model's complexity, you use the same essential steps for creating an aerospace model as you would for creating any other Simulink model.

- 1 Open the Aerospace Blockset Library. You can access this library through the Simulink Library Browser or directly open the Aerospace Blockset window from the MATLAB command line:

```
aerolib
```

Double-click any library in the window to display its contents. The following figure shows the Aerospace Blockset library window.



- 2 *Select and position the blocks.* You must first select the blocks that you need to build your model, and then position the blocks in the model window. For the majority of Simulink models, you select one or more blocks from each of the following categories:
  - a Source blocks generate or import signals into the model, such as a sine wave, a clock, or limited-band white noise.
  - b Simulation blocks can consist of almost any type of block that performs an action in the simulation. A simulation block represents a part of the model functionality to be simulated, such as an actuator block, a mathematical operation, a block from the Aerospace Blockset library, and so on.
  - c Signal Routing blocks route signals from one point in a model to another. If you need to combine or redirect two or more signals in your model, you will probably use a Simulink Signal Routing block, such as Mux and Demux.

As an alternative to the Mux block, you can use the **Vector** option of the Concatenate block **Mode** parameter (also known as the Vector Concatenate block). This block provides a more general way for you to route signals from one point in a model to another. The Vector mode takes as input a vector of signals of the same data type and creates a contiguous output signal. Depending on the input, this block outputs a row or column vector if any of the inputs are row or column vectors, respectively.

- d** Sink blocks display, write, or save model output. To see the results of the simulation, you must use a Sink block.
- 3** *Configure the blocks.* Most blocks feature configuration options that let you customize block functionality to specific simulation parameters. For example, the ISA Atmosphere Model block provides configuration options for setting the height of the troposphere, tropopause, and air density at sea level.
- 4** *Connect the blocks.* To create signal pathways between blocks, you connect the blocks to each other. You can do this manually by clicking and dragging, or you can connect blocks automatically.
- 5** *Encapsulate subsystems.* Systems made with Aerospace Blockset blocks can function as subsystems of larger, more complex models, like subsystems in any Simulink model.

## Build a Simple Actuator System

### In this section...

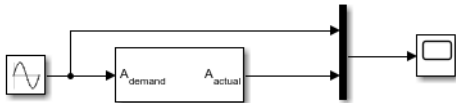
“Build the Model” on page 2-4

“Run the Simulation” on page 2-7

### Build the Model

The Simulink product is a software environment 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, enter `aeroblktutorial` at the MATLAB command line.




The following section (“Create a Model” on page 2-4) explains how to build a model on Windows platforms. You can use this same procedure to build a model on Linux<sup>®</sup> platforms.

The section describes how to build the model. It does not describe how to set the configuration parameters for the model. See “Set Model Configuration Parameters for a Model” (Simulink). That topic describes the Configuration Parameters dialog box for models. If you do not set any configuration parameters, simulating models might cause warnings like:

Warning: Using a default value of 0.2 for maximum step size. The simulation step size will be equal to or less than this value. You can disable this diagnostic by setting 'Automatic solver parameter selection' diagnostic to 'none' in the Diagnostics page of the configuration parameters dialog

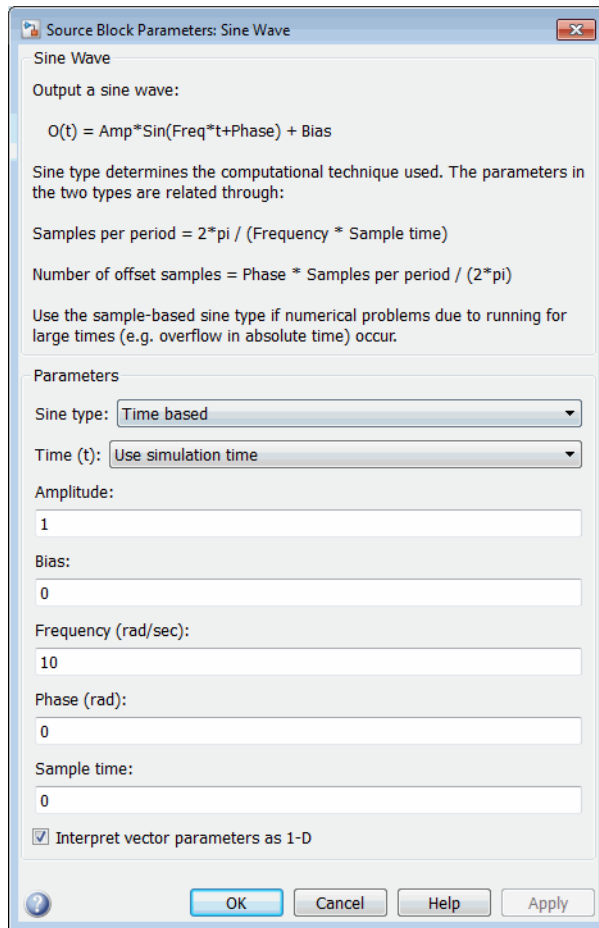
### Create a Model

To create a new blank model and open the Simulink library browser:

- 1 On the MATLAB **Home** tab, click Simulink. In the Simulink start page, click the Blank Model template, and then click Create Model.
- 2 To open the Library Browser, click the browser button.
- 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 Linear Second-Order Actuator block to the model.
  - a Click the  symbol next to **Aerospace Blockset** in the Library Browser to expand the hierarchical list of the aerospace blocks.



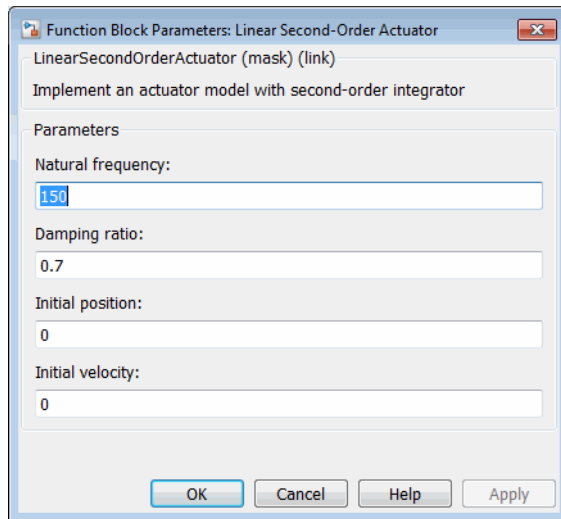
- b** In the expanded list, click **Actuators** to view the blocks in the Actuator library.
  - c** Drag the Linear Second-Order Actuator block into the model window.
- 5** Add a Mux block to the model.
  - a** Click **Signal Routing** in the Library Browser to view the blocks in the Simulink Signals & 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 Linear Second-Order Actuator block. Release the mouse button.
  - b** Using the same technique, connect the output of the Linear Second-Order 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.  
  
For this example, configure the block to generate a 10 rad/s 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 Linear Second-Order Actuator block.

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

- d Click **OK**.



## Run the Simulation

You can now run the model 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 **Run** from the **Simulation** menu in the model window. The signal containing the 10 rad/s sinusoid and the signal containing the actuator position are plotted on the scope.
- 3 Adjust the Scope block's display. 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.
- 4 Vary the Sine Wave block parameters.
  - a While the simulation is running, double-click the Sine Wave block to open its parameter dialog box.
  - b You can then change the frequency of the sinusoid. Try entering 1 or 20 in the **Frequency** field. Close the Sine Wave dialog box to enter your change. You can then 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. However, there are some parameters, like the Sine Wave **Frequency** parameter, that you can *tune* without stopping the simulation.

## Run a Simulation from a Script

You can also modify and run a Simulink simulation from a script. 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 "Run Simulations Programmatically" (Simulink).

## About Aerospace Coordinate Systems

### In this section...

“Fundamental Coordinate System Concepts” on page 2-8

“Coordinate Systems for Modeling” on page 2-9

“Body Coordinates” on page 2-9

“Wind Coordinates” on page 2-10

“Coordinate Systems for Navigation” on page 2-11

“Coordinate Systems for Display” on page 2-13

### Fundamental Coordinate System Concepts

Coordinate systems allow you to keep track of an aircraft or spacecraft position and orientation in space. The Aerospace Blockset coordinate systems are based on these underlying concepts from geodesy, astronomy, and physics.

#### Definitions

The blockset uses right-handed (RH) Cartesian coordinate systems. The right-hand rule establishes the  $x$ - $y$ - $z$  sequence of coordinate axes.

An inertial frame is a nonaccelerating motion reference frame. In an inertial frame, Newton's second law holds:  $\text{force} = \text{mass} \times \text{acceleration}$ . Loosely speaking, acceleration is defined with respect to the distant cosmos, and an inertial frame is often said to be nonaccelerated with respect to the fixed stars. Because the Earth and stars move so slowly with respect to one another, this assumption is a very accurate approximation.

Strictly defined, an inertial frame is a member of the set of all frames not accelerating relative to one another. A noninertial frame is any frame accelerating relative to an inertial frame. Its acceleration, in general, includes both translational and rotational components, resulting in pseudoforces (pseudogravity, as well as Coriolis and centrifugal forces).

The blockset models the Earth shape (the geoid) as an oblate spheroid, a special type of ellipsoid with two longer axes equal (defining the equatorial plane) and a third, slightly shorter (geopolar) axis of symmetry. The equator is the intersection of the equatorial plane and the Earth surface. The geographic poles are the intersection of the Earth surface and the geopolar axis. In general, the Earth geopolar and rotation axes are not identical.

Latitudes parallel the equator. Longitudes parallel the geopolar axis. The zero longitude or prime meridian passes through Greenwich, England.

#### Approximations

The blockset makes three standard approximations in defining coordinate systems relative to the Earth.

- The Earth surface or geoid is an oblate spheroid, defined by its longer equatorial and shorter geopolar axes. In reality, the Earth is slightly deformed with respect to the standard geoid.
- The Earth rotation axis and equatorial plane are perpendicular, so that the rotation and geopolar axes are identical. In reality, these axes are slightly misaligned, and the equatorial plane wobbles as the Earth rotates. This effect is negligible in most applications.

- The only noninertial effect in Earth-fixed coordinates is due to the Earth rotation about its axis. This is a rotating, geocentric system. The blockset ignores the Earth acceleration around the Sun, the Sun acceleration in the Galaxy, and the Galaxy acceleration through the cosmos. In most applications, only the Earth rotation matters.

This approximation must be changed for spacecraft sent into deep space, such as outside the Earth-Moon system, and a heliocentric system is preferred.

### **Motion with Respect to Other Planets**

The blockset uses the standard WGS-84 geoid to model the Earth. You can change the equatorial axis length, the flattening, and the rotation rate.

You can represent the motion of spacecraft with respect to any celestial body that is well approximated by an oblate spheroid by changing the spheroid size, flattening, and rotation rate. If the celestial body is rotating westward (retrogradely), make the rotation rate negative.

### **Coordinate Systems for Modeling**

Modeling aircraft and spacecraft is simplest if you use a coordinate system fixed in the body itself. In the case of aircraft, the forward direction is modified by the presence of wind, and the craft motion through the air is not the same as its motion relative to the ground.

See “Equations of Motion” for further details on how the blockset implements body and wind coordinates.

### **Body Coordinates**

The noninertial body coordinate system is fixed in both origin and orientation to the moving craft. The craft is assumed to be rigid.

The orientation of the body coordinate axes is fixed in the shape of body.

- The  $x$ -axis points through the nose of the craft.
- The  $y$ -axis points to the right of the  $x$ -axis (facing in the pilot's direction of view), perpendicular to the  $x$ -axis.
- The  $z$ -axis points down through the bottom the craft, perpendicular to the  $xy$  plane and satisfying the RH rule.

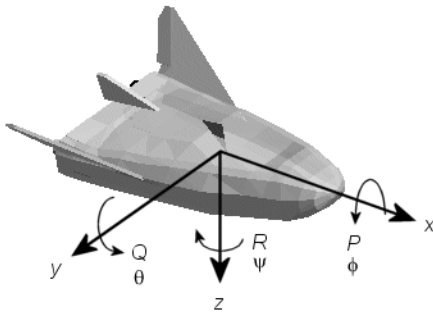
### **Translational Degrees of Freedom**

Translations are defined by moving along these axes by distances  $x$ ,  $y$ , and  $z$  from the origin.

### **Rotational Degrees of Freedom**

Rotations are defined by the Euler angles  $P$ ,  $Q$ ,  $R$  or  $\Phi$ ,  $\Theta$ ,  $\Psi$ . They are:

|                 |                           |
|-----------------|---------------------------|
| $P$ or $\Phi$   | Roll about the $x$ -axis  |
| $Q$ or $\Theta$ | Pitch about the $y$ -axis |
| $R$ or $\Psi$   | Yaw about the $z$ -axis   |



Unless otherwise specified, by default the software uses ZYX rotation order for Euler angles.

## Wind Coordinates

The noninertial wind coordinate system has its origin fixed in the rigid aircraft. The coordinate system orientation is defined relative to the craft velocity  $\mathbf{V}$ .

The orientation of the wind coordinate axes is fixed by the velocity  $\mathbf{V}$ .

- The  $x$ -axis points in the direction of  $\mathbf{V}$ .
- The  $y$ -axis points to the right of the  $x$ -axis (facing in the direction of  $\mathbf{V}$ ), perpendicular to the  $x$ -axis.
- The  $z$ -axis points perpendicular to the  $xy$  plane in whatever way needed to satisfy the RH rule with respect to the  $x$ - and  $y$ -axes.

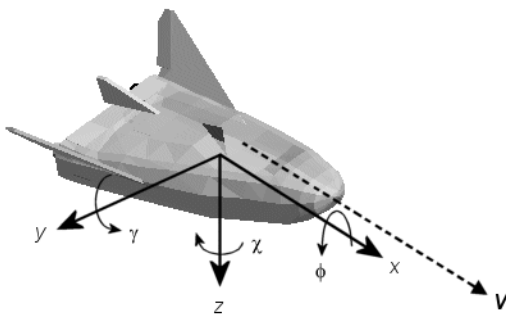
## Translational Degrees of Freedom

Translations are defined by moving along these axes by distances  $x$ ,  $y$ , and  $z$  from the origin.

## Rotational Degrees of Freedom

Rotations are defined by the Euler angles  $\Phi$ ,  $\gamma$ ,  $\chi$ :

|          |                                   |
|----------|-----------------------------------|
| $\Phi$   | Bank angle about the $x$ -axis    |
| $\gamma$ | Flight path about the $y$ -axis   |
| $\chi$   | Heading angle about the $z$ -axis |



Unless otherwise specified, by default the software uses ZYX rotation order for Euler angles.

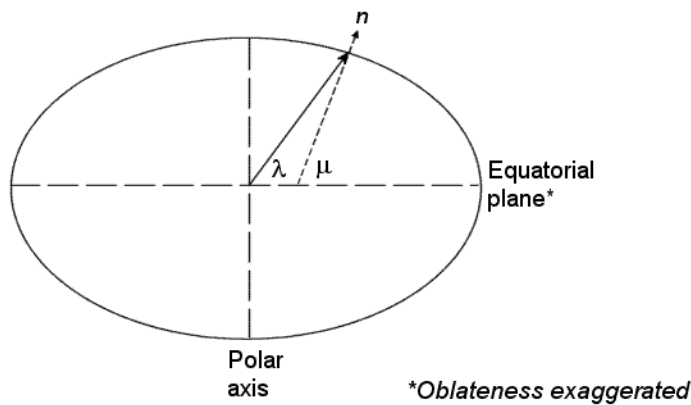
## Coordinate Systems for Navigation

Modeling aerospace trajectories requires positioning and orienting the aircraft or spacecraft with respect to the rotating Earth. Navigation coordinates are defined with respect to the center and surface of the Earth.

### Geocentric and Geodetic Latitudes

The geocentric latitude  $\lambda$  on the Earth surface is defined by the angle subtended by the radius vector from the Earth center to the surface point with the equatorial plane.

The geodetic latitude  $\mu$  on the Earth surface is defined by the angle subtended by the surface normal vector  $n$  and the equatorial plane.

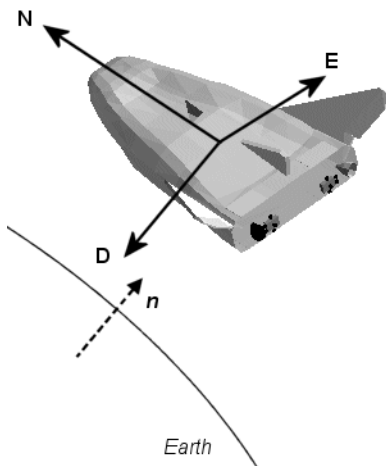


### NED Coordinates

The north-east-down (NED) system is a noninertial system with its origin fixed at the aircraft or spacecraft center of gravity. Its axes are oriented along the geodetic directions defined by the Earth surface.

- The  $x$ -axis points north parallel to the geoid surface, in the polar direction.
- The  $y$ -axis points east parallel to the geoid surface, along a latitude curve.
- The  $z$ -axis points downward, toward the Earth surface, antiparallel to the surface outward normal  $n$ .

Flying at a constant altitude means flying at a constant  $z$  above the Earth surface.

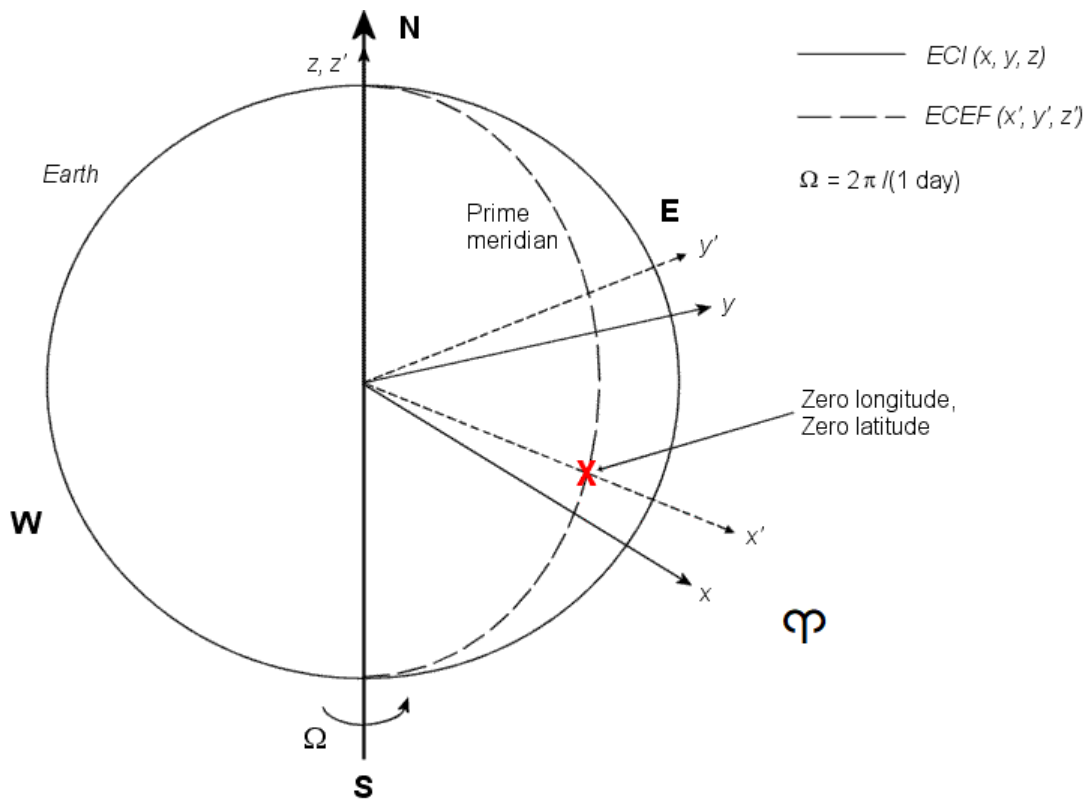


### **ECI Coordinates**

The Earth-centered inertial (ECI) system is non-rotating. For most applications, assume this frame to be inertial, although the equinox and equatorial plane move very slightly over time. The ECI system is considered to be truly inertial for high-precision orbit calculations when the equator and equinox are defined at a particular epoch (e.g. J2000). Aerospace functions and blocks that use a particular realization of the ECI coordinate system provide that information in their documentation. The ECI system origin is fixed at the center of the Earth (see figure).

- The  $x$ -axis points towards the vernal equinox (First Point of Aries  $\Upsilon$ ).
- The  $y$ -axis points 90 degrees to the east of the  $x$ -axis in the equatorial plane.
- The  $z$ -axis points northward along the Earth rotation axis.





## Earth-Centered Coordinates

### ECEF Coordinates

The Earth-center, Earth-fixed (ECEF) system is noninertial and rotates with the Earth. Its origin is fixed at the center of the Earth (see preceding figure).

- The  $x'$ -axis points towards the intersection of Earth equatorial plane and the Greenwich Meridian.
- The  $y'$ -axis points 90 degrees to the east of the  $x'$ -axis in the equatorial plane.
- The  $z'$ -axis points northward along the Earth rotation axis.

## Coordinate Systems for Display

Several display tools are available for use with the Aerospace Blockset product. Each has a specific coordinate system for rendering motion.

### MATLAB Graphics Coordinates

See the “Axes Appearance” (MATLAB) for more information about the MATLAB Graphics coordinate axes.

MATLAB Graphics uses this default coordinate axis orientation:

- The  $x$ -axis points out of the screen.
- The  $y$ -axis points to the right.

- The z-axis points up.

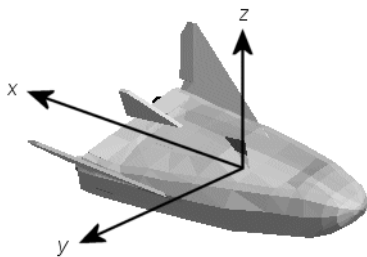
### FlightGear Coordinates

FlightGear is an open-source, third-party flight simulator with an interface supported by the blockset.

- “Work with the Flight Simulator Interface” on page 2-20 discusses the blockset interface to FlightGear.
- See the FlightGear documentation at [www.flightgear.org](http://www.flightgear.org) for complete information about this flight simulator.

The FlightGear coordinates form a special body-fixed system, rotated from the standard body coordinate system about the y-axis by -180 degrees:

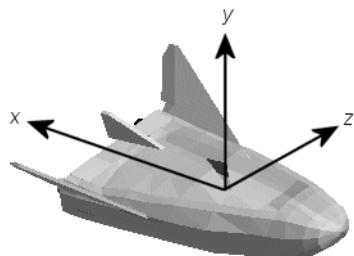
- The x-axis is positive toward the back of the vehicle.
- The y-axis is positive toward the right of the vehicle.
- The z-axis is positive upward, e.g., wheels typically have the lowest z values.



### AC3D Coordinates

AC3D is a low-cost, widely used, geometry editor available from <https://www.inivis.com/>. Its body-fixed coordinates are formed by inverting the three standard body coordinate axes:

- The x-axis is positive toward the back of the vehicle.
- The y-axis is positive upward, e.g., wheels typically have the lowest y values.
- The z-axis is positive to the left of the vehicle.



## References

- [1] *Recommended Practice for Atmospheric and Space Flight Vehicle Coordinate Systems*, R-004-1992, ANSI/AIAA, February 1992.

- [2] Rogers, R. M., *Applied Mathematics in Integrated Navigation Systems*, AIAA, Reston, Virginia, 2000.
- [3] Sobel, D., *Longitude*, Walker & Company, New York, 1995.
- [4] Stevens, B. L., and F. L. Lewis, *Aircraft Control and Simulation*, 2nd ed., *Aircraft Control and Simulation*, Wiley-Interscience, New York, 2003.
- [5] Thomson, W. T., *Introduction to Space Dynamics*, John Wiley & Sons, New York, 1961/Dover Publications, Mineola, New York, 1986.

## **See Also**

## **External Websites**

- World Geodetic System 1984 (WGS 84)
- <https://www.inivis.com/>

## Flight Simulator Interface

| In this section...   |
|--|
| “About the FlightGear Interface” on page 2-16                |
| “Supported FlightGear Versions” on page 2-16                 |
| “Obtain FlightGear” on page 2-17                             |
| “Configure Your Computer for FlightGear” on page 2-17        |
| “FlightGear and Video Cards in Windows Systems” on page 2-17 |
| “Install and Start FlightGear” on page 2-18                  |
| “Install Additional FlightGear Scenery” on page 2-18         |



### About the FlightGear Interface

The Aerospace Blockset product supports an interface to the third-party FlightGear flight simulator, open-source software available through a GNU General Public License (GPL). The FlightGear flight simulator interface included with the blockset is a unidirectional transmission link from the Simulink interface to FlightGear using the FlightGear published `net_fdm` binary data exchange protocol. Data is transmitted via UDP network packets to a running instance of FlightGear. The blockset supports multiple standard binary distributions of FlightGear. See “Run FlightGear with Simulink Models” on page 2-24 for interface details.

FlightGear is a separate software entity not created, owned, or maintained by MathWorks.

- To report bugs in or request enhancements to the Aerospace Blockset FlightGear interface, use the [form](#).
- To report bugs or request enhancements to FlightGear itself, visit [FlightGear website](#).

### Supported FlightGear Versions

The Aerospace Blockset product supports these FlightGear versions:

- v2019.1
- v2018.3
- v2018.2
- v2018.1
- v2017.3
- v2017.1
- v2016.3
- v2016.1
- v3.4
- v3.2
- v3.0
- v2.12
- v2.10

- v2.8
- v2.6
- v2.4
- v2.0

If you are using a FlightGear version older than 2.0, update your FlightGear installation to a supported version. When you open the model, the software returns a warning or error. Obtain updated FlightGear software from <https://www.flightgear.org> in the download area.

## Obtain FlightGear

You can obtain FlightGear from the FlightGear website in the download area or by ordering CDs from FlightGear. The download area contains extensive documentation for installation and configuration. Because FlightGear is an open-source project, source downloads are also available for customizing and porting to custom environments.

## Configure Your Computer for FlightGear

To use FlightGear, you must have a high-performance graphics card with stable drivers. For more information, see the FlightGear CD distribution or the hardware requirements and documentation areas of the FlightGear website.

FlightGear performance and stability can be sensitive to computer video cards, driver versions, and driver settings. You need OpenGL<sup>®</sup> support with hardware acceleration activated. Without proper setup, performance can drop from about a 30 frames-per-second (fps) update rate to less than 1 fps. If your system allows you to update OpenGL settings, modify them to improve performance.

### Graphics Recommendations for Windows

For Windows systems, use the following graphics recommendations:

- A graphics card with acceptable OpenGL performance (as outlined at the FlightGear website).
- The latest tested and stable driver release for your video card. Test the driver thoroughly on a few computers before deploying to others.

For more information, see [FlightGear Hardware Recommendations](#).

### Setup on Linux, Macintosh, and Other Platforms

FlightGear distributions are available for Linux, Macintosh, and other platforms from the FlightGear website, <https://www.flightgear.org>. Installation on these platforms, like Windows, requires careful configuration of graphics cards and drivers. Consult the documentation and hardware requirements sections at the FlightGear website.

## FlightGear and Video Cards in Windows Systems

Your computer built-in video card, such as NVIDIA<sup>®</sup> cards, can conflict with FlightGear shaders. Consider this workaround:

- Disable the FlightGear shaders by selecting the Generate Run Script block **Disable FlightGear shader options** check box.

## Install and Start FlightGear

The extensive FlightGear documentation guides you through the installation in detail. Consult the following:

- Documentation section of the FlightGear website for installation instructions: <https://www.flightgear.org>.
- Hardware recommendations section of the FlightGear website.
- MATLAB system requirements.

Keep the following points in mind:

- Configure your computer graphics card before you install FlightGear. See the preceding section, “Configure Your Computer for FlightGear” on page 2-17.
- Shut down all running applications (including the MATLAB interface) before installing FlightGear.
- Install FlightGear in a folder path name composed of ASCII characters.
- MathWorks tests indicate that the operational stability of FlightGear is especially sensitive during startup. It is best not to move, resize, mouse over, overlap, or cover up the FlightGear window until the initial simulation scene appears after the startup splash screen fades out.

Aerospace Blockset supports FlightGear on several platforms. This table lists the properties to consider before you start to use FlightGear.

| FlightGear Property      | Folder Description              | Platforms      | Typical Location  |
|--------------------------|---------------------------------|----------------|---|
| FlightGearBase-Directory | FlightGear installation folder. | Windows 64-bit | C:\Program Files\FlightGear (default)   |
|                          |                                 | Linux          | Folder into which you installed FlightGear                                      |
|                          |                                 | Mac            | /Applications (folder to which you dragged the FlightGear icon)                 |
| GeometryModelName        | Model geometry folder           | Windows 64-bit | C:\Program Files\FlightGear\data\Aircraft\HL20 (default)                        |
|                          |                                 | Linux          | \$FlightGearBaseDirectory/data/Aircraft/HL20                                    |
|                          |                                 | Mac            | \$FlightGearBaseDirectory/-FlightGear.app/Contents/Resources/data/Aircraft/HL20 |

## Install Additional FlightGear Scenery

When you install the FlightGear software, the installation provides a basic level of scenery files. The FlightGear documentation guides you through installing scenery as part the general FlightGear installation.

If you need to install more FlightGear scenery files, see the instructions at <https://www.flightgear.org>. The instructions describe how to install the additional scenery in a default location. MathWorks® recommends that you follow those instructions.

If you install additional scenery in a nonstandard location, you may need to update the `FG_SCENERY` environment variable in the script output from the Generate Run Script block to include the new path. For a description of the `FG_SCENERY` variable, see the documentation at <https://www.flightgear.org>.

If you do not download scenery in advance, you can direct FlightGear to download it automatically during simulation by selecting the Generate Run Script block **Install FlightGear scenery during simulation (requires Internet connection)** check box.

For Windows systems, you may encounter an error message while launching FlightGear with the `InstallScenery` option enabled:

```
Error creating directory: No such file or directory
```

This error likely indicates that your default FlightGear download folder is not writeable, the path cannot be resolved, or the path contains UNC path names. To work around the issue, edit the `runfg.bat` file to specify a new folder path to store the scenery data:

- 1 Edit `runfg.bat`.
- 2 To the list of command options, append `--download-dir=` and specify a folder to which to download the scenery data. For example:

```
--download-dir=C:\Users\user1\Documents\FlightGear
```

All data downloaded during this FlightGear session is saved to the specified directory. To avoid downloading duplicate scenery data, use the same directory in succeeding FlightGear sessions

- 3 To open FlightGear, run `runfg.bat`.

---

**Note** Each time that you run the Generate Run Script block, it creates a new script. It overwrites any edits that you have added.

---

## Work with the Flight Simulator Interface

### In this section...

“Introduction” on page 2-20

“About Aircraft Geometry Models” on page 2-20

“Work with Aircraft Geometry Models” on page 2-22

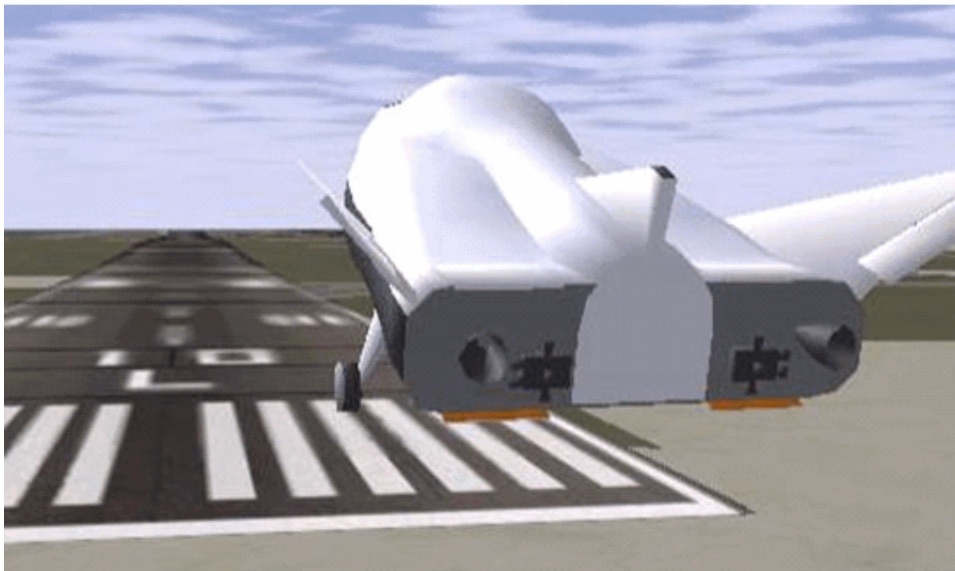
“Run FlightGear with Simulink Models” on page 2-24

“Run the HL-20 Example with FlightGear” on page 2-29

“Send and Receive Data” on page 2-31

### Introduction

Use this section to learn how to use the FlightGear flight simulator and the Aerospace Blockset software to visualize your Simulink aircraft models. If you have not yet installed FlightGear, see “Flight Simulator Interface” on page 2-16 first.



**Simulink Driven HL-20 Model in a Landing Flare at KSFC**

### About Aircraft Geometry Models

Before you can visualize your aircraft dynamics, you need to create or obtain an aircraft model file compatible with FlightGear. This section explains how to do this.

#### Aircraft Geometry Editors and Formats

You have a competitive choice of over twelve 3-D geometry file formats supported by FlightGear.

Currently, the most popular 3-D geometry file format is the AC3D format, which has the suffix \*.ac. AC3D is a low-cost geometry editor available from [www.ac3d.org](http://www.ac3d.org).



## Aircraft Model Structure and Requirements

Aircraft models are contained in the *FlightGearRoot/data/Aircraft/* folder and subfolders. A complete aircraft model must contain a folder linked through the required aircraft master file named *model-set.xml*.

All other model elements are optional. This is a partial list of the optional elements you can put in an aircraft data folder:

- Vehicle objects and their shapes and colors
- Vehicle objects' surface bitmaps
- Variable geometry descriptions
- Cockpit instrument 3-D models
- Vehicle sounds to tie to events (e.g., engine, gear, wind noise)
- Flight dynamics model
- Simulator views
- Submodels (independently movable items) associated with the vehicle

Model behavior reverts to defaults when these elements are not used. For example,

- Default sound: no vehicle-related sounds are emitted.
- Default instrument panel: no instruments are shown.

Models can contain some, all, or even none of the above elements. If you always run FlightGear from the cockpit view, the aircraft geometry is often secondary to the instrument geometries.

A how-to document for including optional elements is included in the FlightGear documentation at:

[http://wiki.flightgear.org/Howto:3D\\_Aircraft\\_Models](http://wiki.flightgear.org/Howto:3D_Aircraft_Models)

## Required Flight Dynamics Model Specification

The flight dynamics model (FDM) specification is a required element for an aircraft model. To set the Simulink software as the source of the flight dynamics model data stream for a given geometry model, you put this line in *data/Aircraft/model/model-set.xml*:

```
<flight-model>network</flight-model>
```

## Obtain and Modify Existing Aircraft Models

You can quickly build models from scratch by referencing instruments, sounds, and other optional elements from existing FlightGear models. Such models provide examples of geometry, dynamics, instruments, views, and sounds. It is simple to copy an aircraft folder to a new name, rename the *model-set.xml* file, modify it for network flight dynamics, and then run FlightGear with the `-aircraft` flag set to the name in *model-set.xml*.

Many existing 3-D aircraft geometry models are available for use with FlightGear. Visit the download area of <https://www.flightgear.org> to see some of the aircraft models available. Additional models can be obtained via Web search. Search key words such as “flight gear aircraft model” are a good starting point. Be sure to comply with copyrights when distributing these files.

## Hardware Requirements for Aircraft Geometry Rendering

When creating your own geometry files, keep in mind that your graphics card can efficiently render a limited number of surfaces. Some cards can efficiently render fewer than 1000 surfaces with bitmaps and specular reflections at the nominal rate of 30 frames per second. Other cards can easily render on the order of 10,000 surfaces.

If your performance slows while using a particular geometry, gauge the effect of geometric complexity on graphics performance by varying the number of aircraft model surfaces. An easy way to check this is to replace the full aircraft geometry file with a simple shape, such as a single triangle, then test FlightGear with this simpler geometry. If a geometry file is too complex for smooth display, use a 3-D geometry editor to simplify your model by reducing the number of surfaces in the geometry.

## Work with Aircraft Geometry Models

Once you have obtained, modified, or created an aircraft data file, you need to put it in the correct folder for FlightGear to access it.

### Import Aircraft Models into FlightGear

To install a compatible model into FlightGear, use one of the following procedures. Choose the one appropriate for your platform. This section assumes that you have read “Install and Start FlightGear” on page 2-18.

If your platform is Windows:

- 1 Go to your installed FlightGear folder. Open the data folder, then the Aircraft folder:  
`\FlightGear\data\Aircraft\`.
- 2 Make a subfolder *model*\ here for your aircraft data.
- 3 Put *model-set.xml* in that subfolder, plus any other files needed.

It is common practice to make subdirectories for the vehicle geometry files (`\model\`), instruments (`\instruments\`), and sounds (`\sounds\`).

If your platform is Linux:

- 1 Go to your installed FlightGear directory. Open the data directory, then the Aircraft directory:  
`$FlightGearBaseDirectory/data/Aircraft/`.
- 2 Make a subdirectory *model/* here for your aircraft data.
- 3 Put *model-set.xml* in that subdirectory, plus any other files needed.

It is common practice to make subdirectories for the vehicle geometry files (`/model/`), instruments (`/instruments/`), and sounds (`/sounds/`).

If your platform is Mac:

- 1 Open a terminal.
- 2 Go to your installed FlightGear folder. Open the data folder, then the Aircraft folder:  
`$FlightGearBaseDirectory/FlightGear.app/Contents/Resources/data/Aircraft/`
- 3 Make a subfolder *model/* here for your aircraft data.
- 4 Put *model-set.xml* in that subfolder, plus any other files needed.

It is common practice to make subdirectories for the vehicle geometry files (*/model/*), instruments (*/instruments/*), and sounds (*/sounds/*).

### Example: Animate Vehicle Geometries

This example illustrates how to prepare hinge line definitions for animated elements such as vehicle control surfaces and landing gear. To enable animation, each element must be a named entity in a geometry file. The resulting code forms part of the HL20 lifting body model presented in “Run the HL-20 Example with FlightGear” on page 2-29.

- 1 The standard body coordinates used in FlightGear geometry models form a right-handed system, rotated from the standard body coordinate system in *Y* by -180 degrees:

- *X* = positive toward the back of the vehicle
- *Y* = positive toward the right of the vehicle
- *Z* = positive is up, e.g., wheels typically have the lowest *Z* values.

See “About Aerospace Coordinate Systems” on page 2-8 for more details.

- 2 Find two points that lie on the desired named-object hinge line in body coordinates and write them down as *XYZ* triplets or put them into a MATLAB calculation like this:

```
a = [2.98, 1.89, 0.53];
b = [3.54, 2.75, 1.46];
```

- 3 Calculate the difference between the points:

```
pdiff = b - a
pdiff =
    0.5600    0.8600    0.9300
```

- 4 The hinge point is either of the points in step 2 (or the midpoint as shown here):

```
mid = a + pdiff/2
mid =
    3.2600    2.3200    0.9950
```

- 5 Put the hinge point into the animation scope in *model-set.xml*:

```
<center>
  <x-m>3.26</x-m>
  <y-m>2.32</y-m>
  <z-m>1.00</z-m>
</center>
```

- 6 Use the difference from step 3 to define the relative motion vector in the animation axis:

```
<axis>
  <x>0.56</x>
  <y>0.86</y>
  <z>0.93</z>
</axis>
```

- 7 Put these steps together to obtain the complete hinge line animation used in the HL20 example model:

```
<animation>
  <type>rotate</type>
  <object-name>RightAileron</object-name>
  <property>/surface-positions/right-aileron-pos-norm</property>
  <factor>30</factor>
  <offset-deg>0</offset-deg>
```

```
<center>
  <x-m>3.26</x-m>
  <y-m>2.32</y-m>
  <z-m>1.00</z-m>
</center>
<axis>
  <x>0.56</x>
  <y>0.86</y>
  <z>0.93</z>
</axis>
</animation>
```

## Run FlightGear with Simulink Models

To run a Simulink model of your aircraft and simultaneously animate it in FlightGear with an aircraft data file *model-set.xml*, you need to configure the aircraft data file and modify your Simulink model with some new blocks.

These are the main steps to connecting and using FlightGear with the Simulink software:

- “Set the Flight Dynamics Model to Network in the Aircraft Data File” on page 2-24 explains how to create the network connection you need.
- “Obtain the Destination IP Address” on page 2-24 starts by determining the IP address of the computer running FlightGear.
- “Send Simulink Data to FlightGear” on page 2-31 shows how to add and connect interface and pace blocks to your Simulink model.
- “Create a FlightGear Run Script” on page 2-25 shows how to write a FlightGear run script compatible with your Simulink model.
- “Start FlightGear” on page 2-27 guides you through the final steps to making the Simulink software work with FlightGear.
- “Improve Performance” on page 2-28 helps you speed your model up.
- “Run FlightGear and Simulink Software on Different Computers” on page 2-29 explains how to connect a simulation from the Simulink software running on one computer to FlightGear running on another computer.

### Set the Flight Dynamics Model to Network in the Aircraft Data File

Be sure to:

- Remove any pre-existing flight dynamics model (FDM) data from the aircraft data file.
- Indicate in the aircraft data file that its FDM is streaming from the network by adding this line:

```
<flight-model>network</flight-model>
```

### Obtain the Destination IP Address

You need the destination IP address for your Simulink model to stream its flight data to FlightGear.

- If you know your computer name, enter at the MATLAB command line:

```
java.net.InetAddress.getByName('www.mathworks.com')
```

- If you are running FlightGear and the Simulink software on the same computer, get your computer name by entering at the MATLAB command line:

```
java.net.InetAddress.getLocalHost
```

- If you are working in Windows, get your computer IP address by entering at the DOS prompt:

```
ipconfig /all
```

Examine the IP address entry in the resulting output. There is one entry per Ethernet device.

### **Create a FlightGear Run Script**

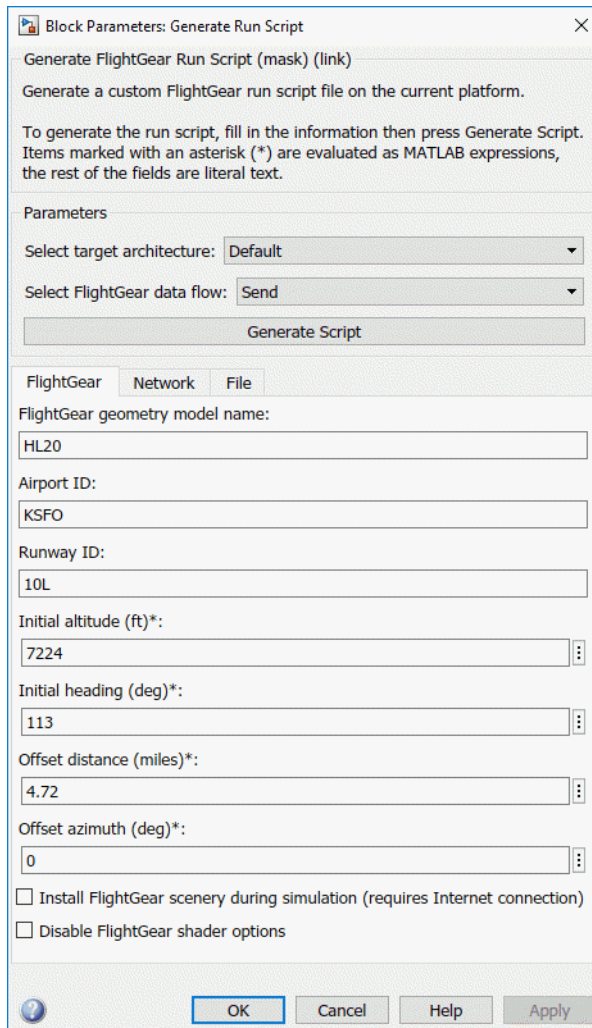
To start FlightGear with the desired initial conditions (location, date, time, weather, operating modes), it is best to create a run script by “Use the Generate Run Script Block” on page 2-25 or “Use the Interface Provided with FlightGear” on page 2-27.

If you make separate run scripts for each model you intend to link to FlightGear and place them in separate directories, run the appropriate script from the MATLAB interface just before starting your Simulink model.

### **Use the Generate Run Script Block**

The easiest way to create a run script is by using the Generate Run Script block. Use the following procedure:

- 1** Open the Flight Simulator Interfaces sublibrary.
- 2** Create a new Simulink model or open an existing model.
- 3** Drag a Generate Run Script block into the Simulink diagram.
- 4** Double-click the Generate Run Script block. Its dialog opens. Observe the three panes, **FlightGear**, **Network**, and **File**.



- 5 In the **Output file name** parameter of the **File** tab, type the name of the output file. This name should be the name of the command you want to use to start FlightGear with these initial parameters. Use the appropriate file extension:

| Platform        | Extension |
|-----------------|-----------|
| Windows         | .bat      |
| Linux and macOS | .sh       |

For example, if your file name is `runfg.bat`, use the `runfg` command to execute the run script and start FlightGear.

- 6 In the **FlightGear base directory** parameter of the **File** tab, specify the name of your FlightGear installation folder.
- 7 In the **FlightGear geometry model name** parameter of the **File** tab, specify the name of the subfolder, in the `FlightGear/data/Aircraft` folder, containing the desired model geometry.
- 8 Specify the initial conditions as needed.
- 9 Click the **Generate Script** button at the top of the **Parameters** area.

The Aerospace Blockset software generates the run script, and saves it in your MATLAB working folder under the file name that you specified in the **File > Output file name** field.

**10** Select or clear these check boxes and

- To direct FlightGear to automatically install required scenery while the simulator is running — Select **Install FlightGear scenery during simulation (requires Internet connection)**. For Windows systems, you may encounter an error message while launching FlightGear with this option enabled. For more information, see “Install Additional FlightGear Scenery” on page 2-18.
- To disable FlightGear shader options — Select **Disable FlightGear shader options**.

**11** Repeat steps 5 through 10 to generate other run scripts, if needed.

**12** Click **OK** to close the dialog box. You do not need to save the Generate Run Script block with the Simulink model.

The Generate Run Script block saves the run script as a text file in your working folder. This is an example of the contents of a run script file:

```
>> cd C:\Applications\FlightGear-2019.1
>> SET FG_ROOT=C:\Applications\FlightGear-2019.1\data
>> cd \bin\
>> fgfs --aircraft=HL20 --fdm=network,localhost,5501,5502,5503
--fog-fastest --disable-clouds --start-date-lat=2004:06:01:09:00:00
--disable-sound --in-air --enable-freeze --airport=KSF0 --runway=10L
--altitude=7224 --heading=113 --offset-distance=4.72 --offset-azimuth=0
```

### Use the Interface Provided with FlightGear

The FlightGear launcher GUI (part of FlightGear, not the Aerospace Blockset product) lets you build simple and advanced options into a visible FlightGear run command.

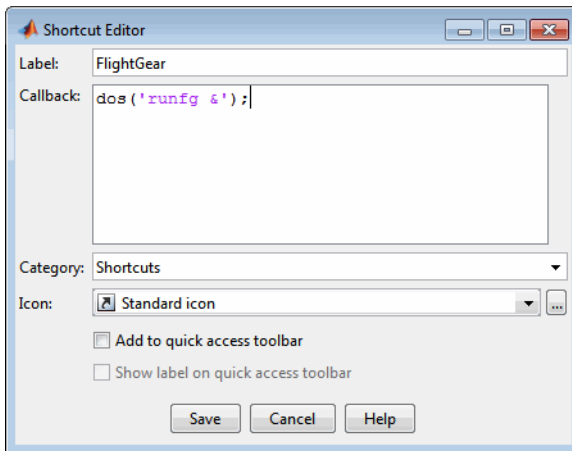
### Start FlightGear

If your computer has enough computational power to run both the Simulink software and FlightGear at the same time, a simple way to start FlightGear on a Windows system is to create a MATLAB desktop button containing the following command to execute a run script like the one created above:

```
system('runfg &')
```

To create a desktop button:

- 1** In the MATLAB Command Window, select **Shortcuts** and click **New Shortcut**. The **Shortcut Editor** dialog opens.
- 2** Set the **Label**, **Callback**, **Category**, and **Icon** fields as shown in the following figure.



- 3 Click **Save**.

The **FlightGear** button appears in your MATLAB desktop. If you click it, the output file, for example `runfg.bat`, runs in the current folder.

Once you have completed the setup, start FlightGear and run your model:

- 1 Make sure your model is in a writable folder. Open the model, and update the diagram. This step ensures that any referenced block code is compiled and that the block diagram is compiled before running. Once you start FlightGear, it uses all available processor power while it is running.
- 2 Click the **FlightGear** button or run the FlightGear run script manually.
- 3 When FlightGear starts, it displays the initial view at the initial coordinates specified in the run script. If you are running the Simulink software and FlightGear on different computers, arrange to view the two displays at the same time.
- 4 Now begin the simulation and view the animation in FlightGear.

### Improve Performance

If your Simulink model is complex and cannot run at the aggregate rate needed for the visualization, you might need to

- Use the Accelerator mode in Simulink (“Perform Acceleration” (Simulink).)
- Free up processor power by running the Simulink model on one computer and FlightGear on another computer. Use the **Destination IP Address** parameter of the Send net\_fdm Packet to FlightGear block to specify the network address of the computer where FlightGear is running.
- Simulate the Simulink model first, then save the resulting translations (x-axis, y-axis, z-axis) and positions (latitude, longitude, altitude), and use the FlightGear Animation object in Aerospace Toolbox to visualize this data.

---

**Tip** If FlightGear uses more computer resources than you want, you can change its scheduling priority to a lesser one. For example, see commands like Windows `start` and Linux `nice` or their equivalents.

---



## Run FlightGear and Simulink Software on Different Computers

It is possible to simulate an aerospace system in the Simulink environment on one computer (the source) and use its simulation output to animate FlightGear on another computer (the target). The steps are similar to those already explained, with certain modifications.

- 1 Obtain the IP address of the computer running FlightGear. See “Obtain the Destination IP Address” on page 2-24.
- 2 Enter this target computer IP address in the Send net\_fdm Packet to FlightGear block. See “Send Simulink Data to FlightGear” on page 2-31.
- 3 Update the Generate Run Script block in your model with the target computer FlightGear base folder. Regenerate the run script to reflect the target computer separate identity.  
  
See “Create a FlightGear Run Script” on page 2-25.
- 4 Copy the generated run script to the target computer. Start FlightGear there. See “Start FlightGear” on page 2-27.
- 5 If you want to also receive data from FlightGear, use the Receive net\_ctrl Packet from FlightGear block. Enter the IP address of the computer running FlightGear in the **Origin IP address** parameter.
- 6 Update the run script for the receive data. Use the Generate Run Script block to regenerate the run script.
- 7 Start your Simulink model on the source computer. FlightGear running on the target displays the simulation motion.

## Run the HL-20 Example with FlightGear

The Aerospace Blockset software contains an example model of the HL-20 lifting body that uses the FlightGear interface and projects. This example illustrates many features of the Aerospace Blockset software. It also contains a Variant Subsystem block that you can use to specify the data source for the simulation. You might want to use the Variant Subsystem block to change the terrain data source or if you do not want to use FlightGear but still want to simulate the model.

To install and configure FlightGear before attempting to simulate this model, see “Flight Simulator Interface” on page 2-16. Also, before attempting to simulate this model, read “Install and Start FlightGear” on page 2-18.

---

**Note** Step 2 of this example copies the preconfigured geometries for the HL-20 simulation from *projectroot*\support to *FlightGear*\data\Aircraft\. It requires that you have system administrator privileges for your machine. If you do not have these privileges, manually copy these files, depending on your platform.

### Windows

Copy the HL20 folder from *projectroot*\support folder to *FlightGear*\data\Aircraft\ folder. This folder contains the preconfigured geometries for the HL-20 simulation and HL20-set.xml. The file *projectroot*\support\HL20\Models\HL20.xml defines the geometry.

For Windows platforms, start the MATLAB app with administrator privileges. For example, in the Start menu, right click the MATLAB app, then select **Run as administrator**.

For more information, see “Import Aircraft Models into FlightGear” on page 2-22.

Linux

Copy the HL20 directory from *projectroot/support* directory to *\$FlightGearBaseDirectory/data/Aircraft/* directory. This directory contains the preconfigured geometries for the HL-20 simulation and *HL20-set.xml*. The file *projectroot/support/HL20/Models/HL20.xml* defines the geometry.

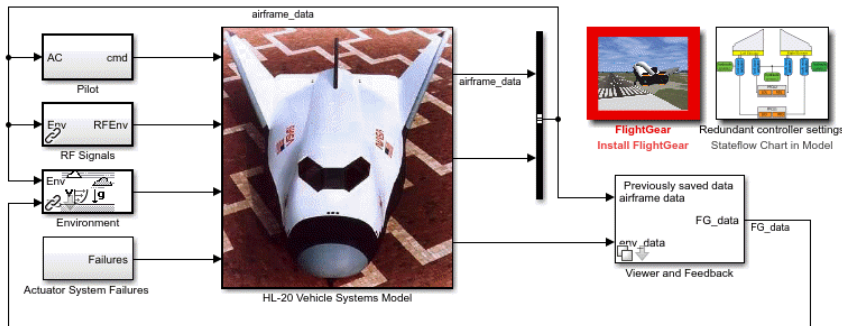
For more about this step, see “Import Aircraft Models into FlightGear” on page 2-22.

Mac

Copy the HL20 folder from *projectroot/support* folder to *\$FlightGearBaseDirectory/FlightGear.app/Contents/Resources/data/Aircraft/* folder. This folder contains the preconfigured geometries for the HL-20 simulation and *HL20-set.xml*. The file *projectroot/support/HL20/Models/HL20.xml* defines the geometry.

For more about this step, see “Import Aircraft Models into FlightGear” on page 2-22.

- 1 Start the MATLAB interface. Open the example either by entering *asbhl20* in the MATLAB Command Window or by finding the example entry (HL-20 with FlightGear Interface) in the Aerospace Blockset Examples page. The project for the model starts and the model opens.



HL-20 Example, version 2.0.425  
Aerodynamic model from Jackson E. B., Cruz C. L., "Preliminary Subsonic Aerodynamic Model for Simulation Studies of the HL-20 Lifting Body", NASA TM4302, August 1992.

How to run the HL20 model:  
See the Aerospace Blockset User's Guide for instructions to set up FlightGear or click on the "FlightGear" block and follow the instructions.  
Note: If FlightGear is not installed, double-click the "Viewer and Feedback" block and select an option: "Previously Saved Data" (for saved data from a previous simulation with FlightGear in the loop), "Signal Editor" (for an existing and editable signal), "Constants" (for a set of constant values), or "Spreadsheet Data" (for data saved in a spreadsheet from a previous simulation with FlightGear in the loop).

- 2 If this is your first time running FlightGear for this model, you need to create and run a customized FlightGear run script. You can do this with one of the following:
  - In the model, double-click the Install FlightGear block and follow the steps in the block. Initially, this block is red. As you follow the steps outlined in the block, the block mask changes.

To start FlightGear for the model, click **Launch HL20 in FlightGear**.

- 3 Now start the simulation and view the animation in FlightGear.

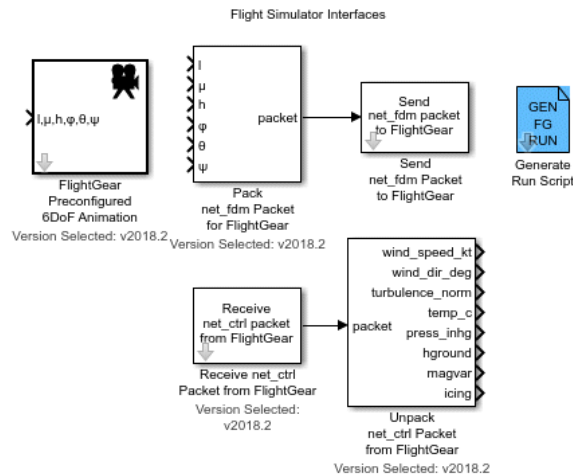
**Note** With the FlightGear window in focus, press the **V** key to alternate between the different aircraft views: cockpit view, helicopter view, chase view, and so on.

## Send and Receive Data

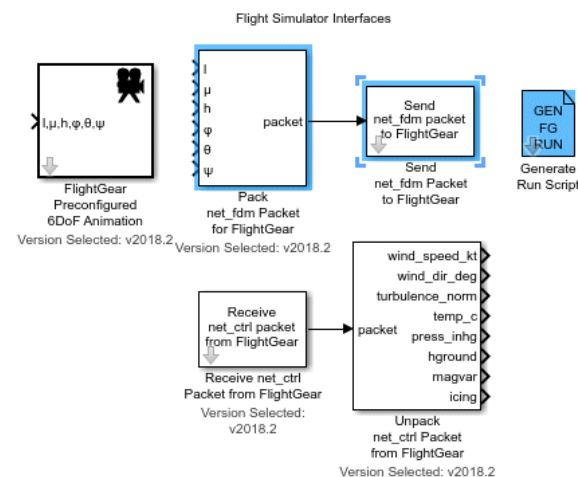
You can send and receive data between a Simulink model and a running FlightGear Flight Simulator.

### Send Simulink Data to FlightGear

The easiest way to connect your model to FlightGear with the blockset is to use the FlightGear Preconfigured 6DoF Animation block:

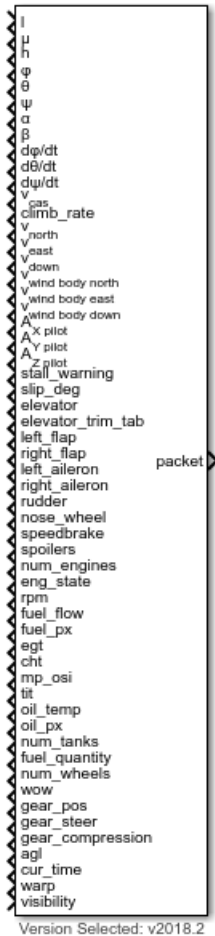


The FlightGear Preconfigured 6DoF Animation block is a subsystem containing the Pack net\_fdm Packet for FlightGear and Send net\_fdm Packet to FlightGear blocks:



These blocks transmit data from a model to a FlightGear session. The blocks are separate for maximum flexibility and compatibility.

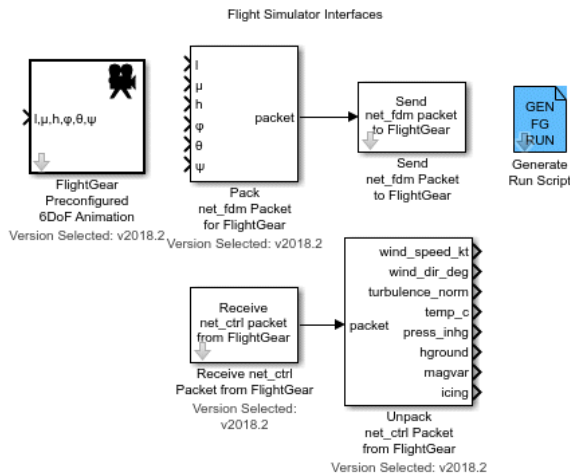
- The Pack net\_fdm Packet for FlightGear block formats a binary structure compatible with FlightGear from model inputs. In the default configuration, the block displays only the 6DoF ports, but you can configure the full FlightGear interface supporting more than 50 distinct signals from the block dialog box:



- The Send net\_fdm Packet to FlightGear block transmits this packet via UDP to the specified IP address and port where a FlightGear session awaits an incoming datastream. Use the IP address you found in “Obtain the Destination IP Address” on page 2-24.
- The Simulation Pace block slows the simulation so that its aggregate run rate is 1 second of simulation time per second of clock time. You can also use it to specify other ratios of simulation time to clock time.

**Send FlightGear Data to Model**

To increase the accuracy of your model simulation, you might want to send FlightGear environment variables to the Simulink model. Use the following blocks:



- Receive net\_ctrl Packet from FlightGear — Receives a network control and environment data packet net\_ctrl from either the simulation of a Simulink model in the FlightGear simulator, or from a FlightGear session.
- Unpack net\_ctrl Packet from FlightGear — Unpacks net\_ctrl variable packets received from FlightGear and makes them available for the Simulink environment.
- Generate Run Script — Generates a customized FlightGear run script on the current platform.

For an example of how to use these blocks to send data to a Simulink model, see “HL-20 Project with Optional FlightGear Interface” on page 6-37.

These blocks use UDP to transfer data from FlightGear to the Simulink environment. Note the following:

- When a host and target are Windows or Linux platforms, you can use any combination of Windows or Linux platforms for the host and target.
- When a host or target is a Mac platform, use only Mac platforms for both the host and target.

### Macintosh Platform and FlightGear Version 2.6 or Later

On a Macintosh system with FlightGear v2.6 or later, you might see unexpected results (for example, very large or very small data values) if your model uses the following blocks:

- Receive net\_ctrl Packet from FlightGear
- Unpack net\_ctrl Packet from FlightGear

To work around this issue:

- 1 In the model, change the **FlightGear version** parameter to v2.4 for both blocks.
- 2 Save and rerun the model.

The results should now be as expected.

## Projects Template for Flight Simulation Applications

### Flight Simulation Applications

Use projects (Simulink) to help organize large flight simulation modeling projects and makes it easier to share projects with others. This template provides a framework for the collaborative development of a flight simulation application. You can customize this project structure for specific applications.

---

**Note** To successfully run this example, install a C/C++ compiler.

---

The Aerospace Blockset software supplies a projects template that you can use to create your own flight simulation application. This template uses variant subsystems, model variants, and referenced models to implement flight simulation application components such as:

- An airframe that contains a 6DOF equation of motion environment model and actuator dynamics
- An inertial measurement unit (IMU) sensor model
- A visualization subsystem oriented for FlightGear
- A model of the nonlinear dynamics of the airframe
- A model of the linear dynamics of the airframe

### Download the Flight Simulation Template

- 1 From the Simulink Start Page, select **Flight Simulation**.
- 2 In the Create Project window, in **Name**, enter a project name, for example `FlightSimProj`.
- 3 In **Folder**, enter a project folder or browse to the folder to contain the project, for example `FlightSimFolder`.
- 4 Click **OK**.

If the folder does not exist, the dialog prompts you to create it. Click **Yes**.

The software compiles the project, populates the project folders, and opens the main model, `flightSimulation`. All models and supporting files are in place for you to customize for your flight simulation application.

### Contents of the Project Template

The flight simulation project template contains the following folders

- **mainModels**

Contains the top-level simulation model, `flightSimulation`. This model opens on startup. This file contains the top-level blocks for the flight simulation environment. Simulink uses the Variant Subsystem, Model Variants, and Model blocks at this level to adapt to the different simulation conditions.

- The aircraft airframe can vary between a nonlinear and a linear approach.
- The commands to the aircraft can vary between a Signal Editor block, a joystick or a variable from the workspace.

- Sensors can vary between models that include sensor dynamics or feedthrough (no associated dynamics).
- Environment values can vary between state-dependent values (the values of temperature, pressure and so on depend on local position, latitude, etc.) or constant values that do not depend on state values.
- The Visualization subsystem provides hooks that let you work with the states. For example, you can visualize the states using FlightGear or they can be recorded in a variable in the workspace for further analysis. States can also be visualized using the Simulation Data Inspector.
- **libraries**
- Contains the libraries used by the models.
- **nonlinearAirframe**
- Contains a model of the nonlinear dynamics of the airframe.
  - A specific subsystem (AC model) that contains a placeholder for the dynamics of your aircraft model . The characteristics of this subsystem are:
    - Actuators and environment inputs. Actuators refer to generic signals that may affect the behavior of the aircraft (for example an electric signal in voltage that will change the position of the hydraulic actuator connected to a control surface such as an aileron).
    - Forces and moments outputs. Effective in the center of gravity of the aircraft in body axis.
  - A 6DOF Body Quaternion block that solves the differential equations of forces and moments to obtain the aircraft states.
- **linearAirframe**
- Contains the linear dynamics of the airframe and the model to obtain these linear dynamics. The example obtains these dynamics by linearizing the nonlinear model using the `trimLinearizeOpPoint` function and `trimNonlinearAirframe` model. This function uses “Simulink Control Design” software to perform the linearization. It performs linearization of the nonlinear model for a given set of known inputs and conditions. For further information regarding trim and linearization, see the Simulink Control Design™ documentation). The `trimLinearizeOpPoint` function stores the output in a MAT-file.
- **controller**
- Contains the models for the Flight Control System (FCS) and its design. These models contain referenced models for different controller architectures needed for the design of aircraft simulation.
- **src**
- Contains source code such as C code. For simulation, it also has two folders that contain S-functions for simulation. These S-functions map buses to vectors and vice versa for the linear airframe model. This mapping can be changed depending on the linearization scheme, and the set of inputs and outputs for the model. To edit the indices for the different signals, you can use the S-Function Builder block
- **tasks**
- Contains scripts to run the model. These scripts do not run continuously during the simulation process.

The folder also contains the non-virtual bus definitions for the states, environment, and sensor buses. These definitions, set the signals and characteristics that different elements in the simulation environment use. This folder also contains the definitions for the variables used in the mask workspace for the Sensors, FlightGear, linearAirframe and nonlinearAirframe blocks. These utilities store parameter values in data structures. For example, if the nonlinear model uses a parameter for a Gain block, the stored variable in the structure is `Vehicle.Nonlinear.Gain.gainValue`, which points to the parameter.

- **tests**

Contains a sample test harness:

- The `linearTest` file contains the actual test point. This file compares a subset of the outputs of the linearized airframe model to the outputs of the nonlinear airframe for the specific trim condition.
- The `runProjectTests` file runs all the available files classified as "Tests" in the project.

- **utilities**

Contains project-specific maintenance task utilities, such as:

- `projectPaths` - Lists the location of folders to be added to the MATLAB path.
- `rebuildSFunction` - Rebuilds S-functions for `linearInputBus` and `linearOutputBus`.
- `startVars` - Defines the variables that the simulation environment requires to be in the base workspace. This utility also controls variants using the `Variants` structure. This structure lets the example switch between the nonlinear and linear airframe from the workspace by changing `VSS_VEHICLE` from 1 (for the nonlinear model) to 0 (for the linear model). For more information on subsystem variants see `Model`.

- **work**

Contains files generated from every run. These files derive from source files, such as the MEX-file that you build from S-function C code.

In Shortcuts, projects creates shortcuts for common tasks:

- **Initialize Variables** — Runs the `startVars` script, which initializes the variables to the base workspace.
- **Rebuild S-functions** —Rebuilds the S-functions in the `src` folder.
- **Run Project Tests** —Runs the test points, labeled **Tests**, for test files in the project.
- **Top Level Simulation Model** — Opens the `flightSimulation` model. It runs on project startup.

### Template Labels

Provides file classification labels for automatic and componentization sorting. This utility adds template labels such as **Tests**, **Airframe Design**, **Flight Controller Design**, and **Calibration Data**.

### Add Airframe Dynamics and Controller Algorithm to the Project

- 1 To open the `linearAirframe` model, in `flightSimulation` double-click the Airframe subsystem.
- 2 Double-click the Nonlinear subsystem.



- 3 In the AC model, add your airframe dynamics.
- 4 Save the model.

### **Add Controller Algorithm to the Project**

- 1 To open the `flightControlSystem` model, in `flightSimulation`, double-click the FCS subsystem.
- 2 In the Controller subsystem, add your controller algorithm.
- 3 Save the model.

Other things to try:

- Simulate your model.
- Explore the **tests** folder for sample tests for your application.

### **See Also**

#### **Related Examples**

- “Create a New Project Using Templates” (Simulink)
- “Quadcopter Project” on page 6-59

## Flight Instrument Gauges

Use the blocks for flight instrument gauges to visualize navigation variables, such as altitude and heading. These blocks, located in the Flight Instruments library, represent standard cockpit instruments:

- Airspeed Indicator
- Altimeter
- Artificial Horizon
- Climb Rate Indicator
- Exhaust Gas Temperature (EGT) Indicator
- Heading Indicator
- Revolutions Per Minute (RPM) Indicator
- Turn Coordinator

### See Also

Airspeed Indicator | Altimeter | Artificial Horizon | Climb Rate Indicator | Exhaust Gas Temperature (EGT) Indicator | Heading Indicator | Revolutions Per Minute (RPM) Indicator | Turn Coordinator

### Related Examples

- “Display Measurements with Cockpit Instruments” on page 2-39
- “Programmatically Interact with Gauge Band Colors” on page 2-41

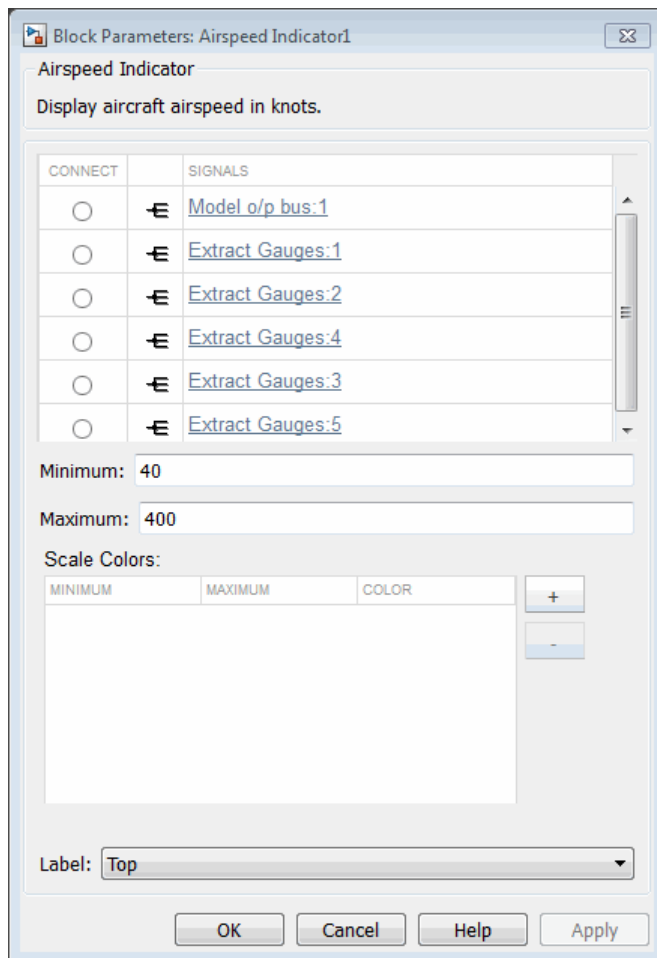
## Display Measurements with Cockpit Instruments

You can view signal data using any of the flight instrument blocks. This example uses the “HL-20 with Flight Instrumentation Blocks” on page 6-27 model. In this example, connect a gauge so that you can view the aircraft heading.

- 1 To open the model, at the MATLAB command window, enter `aeroblk_HL20_Gauges`.
- 2 Open the Visualization subsystem.

There is an existing Airspeed Indicator block in the model.

- 3 Add a second Airspeed Indicator block from the Flight Instruments library to the subsystem.
- 4 Open the new Airspeed Indicator block.
- 5 Select the Extract Flight Instruments block.
- 6 In the new Airspeed Indicator block, observe that the block connection table is filled with signals from the Extract Flight Instruments block that you can observe.



- 7 Select the option button next to `Extract_Gauges : 2` in the connection table.
- 8 To connect the `Extract_Gauges : 2` signal to the Airspeed Indicator block, click **OK**.

**Tip** To directly select the signal to connect, on the Extract Flight Instruments block, select the third output port (Roll Flightpath).

---

- 9** Simulate the model and observe the gauge as it registers the data.
- 10** To change the signal to connect to, you can:
  - Select the same or another block and then select another signal in the updated block connection table.
  - Select another output port for the same or a different block.
- 11** Close the model without saving it.

To create a Simulink model with prewired connections to flight instrument blocks, see `flightControl3DOFAirframeTemplate`.

## See Also

Airspeed Indicator | Altimeter | Artificial Horizon | Climb Rate Indicator | Exhaust Gas Temperature (EGT) Indicator | Heading Indicator | Revolutions Per Minute (RPM) Indicator | Turn Coordinator

## More About

- “Flight Instrument Gauges” on page 2-38
- “Programmatically Interact with Gauge Band Colors” on page 2-41

## Programmatically Interact with Gauge Band Colors

You can programmatically change Airspeed Indicator, EGT Indicator, and RPM Indicator gauge band colors using the `ScaleColors` property. When used with `get_param`, this property returns an  $n$ -by-1 structure containing these elements, where  $n$  is the number of colored bands on the gauge:

- `Min` — Minimum value range for a color band
- `Max` — Maximum value range for a color band
- `Color` — RGB color triplet for a band (range from 0 to 1)

This example describes how to change a color band of the EGT Indicator gauge. By default, the EGT Indicator gauge looks like this.



This gauge has three bands, clockwise 1, 2, and 3.

- 1 Create a blank model and add an EGT Indicator block.
- 2 Select the EGT Indicator block.
- 3 To change the color bands for the EGT Indicator gauge, get the handle of the scale color objects.

```
sc=get_param(gcb, 'ScaleColors')
```

```
sc =
```

```
 3×1 struct array with fields:
```

```
   Min
   Max
   Color
```

- 4 To see the values of the `Min`, `Max`, and `Color` values, use the `sc` handle. For example, to see the values of the first band, `sc(1)`, type:

```
sc(1)
```

```
sc(1)
```

```
ans =
```

```
 struct with fields:
```

```
   Min: 0
```

```
Max: 700  
Color: [0.2980 0.7333 0.0902]
```

- 5 To change the color and size of this band, define a structure with different Min, Max, and Color values and set ScaleColors to that new structure. For example, to change the band range to 1 to 89 and the color to red:

```
sc(1) = struct('Min',1,'Max',89,'Color',[1 0 0]);  
set_param(gcb,'ScaleColors',sc)
```

- 6 Observe the change in the EGT Indicator gauge.



- 7 You can add and change as many color bands as you need. For example, to add a fourth band and set up the gauge with that band:

```
sc(4) = struct('Min',200,'Max',300,'Color',[0 1 .6]);  
set_param(gcb,'ScaleColors',sc)
```



## See Also

Airspeed Indicator | Exhaust Gas Temperature (EGT) Indicator | Revolutions Per Minute (RPM) Indicator

## More About

- “Flight Instrument Gauges” on page 2-38
- “Display Measurements with Cockpit Instruments” on page 2-39

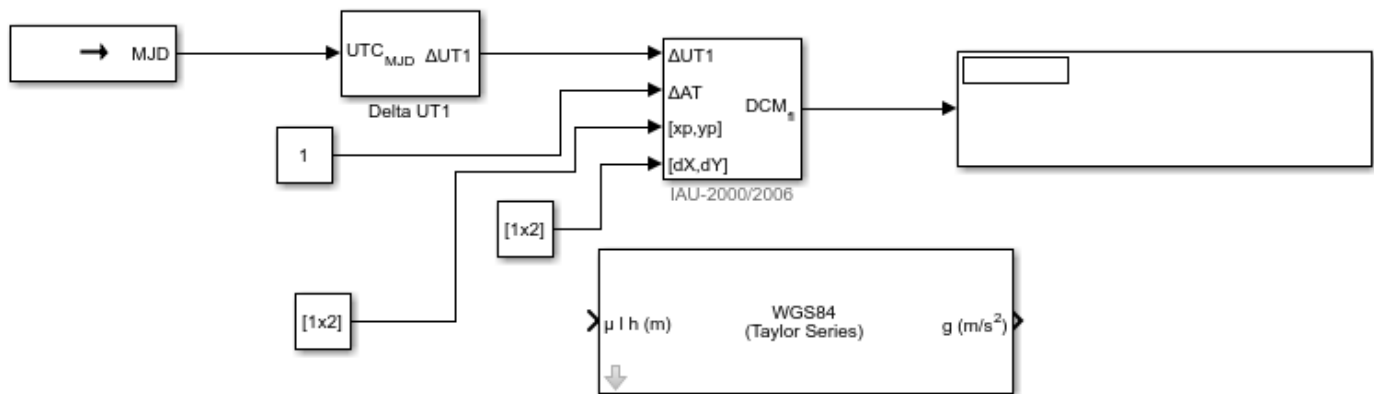
## Calculate UT1 to UTC Values

Calculate the difference between principal Universal Time (UT1) and Coordinated Universal Time (UTC) according to International Earth Rotation Service (IERS) by using the Delta UT1 block. Use the Delta UT1 block with these axes transformation blocks:

- LLA to ECI Position
- ECI Position to LLA
- Direction Cosine Matrix ECI to ECEF
- ECI Position to AER

To calculate the difference between UT1 and UTC, the Delta UT1 block requires the modified Julian date. This example uses the Julian Date Conversion block. However, you can calculate the modified Julian data with other methods. For example, you can use the `mjuliandatemjuliandate` function from the Aerospace Toolbox software to calculate the date and input the result to the Delta UT1 block.

### Use the Delta UT1 Block to Create Difference Values for the Direction Cosine Matrix ECI to ECEF Block



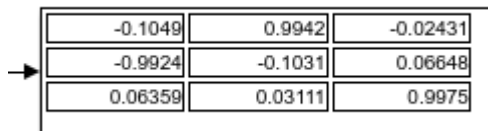
This model shows how a Direction Cosine Matrix ECI to ECEF block uses the output from the Delta UT1 and Julian Date Conversion blocks to obtain the difference between UTC and Universal Time (UT1).

**1** Drag these blocks into a new model and connect them as shown:

- Julian Date Conversion
- Delta UT1
- Direction Cosine Matrix ECI to ECEF
- Display
- Three Constant blocks

**2** Set up the Julian Date Conversion to convert the date December 28, 2015 to its modified Julian date equivalent. This date must match the one specified in the Direction Cosine Matrix ECI to ECEF.

- For **Year**, enter 2015.
  - For **Month**, enter 12.
  - For **Day**, enter 28.
  - To calculate the modified Julian date for December 28, 2015, select the **Calculate modified Julian date** check box.
  - For **Time increment**, select None.
- 3 Leave the default settings for Delta UT1. By default, the block calculates the difference between Universal Time (UT1) and Universal Coordinated Time (UTC) to using the `aeroiersdata.mat` file supplied with the Aerospace Blocksetsoftware.
  - 4 Set up the Direction Cosine Matrix ECI to ECEF block to work with the Universal Coordinated Time (UTC) December 28, 2015. This date must match the one specified in the Julian Date Conversion block:
    - For **Year**, enter 2015.
    - For **Month**, enter 12.
    - For **Day**, enter 28.
    - For **Time increment**, select None.
  - 5 Set up the  $\Delta UT1$ ,  $\Delta AT$ , and polar displacement of the Earth inputs for the Direction Cosine Matrix ECI to ECEF.
    - Constant — Set **Constant value** to 1.
    - Constant1 — Set **Constant value** to 1.
    - Constant2 — Set **Constant value** to [ .05 .05 ].
  - 6 Save and run the model. Observe the resulting direction cosine matrix in the Display block.



|         |         |          |
|---------|---------|----------|
| -0.1049 | 0.9942  | -0.02431 |
| -0.9924 | -0.1031 | 0.06648  |
| 0.06359 | 0.03111 | 0.9975   |

## See Also

Delta UT1 | Direction Cosine Matrix ECI to ECEF | Julian Date Conversion



# Analyze Dynamic Response and Flying Qualities of Aerospace Vehicles

Aerospace Blockset provides flight control analysis tools that you can use to analyze the dynamic response and flying qualities of aerospace vehicles.

- “Flight Control Analysis Live Scripts” on page 2-45 — MATLAB live scripts demonstrate dynamic response and flying quality analysis of Sky Hogg and de Havilland Beaver airframes.
- “Modify Flight Control Analysis Templates” on page 2-47 — You can use templates to analyze the flying qualities of 3 degree-of-freedom and 6 degree-of-freedom airframe models. When you are comfortable running the analysis on the default airframes, you can replace them with your own airframe and analyze it.

---

**Note** Analyzing dynamic response and flying qualities of airframes requires a Simulink Control Design license.

---

## Flight Control Analysis Live Scripts

Each flight control analysis template has an associated MATLAB live script that guides you through a flying quality analysis workflow for the default airframe. You can interact with the script and explore the analysis workflow.

- `DehavillandBeaverFlyingQualityAnalysis` — Compute longitudinal and lateral-directional flying qualities for a Dehavilland Beaver airframe.
- `SkyHoggLongitudinalFlyingQualityAnalysis` — Compute longitudinal flying qualities for a Sky Hogg airframe.

For more information on running live scripts, see “Run Sections in Live Scripts” (MATLAB).

- 1 Open one of the templates, for example:

```
asbFlightControlAnalysis('6DOF')
```

Navigate to the **Getting Started** section and click the first link.

Alternatively, in the Command Window, type:

```
open('DehavillandBeaverFlyingQualityAnalysis')
```

- 2 The script describes how to use eigenvalue analysis to determine the longitudinal flying qualities (long-period phugoid mode and short-period mode) and lateral-directional flying qualities (Dutch roll mode, roll mode, and spiral mode) for an aircraft modeled in Simulink.

As you run the script, when applicable, the results of the runs display inline.

## Modify Flight Analysis Templates

Aerospace Blockset provides these templates:

- `flightControl6DOFAirframeTemplate` — This template uses a six degree-of-freedom airframe configured for linearization and quality analysis. For initialization, the template uses the de Havilland Beaver airframe parameters.

- `flightControl3DOFAirframeTemplate` — This template uses a 3 degree-of-freedom longitudinal airframe configured for linearization and quality analysis. For initialization, the template uses Sky Hogg airframe parameters.

When you are comfortable navigating the flight control analysis templates with the default airframes, consider customizing the templates for your own airframe model.

## Flight Control Analysis Templates

To familiarize yourself with Aerospace Blockset flight control analysis templates:

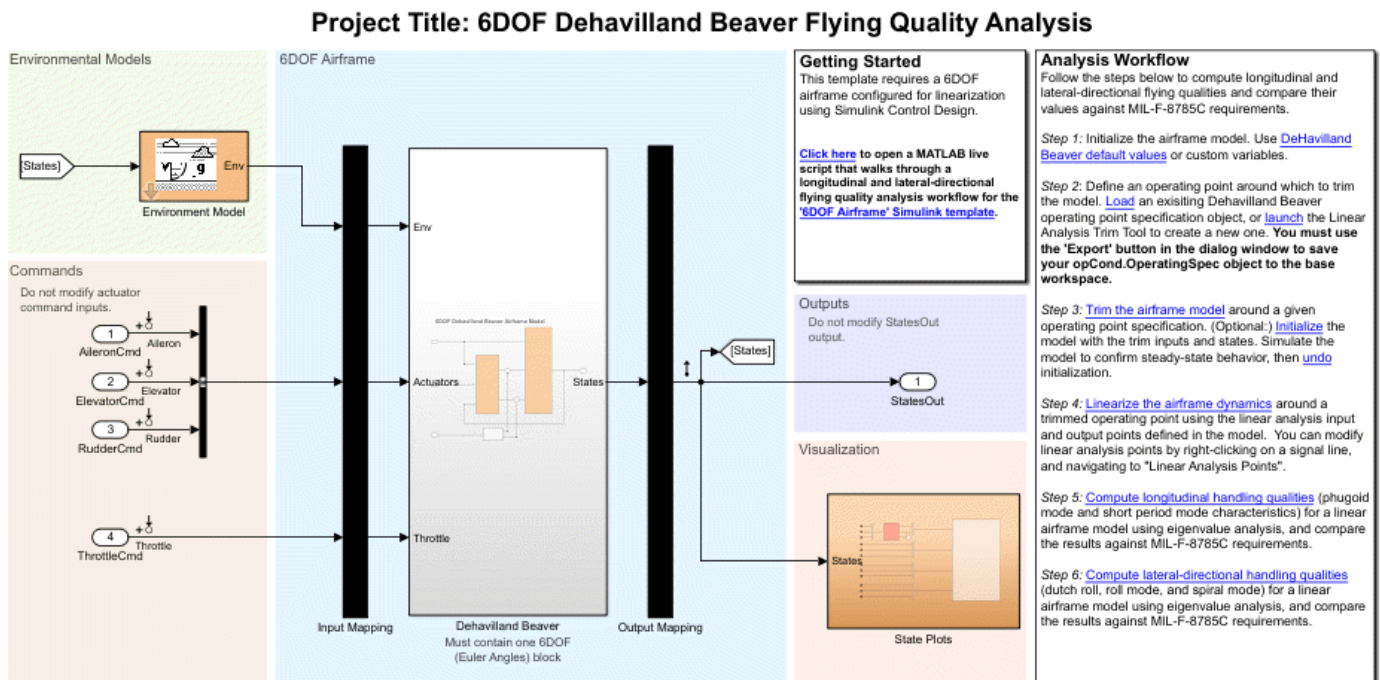
- 1 Open one of the templates. For example, to start a 3DOF template:

```
asbFlightControlAnalysis('3DOF')
```

To open a 6DOF template:

```
asbFlightControlAnalysis('6DOF')
```

The flight control analysis model opens.



- 2 The **Analysis Workflow** section contains a clickable guided workflow to compute longitudinal and lateral-directional flying qualities and compare their values against MIL-F-8785C requirements. Each step creates the necessary variables for the following step. To perform the flying quality analysis, sequentially click the links in the steps.
  - a Create an operating point specification object in the base workspace for the airframe model using the Model Linearizer. Alternatively, load the default object provided in step 2.
  - b To trim the airframe, click **Trim the airframe** in step 3. This action calls the `trimAirframe` function.
  - c To linearize the airframe around a trimmed operating point, click **Linearize the airframe** in step 4. This action calls the `linearizeAirframe` function.

- d To compute the longitudinal flying qualities, click **Compute longitudinal handling qualities**. This action calls the `computeLongitudinalFlyingQualities` function.
- e To compute the lateral-directional handling qualities, click **Compute lateral-directional handling qualities** in step 6. This action calls the `computeLateralDirectionalFlyingQualities` function.

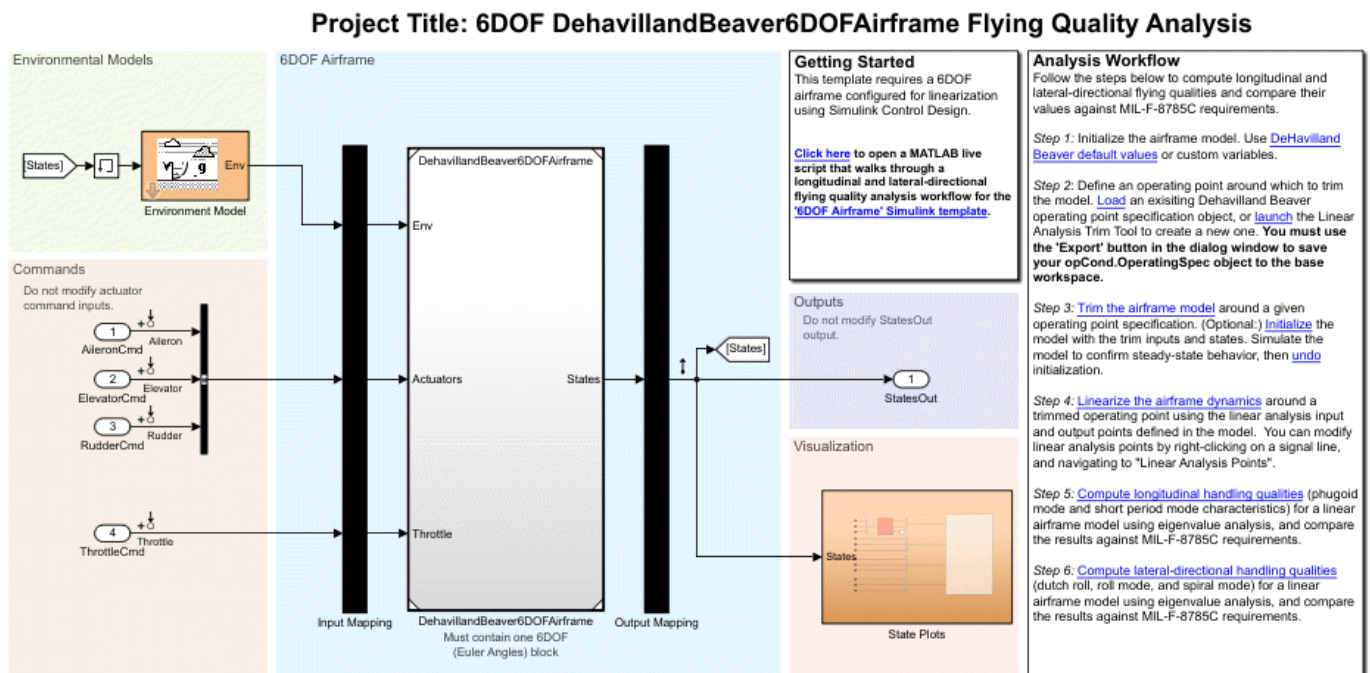
## Modify Flight Control Analysis Templates

When you are comfortable using the 3DOF and/or 6DOF flight control analysis templates on page 2-46 to trim, linearize, and compute the longitudinal and lateral-directional handling qualities for the default airframes, consider customizing the templates to include your own airframe.

- 1 Open a 3DOF or 6DOF template and change the airframe to one of your own. For example, to change the template airframe to an external model:

```
asbFlightControlAnalysis('6DOF', 'sixDOFAirframeExample', 'DehavillandBeaver6DOFAirframe')
```

This command replaces the de Havilland Beaver subsystem with the `DehavillandBeaver6DOFAirframe` model and includes it as a referenced model.



Alternatively, in the corresponding canvas, manually replace the default model airframe in the blue area with your own airframe.

- 2 On the canvas, align the inputs and outputs of the airframe using the Input Mapping and Output Mapping subsystems.
- 3 Create a new operating point specification object. In the **Analysis Workflow** section, go to step 2 and click the **Launch** link to start the Model Linearizer.
- 4 To save your `opCond.OperatingSpec` object to the base workspace, click the **Export** button in the dialog window.
- 5 To trim, linearize, and compute the longitudinal and lateral-directional handling qualities for the airframe model, click the links in workflow steps 3, 4, 5, and 6.

## Explore Flight Control Analysis Functions

The flight control analysis live scripts and template workflows use these functions:

- `asbFlightControlAnalysis`
- `trimAirframe`
- `linearizeAirframe`
- `computeLongitudinalFlyingQualities`
- `computeLateralDirectionalFlyingQualities`

To customize your own scripts to trim airframes around operating points, linearize airframes, and calculate longitudinal and lateral-directional handling qualities, you can use these functions in a workflow as follows:

- 1 Create a flight control analysis template using the `asbFlightControlAnalysis` function.
- 2 Trim the airframe model around an operating point using the `trimAirframe` function.

This step creates a trimmed operating point, required by the `linearizeAirframe` function.

- 3 Linearize the airframe model around the trimmed operating point using the `linearizeAirframe` function.

This step creates a state space model that describes the linearized dynamics of the airframe model at a trimmed operating point.

- 4 Compute the flying qualities for the airframe, including short- and long-period (phugoid) mode characteristics of the specified state space model, using `computeLongitudinalFlyingQualities`, and lateral-directional (Dutch roll, roll, and spiral) mode characteristics, using `computeLateralDirectionalFlyingQualities`.

For example:

```
asbFlightControlAnalysis('6DOF', 'DehavillandBeaverAnalysisModel');
opSpecDefault = DehavillandBeaver6DOF0pSpec('DehavillandBeaverAnalysisModel');
opTrim = trimAirframe('DehavillandBeaverAnalysisModel', opSpecDefault);
linSys = linearizeAirframe('DehavillandBeaverAnalysisModel', opTrim);
lonFlyingQual = computeLongitudinalFlyingQualities('DehavillandBeaverAnalysisModel', linSys)
latFlyingQual = computeLateralDirectionalFlyingQualities('DehavillandBeaverAnalysisModel', linSys)
```

## See Also

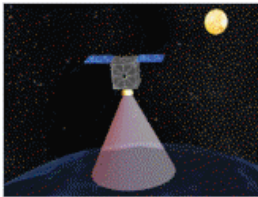
[Model Linearizer](#) | [asbFlightControlAnalysis](#) | [computeLateralDirectionalFlyingQualities](#) | [computeLongitudinalFlyingQualities](#) | [linearizeAirframe](#) | [trimAirframe](#)

## Getting Started with CubeSats

Model, simulate, analyze, and visualize the motion and dynamics of CubeSats and nano satellites. Nano satellites are miniaturized spacecraft designed for space research based on one or more 10 cm cubes of up to 1.33 kg per unit. Use the Aerospace Blockset CubeSat Simulation Library, available as an Add-On.

- 1 Start the Add-On Explorer for the CubeSat Add-on.

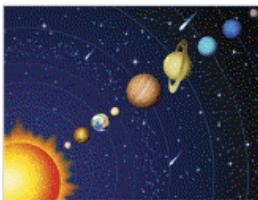
3 RESULTS



### Aerospace Blockset CubeSat Simulation Library by MathWorks Aerospace Products Team **STAFF**

Model, simulate, analyze, and visualize CubeSats using Aerospace Blockset

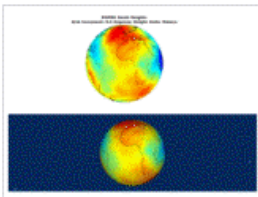
MathWorks Optional Feature



### Ephemeris Data for Aerospace Toolbox by MathWorks Aerospace Products Team **STAFF**

Ephemeris Data for Aerospace Toolbox

MathWorks Optional Feature



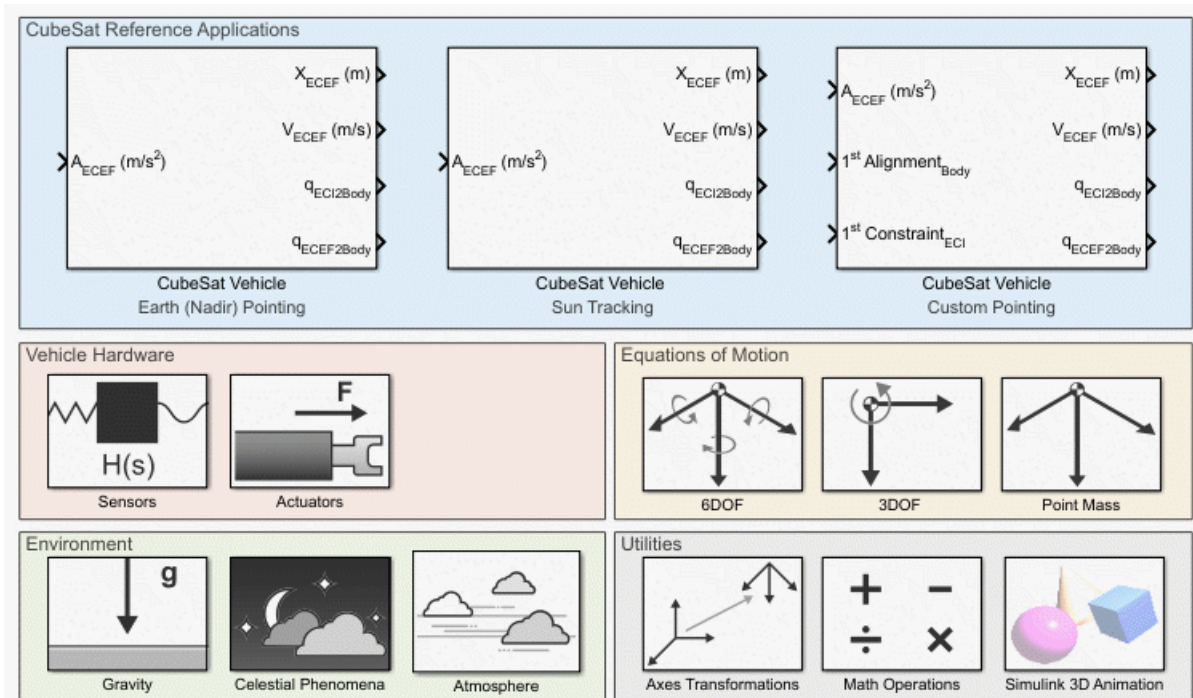
### Geoid Data for Aerospace Toolbox by MathWorks Aerospace Products Team **STAFF**

Geoid Data for Aerospace Toolbox

MathWorks Optional Feature

- 2 In the Add-On Explorer, click Aerospace Blockset CubeSat Simulation Library and follow instructions to install the Add-On.
- 3 To explore the block library for the CubeSat Add-On, at the MATLAB command line, type:
 

```
asbCubeSatBlockLib
```



The CubeSat Simulation Library Add-On contains:

- “Model and Simulate CubeSats” on page 2-51 — To create models, use the CubeSat Vehicle block, propagation model template, and project. Explore the CubeSat example modeling multiple CubeSats.
- “Analyze CubeSat Mission Geometry” on page 2-58 — To analyze CubeSat mission geometries, use the accompanying live script. This live script requires a Mapping Toolbox™ license.
- “Utility Functions” on page 2-56 — For coordinate transformations and mission geometry analysis of CubeSat vehicles.

## References

[1] Vallado, D. A. *Fundamentals of Astrodynamics and Applications*. alg. 1 and eqs. 1-63. New York: McGraw-Hill, 1997.

## See Also

CubeSat Vehicle | asbAddOns | asbCubeSatMissionAnalysis | ecef2eci | eci2ecef | generateAccessTable | greenwichsrt | ijk2keplerian | keplerian2ijk

## More About

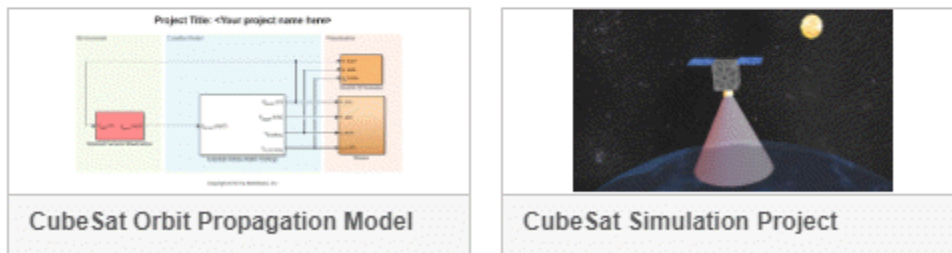
- Model and Simulate CubeSats
- “Analyze CubeSat Mission Geometry” on page 2-58

## Model and Simulate CubeSats

To help you get started modeling and simulating CubeSats, the CubeSat Simulation Library Add-On provides a project and model on the Simulink Start Page. If you have not yet done so, install the Aerospace Blockset CubeSat Simulation Library. For more information, see “Getting Started with CubeSats” on page 2-49.

### > Aerospace Blockset

#### ▼ Aerospace Blockset CubeSat Simulation Library




- CubeSat Orbit Propagation Model template — A model template (CubeSat Simulation Project) that illustrates how to propagate and visualize CubeSat trajectories using the CubeSat Vehicle block. The Spherical Harmonic Gravity Model block performs the orbit propagation. The preconfigured pointing modes set in the CubeSat Vehicle block control the attitude.
- CubeSat Simulation Project — A ready-to-simulate project (CubeSat Simulation Project) that illustrates how to create a detailed CubeSat system design in Simulink by adding in detailed vehicle components to the provided framework.

### CubeSat Orbit Propagation Model Template

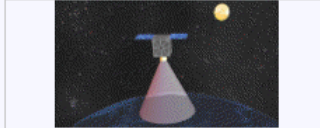
The template model is a ready-to-simulate example that contains a CubeSat Vehicle block with visualization using Simulink 3D Animation.

- 1 Start the CubeSat Orbit Propagation Model template.

▼ Aerospace Blockset CubeSat Simulation Library



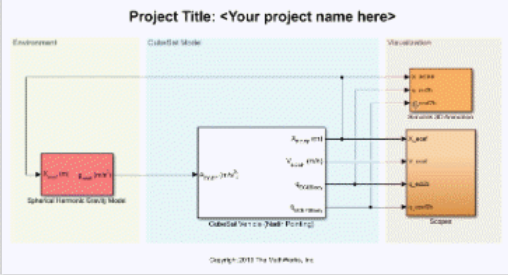
CubeSat Orbit Propagation Model



CubeSat Simulation Project

---

**Project Title: <Your project name here>**



**CubeSat Orbit Propagation Model**  
By The MathWorks, Inc.

Create Model ▼

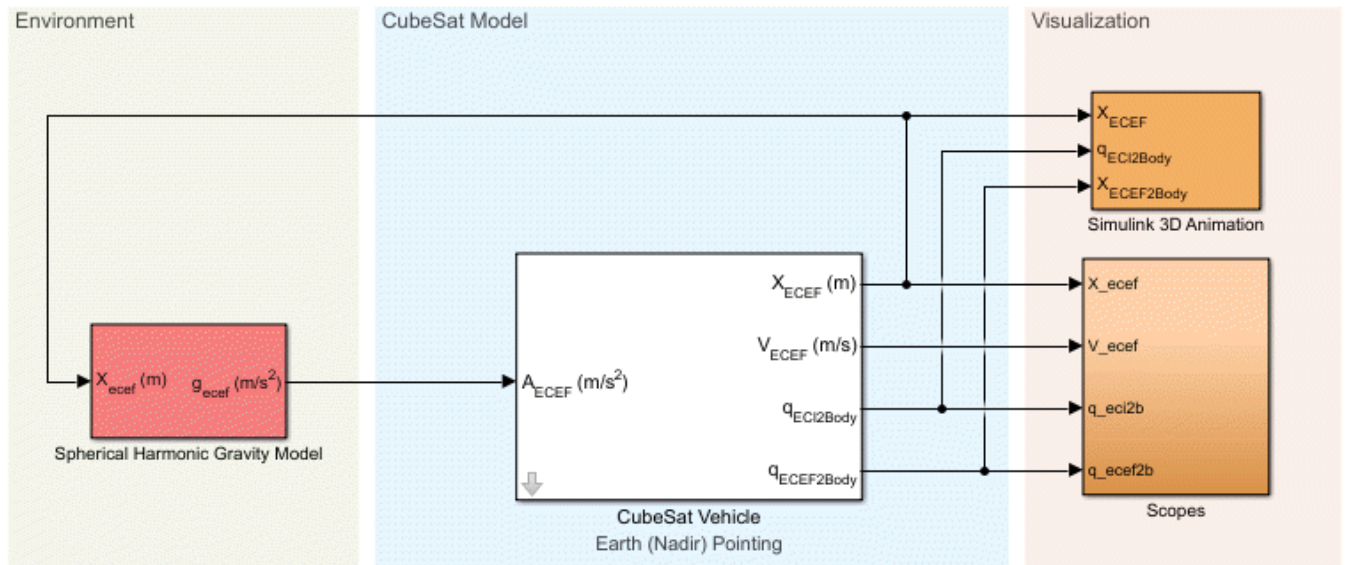
Aerospace Blockset CubeSat Simulation Library lets you model, simulate, analyze, and visualize the motion and dynamics of CubeSats and nano satellites, which are miniaturized spacecraft designed for space research based on one or more 10cm cubes of up to 1.33kg per unit. This model template contains the CubeSat Vehicle block from `asbCubeSatBlockLib.slx` and a Spherical Harmonic Gravity Model from the Aerospace Blockset. Visualization using Simulink 3D Animation is provided if a valid license exists.

The CubeSat Vehicle block models a simple CubeSat vehicle:

- Specify the initial orbital state as a set of Keplerian orbital elements; position and velocity state

2 Click **Create Model**.

### Project Title: <Your project name here>



3 The CubeSat Vehicle block models a simple CubeSat vehicle that you can use as is, with the CubeSat Vehicle block configured to use the initial orbital state as a set of Keplerian orbital elements.



The model uses the Spherical Harmonic Gravity Model block to provide the vehicle gravity for the CubeSat.

To familiarize yourself with CubeSats, experiment with the CubeSat Vehicle block settings.

- On the **CubeSat Orbit** tab of the block, you can optionally use the **Input method** parameter to change the initial orbital state as a set of:
    - Position and velocity state vectors in Earth-centered inertial axes
    - Position and velocity state vectors in Earth-centered Earth-fixed axes
    - Latitude, longitude, altitude, and velocity of the body with respect to ECEF, expressed in the NED frame
  - On the **CubeSat Attitude** tab, you can specify the alignment and constraint vectors to define the CubeSat attitude control.
    - The CubeSat vehicle first aligns the primary alignment vector with the primary constraint vector. The CubeSat vehicle then attempts to align the secondary alignment vector with the secondary constraint vector as closely as possible without affecting primary alignment.
    - The CubeSat Attitude tab also lets you choose between preconfigured Earth (Nadir) Earth Pointing (default) and Sun Tracking attitude control modes.
  - On the **Earth Orientation Parameters** tab, you can direct the block to include higher order earth orientation parameters while computing vehicle initial conditions.
- 4 Run and simulate the model.
  - 5 To view the output signals from the CubeSat, double-click the Scopes subsystem and open the multiple scopes.
  - 6 If you have a valid Simulink 3D Animation license, you can also visualize the orbit in the **CubeSat Orbit Animation** window.
  - 7 Save a copy of the orbit propagation model. You can use this model for the mission analysis live script.

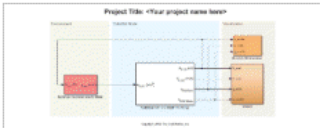
The CubeSat Orbit Propagation Model Propagate CubeSat Vehicle block uses simple preconfigured orbit and attitude control modes. To model and simulate CubeSat vehicles using your own detailed components, consider the CubeSat Simulation Project from the Simulink Start Page. For more information, see “CubeSat Simulation Project” on page 2-53

## CubeSat Simulation Project

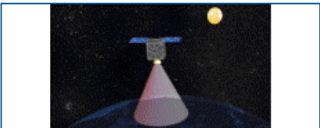
The model is a ready-to-simulate example with visualization using Simulink 3D Animation. This example uses a Vehicle Model subsystem in place of a CubeSat Vehicle block. For a simpler model that illustrates the use of the CubeSat Vehicle block, see “CubeSat Orbit Propagation Model Template” on page 2-51.

- 1 Start the CubeSat Simulation Project.

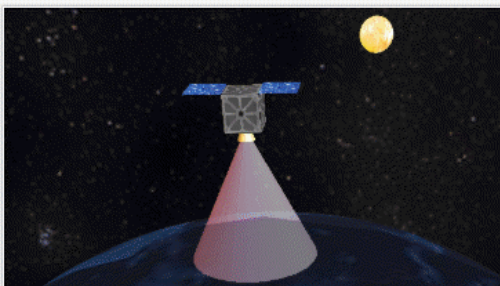
▼ Aerospace Blockset CubeSat Simulation Library



CubeSat Orbit Propagation Model



CubeSat Simulation Project



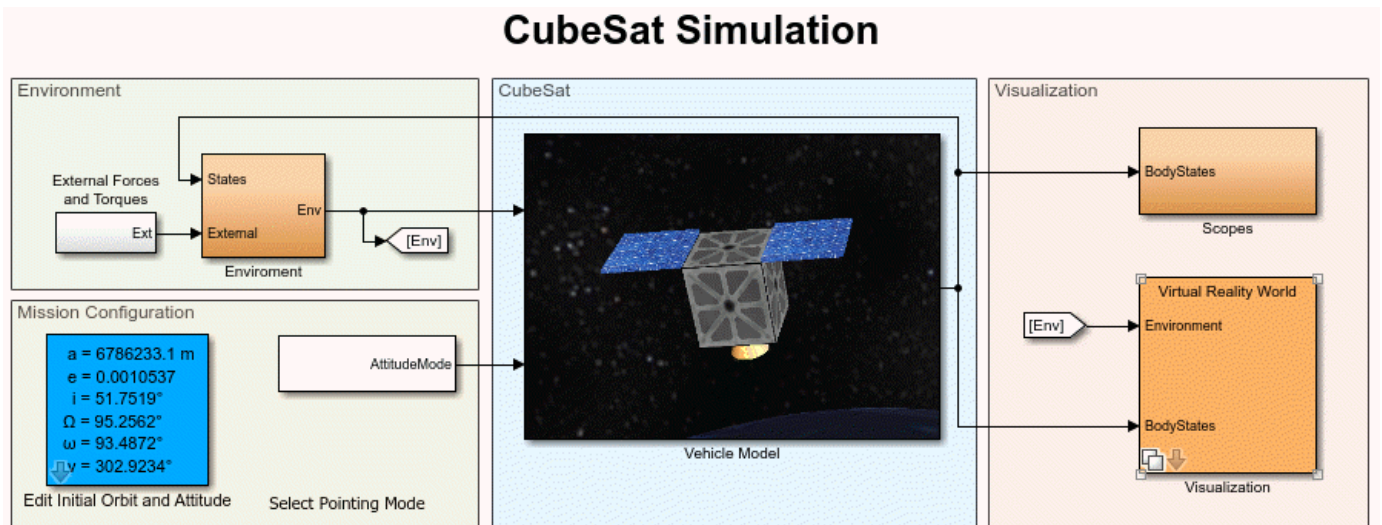
### CubeSat Simulation Project

By The MathWorks, Inc.

Aerospace Blockset CubeSat Simulation Library lets you model, simulate, analyze, and visualize the motion and dynamics of CubeSats and nano satellites, which are miniaturized spacecraft designed for space research based on one or more 10cm cubes of up to 1.33kg per unit. This project includes a ready-to-simulate example with visualization using Simulink 3D Animation. To define the orbit trajectory and attitude of the CubeSat, double-click the `asbCubeSat/Edit Initial Orbit and Attitude` block in the model.

Create Project

2 Click **Create Project**.



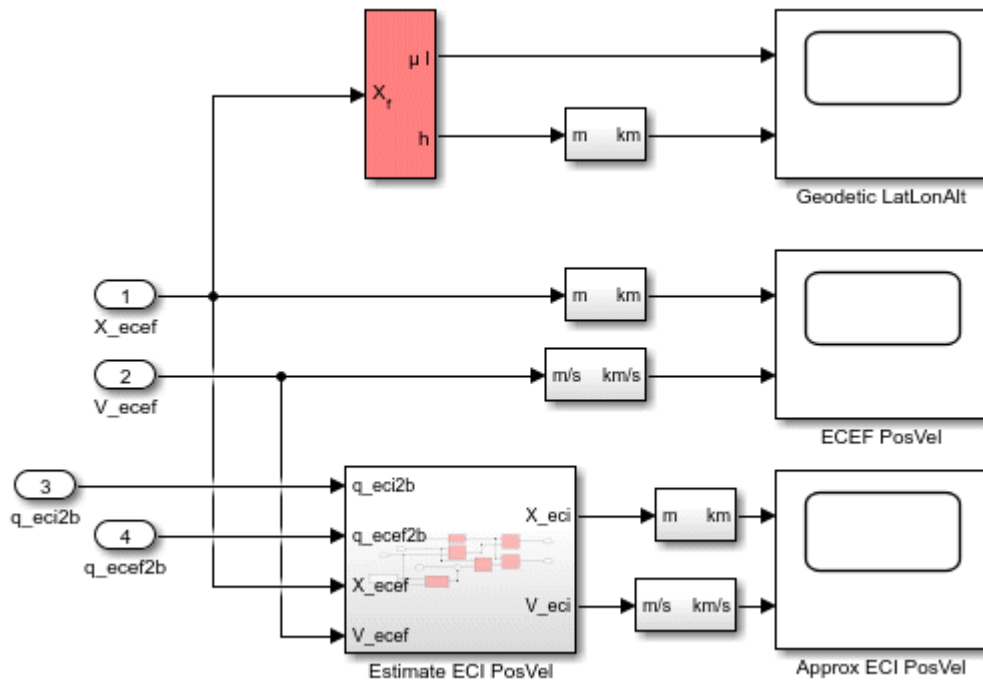
3 The Vehicle Model subsystem models a CubeSat vehicle that you can use as is.

To create your own more sophisticated satellite models, experiment with the Vehicle Model framework. For example, you can replace the perfect thruster model included by default in the actuator subsystem with your own more realistic thruster or reaction wheel model.

4 To change the orbit trajectory and attitude of the CubeSat, in the Mission Configuration section, double-click the Edit Initial Orbit and Attitude block. These parameters have the same intent as the corresponding parameters as the CubeSat Vehicle block.

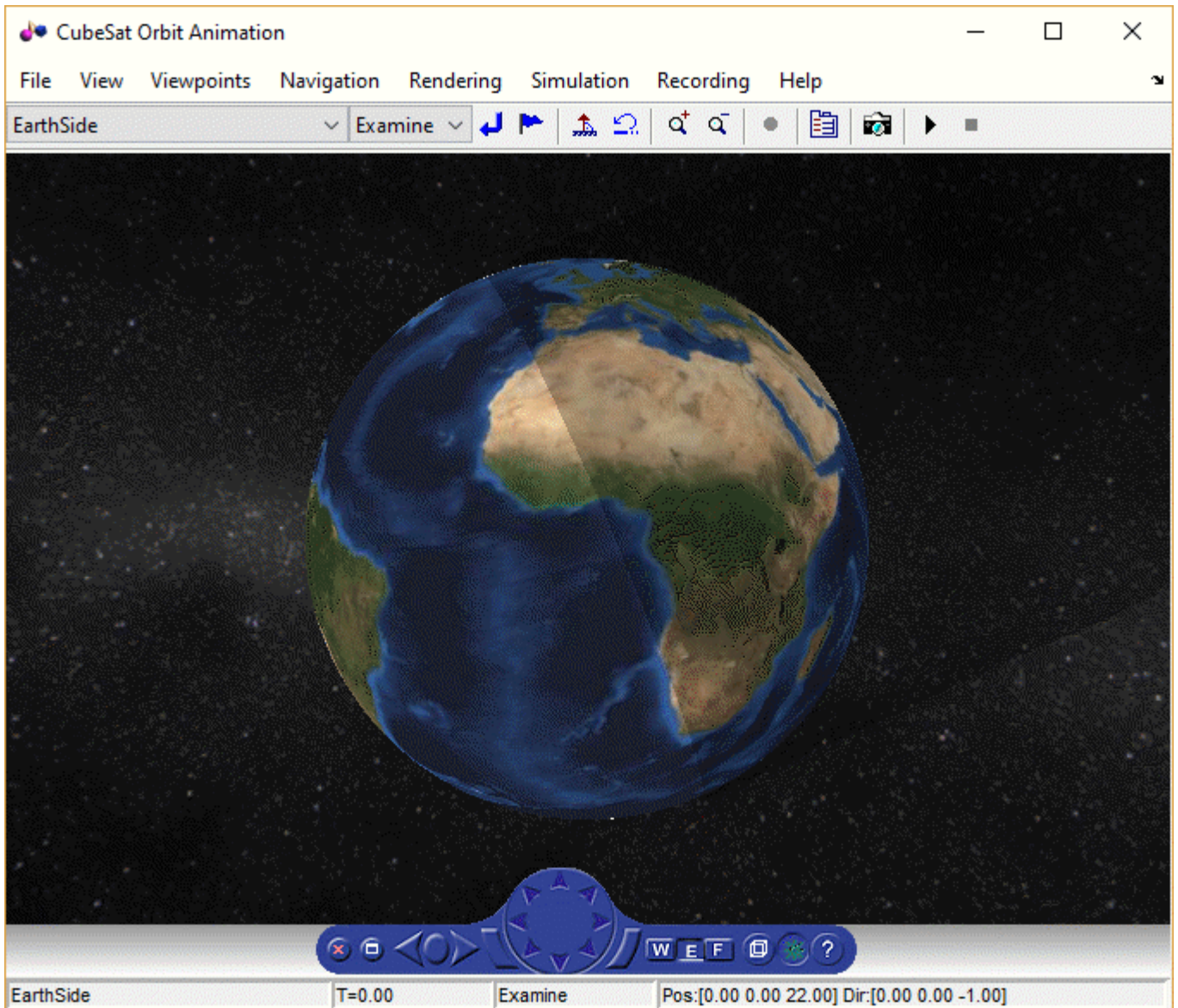
5 Run and simulate the model.

6 To view the output signals from the CubeSat, double-click the Scopes subsystem and open the multiple scopes.



- 7 If you have a license for Simulink 3D Animation, you can also visualize the orbit in an animation window. Double-click the Visualization subsystem and click the **Open Simulink 3D Animation window** button.

The **CubeSat Orbit Animation** window opens.



## Utility Functions

In addition to the existing Aerospace Blockset and Aerospace Toolbox functions, the CubeSat Simulation Library provides utility functions for coordinate transformations. You can use these functions to go between the various initial condition modes of the CubeSat Vehicle block.

| Action  | Function |
|---|----------|
| Calculate position and velocity vectors in Earth-centered inertial mean-equator mean-equinox                  | ecef2eci |
| Calculate position, velocity, and acceleration vectors in Earth-centered Earth-fixed (ECEF) coordinate system | eci2ecef |

| Action  | Function      |
|---|---------------|
| Calculate Greenwich mean and apparent sidereal times  | greenwichsrt  |
| Calculate Keplerian orbit elements using geocentric equatorial position and velocity vectors                      | ijk2keplerian |
| Calculate position and velocity vectors in geocentric equatorial coordinate system using Keplerian orbit elements | keplerian2ijk |

## References

[1] Vallado, D. A. *Fundamentals of Astrodynamics and Applications*. alg. 1 and eqs. 1-63. New York: McGraw-Hill, 1997.

## See Also

CubeSat Vehicle | asbAdd0ns | asbCubeSatMissionAnalysis | ecef2eci | eci2ecef | generateAccessTable | greenwichsrt | ijk2keplerian | keplerian2ijk

## More About

- “Getting Started with CubeSats” on page 2-49
- “Analyze CubeSat Mission Geometry” on page 2-58

## See Also

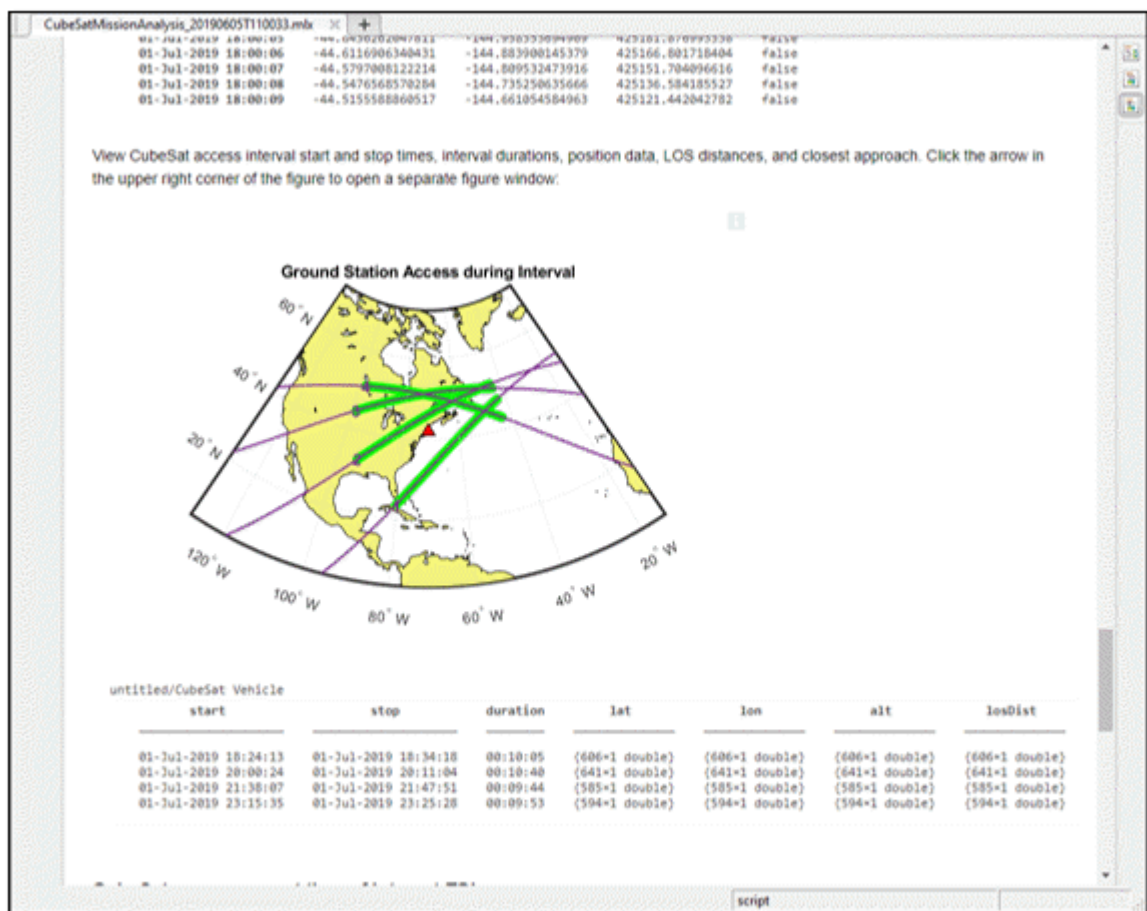
## Analyze CubeSat Mission Geometry

To help you get started analyzing CubeSat mission geometries of CubeSat vehicles modeled in Simulink, the CubeSat Simulation Library provides the `asbCubeSatMissionAnalysis` live script. If you have not yet done so, install the Aerospace Blockset CubeSat Simulation Library. For more information, see “Getting Started with CubeSats” on page 2-49.

This live script uses the CubeSat Simulation Library and Mapping Toolbox software to analyze the mission geometry of one or more CubeSat vehicles modeled in Simulink, and determine CubeSat to ground station line-of-sight (LOS) visibility. To view the underlying MATLAB code for this report, change the display output option.

This live script analyzes mission geometries for CubeSats modeled using the CubeSat Vehicle block.

- 1 Start the CubeSat mission analysis live script. Here is an excerpt of an analysis live script.




This function starts a live script for which you can specify a model. This live script is self-documenting and guides you through the steps to analyze your mission geometry. It contains sections for:

- Mission definition
- Simulation

- Analysis results
- 2 In **Simulink Model(s) containing 'CubeSat Vehicle' block(s)**, enter the name of a model that contains one or more CubeSat Vehicle blocks. For example:
    - The example model saved from the CubeSat Orbit Propagation Model template.
    - The asbDualCubeSatExample model, provided in the Aerospace Blockset CubeSat Simulation Library.
  - 3 As desired, change the other settings in the **Mission Definition** section then, in the **Run Simulation** section, click **Run**.

The live script analyzes the CubeSat vehicles in the specified model and updates the live script with data from the model.

- 4 Explore the live script and change values where prompted to visualize varying data.

All visualizations in the **Mission Analysis Results** section have arrows  that you can click to open the visualization in a separate figure window.

- 5 Optionally save the live script file for this analysis.

## Utility Function

In addition to the existing Aerospace Blockset and Aerospace Toolbox functions, the CubeSat Simulation Library provides the `generateAccessTable` function to calculate line-of-sight (LOS) mutual visibility between CubeSat and ground station.

## References

- [1] Wertz, James R, David F. Everett, and Jeffery J. Puschell. *Space Mission Engineering: The New Smad*. Hawthorne, CA: Microcosm Press, 2011. Print.

## See Also

CubeSat Vehicle | asbAddOns | asbCubeSatMissionAnalysis | ecef2eci | eci2ecef | generateAccessTable | greenwichsrt | ijk2keplerian | keplerian2ijk

## More About

- “Getting Started with CubeSats” on page 2-49
- Model and Simulate CubeSats





# Case Studies

---

- “Ideal Airspeed Correction” on page 3-2
- “1903 Wright Flyer” on page 3-7
- “NASA HL-20 Lifting Body Airframe” on page 3-14

## Ideal Airspeed Correction

### In this section...

“Introduction” on page 3-2  
 “Airspeed Correction Models” on page 3-2  
 “Measure Airspeed” on page 3-2  
 “Model Airspeed Correction” on page 3-3  
 “Simulate Airspeed Correction” on page 3-5

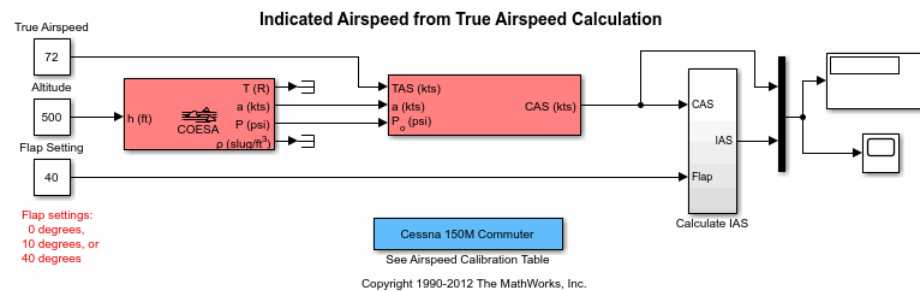
### Introduction

This case study simulates indicated and true airspeed. It constitutes a fragment of a complete aerodynamics problem, including only measurement and calibration.

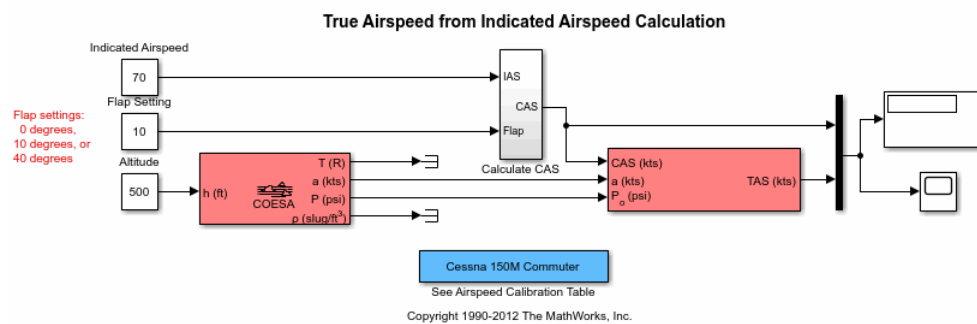
### Airspeed Correction Models

To view the airspeed correction models, enter the following at the MATLAB command line:

```
aeroblk_indicated
aeroblk_calibrated
```



### aeroblk\_indicated Model



### aeroblk\_calibrated Model

### Measure Airspeed

To measure airspeed, most light aircraft designs implement pitot-static airspeed indicators based on Bernoulli's principle. Pitot-static airspeed indicators measure airspeed by an expandable capsule that

expands and contracts with increasing and decreasing dynamic pressure. This is known as *calibrated airspeed* (CAS). It is what a pilot sees in the cockpit of an aircraft.

To compensate for measurement errors, it helps to distinguish three types of airspeed. These types are explained more completely in the following.

| Airspeed Type | Description   |
|---------------|---|
| Calibrated    | Indicated airspeed corrected for calibration error      |
| Equivalent    | Calibrated airspeed corrected for compressibility error |
| True          | Equivalent airspeed corrected for density error         |

### Calibration Error

An airspeed sensor features a static vent to maintain its internal pressure equal to atmospheric pressure. Position and placement of the static vent with respect to the angle of attack and velocity of the aircraft determines the pressure inside the airspeed sensor and therefore the calibration error. Thus, a calibration error is specific to an aircraft's design.

An airspeed calibration table, which is usually included in the pilot operating handbook or other aircraft documentation, helps pilots convert the indicated airspeed to the calibrated airspeed.

### Compressibility Error

The density of air is not constant, and the compressibility of air increases with altitude and airspeed, or when contained in a restricted volume. A pitot-static airspeed sensor contains a restricted volume of air. At high altitudes and high airspeeds, calibrated airspeed is always higher than equivalent airspeed. Equivalent airspeed can be derived by adjusting the calibrated airspeed for compressibility error.

### Density Error

At high altitudes, airspeed indicators read lower than true airspeed because the air density is lower. True airspeed represents the compensation of equivalent airspeed for the density error, the difference in air density at altitude from the air density at sea level, in a standard atmosphere.

## Model Airspeed Correction

The `aeroblk_indicated` and `aeroblk_calibrated` models show how to take true airspeed and correct it to indicated airspeed for instrument display in a Cessna 150M Commuter light aircraft. The `aeroblk_indicated` model implements a conversion to indicated airspeed. The `aeroblk_calibrated` model implements a conversion to true airspeed.

Each model consists of two main components:

- “COESA Atmosphere Model Block” on page 3-3 calculates the change in atmospheric conditions with changing altitude.
- “Ideal Airspeed Correction Block” on page 3-4 transforms true airspeed to calibrated airspeed and vice versa.

### COESA Atmosphere Model Block

The COESA Atmosphere Model block is a mathematical representation of the U.S. 1976 COESA (Committee on Extension to the Standard Atmosphere) standard lower atmospheric values for

absolute temperature, pressure, density, and speed of sound for input geopotential altitude. Below 32,000 meters (104,987 feet), the U.S. Standard Atmosphere is identical with the Standard Atmosphere of the ICAO (International Civil Aviation Organization).

The `aeroblk_indicated` and `aeroblk_calibrated` models use the COESA Atmosphere Model block to supply the speed of sound and air pressure inputs for the Ideal Airspeed Correction block in each model.

**Ideal Airspeed Correction Block**

The Ideal Airspeed Correction block compensates for airspeed measurement errors to convert airspeed from one type to another type. The following table contains the Ideal Airspeed Correction block's inputs and outputs.

| Airspeed Input      | Airspeed Output                            |
|---------------------|--|
| True Airspeed       | Equivalent airspeed<br>Calibrated airspeed |
| Equivalent Airspeed | True airspeed<br>Calibrated airspeed       |
| Calibrated Airspeed | True airspeed<br>Equivalent airspeed       |

In the `aeroblk_indicated` model, the Ideal Airspeed Correction block transforms true to calibrated airspeed. In the `aeroblk_calibrated` model, the Ideal Airspeed Correction block transforms calibrated to true airspeed.

The following sections explain how the Ideal Airspeed Correction block mathematically represents airspeed transformations:

- “True Airspeed Implementation” on page 3-4
- “Calibrated Airspeed Implementation” on page 3-4
- “Equivalent Airspeed Implementation” on page 3-5

**True Airspeed Implementation**

True airspeed (TAS) is implemented as an input and as a function of equivalent airspeed (EAS), expressible as

$$TAS = \frac{EAS \times a}{a_0 \sqrt{\delta}}$$

where

|          |   |
|----------|---|
| $\alpha$ | Speed of sound at altitude in m/s       |
| $\delta$ | Relative pressure ratio at altitude     |
| $a_0$    | Speed of sound at mean sea level in m/s |

**Calibrated Airspeed Implementation**

Calibrated airspeed (CAS), derived using the compressible form of Bernoulli's equation and assuming isentropic conditions, can be expressed as

$$CAS = \sqrt{\frac{2\gamma P_0}{(\gamma - 1)\rho_0} \left[ \left( \frac{q}{P_0} + 1 \right)^{(\gamma - 1)/\gamma} - 1 \right]}$$

where

|          |  |
|----------|--|
| $\rho_0$ | Air density at mean sea level in kg/m <sup>3</sup>     |
| $P_0$    | Static pressure at mean sea level in N/m <sup>2</sup>  |
| $\gamma$ | Ratio of specific heats                                |
| $q$      | Dynamic pressure at mean sea level in N/m <sup>2</sup> |

### Equivalent Airspeed Implementation

Equivalent airspeed (EAS) is the same as CAS, except static pressure at sea level is replaced by static pressure at altitude.

$$EAS = \sqrt{\frac{2\gamma P}{(\gamma - 1)\rho_0} \left[ \left( \frac{q}{P} + 1 \right)^{(\gamma - 1)/\gamma} - 1 \right]}$$

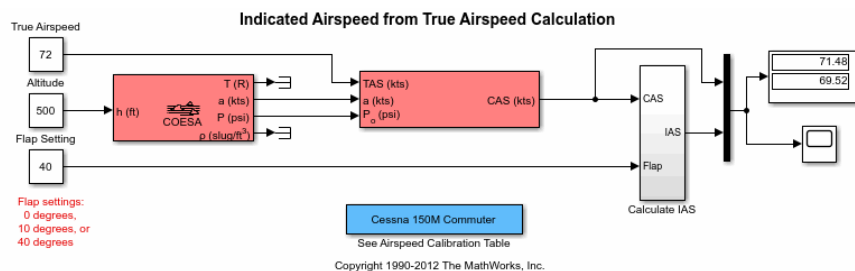
The symbols are defined as follows:

|          |  |
|----------|--|
| $\rho_0$ | Air density at mean sea level in kg/m <sup>3</sup>     |
| $P$      | Static pressure at altitude in N/m <sup>2</sup>        |
| $\gamma$ | Ratio of specific heats                                |
| $q$      | Dynamic pressure at mean sea level in N/m <sup>2</sup> |

### Simulate Airspeed Correction

In the `aeroblk_indicated` model, the aircraft is defined to be traveling at a constant speed of 72 knots (true airspeed) and altitude of 500 feet. The flaps are set to 40 degrees. The COESA Atmosphere Model block takes the altitude as input and outputs the speed of sound and air pressure. Taking the speed of sound, air pressure, and airspeed as inputs, the Ideal Airspeed Correction block converts true airspeed to calibrated airspeed. Finally, the Calculate IAS subsystem uses the flap setting and calibrated airspeed to calculate indicated airspeed.

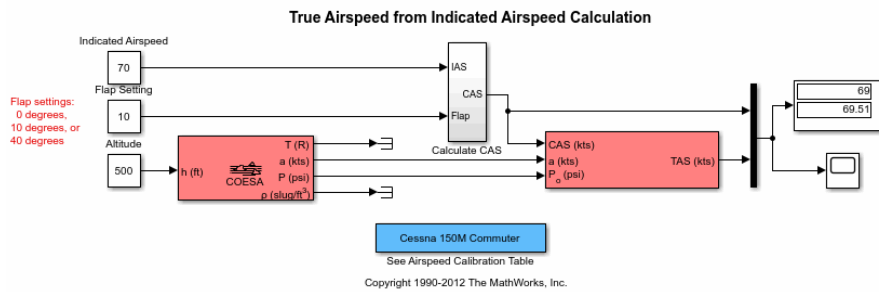
The model's Display block shows both indicated and calibrated airspeeds.



In the `aeroblk_calibrated` model, the aircraft is defined to be traveling at a constant speed of 70 knots (indicated airspeed) and altitude of 500 feet. The flaps are set to 10 degrees. The COESA Atmosphere Model block takes the altitude as input and outputs the speed of sound and air pressure. The Calculate CAS subsystem uses the flap setting and indicated airspeed to calculate the

calibrated airspeed. Finally, using the speed of sound, air pressure, and true calibrated airspeed as inputs, the Ideal Airspeed Correction block converts calibrated airspeed back to true airspeed.

The model's Display block shows both calibrated and true airspeeds.



# 1903 Wright Flyer

**In this section...**

“Introduction” on page 3-7  
“Wright Flyer Model” on page 3-7  
“Airframe Subsystem” on page 3-8  
“Environment Subsystem” on page 3-10  
“Pilot Subsystem” on page 3-11  
“Run the Simulation” on page 3-11  
“References” on page 3-12

## Introduction

---

**Note** The final section of this study requires the Simulink 3D Animation software.

---

This case study describes a model of the 1903 Wright Flyer. Built by Orville and Wilbur Wright, the Wright Flyer took to the skies in December 1903 and opened the age of controlled flight. The Wright brothers' flying machine achieved the following goals:

- Left the ground under its own power
- Moved forward and maintained its speed
- Landed at an elevation no lower than where it started

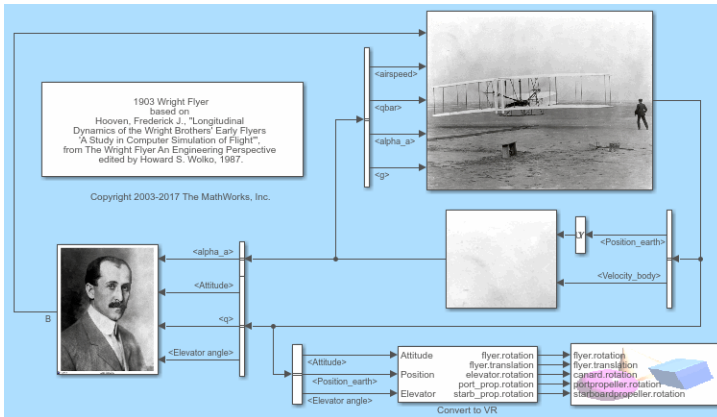
This model is based on an earlier simulation [1] that explored the longitudinal stability of the Wright Flyer and therefore modeled only forward and vertical motion along with the pitch angle. The Wright Flyer suffered from numerous engineering challenges, including dynamic and static instability. Laterally, the Flyer tended to overturn in crosswinds and gusts, and longitudinally, its pitch angle would undulate [2].

Under these constraints, the model recreates the longitudinal flight dynamics that pilots of the Wright Flyer would have experienced. Because they were able to control lateral motion, Orville and Wilbur Wright were able to maintain a relatively straight flight path.

Note, running this model generates information messages in the MATLAB Command Window and assertion warning messages in the Diagnostic Viewer. This is because the model illustrates the use of the Assertion block to indicate that the flyer is hitting the ground when landing.

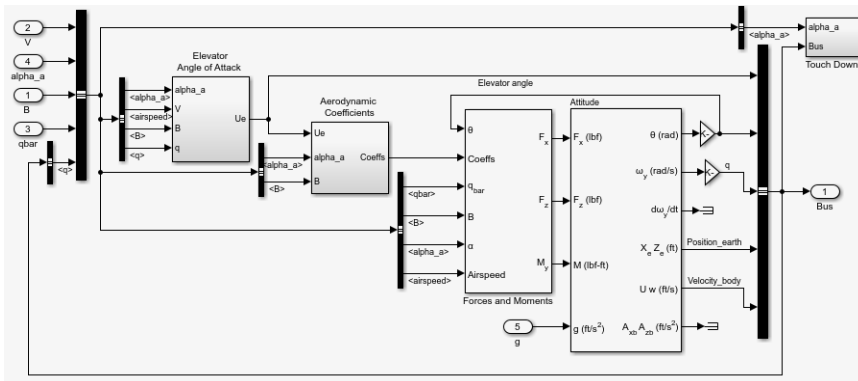
## Wright Flyer Model

Open the Wright Flyer model by entering `aeroblk_wf_3dof` at the MATLAB command line.



### Airframe Subsystem

The Airframe subsystem simulates the rigid body dynamics of the Wright Flyer airframe, including elevator angle of attack, aerodynamic coefficients, forces and moments, and three-degrees-of-freedom equations of motion.

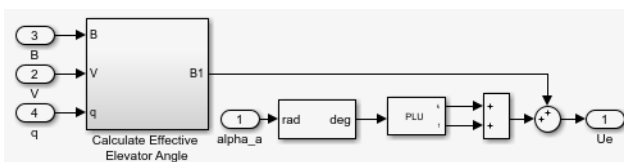


The Airframe subsystem consists of the following parts:

- “Elevator Angle of Attack Subsystem” on page 3-8
- “Aerodynamic Coefficients Subsystem” on page 3-9
- “Forces and Moments Subsystem” on page 3-9
- “3DOF (Body Axes) Block” on page 3-9

### Elevator Angle of Attack Subsystem

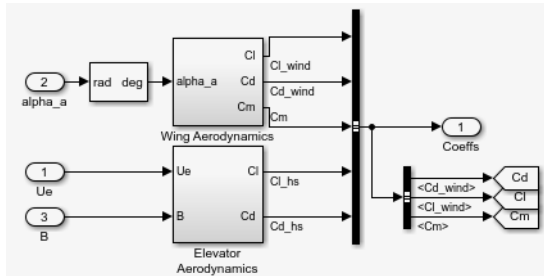
The Elevator Angle of Attack subsystem calculates the effective elevator angle for the Wright Flyer airframe and feeds its output to the Pilot subsystem.





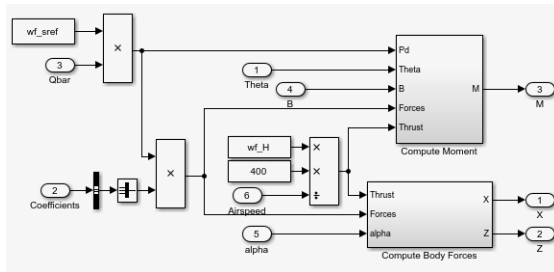
## Aerodynamic Coefficients Subsystem

The Aerodynamic Coefficients subsystem contains aerodynamic data and equations for calculating the aerodynamic coefficients, which are summed and passed to the Forces and Moments subsystem. Stored in data sets, the aerodynamic coefficients are determined by interpolation using Prelookup blocks.



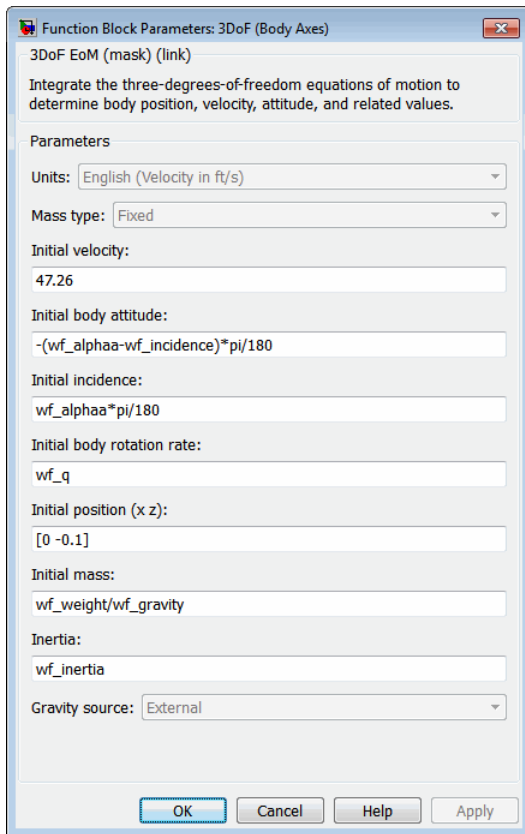
## Forces and Moments Subsystem

The aerodynamic forces and moments acting on the airframe are generated from aerodynamic coefficients. The Forces and Moments subsystem calculates the body forces and body moments acting on the airframe about the center of gravity. These forces and moments depend on the aerodynamic coefficients, thrust, dynamic pressure, and reference airframe parameters.



## 3DOF (Body Axes) Block

The 3DOF (Body Axes) block use equations of motion to define the linear and angular motion of the Wright Flyer airframe. It also performs conversions from the original model's axis system and the body axes.



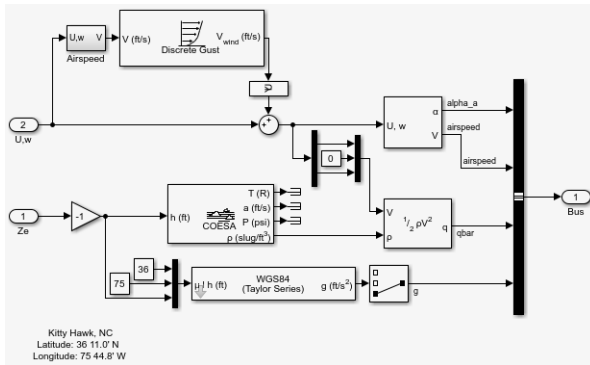
### 3DOF (Body Axes) Block Parameters

## Environment Subsystem

The first and final flights of the Wright Flyer occurred on December 17, 1903. Orville and Wilbur Wright chose an area near Kitty Hawk, North Carolina, situated near the Atlantic coast. Wind gusts of more than 25 miles per hour were recorded that day. After the final flight on that blustery December day, a wind gust caught and overturned the Wright Flyer, damaging it beyond repair.

The Environment subsystem of the Wright Flyer model contains a variety of blocks from the Environment sublibrary of the Aerospace Blockset software, including wind, atmosphere, and gravity, and calculates airspeed and dynamic pressure. The Discrete Wind Gust Model block provides wind gusts to the simulated environment. The other blocks are

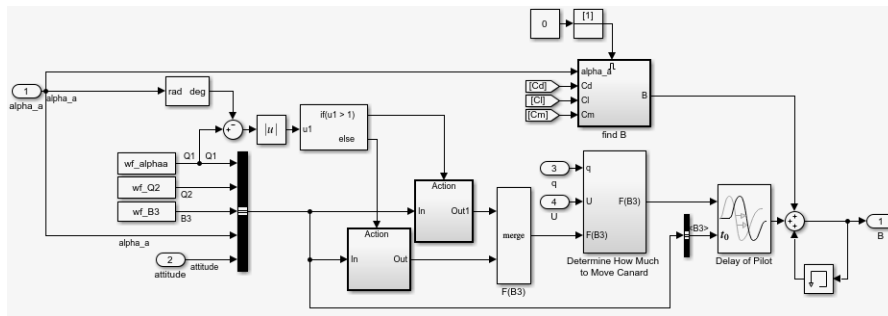
- The Incidence & Airspeed block calculates the angle of attack and airspeed.
- The COESA Atmosphere Model block calculates the air density.
- The Dynamic Pressure block computes the dynamic pressure from the air density and velocity.
- The WGS84 Gravity Model block produces the gravity at the Wright Flyer's latitude, longitude, and height.



## Pilot Subsystem

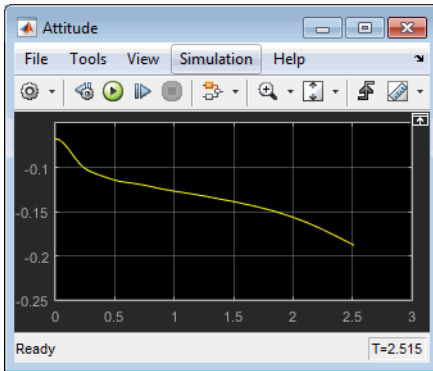
The Pilot subsystem controls the aircraft by responding to both pitch angle (attitude) and angle of attack. If the angle of attack differs from the set angle of attack by more than one degree, the Pilot subsystem responds with a correction of the elevator (canard) angle. When the angular velocity exceeds  $\pm 0.02$  rad/s, angular velocity and angular acceleration are also taken into consideration with additional corrections to the elevator angle.

Pilot reaction time largely determined the success of the flights [1]. Without an automatic controller, a reaction time of 0.06 seconds is optimal for successful flight. The Delay of Pilot (Variable Time Delay) block recreates this effect by producing a delay of no more than 0.08 second.



## Run the Simulation

The default values for this simulation allow the Wright Flyer model to take off and land successfully. The pilot reaction time (`wf_B3`) is set to 0.06 seconds, the desired angle of attack (`wf_alphaa`) is constant, and the altitude attained is low. The Wright Flyer model reacts similarly to the actual Wright Flyer. It leaves the ground, moves forward, and lands on a point as high as that from which it started. This model exhibits the longitudinal undulation in attitude of the original aircraft.



#### Attitude Scope (Measured in Radians)

A pilot with quick reaction times and ideal flight conditions makes it possible to fly the Wright Flyer successfully. The Wright Flyer model confirms that controlling its longitudinal motion was a serious challenge. The longest recorded flight on that day lasted a mere 59 seconds and covered 852 feet.

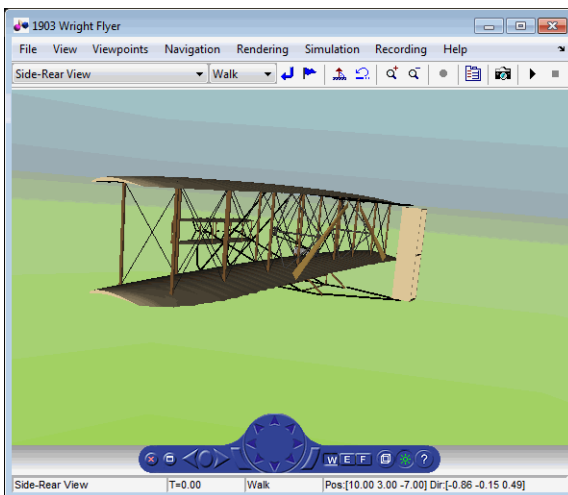
#### Virtual Reality Visualization of the Wright Flyer

---

**Note** This section requires the Simulink 3D Animation.

---

The Wright Flyer model also provides a virtual world visualization, coded in Virtual Reality Modeling Language (VRML) [3]. The VR Sink block in the main model allows you to view the flight motion in three dimensions.



#### 1903 Wright Flyer Virtual Reality World

### References

- [1] Hooven, Frederick J., "Longitudinal Dynamics of the Wright Brothers' Early Flyers: A Study in Computer Simulation of Flight," from *The Wright Flyer: An Engineering Perspective*, ed. Howard S. Wolko, Smithsonian Institution Press, 1987.

[2] Culick, F. E. C. and H. R. Jex, "Aerodynamics, Stability, and Control of the 1903 Wright Flyer," from *The Wright Flyer: An Engineering Perspective*, ed. Howard S. Wolko, Smithsonian Institution Press, 1987.

[3] Thaddeus Beier created the initial Wright Flyer model in Inventor format, and Timothy Rohaly converted it to VRML.

**Additional Information About the 1903 Wright Flyer**

- <https://www.wrightexperience.com>
- <https://wright.nasa.gov>

## NASA HL-20 Lifting Body Airframe

### In this section...

“Introduction” on page 3-14

“NASA HL-20 Lifting Body” on page 3-14

“The HL-20 Airframe and Controller Model” on page 3-15

“References” on page 3-21

### Introduction

This case study models the airframe of a NASA HL-20 lifting body, a low-cost complement to the Space Shuttle orbiter. The HL-20 is unpowered, but the model includes both airframe and controller.

For most flight control designs, the airframe, or plant model, needs to be modeled, simulated, and analyzed. Ideally, this airframe should be modeled quickly, reusing blocks or model structure to reduce validation time and leave more time available for control design. In this study, the Aerospace Blockset software efficiently models portions of the HL-20 airframe. The remaining portions, including calculation of the aerodynamic coefficients, are modeled with the Simulink software. This case study examines the HL-20 airframe model and touches on how the aerodynamic data are used in the model.

### NASA HL-20 Lifting Body

The HL-20, also known as the Personnel Launch System (PLS), is a lifting body reentry vehicle designed to complement the Space Shuttle orbiter. It was developed originally as a low-cost solution for getting to and from low Earth orbit. It can carry up to 10 people and a limited cargo [1].

The HL-20 lifting body can be placed in orbit either by launching it vertically with booster rockets or by transporting it in the payload bay of the Space Shuttle orbiter. The HL-20 lifting body deorbits using a small onboard propulsion system. Its reentry profile is nose first, horizontal, and unpowered.



**Top-Front View of the HL-20 Lifting Body (Photo: NASA Langley)**

The HL-20 design has a number of benefits:

- Rapid turnaround between landing and launch reduces operating costs.
- The HL-20 has exceptional flight safety.
- It can land conventionally on aircraft runways.

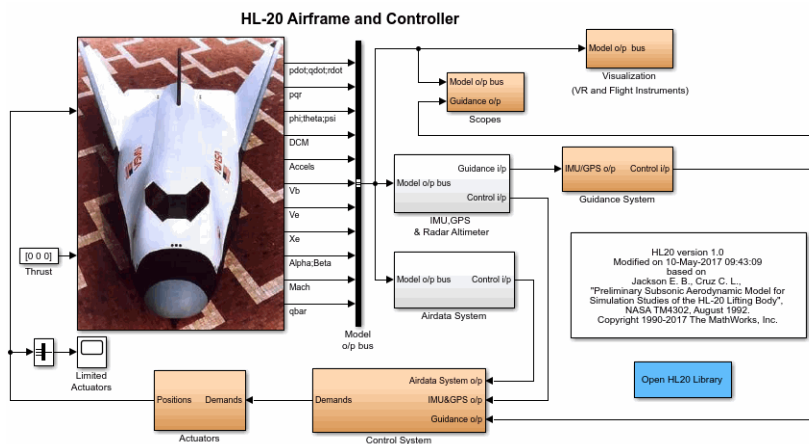
Potential uses for the HL-20 include

- Orbital rescue of stranded astronauts
- International Space Station crew exchanges
- Observation missions
- Satellite servicing missions

Although the HL-20 program is not currently active, the aerodynamic data from HL-20 tests are being used in current NASA projects [2].

## The HL-20 Airframe and Controller Model

You can open the HL-20 airframe and controller model by entering `aeroblk_HL20` at the MATLAB command line.



## Modeling Assumptions and Limitations

Preliminary aerodynamic data for the HL-20 lifting body are taken from NASA document TM4302 [1].

The airframe model incorporates several key assumptions and limitations:

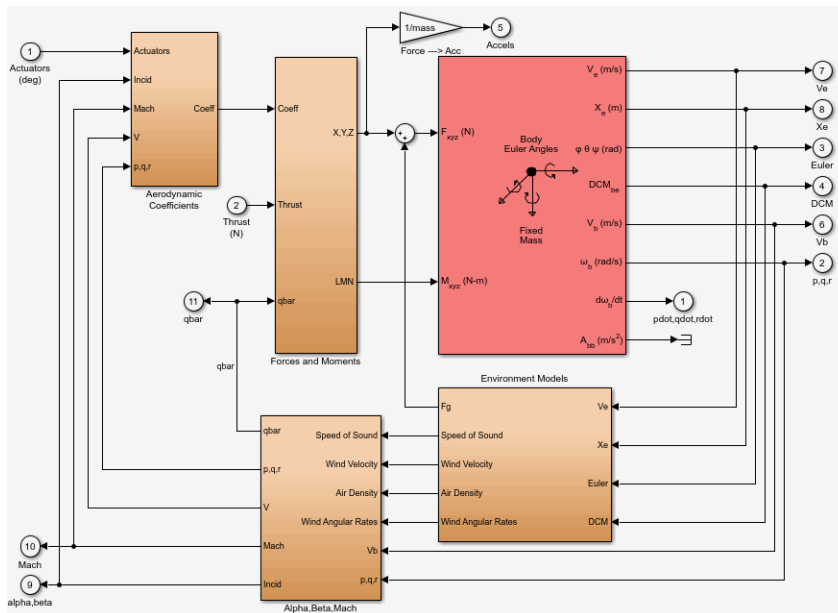
- The airframe is assumed to be rigid and have constant mass, center of gravity, and inertia, since the model represents only the unpowered reentry portion of a mission.
- HL-20 is assumed to be a laterally symmetric vehicle.
- Compressibility (Mach) effects are assumed to be negligible.
- Control effectiveness is assumed to vary nonlinearly with angle of attack and linearly with angle of deflection. Control effectiveness is not dependent on sideslip angle.
- The nonlinear six-degrees-of-freedom aerodynamic model is a representation of an early version of the HL-20. Therefore, the model is not intended for realistic performance simulation of later versions of the HL-20.

The typical airframe model consists of a number of components, such as

- Equations of motion
- Environmental models
- Calculation of aerodynamic coefficients, forces, and moments

The airframe subsystem of the HL-20 model contains five subsystems, which model the typical airframe components:

- “6DOF (Euler Angles) Subsystem” on page 3-16
- “Environmental Models Subsystem” on page 3-16
- “Alpha, Beta, Mach Subsystem” on page 3-18
- “Aerodynamic Coefficients Subsystem” on page 3-19
- “Forces and Moments Subsystem” on page 3-21



### HL-20 Airframe Subsystem

#### 6DOF (Euler Angles) Subsystem

The 6DOF (Euler Angles) subsystem contains the six-degrees-of-freedom equations of motion for the airframe. In the 6DOF (Euler Angles) subsystem, the body attitude is propagated in time using an Euler angle representation. This subsystem is one of the equations of motion blocks from the Aerospace Blockset library. A quaternion representation is also available. See the 6DOF (Euler Angles) and 6DOF (Quaternion) block reference pages for more information on these blocks.

#### Environmental Models Subsystem

The Environmental Models subsystem contains the following subsystems and blocks:

- The WGS84 Gravity Model block implements the mathematical representation of the geocentric equipotential ellipsoid of the World Geodetic System (WGS84).

See the WGS84 Gravity Model block reference page for more information on this block.



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

See the COESA Atmosphere Model block reference page for more information on this block.

- The Wind Models subsystem contains the following blocks:

- The Wind Shear Model block adds wind shear to the model.

See the Wind Shear Model block reference page for more information on this block.

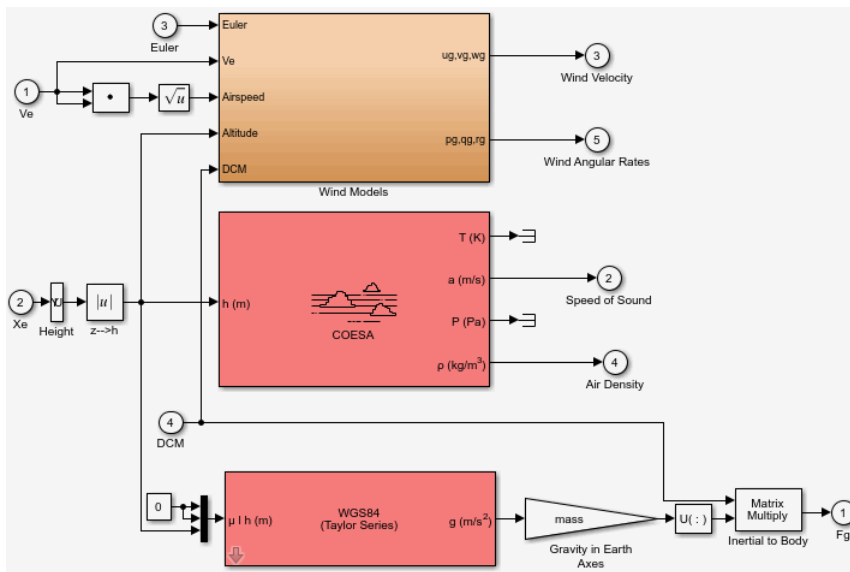
- The Discrete Wind Gust Model block implements a wind gust of the standard “1 - cosine” shape.

See the Discrete Wind Gust Model block reference page for more information on this block.

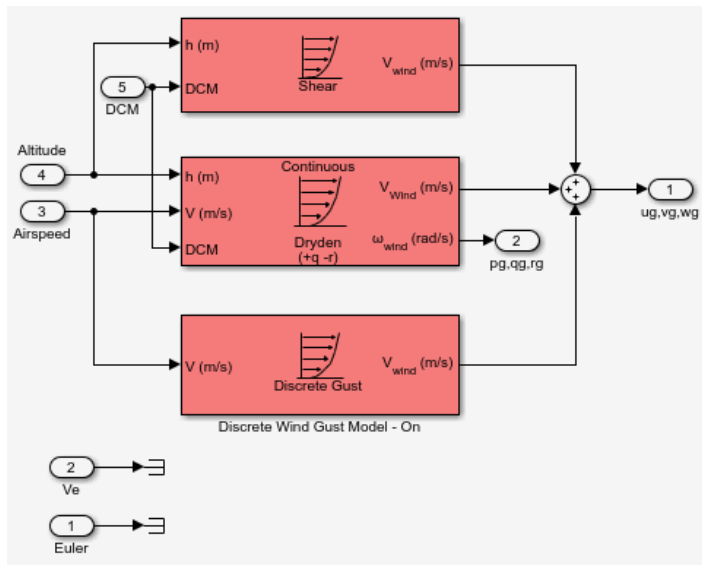
- The Dryden Wind Turbulence Model (Continuous) block uses the Dryden spectral representation to add turbulence to the aerospace model by passing band-limited white noise through appropriate forming filters.

See the Dryden Wind Turbulence Model (Continuous) block reference page for more information on this block.

The environmental models implement mathematical representations within standard references, such as U.S. Standard Atmosphere, 1976.



### Environmental Models in HL-20 Airframe Model



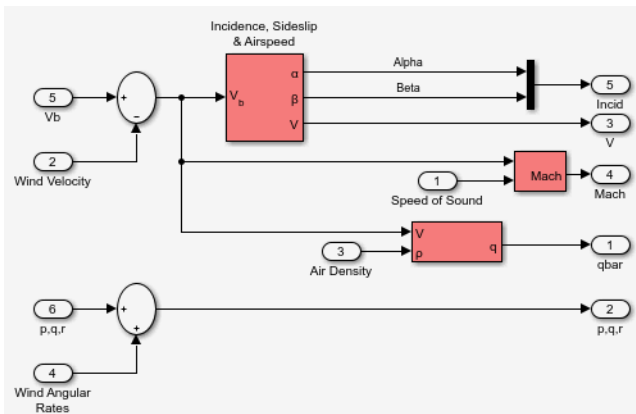
### Wind Models in HL-20 Airframe Model

#### Alpha, Beta, Mach Subsystem

The Alpha, Beta, Mach subsystem calculates additional parameters needed for the aerodynamic coefficient computation and lookup. These additional parameters include

- Mach number
- Incidence angles ( $\alpha$ ,  $\beta$ )
- Airspeed
- Dynamic pressure

The Alpha, Beta, Mach subsystem corrects the body velocity for wind velocity and corrects the body rates for wind angular acceleration.



#### Additional Computed Parameters for HL-20 Airframe Model (Alpha, Beta, Mach Subsystem)

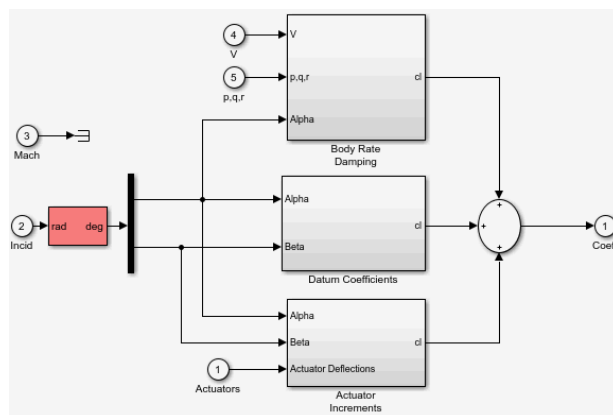
## Aerodynamic Coefficients Subsystem

The Aerodynamic Coefficients subsystem contains aerodynamic data and equations for calculating the six aerodynamic coefficients, which are implemented as in reference [1]. The six aerodynamic coefficients follow.

|       |                             |
|-------|-----------------------------|
| $C_x$ | Axial-force coefficient     |
| $C_y$ | Side-force coefficient      |
| $C_z$ | Normal-force coefficient    |
| $C_l$ | Rolling-moment coefficient  |
| $C_m$ | Pitching-moment coefficient |
| $C_n$ | Yawing-moment coefficient   |

Ground and landing gear effects are not included in this model.

The contribution of each of these coefficients is calculated in the subsystems (body rate, actuator increment, and datum), and then summed and passed to the Forces and Moments subsystem.



## Aerodynamic Coefficients in HL-20 Airframe Model

The aerodynamic data was gathered from wind tunnel tests, mainly on scaled models of a preliminary subsonic aerodynamic model of the HL-20. The data was curve fitted, and most of the aerodynamic coefficients are described by polynomial functions of angle of attack and sideslip angle. In-depth details about the aerodynamic data and the data reduction can be found in reference [1].

The polynomial functions contained in the `aeroblk_init_hl20.m` file are used to calculate lookup tables used by the model's preload function. Lookup tables substitute for polynomial functions. Depending on the order and implementation of the function, using lookup tables can be more efficient than recalculating values at each time step with functions. To further improve efficiency, most tables are implemented as PreLook-up Index Search and Interpolation (n-D) using PreLook-up blocks. These blocks improve performance most when the model has a number of tables with identical breakpoints. These blocks reduce the number of times the model has to search for a breakpoint in a given time step. Once the tables are populated by the preload function, the aerodynamic coefficient can be computed.

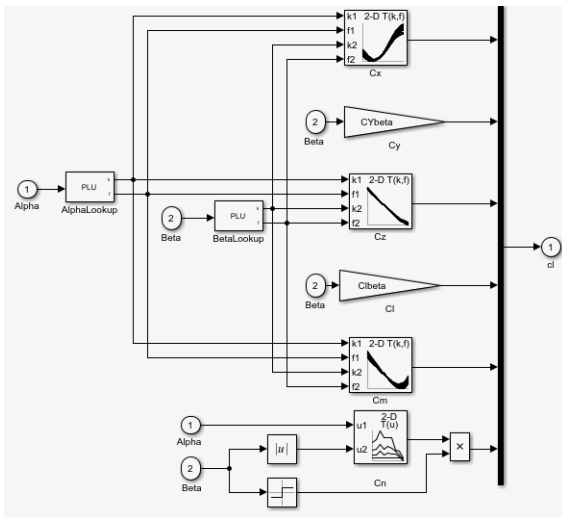
The equations for calculating the six aerodynamic coefficients are divided among three subsystems:

- “Datum Coefficients Subsystem” on page 3-20
- “Body Rate Damping Subsystem” on page 3-20
- “Actuator Increment Subsystem” on page 3-20

Summing the Datum Coefficients, Body Rate Damping, and Actuator Increments subsystem outputs generates the six aerodynamic coefficients used to calculate the airframe forces and moments [1].

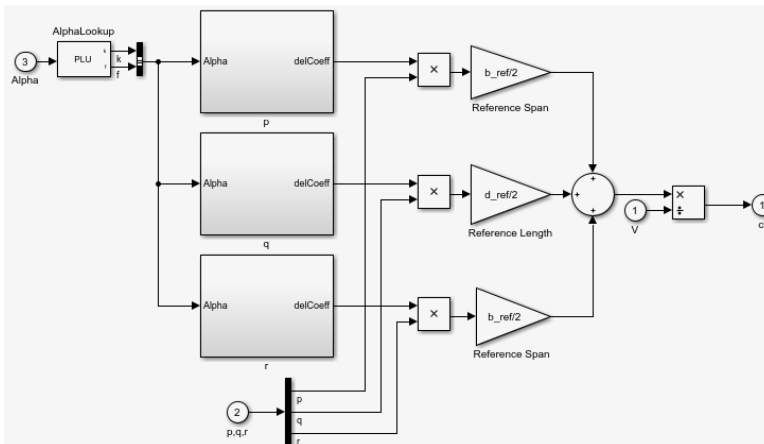
### Datum Coefficients Subsystem

The Datum Coefficients subsystem calculates coefficients for the basic configuration without control surface deflection. These datum coefficients depend only on the incidence angles of the body.



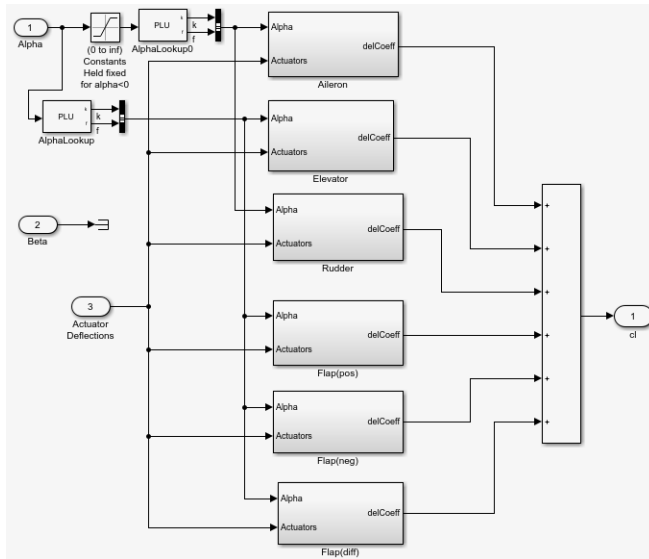
### Body Rate Damping Subsystem

Dynamic motion derivatives are computed in the Body Rate Damping subsystem.



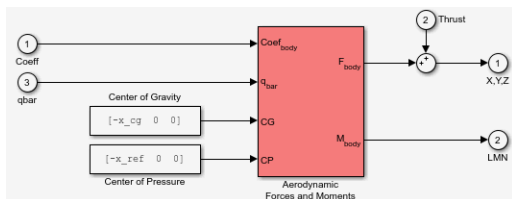
### Actuator Increment Subsystem

Lookup tables determine the incremental changes to the coefficients due to the control surface deflections in the Actuator Increment subsystem. Available control surfaces include symmetric wing flaps (elevator), differential wing flaps (ailerons), positive body flaps, negative body flaps, differential body flaps, and an all-movable rudder.



### Forces and Moments Subsystem

The Forces and Moments subsystem calculates the body forces and body moments acting on the airframe about the center of gravity. These forces and moments depend on the aerodynamic coefficients, thrust, dynamic pressure, and reference airframe parameters.



### Complete the Model

These subsystems that you have examined complete the HL-20 airframe. The next step in the flight control design process is to analyze, trim, and linearize the HL-20 airframe so that a flight control system can be designed for it. You can see an example of an auto-land flight control for the HL-20 airframe in the `aeroblk_HL20` example.

## References

- [1] Jackson, E. B., and C. L. Cruz, "Preliminary Subsonic Aerodynamic Model for Simulation Studies of the HL-20 Lifting Body," NASA TM4302 (August 1992). This document is included in the HL-20 Lifting Body .zip file available from MATLAB Central.
- [2] Moring, F., Jr., "ISS 'Lifeboat' Study Includes ELVs," *Aviation Week & Space Technology* (May 20, 2002).

### Additional Information About the HL-20 Lifting Body

<http://www.astronautix.com/h/hl-20.html>



# Blocks

---

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

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

**Library:** Aerospace Blockset / GNC / Control



### Description

The 1D Controller  $[A(v),B(v),C(v),D(v)]$  block implements a gain-scheduled state-space controller, as described in “Algorithms” on page 4-4.

The output from this block is the actuator demand, which you can input to an actuator block.

### Limitations

If the scheduling parameter inputs to the block go out of range, they are clipped. The state-space matrices are not interpolated out of range.

### Ports

#### Input

##### **y — Aircraft measurements**

vector

Aircraft measurements, specified as a vector.

Data Types: double

##### **v — Scheduling variable**

vector

Scheduling variable, specified as a vector, that conforms to the dimensions of the state-space matrices.

Data Types: double

#### Output

##### **u — Actuator demands**

vector

Actuator demands, specified as a vector.

Data Types: double

### Parameters

##### **A-matrix(v) — A matrix of the state-space implementation**

A1 (default) | array



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

**Programmatic Use**

**Block Parameter:** A

**Type:** character vector

**Values:** vector

**Default:** 'A1'

**B-matrix(v) — B matrix of the state-space implementation**

B1 (default) | array

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

**Programmatic Use**

**Block Parameter:** B

**Type:** character vector

**Values:** vector

**Default:** 'B1'

**C-matrix(v) — C matrix of the state-space implementation**

C1 (default) | array

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

**Programmatic Use**

**Block Parameter:** C

**Type:** character vector

**Values:** vector

**Default:** 'C1'

**D-matrix(v) — D**

D1 (default) | array

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

**Programmatic Use**

**Block Parameter:** D

**Type:** character vector

**Values:** vector

**Default:** 'D1'

**Scheduling variable breakpoints — Breakpoints for scheduling variable**

v\_vec (default) | vector

Breakpoints for the scheduling variable, specified as a vector. The length of  $v$  must be the same as the size of the third dimension of  $A$ ,  $B$ ,  $C$ , and  $D$ .

**Programmatic Use****Block Parameter:** AoA\_vec**Type:** character vector**Values:** vector**Default:** 'v\_vec'**Initial state, x\_initial – Initial states** $0$  (default) | vector

Initial states for the controller, such as initial values for the state vector,  $x$ , specified as a vector. The length of the vector must equal the size of the first dimension of  $A$ .

**Programmatic Use****Block Parameter:** x\_initial**Type:** character vector**Values:** vector**Default:** '0'

## Algorithms

The block implements a gain-scheduled state-space controller as defined by this equation:

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

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

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.

## Extended Capabilities

**C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**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 [A(v),B(v),C(v),D(v)] | 3D Controller [A(v),B(v),C(v),D(v)] | Linear Second-Order Actuator | Nonlinear Second-Order Actuator

**Introduced before R2006a**

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

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

**Library:** Aerospace Blockset / GNC / Control



## Description

The 1D Controller Blend  $u=(1-L).K1.y+L.K2.y$  block implements an array of state-space controller designs. The model runs the controllers in parallel and interpolates their outputs 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. The output from this block is the actuator demand, which you can input to an actuator block.

## Limitations

This block requires the Control System Toolbox™ license.

## Ports

### Input

#### **y** — Aircraft measurements

vector

Aircraft measurements, specified as a vector.

Data Types: double

#### **v** — Scheduling variable

vector

Scheduling variable, specified as a vector, that conforms to the dimensions of the state-space matrices.

Data Types: double

### Output

#### **u** — Actuator demands

vector

Actuator demands, specified as a vector.

Data Types: double

## Parameters

#### **A-matrix(v)** — A-matrix of the state-space implementation

A1 (default) | array

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

**Programmatic Use****Block Parameter:** A**Type:** character vector**Values:** vector**Default:** 'A1'**B-matrix(v) — B-matrix of the state-space implementation**

B1 (default) | array

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

**Programmatic Use****Block Parameter:** B**Type:** character vector**Values:** vector**Default:** 'B1'**C-matrix(v) — C-matrix of the state-space implementation**

C1 (default) | array

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

**Programmatic Use****Block Parameter:** C**Type:** character vector**Values:** vector**Default:** 'C1'**D-matrix(v) — D-matrix of the state-space implementation**

D1 (default) | array

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

**Programmatic Use****Block Parameter:** D**Type:** character vector**Values:** vector**Default:** 'D1'**Scheduling variable breakpoints — Breakpoints for scheduling variable**

[1 1.5 2] (default) | vector

Breakpoints for the scheduling variable, specified as a vector. The length of  $v$  must be same as the size of the third dimension of  $A$ ,  $B$ ,  $C$ , and  $D$ .

**Programmatic Use**

**Block Parameter:** breakpoints\_v

**Type:** character vector

**Values:** vector

**Default:** '[1 1.5 2]'

**Initial state, x\_initial — Initial states**

0 (default) | vector

Initial states for the controller, such as initial values for the state vector,  $x$ , specified as a vector. The length must equal the size of the first dimension of  $A$ .

**Programmatic Use**

**Block Parameter:** x\_initial

**Type:** character vector

**Values:** vector

**Default:** '0'

**Poles of  $A(v)-H(v)*C(v) = [w1 \dots wn]$  — Poles of observer**

[-5 -2] (default) | vector

Poles of observer, specified as a vector. For incoming controllers, the block uses an observer-like structure to ensure that the controller output tracks the current block output,  $u$ . The number of poles must equal the dimension of the  $A$ -matrix. Poles that are too fast result in sensor noise propagation; poles that are too slow result in the failure of the controller output to track  $u$ .

**Programmatic Use**

**Block Parameter:** vec\_w

**Type:** character vector

**Values:** vector

**Default:** '[-5 -2]'

## Algorithms

The block implements

$$\dot{x}_1 = A_1x_1 + B_1y$$

$$u_1 = C_1x_1 + D_1y$$

$$\dot{x}_2 = A_2x_2 + B_2y$$

$$u_2 = C_2x_2 + D_2y$$

$$u = (1 - \lambda)u_1 + \lambda u_2$$

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

For example, suppose two controllers are designed at two operating points  $v=v_{\min}$  and  $v=v_{\max}$ . For longer arrays of design points, the block only implements nearest neighbor designs. At any given instant in time, the block updates three controller designs, reducing computational requirements.

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

## References

- [1] Hyde, R. A., "H-infinity Aerospace Control Design — A VSTOL Flight Application." , *Advances in Industrial Control Series*, Springer Verlag, 1995.

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

## See Also

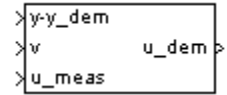
1D Controller [A(v),B(v),C(v),D(v)] | 1D Observer Form [A(v),B(v),C(v),F(v),H(v)] | 1D Self-Conditioned [A(v),B(v),C(v),D(v)] | 2D Controller Blend | Self-Conditioned [A,B,C,D] | Linear Second-Order Actuator | Nonlinear Second-Order Actuator

**Introduced before R2006a**

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

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

**Library:** Aerospace Blockset / GNC / Control



## Description

The 1D Observer Form [A(v),B(v),C(v),F(v),H(v)] block implements a gain-scheduled state-space controller as defined in “Algorithms” on page 4-11.

The output from this block is the actuator demand, which you can input to an actuator block. Use this block to implement a controller designed using  $H$ -infinity loop-shaping, one of the design methods supported by Robust Control Toolbox.

## Limitations

If the scheduling parameter inputs to the block go out of range, they are clipped. The state-space matrices are not interpolated out of range.

## Ports

### Input

#### **y-y\_dem — Set-point error**

vector

Set-point error, specified as a vector, that conforms to the dimensions of the state-space matrices.

Data Types: double

#### **v — Scheduling variable**

vector

Scheduling variable, specified as a vector, that conforms to the dimensions of the state-space matrices.

Data Types: double

#### **u\_meas — Measured actuator position**

vector

Measured actuator position, specified as a vector.

Data Types: double

### Output

#### **u\_dem — Actuator demands**

vector

Actuator demands, specified as a vector.

Data Types: double

## Parameters

### **A-matrix(v) — A-matrix of the state-space implementation**

A (default) | array

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

#### **Programmatic Use**

**Block Parameter:** A

**Type:** character vector

**Values:** vector

**Default:** 'A'

### **B-matrix(v) — B-matrix of the state-space implementation**

B (default) | array

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

#### **Programmatic Use**

**Block Parameter:** B

**Type:** character vector

**Values:** vector

**Default:** 'B'

### **C-matrix(v) — C-matrix of the state-space implementation**

C (default) | array

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  $C(:, :, 1) = [1 \ 0; 0 \ 1];$ .

#### **Programmatic Use**

**Block Parameter:** C

**Type:** character vector

**Values:** vector

**Default:** 'C'

### **F-matrix(v) — F-matrix of the state-space implementation**

F (default) | array

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  $F(:, :, 1) = [1 \ 0; 0 \ 1];$ .

#### **Programmatic Use**

**Block Parameter:** F

**Type:** character vector



**Values:** vector

**Default:** 'F'

### **H-matrix(v) — H-matrix of the state-space implementation**

H (default) | array

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  $H(:, :, 1) = [1 \ 0; 0 \ 1];$ .

#### **Programmatic Use**

**Block Parameter:** H

**Type:** character vector

**Values:** vector

**Default:** 'H'

### **Scheduling variable breakpoints — Breakpoints for scheduling variable**

v\_vec (default) | vector

Breakpoints for the scheduling variable, specified as a vector. The length of  $v$  should be same as the size of the third dimension of  $A$ ,  $B$ ,  $C$ ,  $F$ , and  $H$ .

#### **Programmatic Use**

**Block Parameter:** AoA\_vec

**Type:** character vector

**Values:** vector

**Default:** 'v\_vec'

### **Initial state, x\_initial — Initial states**

0 (default) | vector

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

#### **Programmatic Use**

**Block Parameter:** x\_initial

**Type:** character vector

**Values:** vector

**Default:** '0'

## **Algorithms**

The block implements a gain-scheduled state-space controller defined in the following observer form:

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

## **References**

- [1] Hyde, R. A., "H-infinity Aerospace Control Design — A VSTOL Flight Application," Springer Verlag, *Advances in Industrial Control Series*, 1995.

## **Extended Capabilities**

### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

### **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)] | Linear Second-Order Actuator | Nonlinear Second-Order Actuator

**Introduced before R2006a**

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

Implement gain-scheduled state-space controller in self-conditioned form depending on one scheduling parameter

**Library:** Aerospace Blockset / GNC / Control



## Description

The 1D Self-Conditioned [A(v),B(v),C(v),D(v)] block implements a gain-scheduled state-space controller as defined in “Algorithms” on page 4-15.

The output from this block is the actuator demand, which you can input to an actuator block.

## Limitations

- If the scheduling parameter inputs to the block go out of range, they are clipped. The state-space matrices are not interpolated out of range.
- This block requires the Control System Toolbox license.

## Ports

### Input

#### **y — Aircraft measurements**

vector

Aircraft measurements, specified as a vector.

Data Types: double

#### **v — Scheduling variable**

vector

Scheduling variable, specified as a vector, ordered according to the dimensions of the state-space matrices.

Data Types: double

#### **u\_meas — Measured actuator position**

vector

Measured actuator position, specified as a vector.

Data Types: double

### Output

#### **u\_dem — Actuator demands**

vector

Actuator demands, specified as a vector.

Data Types: double

## Parameters

### **A-matrix(v) — A-matrix of the state-space implementation**

A (default) | array

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

#### **Programmatic Use**

**Block Parameter:** A

**Type:** character vector

**Values:** vector

**Default:** 'A'

### **B-matrix(v) — B-matrix of the state-space implementation**

B (default) | array

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

#### **Programmatic Use**

**Block Parameter:** B

**Type:** character vector

**Values:** vector

**Default:** 'B'

### **C-matrix(v) — C-matrix of the state-space implementation**

C (default) | array

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

#### **Programmatic Use**

**Block Parameter:** C

**Type:** character vector

**Values:** vector

**Default:** 'C'

### **D-matrix(v) — D-matrix of the state-space implementation**

D (default) | array

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

#### **Programmatic Use**

**Block Parameter:** D

**Type:** character vector

**Values:** vector

**Default:** 'D'

### Scheduling variable breakpoints — Breakpoints for scheduling variable

$v\_vec$  (default) | vector

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  $D$ .

#### Programmatic Use

**Block Parameter:** breakpoints\_v

**Type:** character vector

**Values:** vector

**Default:** 'v\_vec'

### Initial state, x\_initial — Initial states

$\theta$  (default) | vector

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

#### Programmatic Use

**Block Parameter:** x\_initial

**Type:** character vector

**Values:** vector

**Default:** '0'

### Poles of $A(v) - H(v) * C(v)$ — Desired poles

[-5 -2] (default) | vector

Desired poles of  $A-HC$ , specified as a vector. 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.

#### Programmatic Use

**Block Parameter:** vec\_w

**Type:** character vector

**Values:** vector

**Default:** '[-5 -2]'

## Algorithms

The 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

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

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

This block implements a gain-scheduled version of the Self-Conditioned [A,B,C,D] block, where  $v$  is 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.

## References

- [1] Kautsky, Nichols, and Van Dooren. "Robust Pole Assignment in Linear State Feedback." *International Journal of Control*, Vol. 41, Number 5, 1985, pp. 1129-1155.

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

## 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)] | Self-Conditioned [A,B,C,D] | Linear Second-Order Actuator | Nonlinear Second-Order Actuator | Self-Conditioned [A,B,C,D]

**Introduced before R2006a**

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

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

**Library:** Aerospace Blockset / GNC / Control



### Description

The 2D Controller [A(v),B(v),C(v),D(v)] block implements a gain-scheduled state-space controller, as described in “Algorithms” on page 4-19.

The output from this block is the actuator demand, which you can input to an actuator block.

### Limitations

If the scheduling parameter inputs to the block go out of range, they are clipped. The state-space matrices are not interpolated out of range.

### Ports

#### Input

##### **y — Aircraft measurements**

vector

Aircraft measurements, specified as a vector.

Data Types: double

##### **v1 — Scheduling variable**

vector

Scheduling variable, specified as a vector, that conforms to the dimensions of the state-space matrices.

Data Types: double

##### **v2 — Scheduling variable**

vector

Scheduling variable, specified as a vector, that conforms to the dimensions of the state-space matrices.

Data Types: double

#### Output

##### **u — Actuator demands**

vector

Actuator demands, specified as a vector.

Data Types: double

## Parameters

### **A-matrix(v1,v2) — A-matrix of the state-space implementation**

A (default) | array

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

#### **Programmatic Use**

**Block Parameter:** A

**Type:** character vector

**Values:** vector

**Default:** 'A'

### **B-matrix(v1,v2) — B-matrix of the state-space implementation**

B (default) | array

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

#### **Programmatic Use**

**Block Parameter:** B

**Type:** character vector

**Values:** vector

**Default:** 'B'

### **C-matrix(v1,v2) — C-matrix of the state-space implementation**

C (default) | array

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

#### **Programmatic Use**

**Block Parameter:** C

**Type:** character vector

**Values:** vector

**Default:** 'C'

### **D-matrix(v1,v2) — D-matrix of the state-space implementation**

D (default) | array

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



**Programmatic Use****Block Parameter:** D**Type:** character vector**Values:** vector**Default:** 'D'**First scheduling variable (v1) breakpoints — Breakpoints for first scheduling variable**

v1\_vec (default) | vector

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  $D$ .

**Programmatic Use****Block Parameter:** AoA\_vec**Type:** character vector**Values:** vector**Default:** 'v1\_vec'**Second scheduling variable (v2) breakpoints — Breakpoints for second scheduling variable**

v2\_vec (default) | vector

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  $D$ .

**Programmatic Use****Block Parameter:** Mach\_vec**Type:** character vector**Values:** vector**Default:** 'v2\_vec'**Initial state, x\_initial — Initial states**

0 (default) | vector

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

**Programmatic Use****Block Parameter:** x\_initial**Type:** character vector**Values:** vector**Default:** '0'**Algorithms**

The block implements a gain-scheduled state-space controller as defined by this equation:

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

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.

## **Extended Capabilities**

### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

### **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)] | Linear Second-Order  
Actuator | Nonlinear Second-Order Actuator

**Introduced before R2006a**

## 2D Controller Blend

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

**Library:** Aerospace Blockset / GNC / Control



### 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. The output from this block is the actuator demand, which you can input to an actuator block.

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.

### Limitations

This block requires the Control System Toolbox license.

### Ports

#### Input

##### **y — Aircraft measurements**

vector

Aircraft measurements, specified as a vector.

Data Types: `double`

##### **v1 — Scheduling variable**

vector

Scheduling variable, specified as a vector, that conforms to the dimensions of the state-space matrices.

Data Types: `double`

##### **v2 — Scheduling variable**

vector

Scheduling variable, specified as a vector, that conforms to the dimensions of the state-space matrices.

Data Types: `double`

## Output

### **u — Actuator demands**

vector

Actuator demands, specified as a vector.

Data Types: `double`

## Parameters

### **A-matrix(v1,v2) — A-matrix of the state-space implementation**

A (default) | array

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

#### **Programmatic Use**

**Block Parameter:** A

**Type:** character vector

**Values:** vector

**Default:** 'A'

### **B-matrix(v1,v2) — B-matrix of the state-space implementation**

A (default) | array

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

#### **Programmatic Use**

**Block Parameter:** B

**Type:** character vector

**Values:** vector

**Default:** 'B'

### **C-matrix(v1,v2) — C-matrix of the state-space implementation**

C (default) | array

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

#### **Programmatic Use**

**Block Parameter:** C

**Type:** character vector

**Values:** vector

**Default:** 'C'

### **D-matrix(v1,v2) — D-matrix of the state-space implementation**

C (default) | array

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

**Programmatic Use**

**Block Parameter:**  $D$

**Type:** character vector

**Values:** vector

**Default:** 'D'

**First scheduling variable (v1) breakpoints – Breakpoints for first scheduling variable**

$v1\_vec$  (default) | vector

Breakpoints for the first scheduling variable, specified as a vector. The length of  $v1$  should be same as the size of the third dimension of  $A$ ,  $B$ ,  $C$ , and  $D$ .

**Programmatic Use**

**Block Parameter:**  $breakpoints\_v1$

**Type:** character vector

**Values:** vector

**Default:** 'v1\_vec'

**Second scheduling variable (v2) breakpoints – Breakpoints for second scheduling variable**

$v2\_vec$  (default) | vector

Breakpoints for the second scheduling variable, specified as a vector. The length of  $v2$  should be same as the size of the fourth dimension of  $A$ ,  $B$ ,  $C$ , and  $D$ .

**Programmatic Use**

**Block Parameter:**  $breakpoints\_v2$

**Type:** character vector

**Values:** vector

**Default:** 'v2\_vec'

**Initial state,  $x\_initial$  – Initial states**

$0$  (default) | vector

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

**Programmatic Use**

**Block Parameter:**  $x\_initial$

**Type:** character vector

**Values:** vector

**Default:** '0'

**Poles of  $A(v) - H(v) * C(v)$  – Desired poles**

$[-5 \ -2]$  (default)

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  $u$ .

**Programmatic Use****Block Parameter:** `vec_w`**Type:** character vector**Values:** vector**Default:** ' [-5 -2] '**References**

[1] Hyde, R. A. "H-infinity Aerospace Control Design - A VSTOL Flight Application." Springer Verlag: *Advances in Industrial Control Series*, 1995.

**Extended Capabilities****C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**See Also**

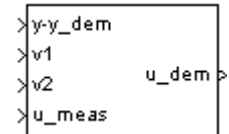
1D Controller Blend:  $u=(1-L).K1.y+L.K2.y$  | 2D Controller [A(v),B(v),C(v),D(v)] | 2D Observer Form [A(v),B(v),C(v),F(v),H(v)] | 2D Self-Conditioned [A(v),B(v),C(v),D(v)] | Linear Second-Order Actuator | Nonlinear Second-Order Actuator | Self-Conditioned [A,B,C,D]

**Introduced before R2006a**

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

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

**Library:** Aerospace Blockset / GNC / Control



### Description

The 2D Observer Form [A(v),B(v),C(v),F(v),H(v)] block implements a gain-scheduled state-space controller as defined in “Algorithms” on page 4-28.

The output from this block is the actuator demand, which you can input to an actuator block. Use this block to implement a controller designed using  $H$ -infinity loop-shaping, one of the design methods supported by Robust Control Toolbox.

### Limitations

If the scheduling parameter inputs to the block go out of range, they are clipped. The state-space matrices are not interpolated out of range.

### Ports

#### Input

##### **y-y\_dem — Set-point error**

vector

Set-point error, specified as a vector.

Data Types: double

##### **v1 — First scheduling variable**

vector

First scheduling variable, specified as a vector, that conforms to the dimensions of the state-space matrices.

Data Types: double

##### **v2 — Second scheduling variable**

vector

Second scheduling variable, specified as a vector, that conforms to the dimensions of the state-space matrices.

Data Types: double

##### **u\_meas — Measured actuator position**

vector

Measured actuator position, specified as a vector.

Data Types: `double`

### Output

#### **u\_dem — Actuator demands**

vector

Actuator demands, specified as a vector.

Data Types: `double`

## Parameters

#### **A-matrix(v1,v2) — A-matrix of the state-space implementation**

A (default) | array

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

#### **Programmatic Use**

**Block Parameter:** A

**Type:** character vector

**Values:** vector

**Default:** 'A'

#### **B-matrix(v1,v2) — B-matrix of the state-space implementation**

B (default) | array

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

#### **Programmatic Use**

**Block Parameter:** B

**Type:** character vector

**Values:** vector

**Default:** 'B'

#### **C-matrix(v1,v2) — C-matrix of the state-space implementation**

C (default) | array

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

#### **Programmatic Use**

**Block Parameter:** C

**Type:** character vector

**Values:** vector



**Default:** 'C'

**F-matrix(v1,v2) – F-matrix of the state-space implementation**

F (default) | array

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

**Programmatic Use**

**Block Parameter:** F

**Type:** character vector

**Values:** vector

**Default:** 'F'

**H-matrix(v1,v2) – H-matrix of the state-space implementation**

H (default) | array

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

**Programmatic Use**

**Block Parameter:** H

**Type:** character vector

**Values:** vector

**Default:** 'H'

**First scheduling variable (v1) breakpoints – Breakpoints for first scheduling variable**

v1\_vec (default)

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  $H.$

**Programmatic Use**

**Block Parameter:** AoA\_vec

**Type:** character vector

**Values:** vector

**Default:** 'v1\_vec'

**Second scheduling variable (v2) breakpoints – Breakpoints for second scheduling variable**

v2\_vec (default)

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  $H.$

**Programmatic Use**

**Block Parameter:** Mach\_vec

**Type:** character vector

**Values:** vector

**Default:** 'v2\_vec'

**Initial state, `x_initial` — Initial states**`0` (default)

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

**Programmatic Use****Block Parameter:** `x_initial`**Type:** character vector**Values:** vector**Default:** `'0'`**Algorithms**

The block implements a gain-scheduled state-space controller defined in the following observer form:

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

**References**

- [1] Hyde, R. A.. "H-infinity Aerospace Control Design — A VSTOL Flight Application." *Advances in Industrial Control Series*, Springer Verlag, 1995.

**Extended Capabilities****C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**See Also**

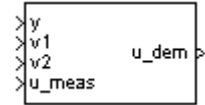
1D Controller [A(v),B(v),C(v),D(v)] | 2D Controller [A(v),B(v),C(v),D(v)] | 2D Controller Blend | 2D Self-Conditioned [A(v),B(v),C(v),D(v)] | 3D Observer Form [A(v),B(v),C(v),F(v),H(v)] | Linear Second-Order Actuator | Nonlinear Second-Order Actuator

**Introduced before R2006a**

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

Implement gain-scheduled state-space controller in self-conditioned form depending on two scheduling parameters

**Library:** Aerospace Blockset / GNC / Control



### Description

The 2D Self-Conditioned [A(v),B(v),C(v),D(v)] block implements a gain-scheduled state-space controller as defined in “Algorithms” on page 4-32.

The output from this block is the actuator demand, which you can input to an actuator block.

### Limitations

- If the scheduling parameter inputs to the block go out of range, they are clipped. The state-space matrices are not interpolated out of range.
- This block requires the Control System Toolbox license.

### Ports

#### Input

##### **y — Aircraft measurements**

vector

Aircraft measurements, specified as a vector.

Data Types: `double`

##### **v1 — First scheduling variable**

vector

First scheduling variable, specified as a vector, ordered according to the dimensions of the state-space matrices.

Data Types: `double`

##### **v2 — Second scheduling variable**

vector

Second scheduling variable, specified as a vector, ordered according to the dimensions of the state-space matrices.

Data Types: `double`

##### **u\_meas — Measured actuator position**

vector

Measured actuator position, specified as a vector.

Data Types: double

## Output

### **u\_dem — Actuator demands**

vector

Actuator demands, specified as a vector.

Data Types: double

## Parameters

### **A-matrix(v1,v2) — A-matrix of the state-space implementation**

A (default) | array

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

#### **Programmatic Use**

**Block Parameter:** A

**Type:** character vector

**Values:** vector

**Default:** 'A'

### **B-matrix(v1,v2) — B-matrix of the state-space implementation**

B (default) | array

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

#### **Programmatic Use**

**Block Parameter:** B

**Type:** character vector

**Values:** vector

**Default:** 'B'

### **C-matrix(v1,v2) — C-matrix of the state-space implementation**

C (default) | array

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

#### **Programmatic Use**

**Block Parameter:** C

**Type:** character vector

**Values:** vector

**Default:** 'C'

**D-matrix(v1,v2) – D-matrix of the state-space implementation**

D (default) | array

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

**Programmatic Use****Block Parameter:** D**Type:** character vector**Values:** vector**Default:** 'D'**First scheduling variable (v1) breakpoints – Breakpoints for first scheduling variable**

v1\_vec (default) | vector

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*.

**Programmatic Use****Block Parameter:** breakpoints\_v1**Type:** character vector**Values:** vector**Default:** 'v1\_vec'**Second scheduling variable (v2) breakpoints – Breakpoints for second scheduling variable**

v2\_vec (default) | vector

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*.

**Programmatic Use****Block Parameter:** breakpoints\_v2**Type:** character vector**Values:** vector**Default:** 'v2\_vec'**Initial state, x\_initial – Initial states**

0 (default) | vector

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

**Programmatic Use****Block Parameter:** x\_initial**Type:** character vector**Values:** vector**Default:** '0'**Poles of A(v)-H(v)\*C(v) – Desired poles**

[-5 -2] (default) | vector

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.

**Programmatic Use**

**Block Parameter:** `vec_w`

**Type:** character vector

**Values:** vector

**Default:** ' [-5 -2] '

## Algorithms

The 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.

## References

- [1] Kautsky, Nichols, and Van Dooren. "Robust Pole Assignment in Linear State Feedback," *International Journal of Control*, Vol. 41, Number 5, 1985, pp 1129-1155.

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

### 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)] | Linear Second-Order Actuator | Nonlinear Second-Order Actuator

**Introduced before R2006a**

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

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

**Library:** Aerospace Blockset / GNC / Control



### Description

The 3D Controller [A(v),B(v),C(v),D(v)] block implements a gain-scheduled state-space controller as described in “Algorithms” on page 4-36.

The output from this block is the actuator demand, which you can input to an actuator block.

### Limitations

If the scheduling parameter inputs to the block go out of range, they are clipped. The state-space matrices are not interpolated out of range.

### Ports

#### Input

##### **y — Aircraft measurements**

vector

Aircraft measurements, specified as a vector.

Data Types: double

##### **v1 — First scheduling variable**

vector

First scheduling variable, specified as a vector, that conforms to the dimensions of the state-space matrices.

Data Types: double

##### **v2 — Second scheduling variable**

vector

Second scheduling variable, specified as a vector, that conforms to the dimensions of the state-space matrices.

Data Types: double

##### **v3 — Third scheduling variable**

vector

Second scheduling variable, specified as a vector, that conforms to the dimensions of the state-space matrices.

Data Types: double

## Output

### **u — Actuator demands**

vector

Actuator demands, specified as a vector.

Data Types: double

## Parameters

### **A-matrix(v1,v2,v3) — A matrix of the state-space implementation**

A (default) | array

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

#### **Programmatic Use**

**Block Parameter:** A

**Type:** character vector

**Values:** vector

**Default:** 'A'

### **B-matrix(v1,v2,v3) — B matrix of the state-space implementation**

B (default) | array

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

#### **Programmatic Use**

**Block Parameter:** B

**Type:** character vector

**Values:** vector

**Default:** 'B'

### **C-matrix(v1,v2,v3) — C matrix of the state-space implementation**

C (default) | array

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

#### **Programmatic Use**

**Block Parameter:** C

**Type:** character vector

**Values:** vector

**Default:** 'C'



**D-matrix(v1,v2,v3) – D matrix of the state-space implementation**

D (default) | array

*D*-matrix of the state-space implementation. In the case of 3-D scheduling, the *D*-matrix should have five dimensions, the last three corresponding to scheduling variables *v1*, *v2*, and *v3*. 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  $D(:, :, 1, 1, 1) = [1 \ 0; 0 \ 1];$ .

**Programmatic Use****Block Parameter:** D**Type:** character vector**Values:** vector**Default:** 'D'**First scheduling variable (v1) breakpoints – Breakpoints for first scheduling variable**

v1\_vec (default) | vector

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*.

**Programmatic Use****Block Parameter:** AoA\_vec**Type:** character vector**Values:** vector**Default:** 'v1\_vec'**Second scheduling variable (v2) breakpoints – Breakpoints for second scheduling variable**

v2\_vec (default) | vector

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*.

**Programmatic Use****Block Parameter:** AoS\_vec**Type:** character vector**Values:** vector**Default:** 'v2\_vec'**Third scheduling variable (v3) breakpoints – Breakpoints for third scheduling variable**

v3\_vec (default) | vector

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*.

**Programmatic Use****Block Parameter:** Mach\_vec**Type:** character vector**Values:** vector**Default:** 'v3\_vec'**Initial state, x\_initial – Initial states**

0 (default) | vector

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  $A$ .

**Programmatic Use****Block Parameter:** `x_initial`**Type:** character vector**Values:** vector**Default:** '0'

## Algorithms

The block implements a gain-scheduled state-space controller as defined by this equation:

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

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

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.

## Extended Capabilities

**C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

## See Also

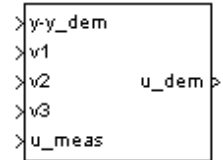
1D Controller [A(v),B(v),C(v),D(v)] | 2D Controller [A(v),B(v),C(v),D(v)] | 3D Observer Form [A(v),B(v),C(v),F(v),H(v)] | 3D Self-Conditioned [A(v),B(v),C(v),D(v)] | Linear Second-Order Actuator | Nonlinear Second-Order Actuator

**Introduced before R2006a**

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

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

**Library:** Aerospace Blockset / GNC / Control



### Description

The 3D Observer Form [A(v),B(v),C(v),F(v),H(v)] block implements a gain-scheduled state-space controller defined in “Algorithms” on page 4-28.

The main application of this block is to implement a controller designed using  $H$ -infinity loop-shaping. Use this block to implement a controller designed using  $H$ -infinity loop-shaping, one of the design methods supported by Robust Control Toolbox.

### Limitations

If the scheduling parameter inputs to the block go out of range, they are clipped. The state-space matrices are not interpolated out of range.

### Ports

#### Input

##### **y-y\_dem — Set-point error**

vector

Set-point error, specified as a vector.

Data Types: double

##### **v1 — First scheduling variable**

vector

First scheduling variable, specified as a vector, that conforms to the dimensions of the state-space matrices.

Data Types: double

##### **v2 — Second scheduling variable**

vector

Second scheduling variable, specified as a vector, that conforms to the dimensions of the state-space matrices.

Data Types: double

**v3 — Third scheduling variable**

vector

Third scheduling variable, specified as a vector, that conforms to the dimensions of the state-space matrices.

Data Types: double

**u\_meas — Measured actuator position**

vector

Measured actuator position, specified as a vector.

Data Types: double

**Output****u\_dem — Actuator demands**

vector

Actuator demands, specified as a vector.

Data Types: double

**Parameters****A-matrix(v1, v2, v3) — A-matrix of the state-space implementation**

A (default) | array

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

**Programmatic Use****Block Parameter:** A**Type:** character vector**Values:** vector**Default:** 'A'**B-matrix(v1, v2, v3) — B-matrix of the state-space implementation**

B (default) | array

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

**Programmatic Use****Block Parameter:** B**Type:** character vector**Values:** vector**Default:** 'B'**C-matrix(v1, v2, v3) — C-matrix of the state-space implementation**

C (default) | array

*C*-matrix of the state-space implementation. In the case of 3-D scheduling, the *C*-matrix should have five dimensions, the last three corresponding to scheduling variables *v1*, *v2*, and *v3*. 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  $C(:, :, 1, 1, 1) = [1 \ 0; 0 \ 1];$ .

**Programmatic Use****Block Parameter:** C**Type:** character vector**Values:** vector**Default:** 'C'**F-matrix(v1, v2, v3) – F-matrix of the state-space implementation**

F (default) | array

State-feedback matrix. In the case of 3-D scheduling, the *F*-matrix should have five dimensions, the last three corresponding to scheduling variables *v1*, *v2*, and *v3*. 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  $F(:, :, 1, 1, 1) = [1 \ 0; 0 \ 1];$ .

**Programmatic Use****Block Parameter:** F**Type:** character vector**Values:** vector**Default:** 'F'**H-matrix(v1, v2, v3) – H-matrix of the state-space implementation**

H (default) | array

Observer (output injection) matrix. In the case of 3-D scheduling, the *H*-matrix should have five dimensions, the last three corresponding to scheduling variables *v1*, *v2*, and *v3*. 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  $H(:, :, 1, 1, 1) = [1 \ 0; 0 \ 1];$ .

**Programmatic Use****Block Parameter:** H**Type:** character vector**Values:** vector**Default:** 'H'**First scheduling variable (v1) breakpoints – Breakpoints for first scheduling variable**

v1\_vec (default)

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*.

**Programmatic Use****Block Parameter:** AoA\_vec**Type:** character vector**Values:** vector**Default:** 'v1\_vec'**Second scheduling variable (v2) breakpoints – Breakpoints for second scheduling variable**

v2\_vec (default)

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  $A$ ,  $B$ ,  $C$ ,  $F$ , and  $H$ .

**Programmatic Use**

**Block Parameter:** `AoS_vec`

**Type:** character vector

**Values:** vector

**Default:** `'v2_vec'`

**Third scheduling variable ( $v_3$ ) breakpoints — Breakpoints for third scheduling variable**

`v3_vec` (default)

Vector of the breakpoints for the third scheduling variable. The length of  $v_3$  should be same as the size of the fifth dimension of  $A$ ,  $B$ ,  $C$ ,  $F$ , and  $H$ .

**Programmatic Use**

**Block Parameter:** `Mach_vec`

**Type:** character vector

**Values:** vector

**Default:** `'v3_vec'`

**Initial state,  $x_{\text{initial}}$  — Initial states**

`0` (default)

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

**Programmatic Use**

**Block Parameter:** `x_initial`

**Type:** character vector

**Values:** vector

**Default:** `'0'`

## Algorithms

The block implements gain-scheduled state-space controller as defined by these equations:

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

## References

- [1] Hyde, R. A. "H-infinity Aerospace Control Design — A VSTOL Flight Application." *Advances in Industrial Control Series*, Springer Verlag, 1995.

## Extended Capabilities

**C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**See Also**

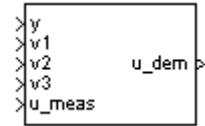
1D Controller  $[A(v),B(v),C(v),D(v)]$  | 2D Observer Form  $[A(v),B(v),C(v),F(v),H(v)]$  | 3D Controller  $[A(v),B(v),C(v),D(v)]$  | 3D Self-Conditioned  $[A(v),B(v),C(v),D(v)]$  | Linear Second-Order Actuator | Nonlinear Second-Order Actuator

**Introduced before R2006a**

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

Implement gain-scheduled state-space controller in self-conditioned form depending on two scheduling parameters

**Library:** Aerospace Blockset / GNC / Control



### Description

The 3D Self-Conditioned  $[A(v),B(v),C(v),D(v)]$  block implements a gain-scheduled state-space controller as defined in “Algorithms” on page 4-45.

If the scheduling parameter inputs to the block go out of range, then they are clipped. The state-space matrices are not interpolated out of range.

The output from this block is the actuator demand, which you can input to an actuator block.

### Limitations

This block requires the Control System Toolbox license.

### Ports

#### Input

##### **y — Aircraft measurements**

vector

Aircraft measurements, specified as a vector.

Data Types: double

##### **v1 — First scheduling variable**

vector

First scheduling variable, specified as a vector, ordered according to the dimensions of the state-space matrices.

Data Types: double

##### **v2 — Second scheduling variable**

vector

Second scheduling variable, specified as a vector, ordered according to the dimensions of the state-space matrices.

Data Types: double

##### **v3 — Third scheduling variable**

vector



Third scheduling variable, specified as a vector, ordered according to the dimensions of the state-space matrices.

Data Types: double

### **u\_meas — Measured actuator position**

vector

Measured actuator position, specified as a vector.

Data Types: double

## **Output**

### **Port\_1 — Actuator demands**

vector

Actuator demands, specified as a vector.

Data Types: double

## **Parameters**

### **A-matrix(v1,v2,v3) — A-matrix of the state-space implementation**

A (default) | array

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

#### **Programmatic Use**

**Block Parameter:** A

**Type:** character vector

**Values:** vector

**Default:** 'A'

### **B-matrix(v1,v2,v3) — B-matrix of the state-space implementation**

B (default) | array

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

#### **Programmatic Use**

**Block Parameter:** B

**Type:** character vector

**Values:** vector

**Default:** 'B'

### **C-matrix(v1,v2,v3) — C-matrix of the state-space implementation**

C (default) | array

C-matrix of the state-space implementation. In the case of 3-D scheduling, the C-matrix should have five dimensions, the last three corresponding to scheduling variables v1, v2, and v3. For example, if

the  $C$ -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  $C(:, :, 1, 1, 1) = [1 \ 0; 0 \ 1];$ .

**Programmatic Use**

**Block Parameter:** C

**Type:** character vector

**Values:** vector

**Default:** 'C'

**D-matrix(v1, v2, v3) — D-matrix of the state-space implementation**

D (default) | array

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

**Programmatic Use**

**Block Parameter:** D

**Type:** character vector

**Values:** vector

**Default:** 'D'

**First scheduling variable (v1) breakpoints — Breakpoints for first scheduling variable**

v1\_vec (default) | vector

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  $D$ .

**Programmatic Use**

**Block Parameter:** breakpoints\_v1

**Type:** character vector

**Values:** vector

**Default:** 'v1\_vec'

**Second scheduling variable (v2) breakpoints — Breakpoints for second scheduling variable**

v2\_vec (default) | vector

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  $D$ .

**Programmatic Use**

**Block Parameter:** breakpoints\_v2

**Type:** character vector

**Values:** vector

**Default:** 'v2\_vec'

**Third scheduling variable (v3) breakpoints — Breakpoints for third scheduling variable**

v3\_vec (default) | vector

Vector of the breakpoints for the third scheduling variable. The length of  $v_3$  should be same as the size of the fifth dimension of  $A$ ,  $B$ ,  $C$ , and  $D$ .

**Programmatic Use****Block Parameter:** breakpoints\_v3**Type:** character vector**Values:** vector**Default:** 'v3\_vec'**Initial state, x\_initial – Initial states**

0 (default) | vector

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

**Programmatic Use****Block Parameter:** x\_initial**Type:** character vector**Values:** vector**Default:** '0'**Poles of  $A(v) - H(v) * C(v)$  – Desired poles**

[-5 -2] (default) | vector

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.

**Programmatic Use****Block Parameter:** vec\_w**Type:** character vector**Values:** vector**Default:** '[-5 -2]'

## Algorithms

The 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.

## References

- [1] Kautsky, Nichols, and Van Dooren. "Robust Pole Assignment in Linear State Feedback." *International Journal of Control*, Vol. 41, Number 5, 1985, pp. 1129-1155.

## **Extended Capabilities**

### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

### **See Also**

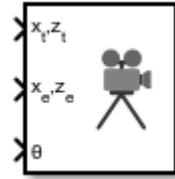
1D Self-Conditioned [A(v),B(v),C(v),D(v)] | 2D Self-Conditioned [A(v),B(v),C(v),D(v)] | 3D Controller [A(v),B(v),C(v),D(v)] | 3D Observer Form [A(v),B(v),C(v),F(v),H(v)] | Linear Second-Order Actuator | Nonlinear Second-Order Actuator

**Introduced before R2006a**

## 3DoF Animation

Create 3-D MATLAB Graphics animation of three-degrees-of-freedom object

**Library:** Aerospace Blockset / Animation / MATLAB-Based 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 MATLAB Graphics.

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

This block does not produce deployable code, but you can use it with Simulink Coder external mode as a SimViewingDevice.

### Ports

#### Input

##### $x_t z_t$ — Target altitude and downrange position

two-element vector

Altitude and downrange position of the target, specified as a two-element vector, in Earth coordinates.

Data Types: double

##### $x_e z_e$ — Craft altitude and downrange position

two-element vector

Altitude and downrange position of the craft, specified as two-element vector, in Earth coordinates.

Data Types: double

##### $\theta$ — Attitude of craft

1-by-1 scalar

Attitude of the craft, specified as 1-by-1 scalar, in radians.

Data Types: double

### Parameters

##### Axes limits [xmin xmax ymin ymax zmin zmax] — Axes limits

[0 5000 -2000 2000 -5050 -3050] (default) | six-element vector

Three-dimensional space to be viewed, specified as a six-element vector.

**Programmatic Use****Block Parameter:** u1**Type:** character vector**Values:** six-element vector**Default:** '[0 5000 -2000 2000 -5050 -3050]'**Time interval between updates – Time interval**

0.05 (default) | scalar

Time interval at which the animation is redrawn, specified as a double scalar.

**Programmatic Use****Block Parameter:** u2**Type:** character vector**Values:** double scalar**Default:** '0.05'**Size of craft displayed – Scale factor**

1.0 (default) | scalar

Scale factor to adjust the size of the craft and target, specified as a double scalar.

**Programmatic Use****Block Parameter:** u3**Type:** character vector**Values:** double scalar**Default:** '1.0'**Enter view – Entrance view**

Fixed position (default) | Cockpit | Fly alongside

Preset entrance views, specified as:

- Fixed position
- Cockpit
- Fly alongside

These preset views are specified by MATLAB Graphics parameters **CameraTarget** and **CameraUpVector** for the figure axes.

---

**Tip** To customize the position and field of view for the selected view, use the **Position of camera** and **View angle** parameters.

---

**Programmatic Use****Block Parameter:** u5**Type:** character vector**Values:** Fixed position | Cockpit | Fly alongside**Default:** 'Fixed position'**Position of camera [xc yc zc] – Camera position**

[2000 500 -3150] (default) | three-element vector

Camera position, specified using the MATLAB Graphics parameter `CameraPosition` for the figure axes as a three-element vector. Used in all cases except for the Cockpit view.

**Programmatic Use****Block Parameter:** `u6`**Type:** character vector**Values:** three-element vector**Default:** `'[2000 500 -3150]'`**View angle – View angle**`10` (default) | scalar

View angle, specified as MATLAB Graphics parameter `CameraViewAngle` for the figure axes in degrees as a double scalar.

**Programmatic Use****Block Parameter:** `u7`**Type:** character vector**Values:** double scalar**Default:** `'10'`**Enable animation – Display animation**`on` (default) | `off`

To display the animation during the simulation, select this check box. If not selected, the animation is not displayed.

**Programmatic Use****Block Parameter:** `u8`**Type:** character vector**Values:** `on` | `off`**Default:** `'on'`

## Extended Capabilities

**C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

## See Also

[6DoF Animation](#) | [CameraPosition](#) | [CameraViewAngle](#) | [FlightGear Preconfigured 6DoF Animation](#)

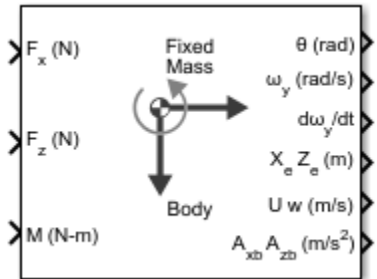
**Topics**

“Designing a Guidance System in MATLAB and Simulink” (Simulink)

**Introduced before R2006a**

## 3DOF (Body Axes)

Implement three-degrees-of-freedom equations of motion with respect to body axes

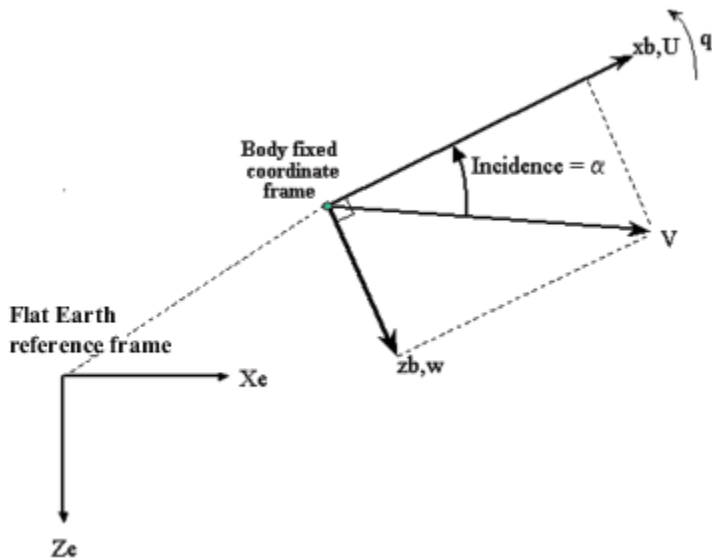


## Library

Equations of Motion/3DOF

## Description

The 3DOF (Body Axes) block considers the rotation in the vertical plane of a body-fixed coordinate frame about a flat Earth reference frame.



The equations of motion are



$$A_{xb} = \dot{u} = \frac{F_x}{m} - qw - g\sin\theta, A_{xe} = \frac{F_x}{m} - \varepsilon\sin\theta$$

$$A_{zb} = \dot{w} = \frac{F_z}{m} + qu + g\cos\theta, A_{ze} = \frac{F_z}{m} + \varepsilon\cos\theta$$

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

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

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

## Parameters

### Main

### Units

Specifies the input and output units:

| Units                      | Forces | Moment       | Acceleration              | Velocity          | Position | Mass     | Inertia                |
|----------------------------|--------|--------------|---------------------------|-------------------|----------|----------|------------------------|
| Metric (MKS)               | Newton | Newton-meter | Meters per second squared | Meters per second | Meters   | Kilogram | Kilogram meter squared |
| English (Velocity in ft/s) | Pound  | Foot- pound  | Feet per second squared   | Feet per second   | Feet     | Slug     | Slug foot squared      |
| English (Velocity in kts)  | Pound  | Foot- pound  | Feet per second squared   | Knots             | Feet     | Slug     | Slug foot squared      |

### Mass Type

Select the type of mass to use:

|                 |  |
|-----------------|--|
| Fixed           | Mass is constant throughout the simulation.                |
| Simple Variable | Mass and inertia vary linearly as a function of mass rate. |
| Custom Variable | Mass and inertia variations are customizable.              |

The Fixed selection conforms to the previously described equations of motion.

### Initial velocity

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

### Initial body attitude

A scalar value for the initial pitch attitude of the body, ( $\theta_0$ ).

### Initial incidence

A scalar value for the initial angle between the velocity vector and the body, ( $\alpha_0$ ).

### Initial body rotation rate

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

### Initial position (x,z)

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

**Initial Mass**

A scalar value for the mass of the body.

**Inertia**

A scalar value for the inertia of the body.

**Gravity Source**

Specify source of gravity:

|          |                                    |
|----------|------------------------------------|
| External | Variable gravity input to block    |
| Internal | Constant gravity specified in mask |

**Acceleration due to gravity**

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

**Include inertial acceleration**

Select this check box to enable an additional output port for the accelerations in body-fixed axes with respect to the inertial frame. You typically connect this signal to the accelerometer.

**State Attributes**

Assign unique name to each state. You can use state names instead of block paths during linearization.

- To assign a name to a single state, enter a unique name between quotes, for example, 'velocity'.
- To assign names to multiple states, enter a comma-delimited list surrounded by braces, for example, {'a', 'b', 'c'}. Each name must be unique.
- If a parameter is empty (' '), no name assignment occurs.
- The state names apply only to the selected block with the name parameter.
- The number of states must divide evenly among the number of state names.
- You can specify fewer names than states, but you cannot specify more names than states.

For example, you can specify two names in a system with four states. The first name applies to the first two states and the second name to the last two states.

- To assign state names with a variable in the MATLAB workspace, enter the variable without quotes. A variable can be a character vector, cell array, or structure.

**Velocity: e.g., {'u', 'w'}**

Specify velocity state names.

Default value is ''.

**Pitch attitude: e.g., 'theta'**

Specify pitch attitude state name.

Default value is ''.

**Position: e.g., {'Xe', 'Ze'}**

Specify position state names.

Default value is ''.

**Pitch angular rate: e.g., 'q'**

Specify pitch angular rate state name.

Default value is ''.

**Inputs and Outputs**

| Input             | Dimension Type     | Description  |
|-------------------|--------------------|--|
| First             |                    | Contains the force acting along the body x-axis, ( $F_x$ ).  |
| Second            |                    | Contains the force acting along the body z-axis, ( $F_z$ ).  |
| Third             |                    | Contains the applied pitch moment, ( $M$ ).  |
| Fourth (Optional) |                    | Contains the block is gravity in the selected units.   |
| Output            | Dimension Type     | Description  |
| First             |                    | Contains the pitch attitude, within $\pm\pi$ , in radians ( $\theta$ ).  |
| Second            |                    | Contains the pitch angular rate, in radians per second ( $q$ ).  |
| Third             |                    | Contains the pitch angular acceleration, in radians per second squared ( $\dot{q}$ ).  |
| Fourth            | Two-element vector | Contains the location of the body, in the flat Earth reference frame, ( $X_e, Z_e$ ).  |
| Fifth             | Two-element vector | Contains the velocity of the body resolved into the body-fixed coordinate frame, ( $u, w$ ).   |
| Sixth             | Two-element vector | Contains the acceleration of the body resolved into the body-fixed coordinate frame, ( $A_x, A_z$ ).   |
| Seventh           | Two-element vector | Contains the accelerations in body-fixed axes with respect to inertial frame (flat Earth). You typically connect this signal to the accelerometer. |

**Examples**

See `aero_guidance` for an example of this block.

**See Also**

3DOF (Wind Axes)

4th Order Point Mass (Longitudinal)

Custom Variable Mass 3DOF (Body Axes)

Custom Variable Mass 3DOF (Wind Axes)

Simple Variable Mass 3DOF (Body Axes)

Simple Variable Mass 3DOF (Wind Axes)

## **Extended Capabilities**

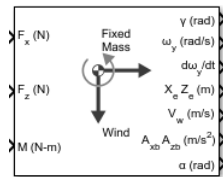
### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**Introduced in R2006a**

## 3DOF (Wind Axes)

Implement three-degrees-of-freedom equations of motion with respect to wind axes

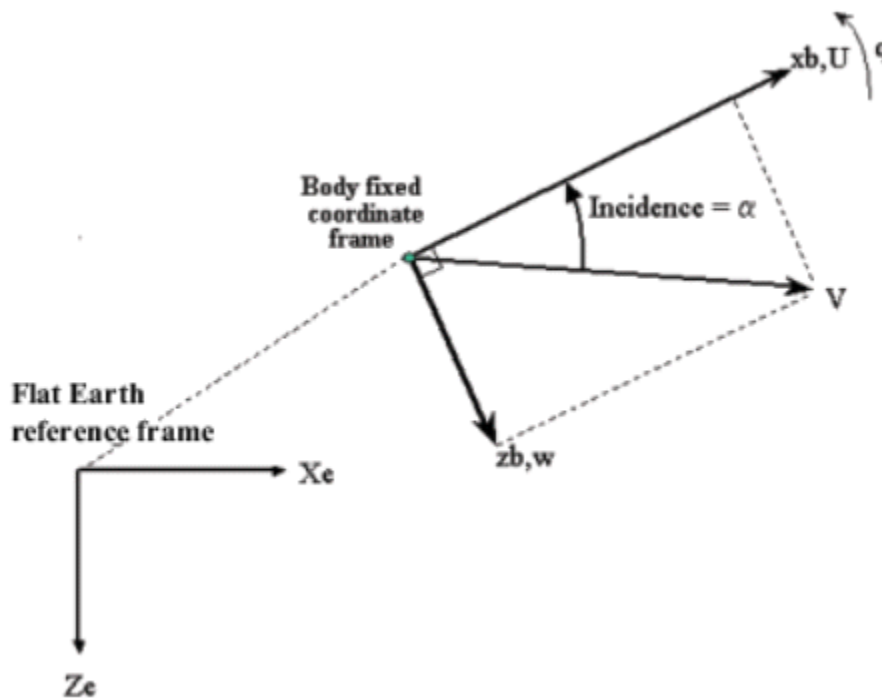


## Library

Equations of Motion/3DOF

## Description

The 3DOF (Wind Axes) block considers the rotation in the vertical plane of a wind-fixed coordinate frame about a flat Earth reference frame.



The equations of motion are

$$\dot{V} = \frac{F_{xwind}}{m} - g \sin \gamma$$

$$\dot{\alpha} = \frac{F_{zwind}}{mV \cos \beta} + q + \frac{g}{V \cos \beta} \cos \gamma$$

$$\dot{q} = \dot{\theta} = \frac{M_{ybody}}{I_{yy}}$$

$$\dot{\gamma} = q - \dot{\alpha}$$

$$A_{be} = \begin{bmatrix} A_{xe} \\ A_{ze} \end{bmatrix} = DCM_{wb} \left[ \frac{F_w}{m} - g \sin \gamma \right]$$

$$A_{bb} = \begin{bmatrix} \dot{u}_b \\ \dot{w}_b \end{bmatrix} = DCM_{wb} \left[ \frac{F_w}{m} - g \sin \gamma - \bar{\omega}_w \times \bar{V}_w \right]$$

$$\bar{F}_w = \begin{bmatrix} F_{xwind} \\ F_{zwind} \end{bmatrix}, \bar{V}_w = \begin{bmatrix} V_{xwind} \\ V_{zwind} \end{bmatrix}, \bar{\omega}_w = q$$

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

## Parameters

### Main

### Units

Specifies the input and output units:

| Units                      | Forces | Moment       | Acceleration              | Velocity          | Position | Mass     | Inertia                |
|----------------------------|--------|--------------|---------------------------|-------------------|----------|----------|------------------------|
| Metric (MKS)               | Newton | Newton meter | Meters per second squared | Meters per second | Meters   | Kilogram | Kilogram meter squared |
| English (Velocity in ft/s) | Pound  | Foot pound   | Feet per second squared   | Feet per second   | Feet     | Slug     | Slug foot squared      |
| English (Velocity in kts)  | Pound  | Foot pound   | Feet per second squared   | Knots             | Feet     | Slug     | Slug foot squared      |

### Mass Type

Select the type of mass to use:

|                 |  |
|-----------------|--|
| Fixed           | Mass is constant throughout the simulation.                |
| Simple Variable | Mass and inertia vary linearly as a function of mass rate. |
| Custom Variable | Mass and inertia variations are customizable.              |

The Fixed selection conforms to the previously described equations of motion.

### Initial airspeed

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

**Initial flight path angle**

A scalar value for the initial flight path angle of the body, ( $\gamma_0$ ).

**Initial incidence**

A scalar value for the initial angle between the velocity vector and the body, ( $\alpha_0$ ).

**Initial body rotation rate**

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

**Initial position (x,z)**

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

**Initial Mass**

A scalar value for the mass of the body.

**Inertia body axes**

A scalar value for the inertia of the body.

**Gravity Source**

Specify source of gravity:

|          |                                    |
|----------|------------------------------------|
| External | Variable gravity input to block    |
| Internal | Constant gravity specified in mask |

**Acceleration due to gravity**

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

**Include inertial acceleration**

Select this check box to enable an additional output port for the accelerations in body-fixed axes with respect to the inertial frame. You typically connect this signal to the accelerometer.

**State Attributes**

Assign unique name to each state. Use state names instead of block paths throughout the linearization process.

- To assign a name to a single state, enter a unique name between quotes, for example, 'velocity'.
- To assign names to multiple states, enter a comma-delimited list surrounded by braces, for example, {'a', 'b', 'c'}. Each name must be unique.
- If a parameter is empty (' '), no name assignment occurs.
- The state names apply only to the selected block with the name parameter.
- The number of states must divide evenly among the number of state names.
- You can specify fewer names than states, but you cannot specify more names than states.

For example, you can specify two names in a system with four states. The first name applies to the first two states and the second name to the last two states.

- To assign state names with a variable in the MATLAB workspace, enter the variable without quotes. A variable can be a character vector, cell array, or structure.

**Velocity: e.g., 'V'**

Specify velocity name for the state.

Default value is ' '.

**Incidence angle: e.g., 'alpha'**

Specify incidence angle name for the state.

Default value is ' '.

**Flight path angle: e.g., 'gamma'**

Specify flight path angle name for the state.

Default value is ' '.

**Body rotation rates: e.g., 'q'**

Specify body rotation rates name for the state.

Default value is ' '.

**Position: e.g., {'Xe', 'Ze'}**

Specify position name for the state.

Default value is ' '.

## Inputs and Outputs

| Input             | Dimension Type     | Description  |
|-------------------|--------------------|--|
| First             |                    | Contains the force acting along the wind x-axis, ( $F_x$ ).  |
| Second            |                    | Contains the force acting along the wind z-axis, ( $F_z$ ).  |
| Third             |                    | Contains the applied pitch moment in body axes, ( $M$ ).   |
| Fourth (Optional) |                    | Contains the block is gravity in the selected units.   |
| Output            | Dimension Type     | Description  |
| First             |                    | Contains the flight path angle, within $\pm\pi$ , in radians ( $\gamma$ ).                           |
| Second            |                    | Contains the pitch angular rate, in radians per second ( $\omega_y$ ).                               |
| Third             |                    | Contains the pitch angular acceleration, in radians per second squared ( $d\omega_y/dt$ ).           |
| Fourth            | Two-element vector | Contains the location of the body, in the flat Earth reference frame, ( $X_e, Z_e$ ).                |
| Fifth             | Two-element vector | Contains the velocity of the body resolved into the wind-fixed coordinate frame, ( $V, 0$ ).         |
| Sixth             | Two-element vector | Contains the acceleration of the body resolved into the body-fixed coordinate frame, ( $A_x, A_z$ ). |
| Seventh           | Scalar             | Contains the angle of attack, ( $\alpha$ ).  |



| Output | Dimension Type     | Description  |
|--------|--------------------|--|
| Eighth | Two-element vector | Contains the accelerations in body-fixed axes with respect to inertial frame (flat Earth). You typically connect this signal to the accelerometer. |

## 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.

## Reference

Stevens, B. L., and F. L. Lewis, *Aircraft Control and Simulation*, John Wiley & Sons, New York, 1992.

## See Also

3DOF (Body Axes)

4th Order Point Mass (Longitudinal)

Custom Variable Mass 3DOF (Body Axes)

Custom Variable Mass 3DOF (Wind Axes)

Simple Variable Mass 3DOF (Body Axes)

Simple Variable Mass 3DOF (Wind Axes)

## Extended Capabilities

### C/C++ Code Generation

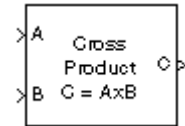
Generate C and C++ code using Simulink® Coder™.

### Introduced in R2006a

## 3x3 Cross Product

Calculate cross product of two 3-by-1 vectors

**Library:** Aerospace Blockset / Utilities / Math Operations



### Description

The 3x3 Cross Product block computes cross (or vector) product of two vectors,  $A$  and  $B$ . The block generates a third vector,  $C$ , in a direction normal to the plane containing  $A$  and  $B$ , 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  $C$  follows the right-hand rule in turning from  $A$  to  $B$ . For related equations, see “Algorithms” on page 4-60.

### Ports

#### Input

##### **A — First cross product input**

3-by-1 vector

First cross product input, specified as a vector.

Example: [10 2 3]

Data Types: double

##### **B — Second cross product input**

3-by-1 vector

Second cross product input, specified as a vector.

Example: [10 2 3]

Data Types: double

#### Output

##### **C — Cross product**

3-by-1 vector

Cross product, output as a vector.

Data Types: double

### Algorithms

The equations used to calculate  $A$ ,  $B$ , and  $C$  are:

$$\begin{aligned}A &= a_1\mathbf{i} + a_2\mathbf{j} + a_3\mathbf{k} \\B &= b_1\mathbf{i} + b_2\mathbf{j} + b_3\mathbf{k} \\C = A \times B &= \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix} \\ &= (a_2b_3 - a_3b_2)\mathbf{i} + (a_3b_1 - a_1b_3)\mathbf{j} + (a_1b_2 - a_2b_1)\mathbf{k}\end{aligned}$$

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

### See Also

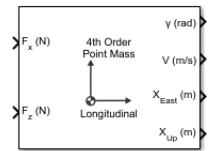
[Adjoint of 3x3 Matrix](#) | [Determinant of 3x3 Matrix](#) | [Invert 3x3 Matrix](#) | [Create 3x3 Matrix](#)

**Introduced before R2006a**

## 4th Order Point Mass (Longitudinal)

Calculate fourth-order point mass

**Library:** Aerospace Blockset / Equations of Motion / Point Mass



### Description

The 4th Order Point Mass (Longitudinal) block performs the calculations for the translational motion of a single point mass or multiple point masses. For more information on the system for the translational motion of a single point mass or multiple mass, see “Algorithms” on page 4-65.

The 4th Order Point Mass (Longitudinal) block port labels change based on the input and output units selected from the **Units** list.

### Limitations

The flat Earth reference frame is considered inertial, an approximation that allows the forces due to the Earth's motion relative to the “fixed stars” to be neglected.

### Ports

#### Input

##### Port\_1 — Force in x-axis

scalar | array

Force in x-axis, specified as a scalar or array, in selected units.

Data Types: double

##### Port\_2 — Force in z-axis

scalar | array

Force in z-axis, specified as a scalar or array, in selected units.

Data Types: double

#### Output

##### Port\_1 — Flight path angle

scalar | array

Flight path angle, returned as a scalar or array, in radians.

Data Types: double

##### Port\_2 — Airspeed

scalar | array

Airspeed, returned as a scalar or array, in selected units.

Data Types: double

### Port\_3 – Downrange or amount traveled east

scalar | array

Downrange or amount traveled east, returned as a scalar or array, in selected units.

Data Types: double

### Port\_4 – Altitude or amount traveled up

scalar | array

Altitude or amount traveled up, returned as a scalar or array, in selected units.

Data Types: double

## Parameters

### Units – Units

Metric (MKS) (default) | English (Velocity in ft/s) | English (Velocity in kts)

Input and output units, specified as:

| Units                      | Forces | Velocity          | Position |
|----------------------------|--------|-------------------|----------|
| Metric (MKS)               | Newton | Meters per second | Meters   |
| English (Velocity in ft/s) | Pound  | Feet per second   | Feet     |
| English (Velocity in kts)  | Pound  | Knots             | Feet     |

### Programmatic Use

**Block Parameter:** units

**Type:** character vector

**Values:** 'Metric (MKS)' | 'English (Velocity in ft/s)' | 'English (Velocity in kts)'

**Default:** 'Metric (MKS)'

### Initial flight path angle – Initial flight path angle

$\theta$  (default) | scalar | vector

Initial flight path angle of the point mass(es), specified as a scalar or vector.

### Programmatic Use

**Block Parameter:** gamma $\theta$

**Type:** character vector

**Values:** scalar | vector

**Default:** ' $\theta$ '

### Initial airspeed – Initial airspeed

100 (default) | scalar | vector

Initial airspeed of the point mass(es), specified as a scalar or vector.

**Programmatic Use****Block Parameter:**  $v_0$ **Type:** character vector**Values:** scalar | vector**Default:** '100'**Initial downrange [East] — Initial downrange** $0$  (default) | scalar | vector

Initial downrange of the point mass(es), specified as a scalar or vector.

**Programmatic Use****Block Parameter:**  $x_0$ **Type:** character vector**Values:** scalar | vector**Default:** '0'**Initial altitude [Up] — Initial altitude of point masses** $0$  (default) | scalar | vector

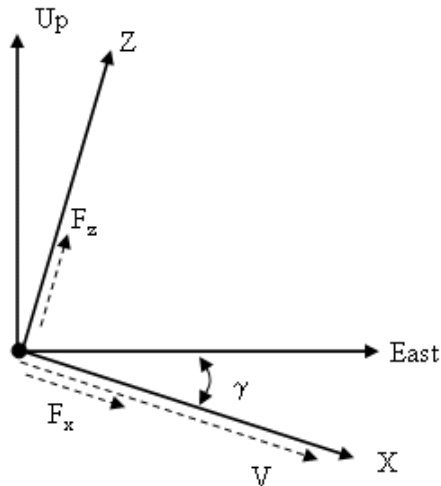
Initial altitude of the point mass(es), specified as a scalar or vector.

**Programmatic Use****Block Parameter:**  $h_0$ **Type:** character vector**Values:** scalar | vector**Default:** '0'**Initial mass — Point mass** $1.0$  (default) | scalar | vector

Mass of the point mass(es), specified as a scalar or vector.

**Programmatic Use****Block Parameter:**  $mass_0$ **Type:** character vector**Values:** scalar | vector**Default:** '1.0'

## Algorithms



The translational motions of the point mass  $[X_{East} X_{Up}]^T$  are functions of airspeed ( $V$ ) and flight path angle ( $\gamma$ ),

$$\begin{aligned} F_x &= m\dot{V} \\ F_z &= mV\dot{\gamma} \\ \dot{X}_{East} &= V\cos\gamma \\ \dot{X}_{Up} &= V\sin\gamma \end{aligned}$$

where the applied forces  $[F_x F_z]^T$  are in a system defined as follows: x-axis is in the direction of vehicle velocity relative to air, z-axis is upward, and y-axis completes the right-handed frame. The mass of the body  $m$  is assumed constant.

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

### See Also

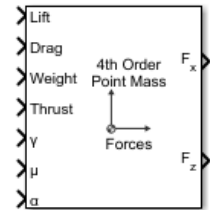
3DOF (Body Axes) | 3DOF (Wind Axes) | 4th Order Point Mass Forces (Longitudinal) | 6th Order Point Mass (Coordinated Flight) | 6th Order Point Mass Forces (Coordinated Flight) | Custom Variable Mass 3DOF (Body Axes) | Custom Variable Mass 3DOF (Wind Axes) | Simple Variable Mass 3DOF (Body Axes) | Simple Variable Mass 3DOF (Wind Axes)

Introduced before R2006a

## 4th Order Point Mass Forces (Longitudinal)

Calculate forces used by fourth-order point mass

**Library:** Aerospace Blockset / Equations of Motion / Point Mass



### Description

The 4th Order Point Mass Forces (Longitudinal) block calculates the applied forces for a single point mass or multiple point masses. For more information on the system for the applied forces, see “Algorithms” on page 4-68.

### Limitations

The flat Earth reference frame is considered inertial, an approximation that allows the forces due to the Earth motion relative to the "fixed stars" to be neglected.

### Ports

#### Input

##### Lift – Lift

scalar | array

Lift, specified as a scalar or array, in units of force.

Data Types: double

##### Drag – Drag

scalar | array

Drag, specified as a scalar or array, in units of force.

Data Types: double

##### Weight – Weight

scalar | array

Weight, specified as a scalar or array, in units of force.

Data Types: double

##### Thrust – Thrust

scalar | array

Thrust, specified as a scalar or array, in units of force.

Data Types: double



**$\chi$  — Flight path angle**

scalar | array

Flight path angle, specified as a scalar or array, in radians.

Data Types: double

 **$\mu$  — Bank angle**

scalar | array

Bank angle, specified as a scalar or array, in radians.

Data Types: double

 **$\alpha$  — Angle of attack**

scalar | array

Angle of attack, specified as a scalar or array, in radians.

Data Types: double

**Output** **$F_x$  — Force in x-axis**

scalar | array

Force in x-axis, returned as a scalar or array, in units of force.

Data Types: double

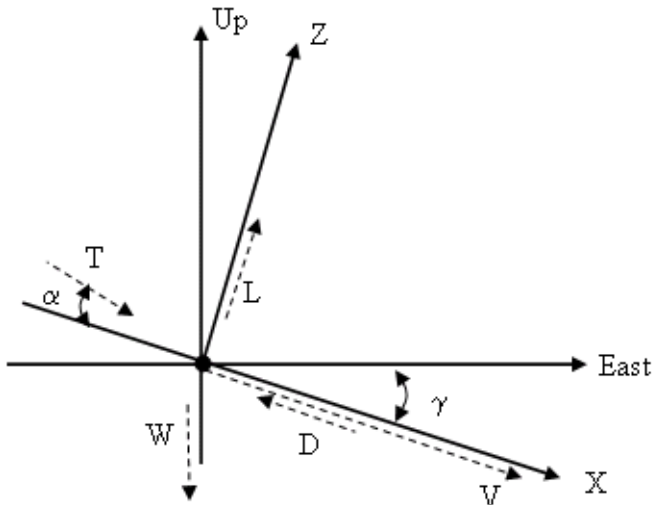
 **$F_z$  — Force in z-axis**

scalar | array

Force in z-axis, returned as a scalar or array, in units of force.

Data Types: double

## Algorithms



The applied forces  $[F_x \ F_z]^T$  are in a system defined as follows:  $x$ -axis is in the direction of vehicle velocity relative to air,  $z$ -axis is upward, and  $y$ -axis completes the right-handed frame. They are functions of lift ( $L$ ), drag ( $D$ ), thrust ( $T$ ), weight ( $W$ ), flight path angle ( $\gamma$ ), angle of attack ( $\alpha$ ), and bank angle ( $\mu$ ).

$$F_z = (L + T \sin \alpha) \cos \mu - W \cos \gamma$$

$$F_x = T \cos \alpha - D - W \sin \gamma$$

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

### See Also

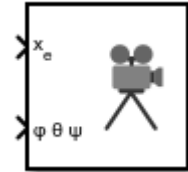
4th Order Point Mass (Longitudinal) | 6th Order Point Mass (Coordinated Flight) | 6th Order Point Mass Forces (Coordinated Flight)

Introduced before R2006a

# 6DoF Animation

Create 3-D MATLAB Graphics animation of six-degrees-of-freedom object

**Library:** Aerospace Blockset / Animation / MATLAB-Based Animation



## Description

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

The 6DoF Animation block uses the input values and the block parameters to create and display the animation. The **Axes limits**, **Static object position**, and **Position of camera** parameters have the same units of length as the input parameters.

This block does not produce deployable code, but you can use it with Simulink Coder external mode as a SimViewingDevice.

## Ports

### Input

#### $x_e$ — Altitude, crossrange position, and downrange position

three-element vector

Altitude, the crossrange position, and the downrange position of the vehicle in Earth coordinates, specified as a three-element vector.

Data Types: double

#### $\varphi \theta \psi$ — Euler angles

three-element vector

Euler angles of the vehicle, specified as a three-element vector.

Data Types: double

## Parameters

#### Axes limits [xmin xmax ymin ymax zmin zmax] — Axes limits

[0 4000 -2000 2000 -5000 -3000] (default) | six-element vector

Three-dimensional space to be viewed, specified as a six-element vector.

#### Programmatic Use

**Block Parameter:** u1

**Type:** character vector

**Values:** six-element vector

**Default:** '[0 4000 -2000 2000 -5000 -3000]'

#### **Time interval between updates — Time interval**

0.1 (default) | scalar

Time interval at which the animation is redrawn, specified as a double scalar.

##### **Programmatic Use**

**Block Parameter:** u2

**Type:** character vector

**Values:** double scalar

**Default:** '0.1'

#### **Size of craft displayed — Scale factor**

1.0 (default) | scalar

Scale factor to adjust the size of the vehicle and target, specified as a double scalar.

##### **Programmatic Use**

**Block Parameter:** u3

**Type:** character vector

**Values:** double scalar

**Default:** '1.0'

#### **Static object position [xp yp zp] — Static object position**

[4000 0 -5000] (default) | three-element vector

Altitude, crossrange position, and downrange position of the target, specified as three-element vector.

##### **Programmatic Use**

**Block Parameter:** u4

**Type:** character vector

**Values:** three-element vector

**Default:** '[4000 0 -5000]'

#### **Enter view — Entrance view**

Fixed position (default) | Cockpit | Fly alongside

Preset entrance views, specified as:

- Fixed position
- Cockpit
- Fly alongside

These preset views are specified by MATLAB Graphics parameters **CameraTarget** and **CameraUpVector** for the figure axes.

---

**Tip** To customize the position and field of view for the selected view, use the **Position of camera** and **View angle** parameters.

---

##### **Programmatic Use**

**Block Parameter:** u5

**Type:** character vector  
**Values:** Fixed position | Cockpit | Fly alongside  
**Default:** 'Fixed position'

### Position of camera [xc yc zc] – Camera position

[2000 500 -3150] (default) | three-element vector

Camera position, specified using the MATLAB Graphics parameter `CameraPosition` for the figure axes as a three-element vector. Used in all cases except for when **Enter view** is set to `Cockpit`.

#### Programmatic Use

**Block Parameter:** u6  
**Type:** character vector  
**Values:** three-element vector  
**Default:** '[2000 500 -3150]'

### View angle – View angle

10 (default) | scalar

View angle for the MATLAB Graphics parameter `CameraViewAngle` for the figure axes in degrees, specified as a double scalar.

#### Programmatic Use

**Block Parameter:** u7  
**Type:** character vector  
**Values:** double scalar  
**Default:** '10'

### Enable animation – Display animation

on (default) | off

Whether to display the animation during the simulation. If not selected, the animation is not displayed.

#### Programmatic Use

**Block Parameter:** u8  
**Type:** character vector  
**Values:** on | off  
**Default:** 'on'

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

## See Also

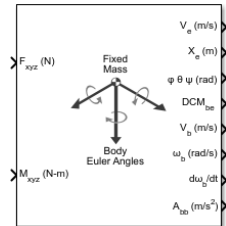
3DoF Animation | `CameraPosition` | `CameraViewAngle` | FlightGear Preconfigured 6DoF Animation

**Introduced before R2006a**

## 6DOF (Euler Angles)

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

**Library:** Aerospace Blockset / Equations of Motion / 6DOF



### Description

The 6DOF (Euler Angles) block implements the Euler angle representation of six-degrees-of-freedom equations of motion, taking into consideration the rotation of a body-fixed coordinate frame ( $X_b$ ,  $Y_b$ ,  $Z_b$ ) about a flat Earth reference frame ( $X_e$ ,  $Y_e$ ,  $Z_e$ ). For more information about these reference points, see “Algorithms” on page 4-78.

### 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.

### Ports

#### Input

##### $F_{xyz}$ (N) — Applied forces

three-element vector

Applied forces, specified as three-element vector.

Data Types: `double`

##### $M_{xyz}$ (N-m) — Applied moments

three-element vector

Applied moments, specified as three-element vector.

Data Types: `double`

#### Output

##### $V_e$ (m/s) — Velocity in flat Earth reference frame

three-element vector

Velocity in the flat Earth reference frame, returned as a three-element vector.

Data Types: `double`

##### $X_e$ (m) — Position in flat Earth reference frame

three-element vector

Position in the flat Earth reference frame, returned as a three-element vector.

Data Types: double

### $\varphi \ \theta \ \psi$ (rad) — Euler rotation angles

three-element vector

Euler rotation angles [roll, pitch, yaw], returned as three-element vector, in radians.

Data Types: double

### DCM<sub>be</sub> — Coordinate transformation

3-by-3 matrix

Coordinate transformation from flat Earth axes to body-fixed axes, returned as a 3-by-3 matrix

Data Types: double

### $V_b$ (m/s) — Velocity in the body-fixed frame

three-element vector

Velocity in the body-fixed frame, returned as a three-element vector.

Data Types: double

### $\omega_b$ (rad/s) — Angular rates in body-fixed axes

three-element vector

Angular rates in body-fixed axes, returned as a three-element vector, in radians per second.

Data Types: double

### $d\omega_b/dt$ — Angular accelerations

three-element vector

Angular accelerations in body-fixed axes, returned as a three-element vector, in radians per second squared.

Data Types: double

### $A_{bb}$ m/s<sup>2</sup> — Accelerations in body-fixed axes

three-element vector

Accelerations in body-fixed axes with respect to body frame, returned as a three-element vector.

Data Types: double

### $A_{be}$ m/s<sup>2</sup> — Accelerations with respect to inertial frame

three-element vector

Accelerations in body-fixed axes with respect to inertial frame (flat Earth), returned as a three-element vector. You typically connect this signal to the accelerometer.

### Dependencies

This port appears only when the **Include inertial acceleration** check box is selected.

Data Types: double

## Parameters

### Main

#### Units – Input and output units

Metric (MKS) (default) | English (Velocity in ft/s) | English (Velocity in kts)

Input and output units, specified as Metric (MKS), English (Velocity in ft/s), or English (Velocity in kts).

| Units                      | Forces | Moment       | Acceleration              | Velocity          | Position | Mass     | Inertia                |
|----------------------------|--------|--------------|---------------------------|-------------------|----------|----------|------------------------|
| Metric (MKS)               | Newton | Newton meter | Meters per second squared | Meters per second | Meters   | Kilogram | Kilogram meter squared |
| English (Velocity in ft/s) | Pound  | Foot pound   | Feet per second squared   | Feet per second   | Feet     | Slug     | Slug foot squared      |
| English (Velocity in kts)  | Pound  | Foot pound   | Feet per second squared   | Knots             | Feet     | Slug     | Slug foot squared      |

#### Programmatic Use

**Block Parameter:** Units

**Type:** character vector

**Values:** Metric (MKS) | English (Velocity in ft/s) | English (Velocity in kts)

**Default:** Metric (MKS)

#### Mass Type – Mass type

Fixed (default) | Simple Variable | Custom Variable

Mass type, specified according to the following table.

| Mass Type       | Description  | Default for  |
|-----------------|--|--|
| Fixed           | Mass is constant throughout the simulation.                | <ul style="list-style-type: none"> <li>6DOF (Quaternion)</li> <li>6DOF (Euler Angles)</li> </ul>   |
| Simple Variable | Mass and inertia vary linearly as a function of mass rate. | <ul style="list-style-type: none"> <li>Simple Variable Mass 6DOF (Euler Angles)</li> <li>Simple Variable Mass 6DOF (Quaternion)</li> </ul> |
| Custom Variable | Mass and inertia variations are customizable.              | <ul style="list-style-type: none"> <li>Custom Variable Mass 6DOF (Quaternion)</li> </ul>   |

The Simple Variable selection conforms to the previously described equations of motion.

#### Programmatic Use

**Block Parameter:** mtype

**Type:** character vector

**Values:** Fixed | Simple Variable | Custom Variable

**Default:** Simple Variable



**Representation — Equations of motion representation**

Quaternion | Euler Angles

Equations of motion representation, specified according to the following table.

| Representation | Description                                  | Default For   |
|----------------|--|---|
| Euler Angles   | Use Euler angles within equations of motion. | <ul style="list-style-type: none"> <li>6DOF (Euler Angles)</li> <li>Simple Variable Mass 6DOF (Euler Angles)</li> </ul>   |
| Quaternion     | Use quaternions within equations of motion.  | <ul style="list-style-type: none"> <li>6DOF (Quaternion)</li> <li>Custom Variable Mass 6DOF (Quaternion)</li> <li>Simple Variable Mass 6DOF (Quaternion)</li> </ul> |

The Quaternion selection conforms to the previously described equations of motion.

**Programmatic Use****Block Parameter:** rep**Type:** character vector**Values:** Euler Angles | Quaternion**Default:** Quaternion**Initial position in inertial axes [Xe,Ye,Ze] — Position in inertial axes**

[0 0 0] (default) | three-element vector

Initial location of the body in the flat Earth reference frame, specified as a three-element vector.

**Programmatic Use****Block Parameter:** xme\_0**Type:** character vector**Values:** [0 0 0] | three-element vector**Default:** [0 0 0]**Initial velocity in body axes [U,v,w] — Velocity in body axes**

[0 0 0] (default) | three-element vector

The three-element vector for the initial velocity in the body-fixed coordinate frame.

**Programmatic Use****Block Parameter:** Vm\_0**Type:** character vector**Values:** [0 0 0] | three-element vector**Default:** [0 0 0]**Initial Euler orientation [roll, pitch, yaw] — Initial Euler orientation**

[0 0 0] (default) | three-element vector

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

**Programmatic Use****Block Parameter:** eul\_0**Type:** character vector

**Values:** [0 0 0] | three-element vector

**Default:** [0 0 0]

### **Initial body rotation rates [p,q,r] — Initial body rotation**

[0 0 0] (default) | three-element vector

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

#### **Programmatic Use**

**Block Parameter:** pm\_0

**Type:** character vector

**Values:** [0 0 0] | three-element vector

**Default:** [0 0 0]

### **Initial mass — Initial mass**

1.0 (default) | scalar

The initial mass of the rigid body.

#### **Programmatic Use**

**Block Parameter:** mass\_0

**Type:** character vector

**Values:** 1.0 | scalar

**Default:** 1.0

### **Inertia — Inertia**

eye(3) (default) | scalar

A scalar value for the inertia of the body.

#### **Programmatic Use**

**Block Parameter:** inertia

**Type:** character vector

**Values:** eye(3) | scalar

**Default:** eye(3)

### **Include inertial acceleration — Include inertial acceleration port**

off (default) | on

Select this check box to add an inertial acceleration port.

#### **Programmatic Use**

**Block Parameter:** abi\_flag

**Type:** character vector

**Values:** off | on

**Default:** off

### **State Attributes**

Assign unique name to each state. You can use state names instead of block paths during linearization.

- To assign a name to a single state, enter a unique name between quotes, for example, 'velocity'.
- To assign names to multiple states, enter a comma-delimited list surrounded by braces, for example, {'a', 'b', 'c'}. Each name must be unique.

- If a parameter is empty ( ' ' ), no name assignment occurs.
- The state names apply only to the selected block with the name parameter.
- The number of states must divide evenly among the number of state names.
- You can specify fewer names than states, but you cannot specify more names than states.

For example, you can specify two names in a system with four states. The first name applies to the first two states and the second name to the last two states.

- To assign state names with a variable in the MATLAB workspace, enter the variable without quotes. A variable can be a character vector, cell array, or structure.

**Position: e.g., {'Xe', 'Ye', 'Ze'} — Position state name**

' ' (default) | comma-delimited list surrounded by braces

Specify position state names.

**Programmatic Use**

**Block Parameter:** xme\_statename

**Type:** character vector

**Values:** ' ' | comma-delimited list surrounded by braces

**Default:** ' '

**Velocity: e.g., {'U', 'v', 'w'} — Velocity state name**

' ' (default) | comma-delimited list surrounded by braces

Specify velocity state names.

**Programmatic Use**

**Block Parameter:** Vm\_statename

**Type:** character vector

**Values:** ' ' | comma-delimited list surrounded by braces

**Default:** ' '

**Euler rotation angles: e.g., {'phi', 'theta', 'psi'} — Euler rotation state name**

' ' (default) | comma-delimited list surrounded by braces

Specify Euler rotation angle state names.

**Programmatic Use**

**Block Parameter:** eul\_statename

**Type:** character vector

**Values:** ' ' | comma-delimited list surrounded by braces

**Default:** ' '

**Body rotation rates: e.g., {'p', 'q', 'r'} — Body rotation state name**

' ' (default) | comma-delimited list surrounded by braces

Specify body rotation rate state names.

**Programmatic Use**

**Block Parameter:** pm\_statename

**Type:** character vector

**Values:** ' ' | comma-delimited list surrounded by braces

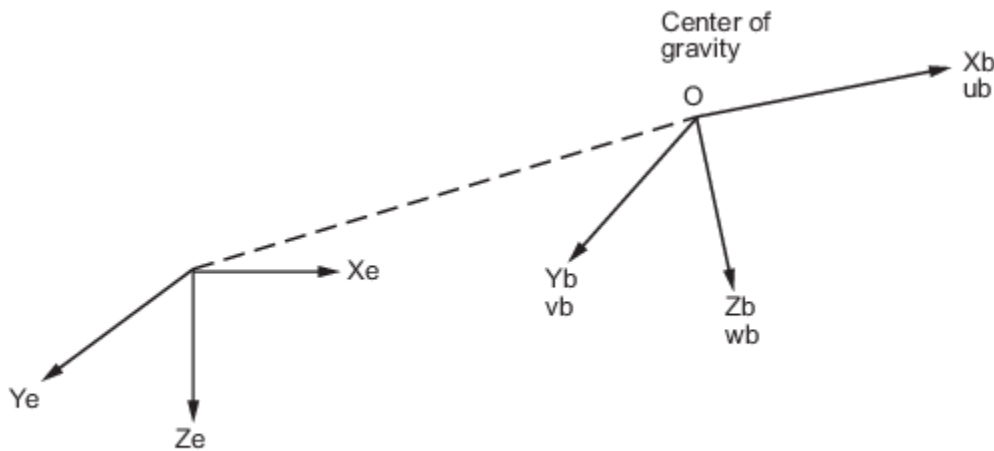
**Default:** ' '

## Algorithms

The 6DOF (Euler Angles) block uses these reference frame concepts.

- 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 flat Earth reference frame is considered inertial, an excellent approximation that allows the forces due to the Earth motion relative to the "fixed stars" to be neglected.



Flat Earth reference frame

- Translational motion of the body-fixed coordinate frame, 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.

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

$$A_{bb} = \begin{bmatrix} \dot{u}_b \\ \dot{v}_b \\ \dot{w}_b \end{bmatrix} = \frac{1}{m}\bar{F}_b - \bar{\omega} \times \bar{V}_b$$

$$A_{be} = \frac{1}{m}\bar{F}_b$$

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

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

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

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

- The relationship between the body-fixed angular velocity vector,  $[p \ q \ r]^T$ , and the rate of change of the Euler angles,  $[\dot{\phi} \ \dot{\theta} \ \dot{\psi}]^T$ , are 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.

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

## References

- [1] Stevens, Brian, and Frank Lewis, *Aircraft Control and Simulation*. Hoboken, NJ: Second Edition, John Wiley & Sons, 2003.
- [2] Zipfel, Peter H., *Modeling and Simulation of Aerospace Vehicle Dynamics*. Reston, Va: Second Edition, AIAA Education Series, 2007.

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

### See Also

6DOF (Quaternion) | 6DOF ECEF (Quaternion) | 6DOF Wind (Quaternion) | 6DOF Wind (Wind Angles) | Custom Variable Mass 6DOF (Euler Angles) | Custom Variable Mass 6DOF (Quaternion) | Custom Variable Mass 6DOF ECEF (Quaternion) | Custom Variable Mass 6DOF Wind (Quaternion) | Custom Variable Mass 6DOF Wind (Wind Angles) | Simple Variable Mass 6DOF (Euler Angles) | Simple Variable Mass 6DOF (Quaternion) | Simple Variable Mass 6DOF ECEF (Quaternion) | Simple Variable Mass 6DOF Wind (Quaternion) | Simple Variable Mass 6DOF Wind (Wind Angles)

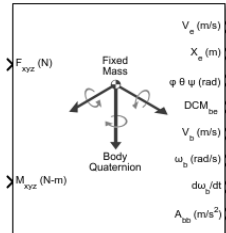
### Topics

“About Aerospace Coordinate Systems” on page 2-8

### Introduced in R2006a

## 6DOF (Quaternion)

Implement quaternion representation of six-degrees-of-freedom equations of motion with respect to body axes



## Library

Equations of Motion/6DOF

## Description

For a description of the coordinate system 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  $\varepsilon$  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.

$$\begin{bmatrix} \dot{q}_0 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 0 & -p & -q & -r \\ p & 0 & r & -q \\ q & -r & 0 & p \\ r & q & -p & 0 \end{bmatrix} \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} + K\varepsilon \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix}$$

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

Aerospace Blockset uses quaternions that are defined using the scalar-first convention.

## Parameters

### Main

### Units

Specifies the input and output units:

| Units                      | Forces | Moment       | Acceleration              | Velocity          | Position | Mass     | Inertia                |
|----------------------------|--------|--------------|---------------------------|-------------------|----------|----------|------------------------|
| Metric (MKS)               | Newton | Newton meter | Meters per second squared | Meters per second | Meters   | Kilogram | Kilogram meter squared |
| English (Velocity in ft/s) | Pound  | Foot pound   | Feet per second squared   | Feet per second   | Feet     | Slug     | Slug foot squared      |
| English (Velocity in kts)  | Pound  | Foot pound   | Feet per second squared   | Knots             | Feet     | Slug     | Slug foot squared      |

### Mass Type

Select the type of mass to use:

|                 |  |
|-----------------|--|
| Fixed           | Mass is constant throughout the simulation.                |
| Simple Variable | Mass and inertia vary linearly as a function of mass rate. |
| Custom Variable | Mass and inertia variations are customizable.              |

The Fixed selection conforms to the previously described equations of motion.

### Representation

Select the representation to use:

|              |  |
|--------------|--|
| Euler Angles | Use Euler angles within equations of motion. |
| Quaternion   | Use quaternions within equations of motion.  |

The Quaternion selection conforms to the previously described equations of motion.

### Initial position in inertial axes

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

### Initial velocity in body axes

The three-element vector for the initial velocity in the body-fixed coordinate frame.

### 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.

### Initial Mass

The mass of the rigid body.

### Inertia matrix

The 3-by-3 inertia tensor matrix  $I$ .

### Gain for quaternion normalization

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

### Include inertial acceleration

Select this check box to enable an additional output port for the accelerations in body-fixed axes with respect to the inertial frame. You typically connect this signal to the accelerometer.

## State Attributes

Assign unique name to each state. You can use state names instead of block paths during linearization.

- To assign a name to a single state, enter a unique name between quotes, for example, 'velocity'.
- To assign names to multiple states, enter a comma-delimited list surrounded by braces, for example, {'a', 'b', 'c'}. Each name must be unique.
- If a parameter is empty (' '), no name assignment occurs.
- The state names apply only to the selected block with the name parameter.
- The number of states must divide evenly among the number of state names.
- You can specify fewer names than states, but you cannot specify more names than states.

For example, you can specify two names in a system with four states. The first name applies to the first two states and the second name to the last two states.

- To assign state names with a variable in the MATLAB workspace, enter the variable without quotes. A variable can be a character vector, cell array, or structure.

### Position: e.g., {'Xe', 'Ye', 'Ze'}

Specify position state names.

Default value is ''.

### Velocity: e.g., {'U', 'v', 'w'}

Specify velocity state names.

Default value is ''.

### Quaternion vector: e.g., {'qr', 'qi', 'qj', 'qk'}

Specify quaternion vector state names.

Default value is ''.

### Body rotation rates: e.g., {'p', 'q', 'r'}

Specify body rotation rate state names.

Default value is ''.

## Inputs and Outputs

| Input  | Dimension Type       | Description  |
|--------|----------------------|--|
| First  | Vector               | Contains the three applied forces.                       |
| Second | Vector               | Contains the three applied moments.                      |
| Output | Dimension Type       | Description  |
| First  | Three-element vector | Contains the velocity in the flat Earth reference frame. |
| Second | Three-element vector | Contains the position in the flat Earth reference frame. |



| Output           | Dimension Type       | Description  |
|------------------|----------------------|--|
| Third            | Three-element vector | Contains the Euler rotation angles [roll, pitch, yaw], in radians.   |
| Fourth           | 3-by-3 matrix        | Contains the coordinate transformation from flat Earth axes to body-fixed axes.  |
| Fifth            | Three-element vector | Contains the velocity in the body-fixed frame.   |
| Sixth            | Three-element vector | Contains the angular rates in body-fixed axes, in radians per second.  |
| Seventh          | Three-element vector | Contains the angular accelerations in body-fixed axes, in radians per second squared.  |
| Eight            | Three-element vector | Contains the accelerations in body-fixed axes with respect to body frame.  |
| Ninth (Optional) | Three-element vector | Contains the accelerations in body-fixed axes with respect to inertial frame (flat Earth). You typically connect this signal to the accelerometer. |

## 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.

## Reference

Stevens, Brian, and Frank Lewis, *Aircraft Control and Simulation*, Second Edition, John Wiley & Sons, 2003.

Zipfel, Peter H., *Modeling and Simulation of Aerospace Vehicle Dynamics*. Second Edition, AIAA Education Series, 2007.

## See Also

6DOF (Euler Angles)

6DOF ECEF (Quaternion)

6DOF Wind (Quaternion)

6DOF Wind (Wind Angles)

6th Order Point Mass (Coordinated Flight)

Custom Variable Mass 6DOF (Euler Angles)

Custom Variable Mass 6DOF (Quaternion)

Custom Variable Mass 6DOF ECEF (Quaternion)

Custom Variable Mass 6DOF Wind (Quaternion)

Custom Variable Mass 6DOF Wind (Wind Angles)

Simple Variable Mass 6DOF (Euler Angles)

Simple Variable Mass 6DOF (Quaternion)

Simple Variable Mass 6DOF ECEF (Quaternion)

Simple Variable Mass 6DOF Wind (Quaternion)

Simple Variable Mass 6DOF Wind (Wind Angles)

“About Aerospace Coordinate Systems” on page 2-8

## **Extended Capabilities**

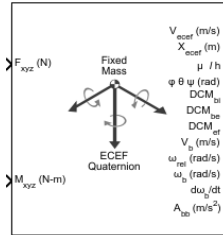
### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**Introduced in R2006a**

## 6DOF ECEF (Quaternion)

Implement quaternion representation of six-degrees-of-freedom equations of motion in Earth-centered Earth-fixed (ECEF) coordinates

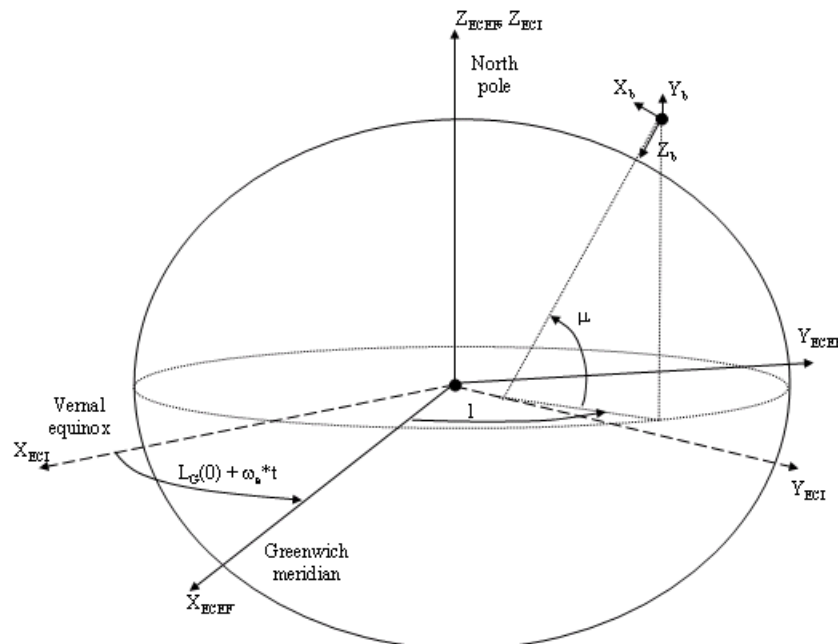


## Library

Equations of Motion/6DOF

## Description

The 6DOF ECEF (Quaternion) block considers the rotation of a Earth-centered Earth-fixed (ECEF) coordinate frame ( $X_{ECEF}$ ,  $Y_{ECEF}$ ,  $Z_{ECEF}$ ) about an Earth-centered inertial (ECI) reference frame ( $X_{ECI}$ ,  $Y_{ECI}$ ,  $Z_{ECI}$ ). The origin of the ECEF coordinate frame is the center of the Earth, additionally the body of interest is assumed to be rigid, an assumption that eliminates the need to consider the forces acting between individual elements of mass. The representation of the rotation of ECEF frame from ECI frame is simplified to consider only the constant rotation of the ellipsoid Earth ( $\omega_e$ ) including an initial celestial longitude ( $L_G(0)$ ). This excellent approximation allows the forces due to the Earth's complex motion relative to the "fixed stars" to be neglected.



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

$$\bar{F}_b = \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} = m \left( \dot{\bar{V}}_b + \bar{\omega}_b \times \bar{V}_b + DCM_{bf} \bar{\omega}_e \times \bar{V}_b + DCM_{bf} (\bar{\omega}_e \times (\bar{\omega}_e \times \bar{X}_f)) \right)$$

where the change of position in ECEF  $\dot{\bar{X}}_f$  is calculated by

$$\dot{\bar{X}}_f = DCM_{fb} \bar{V}_b$$

and the velocity of the body with respect to ECEF frame, expressed in body frame ( $\bar{V}_b$ ), angular rates of the body with respect to ECI frame, expressed in body frame ( $\bar{\omega}_b$ ). Earth rotation rate ( $\bar{\omega}_e$ ), and relative angular rates of the body with respect to north-east-down (NED) frame, expressed in body frame ( $\bar{\omega}_{rel}$ ) are defined as

$$\bar{V}_b = \begin{bmatrix} u \\ v \\ w \end{bmatrix}, \bar{\omega}_{rel} = \begin{bmatrix} p \\ q \\ r \end{bmatrix}, \bar{\omega}_e = \begin{bmatrix} 0 \\ 0 \\ \omega_e \end{bmatrix}, \bar{\omega}_b = \bar{\omega}_{rel} + DCM_{bf} \bar{\omega}_e + DCM_{be} \bar{\omega}_{ned}$$

$$\bar{\omega}_{ned} = \begin{bmatrix} \dot{\mu} \cos \mu \\ -\dot{\mu} \\ -\dot{\mu} \sin \mu \end{bmatrix} = \begin{bmatrix} V_E / (N + h) \\ -V_N / (M + h) \\ -V_E \cdot \tan \mu / (N + h) \end{bmatrix}$$

The rotational dynamics of the body defined in 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.

$$A_{bb} = \begin{bmatrix} \dot{u}_b \\ \dot{v}_b \\ \dot{w}_b \end{bmatrix} = \frac{1}{m} \bar{F}_b - [\bar{\omega}_b \times \bar{V}_b + DCM_{bf} \bar{\omega}_e \times \bar{V}_b + DCM_{bf} (\bar{\omega}_e \times (\bar{\omega}_e \times \bar{X}_f))]$$

$$A_{becef} = \frac{F_b}{m}$$

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

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

The integration of the rate of change of the quaternion vector is given below.

$$\begin{bmatrix} \dot{q}_0 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} = -1/2 \begin{bmatrix} 0 & \omega_b(1) & \omega_b(2) & \omega_b(3) \\ -\omega_b(1) & 0 & -\omega_b(3) & \omega_b(2) \\ -\omega_b(2) & \omega_b(3) & 0 & -\omega_b(1) \\ -\omega_b(3) & -\omega_b(2) & \omega_b(1) & 0 \end{bmatrix} \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix}$$

Aerospace Blockset uses quaternions that are defined using the scalar-first convention.

## Parameters

### Main

#### Units

Specifies the input and output units:

| Units                      | Forces | Moment       | Acceleration              | Velocity          | Position | Mass     | Inertia                |
|----------------------------|--------|--------------|---------------------------|-------------------|----------|----------|------------------------|
| Metric (MKS)               | Newton | Newton meter | Meters per second squared | Meters per second | Meters   | Kilogram | Kilogram meter squared |
| English (Velocity in ft/s) | Pound  | Foot pound   | Feet per second squared   | Feet per second   | Feet     | Slug     | Slug foot squared      |
| English (Velocity in kts)  | Pound  | Foot pound   | Feet per second squared   | Knots             | Feet     | Slug     | Slug foot squared      |

#### Mass type

Select the type of mass to use:

|                 |  |
|-----------------|--|
| Fixed           | Mass is constant throughout the simulation.                |
| Simple Variable | Mass and inertia vary linearly as a function of mass rate. |
| Custom Variable | Mass and inertia variations are customizable.              |

The Fixed selection conforms to the previously described equations of motion.

#### Initial position in geodetic latitude, longitude and altitude

The three-element vector for the initial location of the body in the geodetic reference frame, with latitude, longitude, and altitude. The altitude value depends on the selected units (meters (MKS) or feet (English)). Latitude and longitude values are in degrees and can be any value. However, latitude values of +90 and -90 may return unexpected values because of singularity at the poles.

#### Initial velocity in body axes

The three-element vector containing the initial velocity of the body with respect to ECEF frame, expressed in body frame.

#### Initial Euler orientation

The three-element vector containing the initial Euler rotation angles [roll, pitch, yaw], in radians. Euler rotation angles are those between the body and north-east-down (NED) coordinate systems.

#### Initial body rotation rates

The three-element vector for the initial angular rates of the body with respect to NED frame, expressed in body frame, in radians per second.

#### Initial mass

The mass of the rigid body.

#### Inertia

The 3-by-3 inertia tensor matrix  $I$ , in body-fixed axes.

#### Planet model

Specifies the planet model to use: Custom or Earth (WGS84).

**Flattening**

Specifies the flattening of the planet. This option is only available when **Planet model** is set to Custom.

**Equatorial radius of planet**

Specifies the radius of the planet at its equator. The units of the equatorial radius parameter should be the same as the units for ECEF position. This option is only available when **Planet model** is set to Custom.

**Rotational rate**

Specifies the scalar rotational rate of the planet in rad/s. This option is only available when **Planet model** is set to Custom.

**Celestial longitude of Greenwich source**

Specifies the source of Greenwich meridian's initial celestial longitude:

|          |   |
|----------|---|
| Internal | Use celestial longitude value from mask dialog.   |
| External | Use external input for celestial longitude value. |

**Celestial longitude of Greenwich**

The initial angle between Greenwich meridian and the x-axis of the ECI frame.

**Include inertial acceleration**

Select this check box to enable an additional output port for the accelerations in body-fixed axes with respect to the inertial frame. You typically connect this signal to the accelerometer.

**State Attributes**

Assign unique name to each state. You can use state names instead of block paths during linearization.

- To assign a name to a single state, enter a unique name between quotes, for example, 'velocity'.
- To assign names to multiple states, enter a comma-delimited list surrounded by braces, for example, {'a', 'b', 'c'}. Each name must be unique.
- If a parameter is empty (' '), no name assignment occurs.
- The state names apply only to the selected block with the name parameter.
- The number of states must divide evenly among the number of state names.
- You can specify fewer names than states, but you cannot specify more names than states.

For example, you can specify two names in a system with four states. The first name applies to the first two states and the second name to the last two states.

- To assign state names with a variable in the MATLAB workspace, enter the variable without quotes. A variable can be a character vector, cell array, or structure.

**Quaternion vector: e.g., {'qr', 'qi', 'qj', 'qk'}**

Specify quaternion vector state names.

Default value is ''.

**Body rotation rates: e.g., {'p', 'q', 'r'}**

Specify body rotation rate state names.

Default value is ''.

**Velocity: e.g., {'U', 'v', 'w'}**

Specify velocity state names.

Default value is ''.

**ECEF position: e.g., {'Xecef', 'Yecef', 'Zecef'}**

Specify the ECEF position state names.

Default value is ''.

**Inertial position: e.g., {'Xeci', 'Yeci', 'Zeci'}**

Specify the inertial position state names.

Default value is ''.

**Celestial longitude of Greenwich: e.g., 'LG'**

Specify the Celestial longitude of Greenwich state name.

Default value is ''.

## Inputs and Outputs

| Input   | Dimension Type       | Description  |
|---------|----------------------|--|
| First   | Vector               | Contains the three applied forces in body-fixed axes.  |
| Second  | Vector               | Contains the three applied moments in body-fixed axes.   |
| Output  | Dimension Type       | Description  |
| First   | Vector               | Contains the velocity of the body with respect to ECEF frame, expressed in ECEF frame.   |
| Second  | Three-element vector | Contains the position in ECEF reference frame.   |
| Third   | Three-element vector | Contains the position in geodetic latitude, longitude and altitude, in degrees, degrees and selected units of length respectively.                               |
| Fourth  | Three-element vector | Contains the body rotation angles [roll, pitch, yaw], in radians. Euler rotation angles are those between the body and north-east-down (NED) coordinate systems. |
| Fifth   | 3-by-3 matrix        | Applies to the coordinate transformation from ECI axes to body-fixed axes  |
| Sixth   | 3-by-3 matrix        | Applies to the coordinate transformation from NED axes to body-fixed axes.   |
| Seventh | 3-by-3 matrix        | Applies to the coordinate transformation from ECEF axes to NED axes.   |
| Eighth  | Three-element vector | Contains the velocity of the body with respect to ECEF frame, expressed in the body frame.   |
| Ninth   | Three-element vector | Contains the relative angular rates of the body with respect to NED frame, expressed in the body frame, in radians per second.                                   |

| Output                | Dimension Type       | Description   |
|-----------------------|----------------------|---|
| Tenth                 | Three-element vector | Contains the angular rates of the body with respect to the ECI frame, expressed in body frame, in radians per second.                 |
| Eleventh              | Three-element vector | Contains the angular accelerations of the body with respect to ECI frame, expressed in the body frame, in radians per second squared. |
| Twelfth               | Three-element vector | Contains the accelerations in body-fixed axes with respect to body frame.   |
| Thirteenth (Optional) | Three-element vector | Contains the accelerations in body-fixed axes with respect to ECEF frame.   |

## Assumptions and Limitations

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

This implementation generates a geodetic latitude that lies between  $\pm 90$  degrees, and longitude that lies between  $\pm 180$  degrees. Additionally, the MSL altitude is approximate.

The Earth is assumed to be ellipsoidal. By setting flattening to 0.0, a spherical planet can be achieved. The Earth's precession, nutation, and polar motion are neglected. The celestial longitude of Greenwich is Greenwich Mean Sidereal Time (GMST) and provides a rough approximation to the sidereal time.

The implementation of the ECEF coordinate system assumes that the origin is at the center of the planet, the x-axis intersects the Greenwich meridian and the equator, the z-axis is the mean spin axis of the planet, positive to the north, and the y-axis completes the right-handed system.

The implementation of the ECI coordinate system assumes that the origin is at the center of the planet, the x-axis is the continuation of the line from the center of the Earth toward the vernal equinox, the z-axis points in the direction of the mean equatorial plane's north pole, positive to the north, and the y-axis completes the right-handed system.

## References

Stevens, B. L., and F. L. Lewis, *Aircraft Control and Simulation, Second Edition*, John Wiley & Sons, New York, 2003.

McFarland, Richard E., *A Standard Kinematic Model for Flight simulation at NASA-Ames*, NASA CR-2497.

"Supplement to Department of Defense World Geodetic System 1984 Technical Report: Part I - Methods, Techniques and Data Used in WGS84 Development," DMA TR8350.2-A.

## See Also

6DOF (Euler Angles)

6DOF (Quaternion)

6DOF Wind (Quaternion)



6DOF Wind (Wind Angles)

6th Order Point Mass (Coordinated Flight)

Custom Variable Mass 6DOF (Euler Angles)

Custom Variable Mass 6DOF (Quaternion)

Custom Variable Mass 6DOF ECEF (Quaternion)

Custom Variable Mass 6DOF Wind (Quaternion)

Custom Variable Mass 6DOF Wind (Wind Angles)

Simple Variable Mass 6DOF (Euler Angles)

Simple Variable Mass 6DOF (Quaternion)

Simple Variable Mass 6DOF ECEF (Quaternion)

Simple Variable Mass 6DOF Wind (Quaternion)

Simple Variable Mass 6DOF Wind (Wind Angles)

“About Aerospace Coordinate Systems” on page 2-8

## **Extended Capabilities**

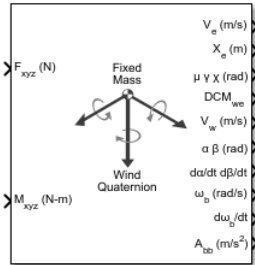
### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**Introduced in R2006a**

## 6DOF Wind (Quaternion)

Implement quaternion representation of six-degrees-of-freedom equations of motion with respect to wind axes

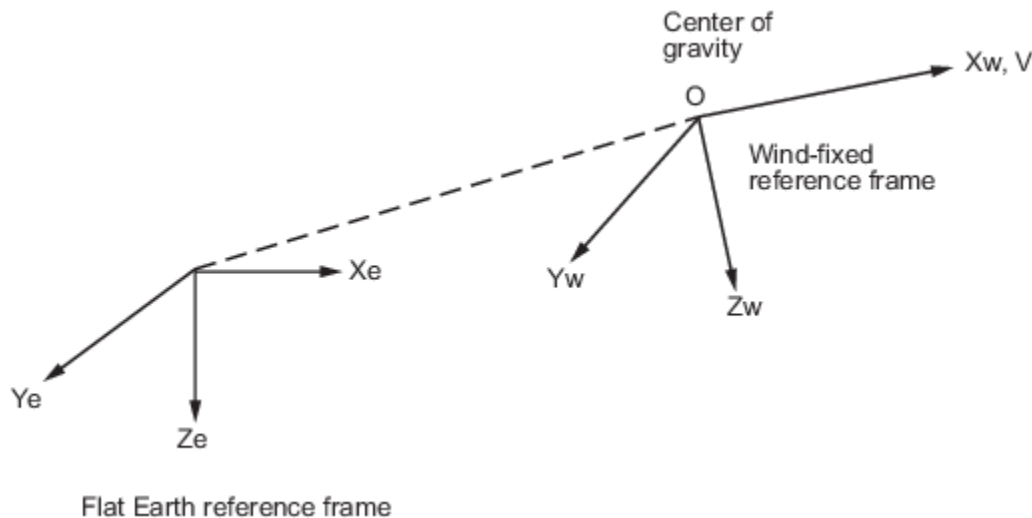


## Library

Equations of Motion/6DOF

## Description

The 6DOF Wind (Quaternion) block considers the rotation of a wind-fixed coordinate frame ( $X_w, Y_w, Z_w$ ) about an flat Earth reference frame ( $X_e, Y_e, Z_e$ ). The origin of the wind-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 flat Earth reference frame is considered inertial, an excellent approximation that allows the forces due to the Earth's motion relative to the “fixed stars” to be neglected.



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

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

$$A_{be} = DCM_{wb} \frac{\bar{F}_w}{m}$$

$$\bar{V}_w = \begin{bmatrix} V \\ 0 \\ 0 \end{bmatrix}, \bar{\omega}_w = \begin{bmatrix} p_w \\ q_w \\ r_w \end{bmatrix} = DMC_{wb} \begin{bmatrix} p_b - \dot{\beta} \sin \alpha \\ q_b - \dot{\alpha} \\ r_b + \dot{\beta} \cos \alpha \end{bmatrix}, \bar{\omega}_b = \begin{bmatrix} p_b \\ q_b \\ r_b \end{bmatrix}$$

$$A_{bb} = \begin{bmatrix} \dot{u}_b \\ \dot{v}_b \\ \dot{w}_b \end{bmatrix} = DCM_{wb} \left[ \frac{\bar{F}_w}{m} - \bar{\omega}_w \times \bar{V}_w \right]$$

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. Inertia tensor  $I$  is much easier to define in body-fixed frame.

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

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

The integration of the rate of change of the quaternion vector is given below.

$$\begin{bmatrix} \dot{q}_0 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} = -1/2 \begin{bmatrix} 0 & p & q & r \\ -p & 0 & -r & q \\ -q & r & 0 & -p \\ -r & -q & p & 0 \end{bmatrix} \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix}$$

Aerospace Blockset uses quaternions that are defined using the scalar-first convention.

## Parameters

### Main

### Units

Specifies the input and output units:

| Units        | Forces | Moment       | Acceleration              | Velocity          | Position | Mass     | Inertia                |
|--------------|--------|--------------|---------------------------|-------------------|----------|----------|------------------------|
| Metric (MKS) | Newton | Newton meter | Meters per second squared | Meters per second | Meters   | Kilogram | Kilogram meter squared |

| Units                            | Forces | Moment     | Acceleration               | Velocity           | Position | Mass | Inertia              |
|----------------------------------|--------|------------|----------------------------|--------------------|----------|------|----------------------|
| English<br>(Velocity<br>in ft/s) | Pound  | Foot pound | Feet per second<br>squared | Feet per<br>second | Feet     | Slug | Slug foot<br>squared |
| English<br>(Velocity<br>in kts)  | Pound  | Foot pound | Feet per second<br>squared | Knots              | Feet     | Slug | Slug foot<br>squared |

### Mass Type

Select the type of mass to use:

|                 |  |
|-----------------|--|
| Fixed           | Mass is constant throughout the simulation.                |
| Simple Variable | Mass and inertia vary linearly as a function of mass rate. |
| Custom Variable | Mass and inertia variations are customizable.              |

The Fixed selection conforms to the previously described equations of motion.

### Representation

Select the representation to use:

|             |   |
|-------------|---|
| Wind Angles | Use wind angles within equations of motion. |
| Quaternion  | Use quaternions within equations of motion. |

The Quaternion selection conforms to the previously described equations of motion.

### Initial position in inertial axes

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

### Initial airspeed, angle of attack, and sideslip angle

The three-element vector containing the initial airspeed, initial angle of attack and initial sideslip angle.

### Initial wind orientation

The three-element vector containing the initial wind angles [bank, flight path, and heading], in radians.

### Initial body rotation rates

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

### Initial mass

The mass of the rigid body.

### Inertia matrix

The 3-by-3 inertia tensor matrix  $I$ , in body-fixed axes.

### Include inertial acceleration

Select this check box to enable an additional output port for the accelerations in body-fixed axes with respect to the inertial frame. You typically connect this signal to the accelerometer.

### State Attributes

Assign unique name to each state. You can use state names instead of block paths during linearization.

- To assign a name to a single state, enter a unique name between quotes, for example, 'velocity'.
- To assign names to multiple states, enter a comma-delimited list surrounded by braces, for example, {'a', 'b', 'c'}. Each name must be unique.
- If a parameter is empty (' '), no name assignment occurs.
- The state names apply only to the selected block with the name parameter.
- The number of states must divide evenly among the number of state names.
- You can specify fewer names than states, but you cannot specify more names than states.

For example, you can specify two names in a system with four states. The first name applies to the first two states and the second name to the last two states.

- To assign state names with a variable in the MATLAB workspace, enter the variable without quotes. A variable can be a character vector, cell array, or structure.

**Position: e.g., {'Xe', 'Ye', 'Ze'}**

Specify position state names.

Default value is ''.

**Velocity: e.g., 'V'**

Specify velocity state name.

Default value is ''.

**Incidence angle: e.g., 'alpha'**

Specify incidence angle state name.

Default value is ''.

**Sideslip angle: e.g., 'beta'**

Specify sideslip angle state name.

Default value is ''.

**Quaternion vector: e.g., {'qr', 'qi', 'qj', 'qk'}**

Specify quaternion vector state names.

Default value is ''.

**Body rotation rates: e.g., {'p', 'q', 'r'}**

Specify body rotation rate state names.

Default value is ''.

## Inputs and Outputs

| Input  | Dimension Type | Description  |
|--------|----------------|--|
| First  | Vector         | Contains the three applied forces in wind-fixed axes.  |
| Second | Vector         | Contains the three applied moments in body-fixed axes. |

| Output              | Dimension Type       | Description  |
|---------------------|----------------------|--|
| First               | Three-element vector | Contains the velocity in the flat Earth reference frame.   |
| Second              | Three-element vector | Contains the position in the flat Earth reference frame.   |
| Third               | Three-element vector | Contains the wind rotation angles [bank, flight path, heading], in radians.  |
| Fourth              | 3-by-3 matrix        | Contains the coordinate transformation from flat Earth axes to wind-fixed axes.  |
| Fifth               | Three-element vector | Contains the velocity in the wind-fixed frame.   |
| Sixth               | Two-element vector   | Contains the angle of attack and sideslip angle, in radians.   |
| Seventh             | Two-element vector   | Contains the rate of change of angle of attack and rate of change of sideslip angle, in radians per second.  |
| Eight               | Three-element vector | Contains the angular rates in body-fixed axes, in radians per second.  |
| Ninth               | Three-element vector | Contains the angular accelerations in body-fixed axes, in radians per second squared.  |
| Tenth               | Three-element vector | Contains the accelerations in body-fixed axes with respect to body frame.  |
| Eleventh (Optional) | Three-element vector | Contains the accelerations in body-fixed axes with respect to inertial frame (flat Earth). You typically connect this signal to the accelerometer. |

## 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.

## Reference

Stevens, B. L., and F. L. Lewis, *Aircraft Control and Simulation*, John Wiley & Sons, New York, 1992.

## See Also

6DOF (Euler Angles)

6DOF (Quaternion)

6DOF ECEF (Quaternion)

6DOF Wind (Wind Angles)

6th Order Point Mass (Coordinated Flight)

Custom Variable Mass 6DOF (Euler Angles)

Custom Variable Mass 6DOF (Quaternion)

Custom Variable Mass 6DOF ECEF (Quaternion)

Custom Variable Mass 6DOF Wind (Quaternion)

Custom Variable Mass 6DOF Wind (Wind Angles)

Simple Variable Mass 6DOF (Euler Angles)

Simple Variable Mass 6DOF (Quaternion)

Simple Variable Mass 6DOF ECEF (Quaternion)

Simple Variable Mass 6DOF Wind (Quaternion)

Simple Variable Mass 6DOF Wind (Wind Angles)

“About Aerospace Coordinate Systems” on page 2-8

## **Extended Capabilities**

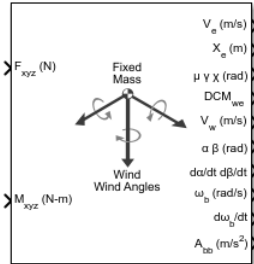
### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

### **Introduced in R2006a**

## 6DOF Wind (Wind Angles)

Implement wind angle representation of six-degrees-of-freedom equations of motion



### Library

Equations of Motion/6DOF

### Description

For a description of the coordinate system employed and the translational dynamics, see the block description for the 6DOF Wind (Quaternion) block.

The relationship between the wind angles,  $[\mu \gamma \chi]^T$ , can be determined by resolving the wind rates into the wind-fixed coordinate frame.

$$\begin{bmatrix} p_w \\ q_w \\ r_w \end{bmatrix} = \begin{bmatrix} \dot{\mu} \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\mu & \sin\mu \\ 0 & -\sin\mu & \cos\mu \end{bmatrix} \begin{bmatrix} 0 \\ \dot{\gamma} \\ 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\mu & \sin\mu \\ 0 & -\sin\mu & \cos\mu \end{bmatrix} \begin{bmatrix} \cos\gamma & 0 & -\sin\gamma \\ 0 & 1 & 0 \\ \sin\gamma & 0 & \cos\gamma \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ \dot{\chi} \end{bmatrix} \equiv J^{-1} \begin{bmatrix} \dot{\mu} \\ \dot{\gamma} \\ \dot{\chi} \end{bmatrix}$$

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

$$\begin{bmatrix} \dot{\mu} \\ \dot{\gamma} \\ \dot{\chi} \end{bmatrix} = J \begin{bmatrix} p_w \\ q_w \\ r_w \end{bmatrix} = \begin{bmatrix} 1 & (\sin\mu \tan\gamma) & (\cos\mu \tan\gamma) \\ 0 & \cos\mu & -\sin\mu \\ 0 & \frac{\sin\mu}{\cos\gamma} & \frac{\cos\mu}{\cos\gamma} \end{bmatrix} \begin{bmatrix} p_w \\ q_w \\ r_w \end{bmatrix}$$

The body-fixed angular rates are related to the wind-fixed angular rate by the following equation.

$$\begin{bmatrix} p_w \\ q_w \\ r_w \end{bmatrix} = DMC_{wb} \begin{bmatrix} p_b - \dot{\beta} \sin\alpha \\ q_b - \dot{\alpha} \\ r_b + \dot{\beta} \cos\alpha \end{bmatrix}$$

Using this relationship in the wind rate vector equations, gives the relationship between the wind rate vector and the body-fixed angular rates.



$$\begin{bmatrix} \dot{\mu} \\ \dot{\gamma} \\ \dot{\chi} \end{bmatrix} = J \begin{bmatrix} p_w \\ q_w \\ r_w \end{bmatrix} = \begin{bmatrix} 1 & (\sin\mu\tan\gamma) & (\cos\mu\tan\gamma) \\ 0 & \cos\mu & -\sin\mu \\ 0 & \frac{\sin\mu}{\cos\gamma} & \frac{\cos\mu}{\cos\gamma} \end{bmatrix} DMC_{wb} \begin{bmatrix} p_b - \dot{\beta}\sin\alpha \\ q_b - \dot{\alpha} \\ r_b + \dot{\beta}\cos\alpha \end{bmatrix}$$

## Parameters

### Main

#### Units

Specifies the input and output units:

| Units                      | Forces | Moment       | Acceleration              | Velocity          | Position | Mass     | Inertia                |
|----------------------------|--------|--------------|---------------------------|-------------------|----------|----------|------------------------|
| Metric (MKS)               | Newton | Newton meter | Meters per second squared | Meters per second | Meters   | Kilogram | Kilogram meter squared |
| English (Velocity in ft/s) | Pound  | Foot pound   | Feet per second squared   | Feet per second   | Feet     | Slug     | Slug foot squared      |
| English (Velocity in kts)  | Pound  | Foot pound   | Feet per second squared   | Knots             | Feet     | Slug     | Slug foot squared      |

#### Mass type

Select the type of mass to use:

|                 |  |
|-----------------|--|
| Fixed           | Mass is constant throughout the simulation.                |
| Simple Variable | Mass and inertia vary linearly as a function of mass rate. |
| Custom Variable | Mass and inertia variations are customizable.              |

The Fixed selection conforms to the previously described equations of motion.

#### Representation

Select the representation to use:

|             |   |
|-------------|---|
| Wind Angles | Use wind angles within equations of motion. |
| Quaternion  | Use quaternions within equations of motion. |

The Wind Angles selection conforms to the previously described equations of motion.

#### Initial position in inertial axes

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

#### Initial airspeed, angle of attack, and sideslip angle

The three-element vector containing the initial airspeed, initial angle of attack and initial sideslip angle.

#### Initial wind orientation

The three-element vector containing the initial wind angles [bank, flight path, and heading], in radians.

**Initial body rotation rates**

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

**Initial mass**

The mass of the rigid body.

**Inertia**

The 3-by-3 inertia tensor matrix  $I$ , in body-fixed axes.

**Include inertial acceleration**

Select this check box to enable an additional output port for the accelerations in body-fixed axes with respect to the inertial frame. You typically connect this signal to the accelerometer.

**State Attributes**

Assign unique name to each state. You can use state names instead of block paths during linearization.

- To assign a name to a single state, enter a unique name between quotes, for example, 'velocity'.
- To assign names to multiple states, enter a comma-delimited list surrounded by braces, for example, {'a', 'b', 'c'}. Each name must be unique.
- If a parameter is empty (' '), no name assignment occurs.
- The state names apply only to the selected block with the name parameter.
- The number of states must divide evenly among the number of state names.
- You can specify fewer names than states, but you cannot specify more names than states.

For example, you can specify two names in a system with four states. The first name applies to the first two states and the second name to the last two states.

- To assign state names with a variable in the MATLAB workspace, enter the variable without quotes. A variable can be a character vector, cell array, or structure.

**Position: e.g., {'Xe', 'Ye', 'Ze'}**

Specify position state names.

Default value is ''.

**Velocity: e.g., 'V'**

Specify velocity state name.

Default value is ''.

**Incidence angle: e.g., 'alpha'**

Specify incidence angle state name.

Default value is ''.

**Sideslip angle: e.g., 'beta'**

Specify sideslip angle state name.

Default value is ''.

**Wind orientation: e.g., {'mu', 'gamma', 'chi'}**

Specify wind orientation state names. This parameter appears if the **Representation** parameter is set to Wind Angles.

Default value is ''.

**Body rotation rates: e.g., {'p', 'q', 'r'}**

Specify body rotation rate state names.

Default value is ''.

**Inputs and Outputs**

| Input  | Dimension Type | Description  |
|--------|----------------|--|
| First  | Vector         | Contains the three applied forces in wind-fixed axes.  |
| Second | Vector         | Contains the three applied moments in body-fixed axes. |

| Output              | Dimension Type       | Description  |
|---------------------|----------------------|--|
| First               | Three-element vector | Contains the velocity in the flat Earth reference frame.   |
| Second              | Three-element vector | Contains the position in the flat Earth reference frame.   |
| Third               | Three-element vector | Contains the wind rotation angles [bank, flight path, heading], within $\pm\pi$ , in radians.  |
| Fourth              | 3-by-3 matrix        | Contains the coordinate transformation from flat Earth axes to wind-fixed axes.  |
| Fifth               | Three-element vector | Contains the velocity in the wind-fixed frame.   |
| Sixth               | Two-element vector   | Contains the angle of attack and sideslip angle, in radians.   |
| Seventh             | Two-element vector   | Contains the rate of change of angle of attack and rate of change of sideslip angle, in radians per second.  |
| Eighth              | Three-element vector | Contains the angular rates in body-fixed axes, in radians per second.  |
| Ninth               | Three-element vector | Contains the angular accelerations in body-fixed axes, in radians per second squared.  |
| Tenth               | Three-element vector | Contains the accelerations in body-fixed axes with respect to body frame.  |
| Eleventh (Optional) | Three-element vector | Contains the accelerations in body-fixed axes with respect to inertial frame (flat Earth). You typically connect this signal to the accelerometer. |

**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.

## Reference

Stevens, B. L., and F. L. Lewis, *Aircraft Control and Simulation*, John Wiley & Sons, New York, 1992.

## See Also

6DOF (Euler Angles)

6DOF (Quaternion)

6DOF ECEF (Quaternion)

6DOF Wind (Quaternion)

6th Order Point Mass (Coordinated Flight)

Custom Variable Mass 6DOF (Euler Angles)

Custom Variable Mass 6DOF (Quaternion)

Custom Variable Mass 6DOF ECEF (Quaternion)

Custom Variable Mass 6DOF Wind (Quaternion)

Custom Variable Mass 6DOF Wind (Wind Angles)

Simple Variable Mass 6DOF (Euler Angles)

Simple Variable Mass 6DOF (Quaternion)

Simple Variable Mass 6DOF ECEF (Quaternion)

Simple Variable Mass 6DOF Wind (Quaternion)

Simple Variable Mass 6DOF Wind (Wind Angles)

“About Aerospace Coordinate Systems” on page 2-8

## Extended Capabilities

### C/C++ Code Generation

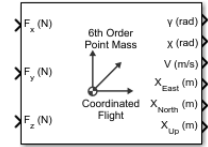
Generate C and C++ code using Simulink® Coder™.

### Introduced in R2006a

# 6th Order Point Mass (Coordinated Flight)

Calculate sixth-order point mass in coordinated flight

**Library:** Aerospace Blockset / Equations of Motion / Point Mass



## Description

The 6th Order Point Mass (Coordinated Flight) block performs the calculations for the translational motion of a single point mass or multiple point masses. For more information on the system for the translational motion of a single point mass or multiple mass, see “Algorithms” on page 4-106.

The 6th Order Point Mass (Coordinated Flight) block port labels change based on the input and output units selected from the **Units** list.

## Limitations

- The block assumes that there is fully coordinated flight, i.e., there is no side force (wind axes) and sideslip is always zero.
- The flat Earth reference frame is considered inertial, an approximation that allows the forces due to the Earth motion relative to the "fixed stars" to be neglected.

## Ports

### Input

#### Port\_1 — Force in x-axis

scalar | array

Force in x-axis, specified as a scalar or vector, in selected units.

Data Types: double

#### Port\_2 — Force in y-axis

scalar | array

Force in y-axis, specified as a scalar or vector, in selected units.

Data Types: double

#### Port\_3 — Force in z-axis

scalar | array

Force in z-axis, specified as a scalar or vector, in selected units.

Data Types: double

**Output****Port\_1 – Flight path angle**

scalar | array

Flight path angle, returned as a scalar or vector, in radians.

Data Types: double

**Port\_2 – Heading angle**

scalar | array

Heading angle, returned as a scalar or vector, in radians.

Data Types: double

**Port\_3 – Airspeed**

scalar | array

Airspeed, returned as a scalar or vector, in selected units.

Data Types: double

**Port\_4 – Downrange or amount traveled east**

scalar | array

Downrange or amount traveled east, returned as a scalar or vector, in selected units.

Data Types: double

**Port\_5 – Crossrange or amount travelled north**

scalar | array

Crossrange or amount traveled north, returned as a scalar or vector, in selected units.

Data Types: double

**Port\_6 – Altitude or amount or travelled up**

scalar | array

Altitude or amount traveled up, returned as a scalar or vector, in selected units.

Data Types: double

**Parameters****Units – Units**

Metric (MKS) (default) | English (Velocity in ft/s) | English (Velocity in kts)

Input and output units, specified as:

| Units                      | Forces | Velocity          | Position |
|----------------------------|--------|-------------------|----------|
| Metric (MKS)               | Newton | Meters per second | Meters   |
| English (Velocity in ft/s) | Pound  | Feet per second   | Feet     |

| Units                     | Forces | Velocity | Position |
|---------------------------|--------|----------|----------|
| English (Velocity in kts) | Pound  | Knots    | Feet     |

**Programmatic Use****Block Parameter:** units**Type:** character vector**Values:** 'Metric (MKS)' | 'English (Velocity in ft/s)' | 'English (Velocity in kts)'**Default:** 'Metric (MKS)'**Initial flight path angle – Initial flight path angle** $\theta$  (default) | scalar | vector

Initial flight path angle of the point mass(es), specified as a scalar or vector.

**Programmatic Use****Block Parameter:** gamma $\theta$ **Type:** character vector**Values:** scalar | vector**Default:** '0'**Initial heading angle – Initial heading angle** $\theta$  (default) | scalar | vector

Initial heading angle of the point mass(es), specified as a scalar or vector.

**Programmatic Use****Block Parameter:** chi $\theta$ **Type:** character vector**Values:** scalar | vector**Default:** '0'**Initial airspeed – Initial airspeed**

100 (default) | scalar | vector

Initial airspeed of the point mass(es), specified as a scalar or vector.

**Programmatic Use****Block Parameter:** V $\theta$ **Type:** character vector**Values:** scalar | vector**Default:** '100'**Initial downrange [East] – Initial downrange** $\theta$  (default) | scalar | vector

Initial downrange of the point mass(es), specified as a scalar or vector.

**Programmatic Use****Block Parameter:** x $\theta$ **Type:** character vector**Values:** scalar | vector**Default:** '0'

**Initial crossrange [North] – Initial cross range** $\theta$  (default) | scalar | vector

Initial crossrange of the point mass(es), specified as a scalar or vector.

**Programmatic Use****Block Parameter:**  $y\theta$ **Type:** character vector**Values:** scalar | vector**Default:** '0'**Initial altitude [Up] – Initial altitude** $\theta$  (default) | scalar | vector

Initial altitude of the point mass(es), specified as a scalar or vector.

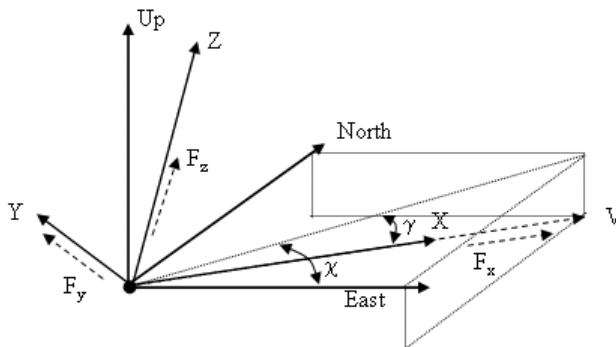
**Programmatic Use****Block Parameter:**  $h\theta$ **Type:** character vector**Values:** scalar | vector**Default:** '0'**Initial mass – Point mass**

1.0 (default) | scalar | vector

Mass of the point mass(es), specified as a scalar or vector.

**Programmatic Use****Block Parameter:**  $mass\theta$ **Type:** character vector**Values:** scalar | vector**Default:** '1.0'**Algorithms**

This figure shows the system for the translational motion of a single point mass or multiple point masses.



The translational motion of the point mass  $[X_{East} X_{North} X_{Up}]^T$  are functions of airspeed ( $V$ ), flight path angle ( $\gamma$ ), and heading angle ( $\chi$ ),



$$F_x = m\dot{V}$$

$$F_y = (mV\cos\gamma)\dot{\chi}$$

$$F_z = mV\dot{\gamma}$$

$$\dot{X}_{East} = V\cos\chi\cos\gamma$$

$$\dot{X}_{North} = V\sin\chi\cos\gamma$$

$$\dot{X}_{Up} = V\sin\gamma$$

where the applied forces  $[F_x F_y F_h]^T$  are in a system is defined by x-axis in the direction of vehicle velocity relative to air, z-axis is upward, and y-axis completes the right-handed frame, and the mass of the body  $m$  is assumed constant.

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

### See Also

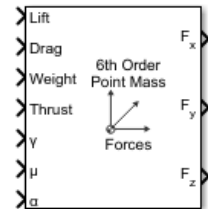
4th Order Point Mass Forces (Longitudinal) | 4th Order Point Mass (Longitudinal) | 6DOF (Euler Angles) | 6DOF (Quaternion) | 6DOF ECEF (Quaternion) | 6DOF Wind (Wind Angles) | 6th Order Point Mass Forces (Coordinated Flight) | Custom Variable Mass 6DOF (Euler Angles) | Custom Variable Mass 6DOF (Quaternion) | Custom Variable Mass 6DOF ECEF (Quaternion) | Custom Variable Mass 6DOF Wind (Quaternion) | Custom Variable Mass 6DOF Wind (Wind Angles) | Simple Variable Mass 6DOF (Euler Angles) | Simple Variable Mass 6DOF (Quaternion) | Simple Variable Mass 6DOF ECEF (Quaternion) | Simple Variable Mass 6DOF Wind (Quaternion) | Simple Variable Mass 6DOF Wind (Wind Angles)

**Introduced before R2006a**

## 6th Order Point Mass Forces (Coordinated Flight)

Calculate forces used by sixth-order point mass in coordinated flight

**Library:** Aerospace Blockset / Equations of Motion / Point Mass



### Description

The 6th Order Point Mass Forces (Coordinated Flight) block calculates the applied forces for a single point mass or multiple point masses. For more information on the system for the applied forces, see "Algorithms" on page 4-109.

### Limitations

- The block assumes that there is fully coordinated flight, i.e., there is no side force (wind axes) and sideslip is always zero.
- The flat Earth reference frame is considered inertial, an approximation that allows the forces due to the Earth motion relative to the "fixed stars" to be neglected.

### Ports

#### Input

##### Lift – Lift

scalar | array

Lift, specified as a scalar or array, in units of force.

Data Types: double

##### Drag – Drag

scalar | array

Drag, specified as a scalar or array, in units of force.

Data Types: double

##### Weight – Weight

scalar | array

Weight, specified as a scalar or array, in units of force.

Data Types: double

##### Thrust – Thrust

scalar | array

Thrust, specified as a scalar or array, in units of force.

Data Types: double

### **$\gamma$ — Flight path angles**

scalar | array

Flight path angle, specified as a scalar or array, in radians.

Data Types: double

### **$\mu$ — Bank angle**

scalar | array

Bank angle, specified as a scalar or array, in radians.

Data Types: double

### **$\alpha$ — Angle of attack**

scalar | array

Angle of attack, specified as a scalar or array, in radians.

Data Types: double

## **Output**

### **$F_x$ — Force in x- axis**

scalar | array

Force in x-axis, specified as a scalar or array, in units of force.

Data Types: double

### **$F_y$ — Force in y- axis**

scalar | array

Force in y-axis, specified as a scalar or array, in units of force.

Data Types: double

### **$F_z$ — Force in z- axis**

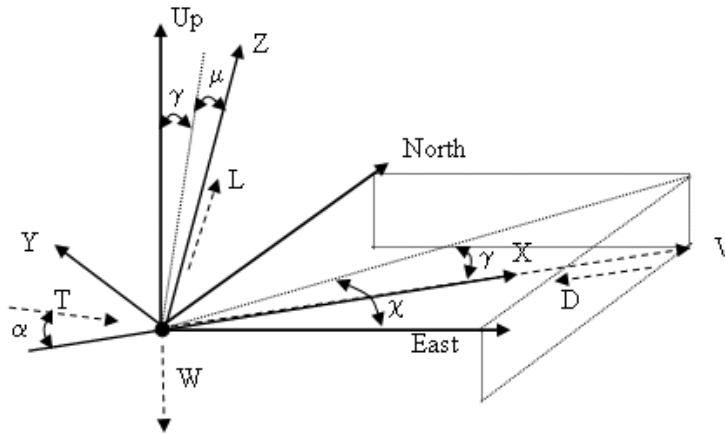
scalar | array

Force in z-axis, specified as a scalar or array, in units of force.

Data Types: double

## **Algorithms**

This figure shows the applied forces in the system used by this block.



The applied forces  $[F_x F_y F_h]^T$  are in a system is defined by  $x$ -axis in the direction of vehicle velocity relative to air,  $z$ -axis is upwards and  $y$ -axis completes the right-handed frame and are functions of lift ( $L$ ), drag ( $D$ ), thrust ( $T$ ), weight ( $W$ ), flight path angle ( $\gamma$ ), angle of attack ( $\alpha$ ), and bank angle ( $\mu$ ).

$$F_x = T \cos \alpha - D - W \sin \gamma$$

$$F_y = (L + T \sin \alpha) \sin \mu$$

$$F_z = (L + T \sin \alpha) \cos \mu - W \cos \gamma$$

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

### See Also

4th Order Point Mass (Longitudinal) | 4th Order Point Mass Forces (Longitudinal) | 6th Order Point Mass (Coordinated Flight)

Introduced before R2006a

# Acceleration Conversion

Convert from acceleration units to desired acceleration units

**Library:** Aerospace Blockset / Utilities / Unit Conversions



## 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 port labels change based on the input and output units selected from the **Initial unit** and **Final unit** parameters.

## Ports

### Input

#### Port\_1 — Acceleration

scalar | array

Acceleration, specified as a scalar or array, in initial acceleration units.

### Dependencies

The input port label depends on the **Initial unit** setting.

Data Types: double

### Output

#### Port\_1 — Acceleration

scalar | array

Acceleration, returned as a scalar or array, in final acceleration units.

### Dependencies

The output port label depends on the **Final unit** setting.

Data Types: double

## Parameters

### Initial unit — Input units

ft/s<sup>2</sup> (default) | m/s<sup>2</sup> | km/s<sup>2</sup> | in/s<sup>2</sup> | km/h-s | mph/s | G's

Input units, specified as:

|                   |                           |
|-------------------|---------------------------|
| m/s <sup>2</sup>  | Meters per second squared |
| ft/s <sup>2</sup> | Feet per second squared   |

|                   |                                |
|-------------------|--------------------------------|
| km/s <sup>2</sup> | Kilometers per second squared  |
| in/s <sup>2</sup> | Inches per second squared      |
| km/h-s            | Kilometers per hour per second |
| mph-s             | Miles per hour per second      |
| G's               | g-units                        |

### Dependencies

The input port label depends on the **Initial unit** setting.

### Programmatic Use

**Block Parameter:** IU

**Type:** character vector

**Values:** 'ft/s<sup>2</sup>' | 'm/s<sup>2</sup>' | 'km/s<sup>2</sup>' | 'in/s<sup>2</sup>' | 'km/h-s' | 'mph/s' | 'G's'

**Default:** 'ft/s<sup>2</sup>'

### Final unit – Output units

ft/s<sup>2</sup>' (default) | m/s<sup>2</sup> | km/s<sup>2</sup> | in/s<sup>2</sup> | km/h-s | mph/s | G's

Output units, specified as:

|                   |                                |
|-------------------|--------------------------------|
| m/s <sup>2</sup>  | Meters per second squared      |
| ft/s <sup>2</sup> | Feet per second squared        |
| km/s <sup>2</sup> | Kilometers per second squared  |
| in/s <sup>2</sup> | Inches per second squared      |
| km/h-s            | Kilometers per hour per second |
| mph-s             | Miles per hour per second      |
| G's               | g-units                        |

### Dependencies

The output port label depends on the **Final unit** setting.

### Programmatic Use

**Block Parameter:** OU

**Type:** character vector

**Values:** 'ft/s<sup>2</sup>' | 'm/s<sup>2</sup>' | 'km/s<sup>2</sup>' | 'in/s<sup>2</sup>' | 'km/h-s' | 'mph/s' | 'G's'

**Default:** 'ft/s<sup>2</sup>'

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

## See Also

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

**Introduced before R2006a**

## Adjoint of 3x3 Matrix

Compute adjoint of matrix

**Library:** Aerospace Blockset / Utilities / Math Operations



### Description

The Adjoint of 3x3 Matrix block computes the adjoint matrix for the input matrix. For related equations, see “Algorithms” on page 4-114.

### Ports

#### Input

##### Port 1 – Input matrix

3-by-3 matrix

Input matrix, specified as a 3-by-3 matrix, in initial acceleration units.

Data Types: double

#### Output

##### Port 1 – Output acceleration

3-by-3 matrix

Output acceleration, returned as a 3-by-3 matrix, in final acceleration units.

Data Types: double

### Algorithms

The input matrix has the form of

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix}$$

The adjoint of the matrix has the form of

$$\text{adj}(A) = \begin{pmatrix} + \begin{vmatrix} A_{22} & A_{23} \\ A_{32} & A_{33} \end{vmatrix} - \begin{vmatrix} A_{12} & A_{13} \\ A_{32} & A_{33} \end{vmatrix} + \begin{vmatrix} A_{12} & A_{13} \\ A_{22} & A_{23} \end{vmatrix} \\ - \begin{vmatrix} A_{21} & A_{23} \\ A_{31} & A_{33} \end{vmatrix} + \begin{vmatrix} A_{11} & A_{13} \\ A_{31} & A_{33} \end{vmatrix} - \begin{vmatrix} A_{11} & A_{13} \\ A_{21} & A_{23} \end{vmatrix} \\ + \begin{vmatrix} A_{21} & A_{22} \\ A_{31} & A_{32} \end{vmatrix} - \begin{vmatrix} A_{11} & A_{12} \\ A_{31} & A_{32} \end{vmatrix} + \begin{vmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{vmatrix} \end{pmatrix}$$



## **Extended Capabilities**

### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

### **See Also**

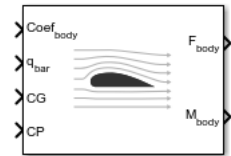
[Create 3x3 Matrix](#) | [Determinant of 3x3 Matrix](#) | [Invert 3x3 Matrix](#)

**Introduced before R2006a**

## Aerodynamic Forces and Moments

Compute aerodynamic forces and moments using aerodynamic coefficients, dynamic pressure, center of gravity, center of pressure, and velocity

**Library:** Aerospace Blockset / Aerodynamics



### Description

The Aerodynamic Forces and Moments block computes the aerodynamic forces and moments about the center of gravity.

The Aerodynamic Forces and Moments block port labels change based on the coordinate system selected from the **Input axes**, **Force axes**, and **Moment axes** list.

### Limitations

- The default state of the block hides the  $V_b$  input port and assumes that the transformation is body-body.
- The center of gravity and the center of pressure are assumed to be in body axes.
- While this block has the ability to output forces and/or moments in the stability axes, the blocks in the Equations of Motion library are currently designed to accept forces and moments in either the body or wind axes only.

### Ports

#### Input

##### Port\_1 — Aerodynamic coefficients

six-element vector

Aerodynamic coefficients (in the chosen input axes) for forces and moments, specified as a vector. These coefficients are ordered into a vector depending on the choice of axes:

| Input Axes | Input Vector  |
|------------|---|
| Body       | (axial force $C_x$ , side force $C_y$ , normal force $C_z$ , rolling moment $C_l$ , pitching moment $C_m$ , yawing moment $C_n$ )         |
| Stability  | (drag force $C_{D(\beta=0)}$ , side force $C_y$ , lift force $C_L$ , rolling moment $C_l$ , pitching moment $C_m$ , yawing moment $C_n$ ) |
| Wind       | (drag force $C_D$ , cross-wind force $C_c$ , lift force $C_L$ , rolling moment $C_l$ , pitching moment $C_m$ , yawing moment $C_n$ )      |

Data Types: double

**Port\_2 – Dynamic pressure**

scalar | three-element vector

Dynamic pressure, specified as a 1-by-3 array.

Data Types: double

**Port\_3 – Center of gravity**

three-element vector

Center of gravity, specified as a 1-by-3 vector.

Data Types: double

**Port\_4 – Center of pressure**

three-element vector

Center of pressure, specified as a 1-by-3 vector. This can also be taken as any general moment reference point as long as the rest of the model reflects the use of the moment reference point.

Data Types: double

**Port\_5 – Velocity in the body axes**

three-element vector

Velocity in the body axes. specified as a 1-by-3 vector.

**Dependencies**This port is enabled if the **Input axes** parameter is set to Wind or Stability.

Data Types: double

**Output****Port\_1 – Aerodynamic forces**

three-element vector

Aerodynamic forces (in the chosen output axes), returned as three-element vector, at the center of gravity in x-, y-, and z-axes.

Data Types: double

**Port\_2 – Aerodynamic moments**

three-element vector

Aerodynamic moments (in the chosen output axes), returned as three-element vector, at the center of gravity in x-, y-, and z-axes.

Data Types: double

**Parameters****Input axes – Coordinate system for input coefficients**

Body (default) | Stability | Wind

Coordinate system for input coefficients, specified as Body (default), Stability, or Wind.

**Dependencies**

Selecting Stability or Wind enables input port Port\_5.

**Programmatic Use**

**Block Parameter:** inputAxes

**Type:** character vector

**Values:** 'Body' | 'Stability' | 'Wind'

**Default:** 'Body'

**Force axes — Coordinate system for aerodynamic force**

Body (default) | Stability | Wind

Coordinate system for aerodynamic force, specified as Body (default), Stability, or Wind.

**Dependencies**

Selecting Stability or Wind enables input port Port\_5.

**Programmatic Use**

**Block Parameter:** outputForcesAxes

**Type:** character vector

**Values:** 'Body' | 'Stability' | 'Wind'

**Default:** 'Body'

**Moment axes — Coordinate system for aerodynamic moment**

Body (default) | Stability | Wind

Coordinate system for aerodynamic moment, specified as Body (default), Stability, or Wind.

**Dependencies**

Selecting Stability or Wind enables input port Port\_5.

**Programmatic Use**

**Block Parameter:** outputMomentAxes

**Type:** character vector

**Values:** 'Body' | 'Stability' | 'Wind'

**Default:** 'Body'

**Reference area — Reference area**

1 (default) | any double value

Reference area for calculating aerodynamic forces and moments, specified as any double value.

**Programmatic Use**

**Block Parameter:** S

**Type:** character vector

**Values:** any double value

**Default:** '1'

**Reference span — Reference span**

1 (default) | any double value

Reference span for calculating aerodynamic moments in x-axes and z-axes, specified as any double value.

**Programmatic Use****Block Parameter:** b**Type:** character vector**Values:** any double value**Default:** ' 1 '**Reference length — Reference length**

1 (default) | any double value

Reference length for calculating aerodynamic moment in the y-axes, specified as any double value.

**Programmatic Use****Block Parameter:** cbar**Type:** character vector**Values:** any double value**Default:** ' 1 '**Algorithms**Let  $\alpha$  be the angle of attack and  $\beta$  the sideslip. The rotation from body to stability axes:

$$C_{s \leftarrow b} = \begin{bmatrix} \cos(\alpha) & 0 & \sin(\alpha) \\ 0 & 1 & 0 \\ -\sin(\alpha) & 0 & \cos(\alpha) \end{bmatrix}$$

can be combined with the rotation from stability to wind axes:

$$C_{w \leftarrow s} = \begin{bmatrix} \cos(\beta) & \sin(\beta) & 0 \\ -\sin(\beta) & \cos(\beta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

to yield the net rotation from body to wind axes:

$$C_{w \leftarrow b} = \begin{bmatrix} \cos(\alpha)\cos(\beta) & \sin(\beta) & \sin(\alpha)\cos(\beta) \\ -\cos(\alpha)\sin(\beta) & \cos(\beta) & -\sin(\alpha)\sin(\beta) \\ -\sin(\alpha) & 0 & \cos(\alpha) \end{bmatrix}$$

Moment coefficients have the same notation in all systems. Force coefficients are given below. Note there are no specific symbols for stability-axes force components. However, the stability axes have two components that are unchanged from the other axes.

$$\mathbf{F}_A^w \equiv \begin{bmatrix} -D \\ -C \\ -L \end{bmatrix} = C_{w \leftarrow b} \cdot \begin{bmatrix} X_A \\ Y_A \\ Z_A \end{bmatrix} \equiv C_{w \leftarrow b} \cdot \mathbf{F}_A^b$$

| Components/Axes | x     | y     | z            |
|-----------------|-------|-------|--------------|
| Wind            | $C_D$ | $C_C$ | $C_L$        |
| Stability       | —     | $C_Y$ | $C_L$        |
| Body            | $C_X$ | $C_Y$ | $C_Z (-C_N)$ |

Given these definitions, to account for the standard definitions of  $D$ ,  $C$ ,  $Y$  (where  $Y = -C$ ), and  $L$ , force coefficients in the wind axes are multiplied by the negative identity  $diag(-1, -1, -1)$ . Forces coefficients

in the stability axes are multiplied by  $diag(-1, 1, -1)$ .  $C_N$  and  $C_X$  are, respectively, the normal and axial force coefficients ( $C_N = -C_Z$ ).

## References

- [1] Stevens, B. L., and F. L. Lewis, *Aircraft Control and Simulation*, John Wiley & Sons, New York, 1992

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

## See Also

Digital DATCOM Forces and Moments | Dynamic Pressure | Estimate Center of Gravity | Moments about CG due to Forces

## Topics

“NASA HL-20 Lifting Body Airframe” on page 3-14

**Introduced before R2006a**

# Airspeed Indicator

Display measurements for aircraft airspeed

**Library:** Aerospace Blockset / Flight Instruments



## Description

The Airspeed Indicator block displays measurements for aircraft airspeed in knots.

By default, minor ticks represent 10-knot increments and major ticks represent 40-knot increments. The parameters **Minimum** and **Maximum** determine the minimum and maximum values on the gauge. The number and distribution of ticks is fixed, which means that the first and last tick display the minimum and maximum values. The ticks in between distribute evenly between the minimum and maximum values. For major ticks, the distribution of ticks is  $(\text{Maximum}-\text{Minimum})/9$ . For minor ticks, the distribution of ticks is  $(\text{Maximum}-\text{Minimum})/36$ .

The airspeed indicator has scale color bars that allow for overlapping for the first bar, displayed at a different radius. This different radius lets the block represent maximum speed with flap extended ( $V_{FE}$ ) and stall speed with flap extended ( $V_{SO}$ ) accurately for aircraft airspeed and stall speed.

---

**Tip** To facilitate understanding and debugging your model, you can modify instrument block connections in your model during normal and accelerator mode simulations.

---

## Parameters

### Connection — Connect to signal

signal name

Connect to signal for display, selected from list of signal names.

To view the data from a signal, select a signal in the model. The signal appears in the **Connection** table. Select the option button next to the signal you want to display. Click **Apply** to connect the signal.

The table has a row for the signal connected to the block. If there are no signals selected in the model, or the block is not connected to any signals, the table is empty.

### Minimum — Minimum tick mark value

40 (default) | finite | double | scalar

Minimum tick mark value, specified as a finite double scalar value, in knots.

### Dependencies

The **Minimum** tick value must be less than the **Maximum** tick value.

**Programmatic Use****Block Parameter:** Limits**Type:** double**Values:** vector**Default:** [40 400], where 40 is the minimum value**Maximum — Maximum tick mark value**

400 (default) | finite | double | scalar

Specify the maximum tick mark value, specified as a finite double scalar value, in knots.

**Dependencies**

The **Maximum** tick value must be greater than the **Minimum** tick value.

**Programmatic Use****Block Parameter:** Limits**Type:** double**Values:** vector**Default:** [40 400], where 400 is the maximum value**Scale Colors — Ranges of color bands**

0 (default) | double | scalar

Ranges of color bands outside the scale, specified as a finite double scalar value. Specify the minimum and maximum color range to display on the gauge.

To add a new color, click +. To remove a color, click -.

**Programmatic Use****Block Parameter:** ScaleColors**Type:** *n*-by-1 struct array**Values:** struct array with elements Min, Max, and Color**Label — Name of connected signal**

Top (default) | Bottom | Hide

Name of connected signal.

- Top

Show label at the top of the block.

- Bottom

Show label at the bottom of the block.

- Hide

Do not show the label or instructional text when the block is not connected.

**Programmatic Use****Block Parameter:** LabelPosition**Type:** character vector**Values:** 'Top' | 'Bottom' | 'Hide'**Default:** 'Top'



## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

This block is ignored for code generation.

### See Also

Indicator | Altimeter | Artificial Horizon | Climb Rate Indicator | Exhaust Gas Temperature (EGT) Indicator | Heading Indicator | Revolutions Per Minute (RPM) Indicator | Turn Coordinator

### Topics

“Display Measurements with Cockpit Instruments” on page 2-39

“Programmatically Interact with Gauge Band Colors” on page 2-41

“Flight Instrument Gauges” on page 2-38

### Introduced in R2016a

# Altimeter

Display measurements for aircraft altitude

**Library:** Aerospace Blockset / Flight Instruments



## Description

The Altimeter Indicator block displays the altitude above sea level in feet, also known as the pressure altitude. The block displays the altitude value with needles on a gauge and a numeric indicator.

- The gauge has 10 major ticks. Within each major tick are five minor ticks. This gauge has three needles. Using the needles, the altimeter can display accurately only altitudes between 0 and 100,000 feet.
  - For the longest needle, an increment of a small tick represents 20 feet and a major tick represents 100 feet.
  - For the second longest needle, a minor tick represents 200 feet and a major tick represents 1,000 feet.
  - For the shortest needle a minor tick represents 2,000 feet and a major tick represents 10,000 feet.
- For the numeric display, the block shows values as numeric characters between 0 and 9,999 feet. When the numeric display value reaches 10,000 feet, the gauge displays the value as the remaining values below 10,000 feet. For example, 12,345 feet displays as 2,345 feet. When a value is less than 0 (below sea level), the block displays 0. The needles show the appropriate value except for when the value is below sea level or over 99999 feet. Below sea level, the needles set to 0, over 99,999, the needles stay set at 99,999.

---

**Tip** To facilitate understanding and debugging your model, you can modify instrument block connections in your model during normal and accelerator mode simulations.

---

## Parameters

### Connection — Connect to signal

signal name

Connect to signal for display, selected from list of signal names.

To view the data from a signal, select a signal in the model. The signal appears in the **Connection** table. Select the option button next to the signal you want to display. Click **Apply** to connect the signal.

The table has a row for the signal connected to the block. If there are no signals selected in the model, or the block is not connected to any signals, the table is empty.

**Label — Block label location**

Top (default) | Bottom | Hide

Block label, displayed at the top or bottom of the block, or hidden.

- Top

Show label at the top of the block.

- Bottom

Show label at the bottom of the block.

- Hide

Do not show the label or instructional text when the block is not connected.

**Programmatic Use****Block Parameter:** LabelPosition**Type:** character vector**Values:** 'Top' | 'Bottom' | 'Hide'**Default:** 'Top'**Extended Capabilities****C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

This block is ignored for code generation.

**See Also**

Airspeed Indicator | Artificial Horizon | Climb Rate Indicator | Exhaust Gas Temperature (EGT) Indicator | Heading Indicator | Revolutions Per Minute (RPM) Indicator | Turn Coordinator

**Topics**

“Display Measurements with Cockpit Instruments” on page 2-39

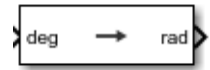
“Flight Instrument Gauges” on page 2-38

**Introduced in R2016a**

# Angle Conversion

Convert from angle units to desired angle units

**Library:** Aerospace Blockset / Utilities / Unit Conversions



## 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 port labels change based on the input and output units selected from the **Initial unit** and the **Final unit** lists.

## Ports

### Input

#### Port\_1 – Angle

scalar | array

Angle, specified as a scalar or array, in initial acceleration units.

### Dependencies

The input port label depends on the **Initial unit** setting.

Data Types: double

### Output

#### Port\_1 – Angle

scalar | array

Angle, returned as a scalar, in final acceleration units.

### Dependencies

The output port label depends on the **Final unit** setting.

Data Types: double

## Parameters

### Initial unit – Input units

deg (default) | rad | rev

Input units, specified as:

|     |         |
|-----|---------|
| deg | Degrees |
| rad | Radians |

|     |             |
|-----|-------------|
| rev | Revolutions |
|-----|-------------|

**Dependencies**

The input port label depends on the **Initial unit** setting.

**Programmatic Use**

**Block Parameter:** IU

**Type:** character vector

**Values:** 'deg' | 'rad' | 'rev'

**Default:** 'deg'

**Final unit – Output units**

rad (default) | deg | rev

Output units, specified as:

|     |             |
|-----|-------------|
| deg | Degrees     |
| rad | Radians     |
| rev | Revolutions |

**Dependencies**

The output port label depends on the **Final unit** setting.

**Programmatic Use**

**Block Parameter:** OU

**Type:** character vector

**Values:** 'deg' | 'rad' | 'rev'

**Default:** 'rad'

**Extended Capabilities****C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**See Also**

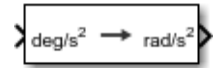
Acceleration Conversion | Angular Acceleration Conversion | Angular Velocity Conversion | Density Conversion | Force Conversion | Length Conversion | Mass Conversion | Pressure Conversion | Temperature Conversion | Velocity Conversion

**Introduced before R2006a**

# Angular Acceleration Conversion

Convert from angular acceleration units to desired angular acceleration units

**Library:** Aerospace Blockset / Utilities / Unit Conversions



## 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 port labels change based on the input and output units selected from the **Initial unit** and the **Final unit** lists.

## Ports

### Input

#### Port\_1 – Angular input acceleration

scalar | array

Angle, specified as a scalar, in initial acceleration units.

#### Dependencies

The input port label depends on the **Initial unit** setting.

Data Types: double

### Output

#### Port\_1 – Angular output acceleration

scalar | array

Angle, returned as a scalar, in final acceleration units.

#### Dependencies

The output port label depends on the **Final unit** setting.

Data Types: double

## Parameters

#### Initial unit – Input units

deg/s<sup>2</sup> (default) | rad/s<sup>2</sup> | rpm/s

Specifies the input units, specified as:

|                    |                            |
|--------------------|----------------------------|
| deg/s <sup>2</sup> | Degrees per second squared |
|--------------------|----------------------------|

|                    |                                   |
|--------------------|-----------------------------------|
| rad/s <sup>2</sup> | Radians per second squared        |
| rpm/s              | Revolutions per minute per second |

**Dependencies**

The input port label depends on the **Initial unit** setting.

**Programmatic Use**

**Block Parameter:** IU

**Type:** character vector

**Values:** 'deg/s<sup>2</sup>' | 'rad/s<sup>2</sup>' | 'rpm/s'

**Default:** 'deg/s<sup>2</sup>'

**Final unit – Output units**

rad/s<sup>2</sup> (default) | deg/s<sup>2</sup> | rpm/s

Output units, specified as:

|                    |                                   |
|--------------------|-----------------------------------|
| deg/s <sup>2</sup> | Degrees per second squared        |
| rad/s <sup>2</sup> | Radians per second squared        |
| rpm/s              | Revolutions per minute per second |

**Dependencies**

The output port label depends on the **Final unit** setting.

**Programmatic Use**

**Block Parameter:** OU

**Type:** character vector

**Values:** 'deg/s<sup>2</sup>' | 'rad/s<sup>2</sup>' | 'rpm/s'

**Default:** 'rad/s<sup>2</sup>'

**Extended Capabilities****C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**See Also**

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

**Introduced before R2006a**

# Angular Velocity Conversion

Convert from angular velocity units to desired angular velocity units

**Library:** Aerospace Blockset / Utilities / Unit Conversions



## 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 port labels change based on the input and output units selected from the **Initial unit** and the **Final unit** lists.

## Ports

### Input

#### Port\_1 – Angular acceleration

scalar | array

Angular acceleration, specified as a scalar, in initial angular acceleration units.

### Dependencies

The input port label depends on the **Initial unit** setting.

Data Types: double

### Output

#### Port\_1 – Angular acceleration

scalar | array

Angular acceleration, returned as a scalar, in final angular acceleration units.

### Dependencies

The output port label depends on the **Final unit** setting.

Data Types: double

## Parameters

### Initial unit – Input units

deg/s (default) | rad/s | rpm

Input units, specified as:

|       |                    |
|-------|--------------------|
| deg/s | Degrees per second |
|-------|--------------------|



|       |                        |
|-------|------------------------|
| rad/s | Radians per second     |
| rpm   | Revolutions per minute |

**Dependencies**

The input port label depends on the **Initial unit** setting.

**Programmatic Use**

**Block Parameter:** IU

**Type:** character vector

**Values:** 'deg/s' | 'rad/s' | 'rpm/s'

**Default:** 'deg/s'

**Final unit – Output units**

rad/s (default) | deg/s | rpm

Output units, specified as:

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

**Dependencies**

The output port label depends on the **Final unit** setting.

**Programmatic Use**

**Block Parameter:** OU

**Type:** character vector

**Values:** 'deg/s' | 'rad/s' | 'rpm/s'

**Default:** 'deg/s'

**Extended Capabilities****C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**See Also**

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

**Introduced before R2006a**

## Artificial Horizon

Represent aircraft attitude relative to horizon

**Library:** Aerospace Blockset / Flight Instruments



### Description

The Artificial Horizon block represents aircraft attitude relative to horizon and displays roll and pitch in degrees:

- Values for roll cannot exceed +/- 90 degrees.
- Values for pitch cannot exceed +/- 30 degrees.

If the values exceed the maximum values, the gauge maximum and minimum values do not change.

Changes in roll value affect the gauge semicircles and the ticks located on the black arc turn accordingly. Changes in pitch value affect the scales and the distribution of the semicircles.

Combine the roll and pitch signals in a Mux block in the order:

- 1 Roll
- 2 Pitch

---

**Tip** To facilitate understanding and debugging your model, you can modify instrument block connections in your model during normal and accelerator mode simulations.

---

### Parameters

#### Connection — Connect to signal

signal name | 2-element signal

Connect to 2-element signal for display, selected from list of signal names. The 2-element signal consists of roll and pitch combined together in a Mux block, in degrees. You connect and display this combined signal. This input cannot be a bus signal.

To view the data from a signal, select a signal in the model. The signal appears in the **Connection** table. Select the option button next to the signal you want to display. Click **Apply** to connect the signal.

The table has a row for the signal connected to the block. If there are no signals selected in the model, or the block is not connected to any signals, the table is empty.

To view the data from a signal, select a signal in the model. The signal appears in the **Connection** table. Select the option button next to the signal you want to display. Click **Apply** to connect the signal.

The table has a row for the signal connected to the block. If there are no signals selected in the model, or the block is not connected to any signals, the table is empty.

**Label — Block label location**

Top (default) | Bottom | Hide

Block label, displayed at the top or bottom of the block, or hidden.

- Top  
Show label at the top of the block.
- Bottom  
Show label at the bottom of the block.
- Hide  
Do not show the label or instructional text when the block is not connected.

**Programmatic Use**

**Block Parameter:** LabelPosition

**Type:** character vector

**Values:** 'Top' | 'Bottom' | 'Hide'

**Default:** 'Top'

**Extended Capabilities****C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**See Also**

Airspeed Indicator | Altimeter | Artificial Horizon | Climb Rate Indicator | Exhaust Gas Temperature (EGT) Indicator | Revolutions Per Minute (RPM) Indicator | Turn Coordinator

**Topics**

“Display Measurements with Cockpit Instruments” on page 2-39

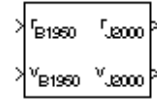
“Flight Instrument Gauges” on page 2-38

**Introduced in R2016a**

## Besselian Epoch to Julian Epoch

Transform position and velocity components from discontinued Standard Besselian Epoch (B1950) to Standard Julian Epoch (J2000)

**Library:** Aerospace Blockset / Utilities / Axes Transformations



### Description

The Besselian Epoch to Julian Epoch block transforms two 3-by-1 vectors of Besselian Epoch position ( $\bar{r}_{B1950}$ ), and Besselian Epoch velocity ( $\bar{v}_{B1950}$ ) into Julian Epoch position ( $\bar{r}_{J2000}$ ), and Julian Epoch velocity ( $\bar{v}_{J2000}$ ). For more information on the transformation, see “Algorithms” on page 4-134.

### Ports

#### Input

**$r_{B1950}$  — Position**  
3-by-1 vector

Position in Standard Besselian Epoch (B1950), specified as a 3-by-1 vector.

Data Types: `double`

**$v_{B1950}$  — Velocity**  
3-by-1 vector

Velocity in Standard Besselian Epoch (B1950), specified as a 3-by-1 vector.

Data Types: `double`

#### Output

**$r_{J2000}$  — Position**  
3-by-1 vector

Position in Standard Julian Epoch (J2000), returned as a 3-by-1 vector.

Data Types: `double`

**$v_{J2000}$  — Velocity**  
3-by-1 vector

Velocity in Standard Julian Epoch (J2000), returned as a 3-by-1 vector.

Data Types: `double`

### Algorithms

The transformation is calculated using:

$$\begin{bmatrix} \bar{r}_{J2000} \\ \bar{v}_{J2000} \end{bmatrix} = \begin{bmatrix} \bar{M}_{rr} & \bar{M}_{vr} \\ \bar{M}_{rv} & \bar{M}_{vv} \end{bmatrix} \begin{bmatrix} \bar{r}_{B1950} \\ \bar{v}_{B1950} \end{bmatrix}$$

where  $(\bar{M}_{rr}, \bar{M}_{vr}, \bar{M}_{rv}, \bar{M}_{vv})$  are defined as:

$$\bar{M}_{rr} = \begin{bmatrix} 0.9999256782 & -0.0111820611 & -0.0048579477 \\ 0.0111820610 & 0.9999374784 & -0.0000271765 \\ 0.0048579479 & -0.0000271474 & 0.9999881997 \end{bmatrix}$$

$$\bar{M}_{vr} = \begin{bmatrix} 0.00000242395018 & -0.00000002710663 & -0.00000001177656 \\ 0.00000002710663 & 0.00000242397878 & -0.00000000006587 \\ 0.00000001177656 & -0.00000000006582 & 0.00000242410173 \end{bmatrix}$$

$$\bar{M}_{rv} = \begin{bmatrix} -0.000551 & -0.238565 & 0.435739 \\ 0.238514 & -0.002667 & -0.008541 \\ -0.435623 & 0.012254 & 0.002117 \end{bmatrix}$$

$$\bar{M}_{vv} = \begin{bmatrix} 0.99994704 & -0.01118251 & -0.00485767 \\ 0.01118251 & 0.99995883 & -0.00002718 \\ 0.00485767 & -0.00002714 & 1.00000956 \end{bmatrix}$$

## References

- [1] "Supplement to Department of Defense World Geodetic System 1984 Technical Report: Part I - Methods, Techniques and Data Used in WGS84 Development," DMA TR8350.2-A.

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

## See Also

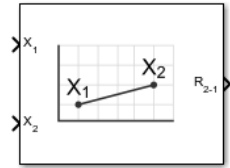
Julian Epoch to Besselian Epoch

Introduced before R2006a

## Calculate Range

Calculate range between two vehicles given their respective positions

**Library:** Aerospace Blockset / GNC / Guidance



### Description

The Calculate Range block computes the range between two vehicles. The equation used for the range calculation is

$$\text{Range} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

### Ports

#### Input

##### $x_1$ — Vehicle 1 position

3-element vector

Contains the (x, y, and z) position of vehicle 1, specified as a three-element vector. These values are of the double data type.

Data Types: double

##### $x_2$ — Vehicle 2 position

3-element vector

The (x, y, and z) position of vehicle 2, specified as a three-element vector. These values are of the double data type.

Data Types: double

#### Output

##### $R_{2-1}$ — Range

scalar

Range from vehicle 2 and vehicle 1, returned as a scalar of double data type. The calculated range is the magnitude of the distance, but not the direction. It is always positive or zero.

Data Types: double

### Extended Capabilities

#### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

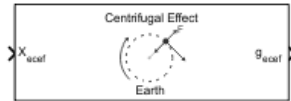
**See Also**

Three-axis Inertial Measurement Unit

**Introduced before R2006a**

## Centrifugal Effect Model

Implement mathematical representation of centrifugal effect for planetary gravity



## Library

Environment/Gravity

## Description

The Centrifugal Effect Model block implements the mathematical representation of centrifugal effect for planetary gravity. The gravity centrifugal effect is the acceleration portion of centrifugal force effects due to the rotation of a planet. This block implements this representation using planetary rotation rates. You use centrifugal force values in rotating or non-inertial coordinate systems.

## Parameters

### Planet model

Specify the planetary model. From the list, select Mercury, Venus, Earth, Moon, Mars, Jupiter, Saturn, Uranus, Neptune, or Custom. The block uses the rotation of the selected planet to implement the mathematical representation of the centrifugal effect.

Selecting Custom enables you to specify your own planetary model. This option enables the **Planetary rotational rate (rad/sec)** and **Input planetary rotation rate** parameters.

### Planetary rotational rate (rad/sec)

Specify the planetary rotational rate in radians per second.

If you want to specify the planetary rotational rate as an input to the block, see the **Input planetary rotation rate** parameter.

Selecting the **Input planetary rotation rate** check box disables the **Planetary rotational rate (rad/sec)** parameter.

### Input planetary rotation rate

Select this check box to enable a block input. You can then input a planetary rotation rate as a block input. When you select this check box, the block mask updates to display an input port for the rotation rate.

Selecting the **Input planetary rotation rate** check box disables the **Planetary rotational rate (rad/sec)** parameter.

## Inputs and Outputs

This block accepts only scalar inputs ( $m=1$ ).



| Input  | Dimension Type | Description  |
|--------|----------------|--|
| First  | m-by-3 matrix  | Contains planet-centered planet-fixed coordinates from the center of the planet. If <b>Planet model</b> has a value of Earth, this matrix contains Earth-centered Earth-fixed (ECEF) coordinates. The block does not use explicit units. |
| Output | Dimension Type | Description  |
| First  | m-by-3 array   | Contains gravity values in the x-axis, y-axis and z-axis of the planet-centered planet-fixed coordinates in input distance units per second squared.   |

## References

Vallado, D. A., *Fundamentals of Astrodynamics and Applications*, McGraw-Hill, New York, 1997.

NIMA TR8350.2: *Department of Defense World Geodetic System 1984, Its Definition and Relationship with Local Geodetic Systems*.

## Extended Capabilities

### C/C++ Code Generation

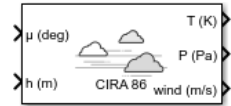
Generate C and C++ code using Simulink® Coder™.

### Introduced in R2010a

## CIRA-86 Atmosphere Model

Implement mathematical representation of 1986 CIRA atmosphere

**Library:** Aerospace Blockset / Environment / Atmosphere



### Description

The CIRA-86 Atmosphere Model block implements the mathematical representation of the 1986 Committee on Space Research (COSPAR) International Reference Atmosphere (CIRA). The block provides values for absolute temperature, pressure, density, and speed of sound for the input geopotential altitude.

The CIRA-86 Atmosphere Model block port labels change based on the input and output units selected from the **Units** list.

### Limitations

- This function uses a corrected version of the CIRA data files provided by J. Barnett in July 1990 in ASCII format.
- This function has the limitations of the CIRA 1986 model. The values for the CIRA 1986 model are limited to the regions of 80 degrees S to 80 degrees N on the Earth and geopotential heights of 0 to 120 kilometers. In each monthly mean data set, values at 80 degrees S for 101,300 pascal or 0 meters were omitted because these levels are within the Antarctic land mass. For zonal mean pressure in constant altitude coordinates, pressure data is not available below 20 kilometers. Therefore, this is the bottom level of the CIRA climatology.

### Ports

#### Input

##### Port\_1 – Latitude

array

Contains the latitude in degrees (limited to +/-80 degrees).

Data Types: double

##### Port\_2 – Geopotential heights or pressures

array

Contains an *m* array of either:

- Geopotential heights in selected length units (**Coordinate type** is GPHeight)
- Pressures in selected pressure units (**Coordinate type** is Pressure)

Data Types: double

**Output****Port\_1 – mean temperature**

array

Mean temperature, specified as an array, in selected units.

Data Types: double

**Port\_2 – Pressures or geopotential heights**

array

m array of either:

- Pressures in selected pressure units (**Coordinate type** is GPHeight)
- Geopotential heights in selected length units (**Coordinate type** is Pressure)

Data Types: double

**Port\_3 – Mean zonal winds**

array

Mean zonal winds, specified as an array, in selected units.

Data Types: double

**Parameters****Units – Units**

Metric (MKS) (default) | English (Velocity in ft/s) | English (Velocity in kts)

Input and output units, specified as:

| Units                      | Height | Temperature     | Speed of Sound    | Air Pressure                | Air Density               |
|----------------------------|--------|-----------------|-------------------|-----------------------------|---------------------------|
| Metric (MKS)               | Meters | Kelvin          | Meters per second | Pascal                      | Kilograms per cubic meter |
| English (Velocity in ft/s) | Feet   | Degrees Rankine | Feet per second   | Pound-force per square inch | Slug per cubic foot       |
| English (Velocity in kts)  | Feet   | Degrees Rankine | Knots             | Pound-force per square inch | Slug per cubic foot       |

**Programmatic Use****Block Parameter:** units**Type:** character vector**Values:** 'Metric (MKS)' | 'English (Velocity in ft/s)' | 'English (Velocity in kts)'**Default:** 'Metric (MKS)'**Coordinate type – Coordinate type representation**

Pressure (default) | GPHeight

Coordinate type representation, specified as:

- Pressure

Indicates pressure in pascal.

- GPHeight

Indicates geopotential height in meters.

**Programmatic Use**

**Block Parameter:** ctype

**Type:** character vector

**Values:** 'GPHeight' | 'Pressure'

**Default:** 'GPHeight'

**Mean value type — Mean value types**

Monthly (default) | Annual

Specify mean value types. The default is Monthly.

- Monthly

Indicates monthly values. If you select Monthly, you must also set the **Month** parameter.

- Annual

Indicates annual values. Valid when **Coordinate type** has a value of Pressure.

**Dependencies**

Setting this parameter to Monthly enables the **Month** parameter.

**Programmatic Use**

**Block Parameter:** mtype

**Type:** character vector

**Values:** 'Monthly' | 'Annual'

**Default:** 'Monthly'

**Month — Month of mean value**

January (default) | February | March | April | May | June | July | August | September |  
October | November | December

Indicates the month in which the mean values are taken. From the list, select the desired month. This parameter applies only when **Mean value type** has a value of Monthly.

**Dependencies**

This parameter is enabled when **Mean value type** is set to Monthly.

**Programmatic Use**

**Block Parameter:** month

**Type:** character vector

**Values:** 'January' | 'February' | 'March' | 'April' | 'May' | 'June' | 'July' | 'August' |  
'September' | 'October' | 'November' | 'December'

**Default:** 'January'

**Action for out-of-range input — Out-of-range block behavior**

None (default) | Warning | Error

Out-of-range block behavior, specified as follows.

| Action          | Description   |
|-----------------|---|
| None            | No action.  |
| Warning         | Warning in the MATLAB Command Window, model simulation continues. |
| Error (default) | MATLAB returns an exception, model simulation stops.              |

#### Programmatic Use

**Block Parameter:** action

**Type:** character vector

**Values:** 'None' | 'Warning' | 'Error'

**Default:** 'Warning'

## References

- [1] Fleming, E. L., Chandra, S., Shoerberl, M. R., Barnett, J. J., *Monthly Mean Global Climatology of Temperature, Wind, Geopotential Height and Pressure for 0-120 km*, NASA TM100697, February 1988

## Extended Capabilities

#### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

## See Also

ISA Atmosphere Model | COESA Atmosphere Model

#### External Websites

<https://ccmc.gsfc.nasa.gov/modelweb/atmos/cospar1.html>

**Introduced in R2007b**

# Climb Rate Indicator

Display measurements for aircraft climb rate

**Library:** Aerospace Blockset / Flight Instruments



## Description

The Climb Rate Indicator block displays measurements for an aircraft climb rate in ft/min.

The needle covers the top semicircle, if the velocity is positive, and the lower semicircle, if the climb rate is negative. The range of the indicator is from **-Maximum** feet per minute to **Maximum** feet per minute. Major ticks indicate **Maximum/4**. Minor ticks indicate **Maximum/8** and **Maximum/80**.

---

**Tip** To facilitate understanding and debugging your model, you can modify instrument block connections in your model during normal and accelerator mode simulations.

---

## Parameters

### Connection — Connect to signal

signal name

Connect to signal for display, selected from list of signal names.

To view the data from a signal, select a signal in the model. The signal appears in the **Connection** table. Select the option button next to the signal you want to display. Click **Apply** to connect the signal.

The table has a row for the signal connected to the block. If there are no signals selected in the model, or the block is not connected to any signals, the table is empty.

### Maximum — Maximum tick mark value

4000 (default) | finite | double | scalar

Maximum tick mark value, specified as a finite double scalar value, in ft/min.

The minimum tick value is always 0.

### Programmatic Use

**Block Parameter:** MaximumRate

**Type:** character vector

**Values:** scalar

**Default:** '4000'

### Label — Block label location

Top (default) | Bottom | Hide

Block label, displayed at the top or bottom of the block, or hidden.

- **Top**

Show label at the top of the block.

- **Bottom**

Show label at the bottom of the block.

- **Hide**

Do not show the label or instructional text when the block is not connected.

**Programmatic Use**

**Block Parameter:** LabelPosition

**Type:** character vector

**Values:** 'Top' | 'Bottom' | 'Hide'

**Default:** 'Top'

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

This block is ignored for code generation.

## See Also

Indicator | Airspeed Indicator | Altimeter | Artificial Horizon | Exhaust Gas Temperature (EGT) | Heading Indicator | Revolutions Per Minute (RPM) Indicator | Turn Coordinator

### Topics

“Display Measurements with Cockpit Instruments” on page 2-39

“Programmatically Interact with Gauge Band Colors” on page 2-41

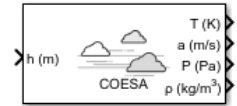
“Flight Instrument Gauges” on page 2-38

### Introduced in R2016a

# COESA Atmosphere Model

Implement 1976 COESA lower atmosphere

**Library:** Aerospace Blockset / Environment / Atmosphere



## Description

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

The COESA Atmosphere Model, Non-Standard Day 210C, and Non-Standard Day 310 blocks are identical blocks. When configured for COESA Atmosphere Model, the block implements the COESA mathematical representation. When configured for Non-Standard Day 210C, the block implements MIL-STD-210C climatic data. When configured for Non-Standard Day 310, the block implements MIL-HDBK-310 climatic data.

Below 32,000 meters (approximately 104,987 feet), the U.S. Standard Atmosphere is identical with the Standard Atmosphere of the International Civil Aviation Organization (ICAO).

The COESA Atmosphere Model block port labels change based on the input and output units selected from the **Units** list.

## Limitations

Below the geopotential altitude of 0 m (0 feet) and above the geopotential altitude of 84,852 m (approximately 278,386 feet), temperature values are extrapolated linearly and pressure values are extrapolated logarithmically. Density and speed of sound are calculated using a perfect gas relationship.

## Ports

### Input

#### Port\_1 – Geopotential height

scalar | array

Geopotential height, specified as a scalar or array, in specified units.

Data Types: double

### Output

#### Port\_1 – Temperature

scalar | array

Temperature, specified as a scalar or array, in specified units.



Data Types: double

### Port\_2 – Speed of sound

scalar | array

Speed of sound, specified as a scalar or array, in specified units.

Data Types: double

### Port\_3 – Air pressure

scalar | array

Air pressure, specified as a scalar or array, in specified units.

Data Types: double

### Port\_4 – Air density

scalar | array

Air density, specified as a scalar or array, in specified units.

Data Types: double

## Parameters

### Units – Input and output units

Metric (MKS) (default) | English (Velocity in ft/s) | English (Velocity in kts)

Input and output units, specified as:

| Units                      | Height | Temperature     | Speed of Sound    | Air Pressure                | Air Density               |
|----------------------------|--------|-----------------|-------------------|-----------------------------|---------------------------|
| Metric (MKS)               | Meters | Kelvin          | Meters per second | Pascal                      | Kilograms per cubic meter |
| English (Velocity in ft/s) | Feet   | Degrees Rankine | Feet per second   | Pound-force per square inch | Slug per cubic foot       |
| English (Velocity in kts)  | Feet   | Degrees Rankine | Knots             | Pound-force per square inch | Slug per cubic foot       |

#### Programmatic Use

**Block Parameter:** units

**Type:** character vector

**Values:** 'Metric (MKS)' | 'English (Velocity in ft/s)' | 'English (Velocity in kts)'

**Default:** 'Metric (MKS)'

#### Specification – Atmosphere model type

1976 COESA-extended U.S. Standard Atmosphere (default) | MIL-HDBK-310 | MIL-STD-210C

Atmosphere model type, specified as 1976 COESA-extended U.S. Standard Atmosphere, MIL-HDBK-310, or MIL-STD-210C. For the MIL-HDBK-310 and MIL-STD-210C options:

|                |   |
|----------------|---|
| MIL -HDBK -310 | This selection is linked to the Non-Standard Day 310 block. See the block reference for more information. Selecting MIL -HDBK -310 enables the parameters <b>Atmospheric model type</b> , <b>Extreme parameter</b> , <b>Frequency of occurrence</b> , and <b>Altitude of extreme value</b> .  |
| MIL -STD -210C | This selection is linked to the Non-Standard Day 210C block. See the block reference for more information. Selecting MIL -HDBK -310 enables the parameters <b>Atmospheric model type</b> , <b>Extreme parameter</b> , <b>Frequency of occurrence</b> , and <b>Altitude of extreme value</b> . |

#### Dependencies

Selecting MIL -HDBK -310 or MIL -STD -210C enables these parameters:

- **Atmospheric model type**
- **Extreme parameter**
- **Frequency of occurrence**
- **Altitude of extreme value**

#### Programmatic Use

**Block Parameter:** spec

**Type:** character vector

**Values:** '1976 COESA-extended U.S. Standard Atmosphere' | 'MIL-HDBK-310' | 'MIL-STD-210C'

**Default:** '1976 COESA-extended U.S. Standard Atmosphere'

#### Atmospheric model type — Model type

Profile (default) | Envelope

Representation of atmospheric model type, specified as:

|          |  |
|----------|--|
| Profile  | Realistic atmospheric profiles associated with extremes at specified altitudes. Recommended for simulation of vehicles vertically traversing the atmosphere or when the total influence of the atmosphere is needed. |
| Envelope | Uses extreme atmospheric values at each altitude. Recommended for vehicles only horizontally traversing the atmosphere without much change in altitude.  |

#### Dependencies

- Selecting MIL -HDBK -310 or MIL -STD -210C for the **Specification** parameter enables this parameter.
- Selecting Profile enables the **Attitude of extreme value** parameter.

#### Programmatic Use

**Block Parameter:** model

**Type:** character vector

**Values:** 'Profile' | 'Envelope'

**Default:** 'Profile'

**Extreme parameter – Model type**

High temperature (default) | Low temperature | High density | Low density | High pressure | Low pressure

Atmospheric parameter that is the extreme value.

**Dependencies**

- Selecting MIL-HDBK-310 or MIL-STD-210C for the **Specification** parameter enables this parameter.
- The High pressure and Low pressure options appear only when **Atmospheric model type** is set to Envelope.

**Programmatic Use**

**Block Parameter:** profile\_var

**Type:** character vector

**Values:** 'High temperature' | 'Low temperature' | 'High density' | 'Low density' | 'High pressure' | 'Low pressure'

**Default:** 'High temperature'

**Frequency of occurrence – Model type**

1% (default) | Extreme values | 5% | 10% | 20%

Percent of time the values would occur.

**Dependencies**

- Selecting MIL-HDBK-310 or MIL-STD-210C for the **Specification** parameter enables this parameter.
- Extreme values, 5%, and 20% are available only when Envelope is selected for **Atmospheric model type**.
- 1% and 10% are always available.

**Programmatic Use**

**Block Parameter:** profile\_percent

**Type:** character vector

**Values:** 'Extreme values' | '1%' | '5%' | '10%' | '20%'

**Default:** '1%'

**Altitude of extreme value – Geometric altitude**

5 km (16404 ft) (default) | 10 km (32808 ft) | 20 km (65617 ft) | 30 km (98425 ft) | 40 km (131234 ft)

Geometric altitude at which the extreme values occur, specified as 5 km (16404 ft), 10 km (32808 ft), 20 km (65617 ft), 30 km (98425 ft), or 40 km (131234 ft).

**Dependencies**

This parameter appears if the **Atmospheric model type** is set to Profile.

**Programmatic Use**

**Block Parameter:** profile\_alt

**Type:** character vector

**Values:** 5 km (16404 ft) | 10 km (32808 ft) | 20 km (65617 ft) | 30 km (98425 ft) | 40 km (131234 ft)

**Default:** 40 km (131234 ft)

### Action for out-of-range input – Out-of-range block behavior

Warning (default) | None | Error

Out-of-range block behavior, specified as follows.

| Action          | Description   |
|-----------------|---|
| None            | No action.  |
| Warning         | Warning in the MATLAB Command Window, model simulation continues. |
| Error (default) | MATLAB returns an exception, model simulation stops.              |

#### Programmatic Use

**Block Parameter:** action

**Type:** character vector

**Values:** 'None' | 'Warning' | 'Error'

**Default:** 'Warning'

## References

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

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

## See Also

CIRA-86 Atmosphere Model | ISA Atmosphere Model | Non-Standard Day 210C | Non-Standard Day 310

### Topics

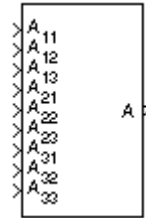
“Explore the NASA HL-20 Model” on page 1-5

**Introduced before R2006a**

# Create 3x3 Matrix

Create 3-by-3 matrix from nine input values

**Library:** Aerospace Blockset / Utilities / Math Operations



## Description

The Create 3x3 Matrix block creates a 3-by-3 matrix from nine input values where each input corresponds to an element of the matrix.

The output matrix has the form of

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix}$$

## Ports

### Input

#### **A<sub>11</sub>** — First row, first column of matrix

matrix element

First row, first column of the matrix, specified as a matrix element.

Example: 1

Data Types: double

#### **A<sub>12</sub>** — First row, second column of matrix

matrix element

First row, second column of the matrix, specified as a matrix element.

Example: 2

Data Types: double

#### **A<sub>13</sub>** — First row, third column of matrix

matrix element

First row, third column of the matrix, specified as a matrix element.

Example: 3

Data Types: double

**A<sub>21</sub> — Second row, first column of matrix**

matrix element

Second row, first column of the matrix, specified as a matrix element.

Example: 4

Data Types: double

**A<sub>22</sub> — Second row, second column of matrix**

matrix element

Second row, second column of the matrix, specified as a matrix element.

Example: 5

Data Types: double

**A<sub>23</sub> — Second row, third column of matrix**

matrix element

Second row, third column of the matrix, specified as a matrix element.

Example: 6

Data Types: double

**A<sub>31</sub> — Third row, first column of matrix**

matrix element

Third row, first column of the matrix, specified as a matrix element.

Example: 7

Data Types: double

**A<sub>32</sub> — Third row, second column of matrix**

matrix element

Third row, second column of the matrix, specified as a matrix element.

Example: 8

Data Types: double

**A<sub>33</sub> — Third row, third column of matrix**

matrix element

Third row, third column of the matrix, specified as a matrix element.

Example: 9

Data Types: double

**Output****A — Matrix**

3-by-3 matrix

Matrix, output as a 3-by-3 matrix.

Data Types: double

## **Extended Capabilities**

### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

### **See Also**

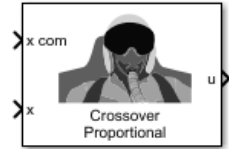
[Adjoint of 3x3 Matrix](#) | [Determinant of 3x3 Matrix](#) | [Invert 3x3 Matrix](#) | [Symmetric Inertia Tensor](#)

**Introduced before R2006a**

## Crossover Pilot Model

Represent crossover pilot model

**Library:** Aerospace Blockset / Pilot Models



### Description

The Crossover Pilot Model block represents the pilot model described in *Mathematical Models of Human Pilot Behavior* [1]). This pilot model is a single input, single output (SISO) model that represents some aspects of human behavior when controlling aircraft.

The Crossover Pilot Model takes into account the combined dynamics of the human pilot and the aircraft, using the form described in “Algorithms” on page 4-158 around the crossover frequency.

This block has nonlinear behavior. If you want to linearize the block (for example, with one of the `linmod` functions), you might need to change the Pade approximation order. The Crossover Pilot Model block implementation incorporates the Transport Delay block with the **Pade order (for linearization)** parameter set to 2 by default. To change this value, use the `set_param` function, for example:

```
set_param(gcb, 'pade', '3')
```

When modeling human pilot models, use this block for more accuracy than that provided by the Tustin Pilot Model block. This block is also less accurate than the Precision Pilot Model block.

### Ports

#### Input

##### **x com — Signal command**

scalar

Signal command that the pilot model controls, specified as a scalar.

Data Types: `double`

##### **x — Signal**

scalar

Signal that the pilot model controls, specified as a scalar.

Data Types: `double`

#### Output

##### **u — Aircraft command**

scalar

Aircraft command, returned as a scalar.



Data Types: double

## Parameters

### Type of control – Dynamics control

Proportional (default) | Rate or velocity | Spiral divergence | Second order - Short period | Acceleration(\*) | Roll attitude(\*) | Unstable short period(\*) | Second order - Phugoid(\*)

Dynamics control that you want the pilot to have over the aircraft. This table lists the options and associated dynamics.

| Option (Controlled Element Transfer Function) | Transfer Function of Controlled Element ( $Y_c$ )             | Transfer Function of Pilot ( $Y_p$ ) | $Y_c Y_p$  | Notes  |
|---|---|--------------------------------------|--|--|
| Proportional                                  | $K_c$   | $\frac{K_p e^{-\tau s}}{s}$          | $\frac{K_c K_p e^{-\tau s}}{s}$  |  |
| Rate or velocity                              | $\frac{K_c}{s}$   | $K_p e^{-\tau s}$                    | $\frac{K_c K_p e^{-\tau s}}{s}$  |  |
| Spiral divergence                             | $\frac{K_c}{T_I s - 1}$                                       | $K_p e^{-\tau s}$                    | $\frac{K_c K_p e^{-\tau s}}{(T_I s - 1)}$  |  |
| Second order - Short period                   | $\frac{K_c \omega_n^2}{s^2 + 2\zeta \omega_n s + \omega_n^2}$ | $\frac{K_p e^{-\tau s}}{T_I s + 1}$  | $\frac{K_c \omega_n^2}{s^2 + 2\zeta \omega_n s + \omega_n^2} \times \frac{K_p e^{-\tau s}}{T_I s + 1}$ | Short period, with $\omega_n > 1/\tau$                               |
| Acceleration (*)                              | $\frac{K_c}{s^2}$   | $K_p s e^{-\tau s}$                  | $\frac{K_c K_p e^{-\tau s}}{s}$  |  |
| Roll attitude (*)                             | $\frac{K_c}{s(T_I s + 1)}$                                    | $K_p (T_L s + 1) e^{-\tau s}$        | $\frac{K_c K_p e^{-\tau s}}{s}$  | With $T_L \approx T_I$   |
| Unstable short period(*)                      | $\frac{K_c}{(T_{I1} s + 1)(T_{I2} s - 1)}$                    | $K_p (T_L s + 1) e^{-\tau s}$        | $\frac{K_c K_p e^{-\tau s}}{(T_{I2} s - 1)}$   | With $T_L \approx T_{I1}$  |
| Second order - Phugoid(*)                     | $\frac{K_c \omega_n^2}{s^2 + 2\zeta \omega_n s + \omega_n^2}$ | $K_p (T_L s + 1) e^{-\tau s}$        | $\frac{K_c K_p \omega_n^2 e^{-\tau s}}{s}$   | Phugoid, with $\omega_n \ll 1/\tau$ , $1/T_L \approx \zeta \omega_n$ |

\* Indicates that the pilot model includes a Derivative block, which produces a numerical derivative. For this reason, do not send discontinuous (such as a step) or noisy input to the Crossover Pilot Model block. Such inputs can cause large outputs that might render the system unstable.

This table defines the variables used in the list of control options.

| Variable   | Description                        |
|------------|------------------------------------|
| $K_c$      | Aircraft gain.                     |
| $K_p$      | Pilot gain.                        |
| $\tau$     | Pilot time delay.                  |
| $T_I$      | Lag constant.                      |
| $T_L$      | Lead constant.                     |
| $\zeta$    | Damping ratio for the aircraft.    |
| $\omega_n$ | Natural frequency of the aircraft. |

### Dependencies

The Crossover Pilot Model parameters are enabled and disabled according to the **Type of control** options. The **Calculated value**, **Controlled element gain**, **Pilot gain**, **Crossover frequency (rad/s)**, and **Pilot time delay(s)** parameters are always enabled.

### Programmatic Use

**Block Parameter:** sw\_popup

**Type:** character vector

**Values:** 'Proportion' | 'Rate or velocity' | 'Spiral divergence' | 'Second order - Short period' | 'Acceleration(\*)' | 'Roll attitude(\*)' | 'Unstable short period(\*)' | 'Second order - Phugoid(\*)'

**Default:** 'Proportion'

### Calculated value — Crossover frequency or pilot gain

Crossover frequency (default) | Pilot gain

Crossover frequency or pilot gain value you want the block to calculate:

- Crossover frequency — The block calculates the crossover frequency value. The parameter value is disabled.
- Pilot gain — The block calculates the pilot gain value. The parameter value is disabled.

### Programmatic Use

**Block Parameter:** freq\_gain\_popup

**Type:** character vector

**Values:** 'Crossover frequency' | 'Pilot gain'

**Default:** 'Crossover frequency'

### Controlled element gain — Controlled element gain

1 (default) | scalar

Controlled element gain, specified as a double scalar.

### Programmatic Use

**Block Parameter:** Kc

**Type:** character vector

**Values:** double scalar

**Default:** '1'

### Pilot gain — Pilot gain

3 (default) | scalar

Pilot gain, specified as a double scalar.

**Dependencies**

To enable this parameter, set **Calculated value** to Pilot gain.

**Programmatic Use**

**Block Parameter:** Kp

**Type:** character vector

**Values:** double scalar

**Default:** '3'

**Crossover frequency (rad/s) – Crossover frequency**

3 (default) | scalar in the range of 1 and 10

Crossover frequency value, specified as double scalar, in rad/s. The value must be in the range between 1 and 10.

**Dependencies**

To enable this parameter, set **Calculated value** to Crossover frequency.

**Programmatic Use**

**Block Parameter:** omega\_c

**Type:** character vector

**Values:** double scalar

**Default:** '3'

**Pilot time delay(s) – Pilot time delay**

0.1 (default) | scalar

Total pilot time delay, specified as a double scalar, in seconds. This value typically ranges from 0.1 s to 0.2 s.

**Programmatic Use**

**Block Parameter:** time\_delay

**Type:** character vector

**Values:** double scalar

**Default:** '0.1'

**Pilot lead constant – Pilot lead constant**

1 (default) | scalar

Pilot lead constant, specified as a double scalar.

**Dependencies**

To enable this parameter, set **Type of control** to one of the following options:

- Roll attitude (\*)
- Unstable short period (\*)
- Second order - Phygoid(\*)

**Programmatic Use**

**Block Parameter:** T

**Type:** character vector

**Values:** double scalar

**Default:** '1'

### Pilot lag constant — Pilot lag constant

5 (default) | scalar

Pilot lag constant, specified as a double scalar.

#### Dependencies

To enable this parameter, set **Type of control** to Second order - Short period.

#### Programmatic Use

**Block Parameter:** Ti

**Type:** character vector

**Values:** double scalar

**Default:** '5'

## Algorithms

The Crossover Model takes into account the combined dynamics of the human pilot and the aircraft, using the following form around the crossover frequency:

$$Y_p Y_c = \frac{\omega_c e^{-\tau s}}{s},$$

Where:

| Variable   | Description  |
|------------|--|
| $Y_p$      | Pilot transfer function.                                       |
| $Y_c$      | Aircraft transfer function.                                    |
| $\omega_c$ | Crossover frequency.   |
| $\tau$     | Transport delay time caused by the pilot neuromuscular system. |

If the dynamics of the aircraft ( $Y_c$ ) change,  $Y_p$  changes correspondingly.

---

**Note** This block is valid only around the crossover frequency. It is not valid for discrete inputs such as a step.

---

## References

- [1] McRuer, D. T., Krendel, E., *Mathematical Models of Human Pilot Behavior*. Advisory Group on Aerospace Research and Development AGARDograph 188, Jan. 1974.

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

## **See Also**

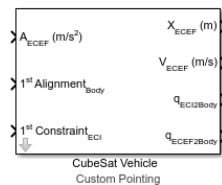
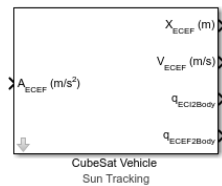
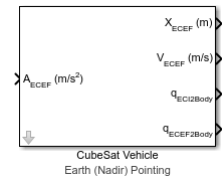
Precision Pilot Model | Transport Delay | Tustin Pilot Model | `linmod`

**Introduced in R2012b**

# CubeSat Vehicle

Model CubeSat vehicle

**Library:** Aerospace Blockset / Aerospace Blockset CubeSat Simulation Library



## Description

The CubeSat Vehicle block models CubeSat vehicles to provide a high level mission planning/rapid prototyping option to quickly model and propagate satellite orbits. To accommodate constellation planning workflows, you can also use these blocks multiple times in a model. Specify this information for the vehicle:

- Initial orbital state
- Attitude control (pointing) mode

The Aerospace Blockset CubeSat Simulation Library contains three versions of the CubeSat Vehicle block preconfigured for common attitude control modes:

- Earth (Nadir) Pointing — Primary alignment vector points towards the center of the Earth
- Sun Tracking — Primary alignment vector points toward the Sun
- Custom Pointing — Custom alignment and constraint vectors

To access the Aerospace Blockset CubeSat Simulation Library, type `asbCubeSatBlockLib` in the MATLAB Command Window.

## Limitations

The CubeSat Vehicle block is available only through the Add-On Explorer.

## Ports

### Input

#### $A_{ECEF}$ (m/s<sup>2</sup>) — Vehicle accelerations

vector of size 3

Vehicle gravity accelerations (including gravity) used for orbit propagation, specified as a vector of size 3, in m/s<sup>2</sup>.

Data Types: single | double

#### 1st Alignment<sub>Body</sub> — Primary alignment vector

three-element vector

Primary alignment vector, in the Body frame, to align with primary constraint vector.

Data Types: double

#### 1st Constraint<sub>ECI</sub> — Primary constraint vector

three-element vector

Primary constraint vector specifying the direction in which to align the primary alignment vector.

### Dependencies

This port is not available when **Pointing mode** is set to Earth (Nadir) Pointing or Sun Tracking, which have implied primary constraint vectors.

Data Types: double

#### 1st Alignment<sub>Body</sub> — Primary alignment vector

three-element vector

Primary alignment vector, in the Body frame, to align with primary constraint vector.

Data Types: double

#### 1st Constraint<sub>ECI</sub> — Primary constraint vector

three-element vector

Primary constraint vector specifying the direction in which to align the primary alignment vector.

### Dependencies

- The direction depends on the **Constraint coordinate system**.
- This port is not available when **Pointing mode** is set to Earth (Nadir) Pointing or Sun Tracking, which have implied primary constraint vectors.

Data Types: double

#### 2nd Alignment<sub>Body</sub> — Secondary alignment vector

three-element vector

Secondary alignment vector, in the Body frame, to align with secondary constraint vector.

Data Types: double

**2nd Constraint<sub>ECI</sub> — Secondary constraint vector**

three-element vector

Secondary constraint vector specifying the direction in which to align the secondary alignment vector.

**Dependencies**

The direction depends on the **Constraint coordinate system**.

Data Types: double

**Output****X<sub>ECEF</sub> — CubeSat position**

three-element vector

Earth-centered Earth-fixed CubeSat position components, specified as a 3-by-1 array.

Data Types: double

**V<sub>ECEF</sub> — Velocity components**

3-by-1 array

Earth-centered Earth-fixed velocity components, specified as a 3-by-1 array.

Data Types: double

**q<sub>ECI2Body</sub> — Quaternion rotation**

4-by-1 array

Quaternion rotation from Earth-centered inertial frame to Body frame.

Data Types: double

**q<sub>ECEF2Body</sub> — Quaternion array**

4-by-1 array

Quaternion rotation from Earth-centered Earth-fixed frame to Body frame.

Data Types: double

**Parameters****Start date [Julian date] — Initial start date of simulation**

2458488 (default) | Julian date

Initial start date of simulation. The block defines initial conditions using this date.

---

**Tip** To calculate the Julian date, use the `juliandate` function.

---

**Programmatic Use****Block Parameter:** `sim_t0`**Type:** character vector**Values:** Julian date



**Default:** '2458488'

## CubeSat Orbit

### Input method — Initial vehicle

Keplerian Orbital Elements (default) | ECI Position and Velocity | ECEF Position and Velocity | Geodetic LatLonAlt and Velocity in NED

Initial vehicle position and velocity input method.

### Dependencies

Selecting the Keplerian Orbital Elements input method enables these parameters:

- **Epoch of ECI frame [Julian date]**
- **Semi-major axis [m]**
- **Eccentricity**
- **Inclination [deg]**
- **Right ascension of the ascending node [deg]**
- **Argument of periapsis [deg]**
- **True anomaly [deg]**
- **True longitude [deg] (circular equatorial)**
- **Argument of latitude [deg] (circular inclined)**
- **Longitude of periapsis [deg] (elliptical equatorial)**

Selecting the ECI Position and Velocity input method enables these parameters:

- **Epoch of ECI frame [Julian date]**
- **ECI position vector [m]**
- **ECI velocity vector [m/s]**

Selecting the ECEF Position and Velocity input method enables these parameters:

- **ECEF position vector [m]**
- **ECEF velocity vector [m/s]**

Selecting the Geodetic LatLonAlt and Velocity in NED input method enables these parameters:

- **Geodetic latitude, longitude, altitude [deg, deg, m]**
- **NED velocity vector [m/s]**

### Programmatic Use

**Block Parameter:** method

**Type:** character vector

**Values:** 'Keplerian Orbital Elements' | 'ECI Position and Velocity' | 'ECEF Position and Velocity' | 'Geodetic LatLonAlt and Velocity in NED'

**Default:** 'Keplerian Orbital Elements'

### Epoch of ECI frame [Julian date] — Epoch of ECI frame

2451545 (default) | Julian date

Epoch of ECI frame, specified as a Julian date.

---

**Tip** To calculate the Julian date for a particular date, use the `juliandate` function.

---

**Programmatic Use**

**Block Parameter:** epoch

**Type:** character vector

**Values:** Julian date format

**Default:** '2451545'

**Semi-major axis [m] — CubeSat semi-major axis**

6878137 (default) | axis in meters

CubeSat semi-major axis (half of the longest orbit diameter), specified in m.

**Programmatic Use**

**Block Parameter:** a

**Type:** character vector

**Values:** scalar

**Default:** '6878137'

**Eccentricity — Orbital eccentricity**

0 (default) | eccentricity greater than or equal to 0

Deviation of the CubeSat orbit from a perfect circle.

**Programmatic Use**

**Block Parameter:** ecc

**Type:** character vector

**Values:** scalar

**Default:** '0'

**Inclination [deg] — Tilt angle of CubeSat orbital plane**

0 | degrees between 0 and 180

Tilt angle of CubeSat orbital plane, specified between 0 and 180 deg.

**Programmatic Use**

**Block Parameter:** incl

**Type:** character vector

**Values:** scalar

**Default:** '0'

**Right ascension of the ascending node [deg] — Angular distance in equatorial plane**

0 (default) | degrees between 0 and 360

Angular distance in equatorial plane from x-axis to location of the ascending node (point at which the satellite crosses the equator from south to north), specified between 0 and 360 deg.

**Programmatic Use**

**Block Parameter:** omega

**Type:** character vector

**Values:** scalar

**Default:** '0'

**Argument of periapsis [deg] – Angle from CubeSat body ascending node to periapsis**

0 (default) | degrees between 0 and 360

Angle from the CubeSat body ascending node to the periapsis (closest point of orbit to Earth), specified between 0 and 360 deg.

**Programmatic Use****Block Parameter:** argp**Type:** character vector**Values:** scalar**Default:** '0'**True anomaly [deg] – Angle between periapsis and current position of CubeSat**

0 (default) | degrees between 0 and 360

Angle between the periapsis (closest point of orbit to Earth) and the current position of CubeSat, specified between 0 and 360 deg.

**Programmatic Use****Block Parameter:** nu**Type:** character vector**Values:** scalar**Default:** '0'**True longitude [deg] (circular equatorial) – Angle between x-axis of periapsis and position of CubeSat vector**

0 (default) | degrees between 0 and 360

Angle between x-axis of periapsis and position of CubeSat vector, specified between 0 and 360 deg.

**Programmatic Use****Block Parameter:** trueLon**Type:** character vector**Values:** scalar**Default:** '0'**Argument of latitude [deg] (circular inclined) – Angle between ascending node and satellite position vector**

0 (default) | degrees between 0 and 360

Angle between ascending node and satellite position vector, specified between 0 and 360 deg.

**Programmatic Use****Block Parameter:** argLat**Type:** character vector**Values:** scalar**Default:** '0'**Longitude of periapsis [deg] (elliptical equatorial) – Angle between x-axis of periapsis and eccentricity vector**

0 (default) | degrees between 0 and 360

Angle between the x-axis of the periapsis and the eccentricity vector, specified between 0 and 360 deg.

**Programmatic Use****Block Parameter:** lonper**Type:** character vector**Values:** scalar**Default:** '0'**ECI position vector [m] — Cartesian position vector**

[0 0 0] (default) | vector

Cartesian position vector of satellite in ECI coordinate frame at **Start Date**.**Programmatic Use****Block Parameter:** r\_eci**Type:** character vector**Values:** scalar**Default:** '[0 0 0]'**ECI velocity vector [m/s] — Cartesian velocity vector**

[0 0 0] (default) | velocity vector

Cartesian velocity vector of satellite in ECI coordinate frame at **Start Date**.**Programmatic Use****Block Parameter:** v\_eci**Type:** character vector**Values:** scalar**Default:** '[0 0 0]'**ECEF position vector [m] — Cartesian position vector**

[0 0 0] (default) | vector

Cartesian position vector of satellite in ECEF coordinate frame at **Start Date**.**Programmatic Use****Block Parameter:** r\_ecef**Type:** character vector**Values:** scalar**Default:** '[0 0 0]'**ECEF velocity vector [m/s] — Cartesian velocity vector**

[0 0 0] (default) | velocity vector

Cartesian velocity vector of satellite in ECEF coordinate frame at **Start Date**.**Programmatic Use****Block Parameter:** v\_ecef**Type:** character vector**Values:** scalar**Default:** '[0 0 0]'**Geodetic latitude, longitude, altitude [deg, deg, m] — Geodetic latitude and longitude, and altitude**

[0 0 0] (default) | velocity vector

Geodetic latitude and longitude, in deg, and altitude above WGS84 ellipsoid, in m.

**Programmatic Use****Block Parameter:** `lla`**Type:** character vector**Values:** scalar**Default:** `'[0 0 0]'`**NED velocity vector [m/s] — Body velocity**`[0 0 0]` (default) | velocity vector

Body velocity with respect to Earth-centered Earth-fixed (ECEF), expressed in the north-east-down (NED) coordinate frame, specified as a vector, in m/s.

**Programmatic Use****Block Parameter:** `v_ned`**Type:** character vector**Values:** scalar**Default:** `'[0 0 0]'`**CubeSat Attitude****Initial Euler angles (roll, pitch, yaw) [deg] — Initial Euler rotation angles**`[0 0 0]` (default) | vector | degrees

Initial Euler rotation angles (roll, pitch, yaw) between Body and NED coordinate frames, specified in degrees.

**Programmatic Use****Block Parameter:** `euler`**Type:** character vector**Values:** scalar**Default:** `'[0 0 0]'`**Initial body angular rates [deg/s] — Initial angular rates**`[0 0 -0.05168]` (default) | vector

Initial angular rates with respect to NED frame, expressed in Body frame, specified as a vector.

**Programmatic Use****Block Parameter:** `pqr`**Type:** character vector**Values:** scalar**Default:** `'[0 0 0]'`**Pointing mode — CubeSat vehicle pointing mode**

Earth (Nadir) Pointing (default) | Sun Tracking | Custom Pointing | Standby (Off)

CubeSat vehicle pointing mode, specified as Earth (Nadir) Pointing, Sun Tracking, or Custom Pointing. The CubeSat vehicle uses the pointing mode for precise attitude control. For no attitude control, select Standby (Off).

**Programmatic Use****Block Parameter:** `pointingMode`**Type:** character vector**Values:** `'Earth (Nadir) Pointing' | 'Sun Tracking' | 'Custom Pointing' | 'Standby (Off)'`

**Default:** 'Earth (Nadir) Pointing'

**Primary alignment vector (Body wrt  $B_{CM}$ ) — Primary alignment vector**

Dialog (default) | Input port

Primary alignment vector, in Body frame, to align with primary constraint vector.

**Dependencies**

- Selecting Dialog enables a text box in which you specify the primary alignment vector. The default value is [0 0 1].
- Selecting Input port enables the 1<sup>st</sup> Alignment<sub>Body</sub> input port, at which you specify the primary alignment vector.

**Programmatic Use**

**Block Parameter:** firstAlign

**Type:** character vector

**Values:** vector

**Default:** '[0 0 1]'

**Programmatic Use**

**Block Parameter:** firstAlignExt

**Type:** character vector

**Values:** 'Input port' | 'Dialog'

**Default:** 'Dialog'

**Secondary alignment vector (Body wrt  $B_{CM}$ ) — Secondary alignment vector**

Dialog (default) | Input port

Secondary alignment vector, in Body frame, to align with secondary constraint vector.

**Dependencies**

- Selecting Dialog enables a text box in which you specify the secondary alignment vector. The default value is [0 1 0].
- Selecting Input port enables the 2<sup>nd</sup> Alignment<sub>Body</sub> input port, at which you specify the secondary alignment vector.

**Programmatic Use**

**Block Parameter:** secondAlign

**Type:** character vector

**Values:** vector

**Default:** '[0 1 0]'

**Programmatic Use**

**Block Parameter:** secondAlignExt

**Type:** character vector

**Values:** 'Input port' | 'Dialog'

**Default:** 'Dialog'

**Constraint coordinate system — Constraint coordinate system**

ECI Axes (default) | ECEF Axes | NED Axes | Body-Fixed Axes

Constraint coordinate system, specified as ECI Axes, ECEF Axes, NED Axes, or Body-Fixed Axes.

**Programmatic Use****Block Parameter:** constraintCoord**Type:** character vector**Values:** 'ECI Axes' | 'ECEF Axes' | 'NED Axes' | 'Body-Fixed Axes'**Default:** 'ECI Axes'**Primary constraint vector (wrt  $B_{CM}$ ) — Primary constraint vector**

Dialog (default) | Input port

Primary constraint vector, in the Body frame, to align with the primary alignment vector.

**Dependencies**

- This parameter is disabled when **Pointing mode** is Earth (Nadir) Pointing or Sun Tracking.
- Selecting Dialog enables a text box in which you specify the primary constraint vector. The default value is [1 0 0].
- Selecting Input port enables the 1<sup>st</sup> constraint<sub>Body</sub> input port, at which you specify the primary constraint vector.

**Programmatic Use****Block Parameter:** firstRef**Type:** character vector**Values:** vector**Default:** '[1 0 0]'**Programmatic Use****Block Parameter:** firstRefExt**Type:** character vector**Values:** 'Input port' | 'Dialog'**Default:** 'Dialog'**Secondary constraint vector (wrt  $B_{CM}$ ) — Secondary constraint vector**

Dialog (default) | Input port

Secondary constraint vector, in the Body frame, to align with the secondary alignment vector.

**Dependencies**

- Selecting Dialog enables a text box in which you specify the secondary constraint vector. The default value is [0 0 1].
- Selecting Input port enables the 2<sup>nd</sup> constraint<sub>Body</sub> input port, at which you specify the secondary constraint vector.

**Programmatic Use****Block Parameter:** secondRef**Type:** character vector**Values:** vector**Default:** '[0 0 1]'**Programmatic Use****Block Parameter:** secondRefExt**Type:** character vector**Values:** 'Input port' | 'Dialog'**Default:** 'Dialog'

## Earth Orientation Parameters

### Difference between TAI and UTC [sec] — Difference between International Atomic Time and Coordinated Universal Time

0 (default) | scalar

Difference between International Atomic Time (TAI) and Coordinated Universal Time (UTC), specified as a scalar, in sec.

#### Programmatic Use

**Block Parameter:** dAT

**Type:** character vector

**Values:** scalar

**Default:** '0'

### Difference between UTC and UT1 [sec] — Difference between Coordinated Universal Time and Universal Time

0 (default) | scalar

Scalar value specifying the difference between UTC and UT1, specified as a scalar, in sec.

#### Programmatic Use

**Block Parameter:** dUT1

**Type:** character vector

**Values:** scalar

**Default:** '0'

### Polar displacement [deg deg] — Polar displacements due to motion of Earth crust

[0 0] (default) | 1-by-2 array

Polar displacements due to the motion of Earth crust along the x- and y-axis, specified as a 1-by-2 array, in deg.

#### Programmatic Use

**Block Parameter:** pm

**Type:** character vector

**Values:** scalar

**Default:** '[0 0]'

### CIP adjustment [deg deg] — Adjustments to location of Celestial Intermediate Pole

[0 0] (default) | 1-by-2 array

Adjustments to the location of the Celestial Intermediate Pole (CIP) along the x- and y-axis, specified as a 1-by-2 array, in deg.

#### Programmatic Use

**Block Parameter:** dCIP

**Type:** character vector

**Values:** scalar

**Default:** '[0 0]'

### Excess length of day [sec] — Excess length of day

0 (default) | scalar

Excess length of day (difference between astronomically determined duration of day and 86400 seconds), specified as a scalar, in International System of Units (SI) sec.



**Programmatic Use****Block Parameter:** lod**Type:** character vector**Values:** scalar**Default:** '0'**Mission Analysis****Analysis run time source — Source of run time for mission analysis live script**

Dialog (default) | Model Stop Time

Source of run time for mission analysis live script, specified as:

- Dialog — Defined in **Run time** parameter.
- Model Stop Time — Defined in model configuration parameter **Stop Time**.

**Programmatic Use****Block Parameter:** missionRTSource**Type:** character vector**Values:** 'Dialog' | 'Model StopTime'**Default:** 'Dialog'**Run time [sec] — Run time for mission analysis live script**

6\*60\*60 (default) | scalar

Run time for mission analysis live script, specified as a scalar.

**Programmatic Use****Block Parameter:** missionRT**Type:** character vector**Values:** scalar**Default:** '6\*60\*60'**Ground station geodetic latitude, longitude [deg, deg] — Ground station location**

[42, -71] (default) | vector

Ground station location, specified as a vector, in geodetic latitude and longitude in deg, deg.

**Programmatic Use****Block Parameter:** missionGS**Type:** character vector**Values:** vector**Default:** '[42, -71]'**Run TOI analysis — Enable time of interest mission analysis**

on (default) | off

Select this check box to enable time of interest analysis in mission analysis.live script

**Programmatic Use****Block Parameter:** missionTOICheck**Type:** character vector**Values:** 'on' | 'off'**Default:** 'on'

**Time of interest [Julian date] – Time of interest for mission analysis live script**

[] (default) | Julian date

Time of interest mission analysis, specified as a Julian date. To use the simulation start date, enter an empty array ( []).

---

**Tip** To calculate the Julian date, use the `juliandate` function.

---

**Programmatic Use****Block Parameter:** missionTOI**Type:** character vector**Values:** Julian date**Default:** ' [] '**Camera field-of-view (FOV) half angle (deg) – Half angle of field of view**

55 (default) | [] | scalar

Half angle of field of view for nadir on-pointed camera. To exclude from analysis, enter an empty array ( []).

**Programmatic Use****Block Parameter:** missionEta**Type:** character vector**Values:** ' [] ' | scalar**Default:** '55'**Live script file name – File name for mission analysis live script report**

blank entry (default) | live script file name

File name for mission analysis live script report, generated as a live script. To create a default mission analysis report with the format `CubeSatMissionReport_currentdate.mlx`, leave the parameter blank. To create a live script of the mission analysis report, click the **Create Live Script Report** button.

**Dependencies**

To create the live script with the specified file name, click the **Create Live Script Report** button. If this parameter is blank, the block creates a live script with a default file name.

**Programmatic Use****Block Parameter:** missionName**Type:** character vector**Values:** blank entry | file name**Default:** blank entry**Create Live Script Report – Analyze mission and create live script report**

button

To analyze mission and create report in live script format, click this button. To create a default mission analysis report with the format `CubeSatMissionReport_currentdate.mlx`, leave the parameter blank. To create a live script of the mission analysis report, click the **Create Live Script Report** button.

## Dependencies

To create the live script with the file name specified in **Live script file name**, click the **Create Live Script Report** button. If **Live script file name** is blank, the block creates a live script with a default file name.

## References

[1] Wertz, James R, David F. Everett, and Jeffery J. Puschell. Space Mission Engineering: The New Smad. Hawthorne, CA: Microcosm Press, 2011. Print.

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

## See Also

asbCubeSatMissionAnalysis | ecef2eci | eci2ecef | generateAccessTable | greenwichsrt | ijk2keplerian | juliandate | keplerian2ijk

## Topics

“Getting Started with CubeSats” on page 2-49

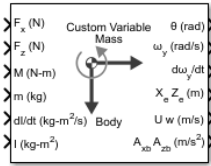
“Model and Simulate CubeSats” on page 2-51

“Analyze CubeSat Mission Geometry” on page 2-58

## Introduced in R2019a

## Custom Variable Mass 3DOF (Body Axes)

Implement three-degrees-of-freedom equations of motion of custom variable mass with respect to body axes

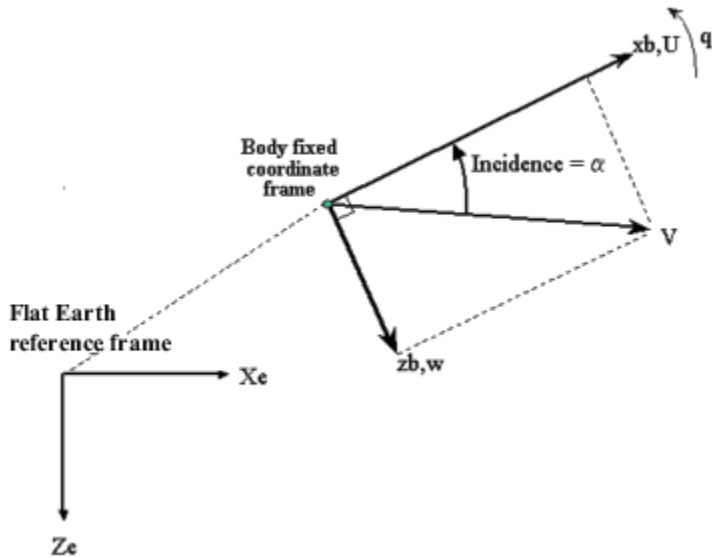


## Library

Equations of Motion/3DOF

## Description

The Custom Variable Mass 3DOF (Body Axes) block considers the rotation in the vertical plane of a body-fixed coordinate frame about a flat Earth reference frame.



The equations of motion are

$$A_{xb} = \dot{u} = \frac{F_x}{m} - \frac{\dot{m}U_{re_b}}{m} - qw - g\sin\theta$$

$$V_{re_b} = [U_{re} \ W_{re}]_b$$

$$A_{yb} = \dot{w} = \frac{F_z}{m} - \frac{\dot{m}W_{re_b}}{m} + qu + g\cos\theta$$

$$\dot{q} = \frac{M - \dot{I}_{yy}q}{I_{yy}}$$

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

$$A_{be} = \begin{bmatrix} A_{xe} \\ A_{ze} \end{bmatrix} = \frac{F_b - \dot{m}U_{re_b}}{m} - \bar{g}$$

$$A_{bb} = \begin{bmatrix} A_{xb} \\ A_{yb} \end{bmatrix}$$

where the applied forces are assumed to act at the center of gravity of the body.  $U_{re_b}$  and  $W_{re_b}$  are the relative velocities of the mass flow ( $\dot{m}$ ) being added to or ejected from the body in body-fixed axes.

## Parameters

### Main

### Units

Specifies the input and output units:

| Units                      | Forces | Moment       | Acceleration              | Velocity          | Position | Mass     | Inertia                |
|----------------------------|--------|--------------|---------------------------|-------------------|----------|----------|------------------------|
| Metric (MKS)               | Newton | Newton meter | Meters per second squared | Meters per second | Meters   | Kilogram | Kilogram meter squared |
| English (Velocity in ft/s) | Pound  | Foot pound   | Feet per second squared   | Feet per second   | Feet     | Slug     | Slug foot squared      |
| English (Velocity in kts)  | Pound  | Foot pound   | Feet per second squared   | Knots             | Feet     | Slug     | Slug foot squared      |

### Mass Type

Select the type of mass to use:

|                 |  |
|-----------------|--|
| Fixed           | Mass is constant throughout the simulation.                |
| Simple Variable | Mass and inertia vary linearly as a function of mass rate. |
| Custom Variable | Mass and inertia variations are customizable.              |

The Custom Variable selection conforms to the previously described equations of motion.

### Initial velocity

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

**Initial body attitude**

A scalar value for the initial pitch attitude of the body, ( $\theta_0$ ).

**Initial incidence**

A scalar value for the initial angle between the velocity vector and the body, ( $\alpha_0$ ).

**Initial body rotation rate**

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

**Initial position (x,z)**

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

**Gravity Source**

Specify source of gravity:

|          |  |
|----------|--|
| External | Variable gravity input to block                                  |
| Internal | Constant gravity specified in <b>Acceleration due to gravity</b> |

**Acceleration due to gravity**

A scalar value for the acceleration due to gravity used if internal gravity source is selected. If gravity is to be neglected in the simulation, this value can be set to 0. This parameter appears if you set **Gravity source** to Internal.

**Include mass flow relative velocity**

Select this check box to add a mass flow relative velocity port. This is the relative velocity at which the mass is accreted or ablated.

**Include inertial acceleration**

Select this check box to enable an additional output port for the accelerations in body-fixed axes with respect to the inertial frame. You typically connect this signal to the accelerometer.

**State Attributes**

Assign unique name to each state. You can use state names instead of block paths during linearization.

- To assign a name to a single state, enter a unique name between quotes, for example, 'velocity'.
- To assign names to multiple states, enter a comma-delimited list surrounded by braces, for example, {'a', 'b', 'c'}. Each name must be unique.
- If a parameter is empty (' '), no name assignment occurs.
- The state names apply only to the selected block with the name parameter.
- The number of states must divide evenly among the number of state names.
- You can specify fewer names than states, but you cannot specify more names than states.

For example, you can specify two names in a system with four states. The first name applies to the first two states and the second name to the last two states.

- To assign state names with a variable in the MATLAB workspace, enter the variable without quotes. A variable can be a character vector, cell array, or structure.

**Velocity: e.g., {'u', 'w'}**

Specify velocity state names.

Default value is ''.

**Pitch attitude: e.g., 'theta'**

Specify pitch attitude state name.

Default value is ''.

**Position: e.g., {'Xe', 'Ze'}**

Specify position state names.

Default value is ''.

**Pitch angular rate: e.g., 'q'**

Specify pitch angular rate state name.

Default value is ''.

**Inputs and Outputs**

| Input                | Dimension Type     | Description  |
|----------------------|--------------------|--|
| First                | Scalar             | Contains the force acting along the body $x$ -axis, ( $F_x$ ).   |
| Second               | Scalar             | Contains the force acting along the body $z$ -axis, ( $F_z$ ).   |
| Third                | Scalar             | Contains the applied pitch moment, ( $M$ ).  |
| Fourth<br>(Optional) | Vector             | Contains the rate of change of mass, ( $\dot{m}$ ) (positive if accreted, negative if ablated).                  |
| Fifth                | Scalar             | Contains the mass, ( $m$ ).  |
| Sixth                | Scalar             | Contains the rate of change of inertia tensor matrix, ( $\dot{I}_{yy}$ ).  |
| Seventh              | Scalar             | Contains the inertia tensor matrix, ( $I_{yy}$ ).  |
| Eighth<br>(Optional) | Scalar             | Contains the gravity in the selected units.  |
| Ninth<br>(Optional)  | Two-element vector | Contains one or more relative velocities at which the mass is accreted to or ablated from the body in body axes. |

| Output | Dimension Type     | Description  |
|--------|--------------------|--|
| First  | Scalar             | Contains the pitch attitude, within $\pm\pi$ , in radians ( $\theta$ ).                              |
| Second | Scalar             | Contains the pitch angular rate, in radians per second ( $q$ ).                                      |
| Third  | Scalar             | Contains the pitch angular acceleration, in radians per second squared ( $\dot{q}$ ).                |
| Fourth | Two-element vector | Contains the location of the body, in the flat Earth reference frame, ( $X_e, Z_e$ ).                |
| Fifth  | Two-element vector | Contains the velocity of the body resolved into the body-fixed coordinate frame, ( $u, w$ ).         |
| Sixth  | Two-element vector | Contains the acceleration of the body resolved into the body-fixed coordinate frame, ( $A_x, A_z$ ). |

| <b>Output</b>      | <b>Dimension Type</b> | <b>Description</b>   |
|--------------------|-----------------------|--|
| Seventh (Optional) | Two-element vector    | Contains the accelerations in body-fixed axes with respect to inertial frame (flat Earth). You typically connect this signal to the accelerometer. |

## **See Also**

3DOF (Body Axes)

Incidence & Airspeed

Simple Variable Mass 3DOF (Body Axes)

## **Extended Capabilities**

### **C/C++ Code Generation**

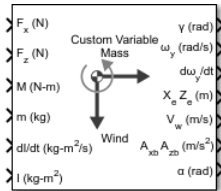
Generate C and C++ code using Simulink® Coder™.

**Introduced in R2006a**



# Custom Variable Mass 3DOF (Wind Axes)

Implement three-degrees-of-freedom equations of motion of custom variable mass with respect to wind axes

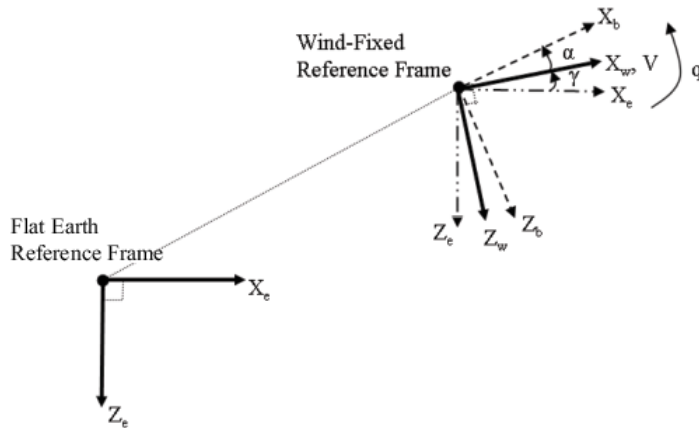


## Library

Equations of Motion/3DOF

## Description

The Custom Variable Mass 3DOF (Wind Axes) block considers the rotation in the vertical plane of a wind-fixed coordinate frame about a flat Earth reference frame.



The equations of motion are

$$\dot{V} = \frac{F_{xwind}}{m} - \frac{\dot{m}Vre_{xwind}}{m} - g\sin\gamma$$

$$A_{be} = \begin{bmatrix} A_{xc} \\ A_{zc} \end{bmatrix} = DCM_{wb} \left[ \frac{F_w - \dot{m}V_{rew}}{m} - \bar{g} \right]$$

$$A_{bb} = \begin{bmatrix} A_{xb} \\ A_{zb} \end{bmatrix} = DCM_{wb} \left[ \frac{F_w - \dot{m}V_{rew}}{m} - g - \bar{\omega}_w \times \bar{V}_w \right]$$

$$\dot{\alpha} = \frac{F_{zwind}}{mV} + q + \frac{g}{V}\cos\gamma - \frac{\dot{m}Vre_{zwind}}{mV}$$

$$\dot{q} = \dot{\theta} = \frac{M_{ybody} - \dot{I}_{yy}q}{I_{yy}}$$

$$\dot{\gamma} = q - \dot{\alpha}$$

where the applied forces are assumed to act at the center of gravity of the body.  $Vre_w$  is the relative velocity in the wind axes at which the mass flow ( $\dot{m}$ ) is ejected or added to the body in wind axes.

## Parameters

### Main

### Units

Specifies the input and output units:

| Units                      | Forces | Moment       | Acceleration              | Velocity          | Position | Mass     | Inertia                |
|----------------------------|--------|--------------|---------------------------|-------------------|----------|----------|------------------------|
| Metric (MKS)               | Newton | Newton meter | Meters per second squared | Meters per second | Meters   | Kilogram | Kilogram meter squared |
| English (Velocity in ft/s) | Pound  | Foot pound   | Feet per second squared   | Feet per second   | Feet     | Slug     | Slug foot squared      |
| English (Velocity in kts)  | Pound  | Foot pound   | Feet per second squared   | Knots             | Feet     | Slug     | Slug foot squared      |

### Mass Type

Select the type of mass to use:

|                 |  |
|-----------------|--|
| Fixed           | Mass is constant throughout the simulation.                |
| Simple Variable | Mass and inertia vary linearly as a function of mass rate. |
| Custom Variable | Mass and inertia variations are customizable.              |

The Custom Variable selection conforms to the previously described equations of motion.

### Initial airspeed

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

**Initial flight path angle**

A scalar value for the initial pitch attitude of the body, ( $\gamma_0$ ).

**Initial incidence**

A scalar value for the initial angle between the velocity vector and the body, ( $\alpha_0$ ).

**Initial body rotation rate**

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

**Initial position (x,z)**

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

**Gravity Source**

Specify source of gravity:

|          |  |
|----------|--|
| External | Variable gravity input to block                                  |
| Internal | Constant gravity specified in <b>Acceleration due to gravity</b> |

**Acceleration due to gravity**

A scalar value for the acceleration due to gravity used if internal gravity source is selected. If gravity is to be neglected in the simulation, this value can be set to 0. This parameter appears if you set **Gravity source** to Internal.

**Include mass flow relative velocity**

Select this check box to add a mass flow relative velocity port. This is the relative velocity at which the mass is accreted or ablated.

**Include inertial acceleration**

Select this check box to enable an additional output port for the accelerations in body-fixed axes with respect to the inertial frame. You typically connect this signal to the accelerometer.

**State Attributes**

Assign unique name to each state. You can use state names instead of block paths during linearization.

- To assign a name to a single state, enter a unique name between quotes, for example, 'velocity'.
- To assign names to multiple states, enter a comma-delimited list surrounded by braces, for example, {'a', 'b', 'c'}. Each name must be unique.
- If a parameter is empty (' '), no name assignment occurs.
- The state names apply only to the selected block with the name parameter.
- The number of states must divide evenly among the number of state names.
- You can specify fewer names than states, but you cannot specify more names than states.

For example, you can specify two names in a system with four states. The first name applies to the first two states and the second name to the last two states.

- To assign state names with a variable in the MATLAB workspace, enter the variable without quotes. A variable can be a character vector, cell array, or structure.

**Velocity: e.g., 'V'**

Specify velocity state name.

Default value is ''.

**Incidence angle: e.g., 'alpha'**

Specify incidence angle state name.

Default value is ''.

**Flight path angle: e.g., 'gamma'**

Specify flight path angle state name.

Default value is ''.

**Body rotation rate: e.g., 'q'**

Specify body rotation rates state name.

Default value is ''.

**Position: e.g., {'Xe', 'Ze'}**

Specify position state names.

Default value is ''.

**Inputs and Outputs**

| Input                | Dimension Type     | Description  |
|----------------------|--------------------|--|
| First                |                    | Contains the force acting along the wind x-axis, ( $F_x$ ).  |
| Second               |                    | Contains the force acting along the wind z-axis, ( $F_z$ ).  |
| Third                |                    | Contains the applied pitch moment in body axes, ( $M$ ).   |
| Fourth<br>(Optional) | Vector             | Contains one or more rates of change of mass, ( $\dot{m}$ ) (positive if accreted, negative if ablated).         |
| Fifth                |                    | Contains the mass, ( $m$ ).  |
| Sixth                |                    | Contains the rate of change of inertia tensor matrix, ( $\dot{I}_{yy}$ ).  |
| Seventh              |                    | Contains the inertia tensor matrix, ( $I_{yy}$ ).  |
| Eighth<br>(Optional) |                    | Contains the gravity in the selected units.  |
| Ninth<br>(Optional)  | Two-element vector | Contains one or more relative velocities at which the mass is accreted to or ablated from the body in wind axes. |

| Output | Dimension Type     | Description  |
|--------|--------------------|--|
| First  |                    | Contains the flight path angle, within $\pm\pi$ , in radians ( $\gamma$ ).                 |
| Second |                    | Contains the pitch angular rate, in radians per second ( $\omega_y$ ).                     |
| Third  |                    | Contains the pitch angular acceleration, in radians per second squared ( $d\omega_y/dt$ ). |
| Fourth | Two-element vector | Contains the location of the body, in the flat Earth reference frame, ( $X_e, Z_e$ ).      |

| Output           | Dimension Type     | Description  |
|------------------|--------------------|--|
| Fifth            | Two-element vector | Contains the velocity of the body resolved into the wind-fixed coordinate frame, $(V, 0)$ .  |
| Sixth            | Two-element vector | Contains the acceleration of the body resolved into the body-fixed coordinate frame, $(A_x, A_z)$ .  |
| Seventh          | Scalar             | Contains the angle of attack, $(\alpha)$ .   |
| Eight (Optional) | Two-element vector | Contains the accelerations in body-fixed axes with respect to inertial frame (flat Earth). You typically connect this signal to the accelerometer. |

## Reference

Stevens, B. L., and F. L. Lewis, *Aircraft Control and Simulation*, John Wiley & Sons, New York, 1992.

## See Also

3DOF (Body Axes)

3DOF (Wind Axes)

4th Order Point Mass (Longitudinal)

Custom Variable Mass 3DOF (Body Axes)

Simple Variable Mass 3DOF (Body Axes)

Simple Variable Mass 3DOF (Wind Axes)

## Extended Capabilities

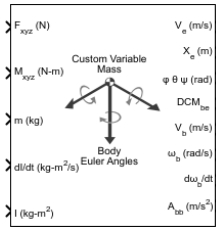
### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

### Introduced in R2006a

## Custom Variable Mass 6DOF (Euler Angles)

Implement Euler angle representation of six-degrees-of-freedom equations of motion of custom variable mass

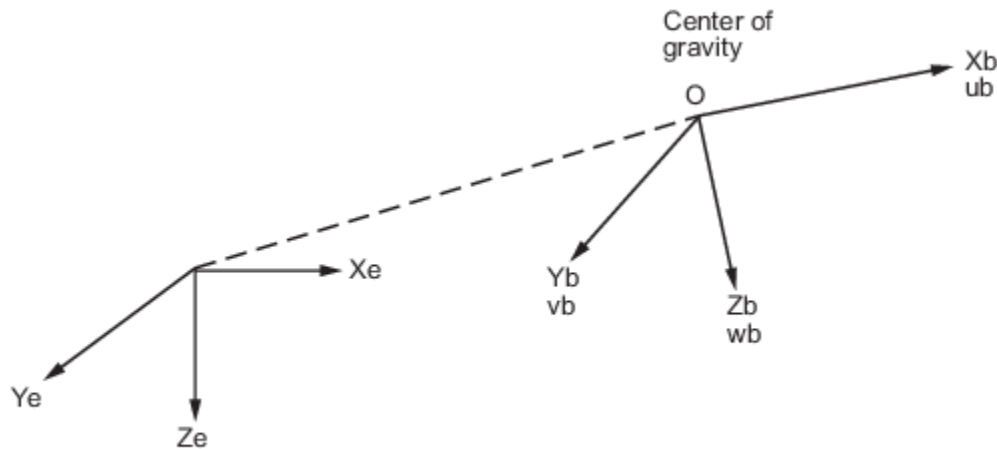


### Library

Equations of Motion/6DOF

### Description

The Custom Variable Mass 6DOF (Euler Angles) block considers the rotation of a body-fixed coordinate frame ( $X_b, Y_b, Z_b$ ) about a flat Earth 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 flat Earth reference frame is considered inertial, an excellent approximation that allows the forces due to the Earth's motion relative to the "fixed stars" to be neglected.



Flat Earth 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.  $V_{re_b}$  is the relative velocity in the body axes at which the mass flow ( $\dot{m}$ ) is ejected or added to the body-fixed axes.

$$\begin{aligned}\bar{F}_b &= \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} = m(\dot{\bar{V}}_b + \bar{\omega} \times \bar{V}_b) + \dot{m}\bar{V}_{reb} \\ A_{be} &= \frac{\bar{F}_b - \dot{m}\bar{V}_{reb}}{m} \\ A_{bb} &= \begin{bmatrix} \dot{u}_b \\ \dot{v}_b \\ \dot{w}_b \end{bmatrix} = \frac{\bar{F}_b - \dot{m}\bar{V}_{reb}}{m} - \bar{\omega} \times \bar{V}_b \\ \bar{V}_b &= \begin{bmatrix} u_b \\ v_b \\ w_b \end{bmatrix}, \bar{\omega} = \begin{bmatrix} p \\ q \\ r \end{bmatrix}\end{aligned}$$

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  $O$ .

$$\begin{aligned}\bar{M}_B &= \begin{bmatrix} L \\ M \\ N \end{bmatrix} = I\dot{\bar{\omega}} + \bar{\omega} \times (I\bar{\omega}) + \dot{I}\bar{\omega} \\ I &= \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{yx} & I_{yy} & -I_{yz} \\ -I_{zx} & -I_{zy} & I_{zz} \end{bmatrix} \\ \dot{I} &= \begin{bmatrix} \dot{I}_{xx} & -\dot{I}_{xy} & -\dot{I}_{xz} \\ -\dot{I}_{yx} & \dot{I}_{yy} & -\dot{I}_{yz} \\ -\dot{I}_{zx} & -\dot{I}_{zy} & \dot{I}_{zz} \end{bmatrix}\end{aligned}$$

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.

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

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

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

For more information on aerospace coordinate systems, see "About Aerospace Coordinate Systems" on page 2-8.

## Parameters

### Main

#### Units

Specifies the input and output units:

| Units                      | Forces | Moment       | Acceleration              | Velocity          | Position | Mass     | Inertia                |
|----------------------------|--------|--------------|---------------------------|-------------------|----------|----------|------------------------|
| Metric (MKS)               | Newton | Newton meter | Meters per second squared | Meters per second | Meters   | Kilogram | Kilogram meter squared |
| English (Velocity in ft/s) | Pound  | Foot pound   | Feet per second squared   | Feet per second   | Feet     | Slug     | Slug foot squared      |
| English (Velocity in kts)  | Pound  | Foot pound   | Feet per second squared   | Knots             | Feet     | Slug     | Slug foot squared      |

#### Mass Type

Select the type of mass to use:

|                 |  |
|-----------------|--|
| Fixed           | Mass is constant throughout the simulation.                |
| Simple Variable | Mass and inertia vary linearly as a function of mass rate. |
| Custom Variable | Mass and inertia variations are customizable.              |

The Custom Variable selection conforms to the previously described equations of motion.

#### Representation

Select the representation to use:

|              |  |
|--------------|--|
| Euler Angles | Use Euler angles within equations of motion. |
| Quaternion   | Use quaternions within equations of motion.  |

The Euler Angles selection conforms to the previously described equations of motion.

#### Initial position in inertial axes

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

#### Initial velocity in body axes

The three-element vector for the initial velocity in the body-fixed coordinate frame.

#### 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.

#### Include mass flow relative velocity

Select this check box to add a mass flow relative velocity port. This is the relative velocity at which the mass is accreted or ablated.



**Include inertial acceleration**

Select this check box to enable an additional output port for the accelerations in body-fixed axes with respect to the inertial frame. You typically connect this signal to the accelerometer.

**State Attributes**

Assign unique name to each state. You can use state names instead of block paths during linearization.

- To assign a name to a single state, enter a unique name between quotes, for example, 'velocity'.
- To assign names to multiple states, enter a comma-delimited list surrounded by braces, for example, {'a', 'b', 'c'}. Each name must be unique.
- If a parameter is empty (' '), no name assignment occurs.
- The state names apply only to the selected block with the name parameter.
- The number of states must divide evenly among the number of state names.
- You can specify fewer names than states, but you cannot specify more names than states.

For example, you can specify two names in a system with four states. The first name applies to the first two states and the second name to the last two states.

- To assign state names with a variable in the MATLAB workspace, enter the variable without quotes. A variable can be a character vector, cell array, or structure.

**Position: e.g., {'Xe', 'Ye', 'Ze'}**

Specify position state names.

Default value is ''.

**Velocity: e.g., {'U', 'v', 'w'}**

Specify velocity state names.

Default value is ''.

**Euler rotation angles: e.g., {'phi', 'theta', 'psi'}**

Specify Euler rotation angles state names. This parameter appears if the **Representation** parameter is set to Euler Angles.

Default value is ''.

**Body rotation rates: e.g., {'p', 'q', 'r'}**

Specify body rotation rate state names.

Default value is ''.

**Inputs and Outputs**

| Input  | Dimension Type | Description                         |
|--------|----------------|-------------------------------------|
| First  | Vector         | Contains the three applied forces.  |
| Second | Vector         | Contains the three applied moments. |

| Input                 | Dimension Type          | Description  |
|-----------------------|-------------------------|--|
| Third<br>(Optional)   | Vector                  | Contains one or more rates of change of mass (positive if accreted, negative if ablated).                              |
| Fourth                | Scalar                  | Contains the mass.   |
| Fifth                 | 3-by-3 matrix           | Contains the rate of change of inertia tensor matrix.  |
| Sixth                 | 3-by-3 matrix           | Contains the inertia tensor matrix.  |
| Seventh<br>(Optional) | Three-element<br>vector | Contains one or more relative velocities at which the mass is accreted to or ablated from the body in body-fixed axes. |

| Output              | Dimension Type       | Description  |
|---------------------|----------------------|--|
| First               | Three-element vector | Contains the velocity in the flat Earth reference frame.   |
| Second              | Three-element vector | Contains the position in the flat Earth reference frame.   |
| Third               | Three-element vector | Contains the Euler rotation angles [roll, pitch, yaw], within $\pm\pi$ , in radians.   |
| Fourth              | 3-by-3 matrix        | Contains the coordinate transformation from flat Earth axes to body-fixed axes.  |
| Fifth               | Three-element vector | Contains the velocity in the body-fixed frame.   |
| Sixth               | Three-element vector | Contains the angular rates in body-fixed axes, in radians per second.  |
| Seventh             | Three-element vector | Contains the angular accelerations in body-fixed axes, in radians per second squared.  |
| Eight               | Three-element vector | Contains the accelerations in body-fixed axes with respect to body frame.  |
| Ninth<br>(Optional) | Three-element vector | Contains the accelerations in body-fixed axes with respect to inertial frame (flat Earth). You typically connect this signal to the accelerometer. |

## Assumptions and Limitations

The block assumes that the applied forces are acting at the center of gravity of the body.

## Reference

Stevens, Brian, and Frank Lewis, *Aircraft Control and Simulation*, Second Edition, John Wiley & Sons, 2003.

Zipfel, Peter H., *Modeling and Simulation of Aerospace Vehicle Dynamics*. Second Edition, AIAA Education Series, 2007.

## See Also

6DOF (Euler Angles)

6DOF (Quaternion)

6DOF ECEF (Quaternion)

6DOF Wind (Quaternion)

6DOF Wind (Wind Angles)

6th Order Point Mass (Coordinated Flight)

Custom Variable Mass 6DOF (Quaternion)

Custom Variable Mass 6DOF ECEF (Quaternion)

Custom Variable Mass 6DOF Wind (Quaternion)

Custom Variable Mass 6DOF Wind (Wind Angles)

Simple Variable Mass 6DOF (Euler Angles)

Simple Variable Mass 6DOF (Quaternion)

Simple Variable Mass 6DOF ECEF (Quaternion)

Simple Variable Mass 6DOF Wind (Quaternion)

Simple Variable Mass 6DOF Wind (Wind Angles)

“About Aerospace Coordinate Systems” on page 2-8

## **Extended Capabilities**

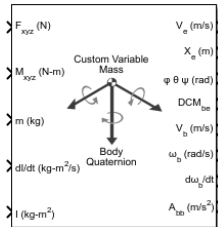
### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**Introduced in R2006a**

## Custom Variable Mass 6DOF (Quaternion)

Implement quaternion representation of six-degrees-of-freedom equations of motion of custom variable mass with respect to body axes



### Library

Equations of Motion/6DOF

### Description

For a description of the coordinate system and the translational dynamics, see the block description for the Custom Variable Mass 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  $\varepsilon$  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.

$$\begin{bmatrix} \dot{q}_0 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 0 & -p & -q & -r \\ p & 0 & r & -q \\ q & -r & 0 & p \\ r & q & -p & 0 \end{bmatrix} \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} + K\varepsilon \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix}$$

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

Aerospace Blockset uses quaternions that are defined using the scalar-first convention.

### Parameters

#### Main

#### Units

Specifies the input and output units:

| Units                      | Forces | Moment       | Acceleration              | Velocity          | Position | Mass     | Inertia                |
|----------------------------|--------|--------------|---------------------------|-------------------|----------|----------|------------------------|
| Metric (MKS)               | Newton | Newton meter | Meters per second squared | Meters per second | Meters   | Kilogram | Kilogram meter squared |
| English (Velocity in ft/s) | Pound  | Foot pound   | Feet per second squared   | Feet per second   | Feet     | Slug     | Slug foot squared      |
| English (Velocity in kts)  | Pound  | Foot pound   | Feet per second squared   | Knots             | Feet     | Slug     | Slug foot squared      |

### Mass Type

Select the type of mass to use:

|                 |  |
|-----------------|--|
| Fixed           | Mass is constant throughout the simulation.                |
| Simple Variable | Mass and inertia vary linearly as a function of mass rate. |
| Custom Variable | Mass and inertia variations are customizable.              |

The Custom Variable selection conforms to the previously described equations of motion.

### Representation

Select the representation to use:

|              |  |
|--------------|--|
| Euler Angles | Use Euler angles within equations of motion. |
| Quaternion   | Use quaternions within equations of motion.  |

The Quaternion selection conforms to the previously described equations of motion.

### Initial position in inertial axes

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

### Initial velocity in body axes

The three-element vector for the initial velocity in the body-fixed coordinate frame.

### 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.

### Gain for quaternion normalization

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

### Include mass flow relative velocity

Select this check box to add a mass flow relative velocity port. This is the relative velocity at which the mass is accreted or ablated.

### Include inertial acceleration

Select this check box to enable an additional output port for the accelerations in body-fixed axes with respect to the inertial frame. You typically connect this signal to the accelerometer.

## State Attributes

Assign unique name to each state. You can use state names instead of block paths during linearization.

- To assign a name to a single state, enter a unique name between quotes, for example, 'velocity'.
- To assign names to multiple states, enter a comma-delimited list surrounded by braces, for example, {'a', 'b', 'c'}. Each name must be unique.
- If a parameter is empty (' '), no name assignment occurs.
- The state names apply only to the selected block with the name parameter.
- The number of states must divide evenly among the number of state names.
- You can specify fewer names than states, but you cannot specify more names than states.

For example, you can specify two names in a system with four states. The first name applies to the first two states and the second name to the last two states.

- To assign state names with a variable in the MATLAB workspace, enter the variable without quotes. A variable can be a character vector, cell array, or structure.

### Position: e.g., {'Xe', 'Ye', 'Ze'}

Specify position state names.

Default value is ''.

### Velocity: e.g., {'U', 'v', 'w'}

Specify velocity state names.

Default value is ''.

### Quaternion vector: e.g., {'qr', 'qi', 'qj', 'qk'}

Specify quaternion vector state names. This parameter appears if the **Representation** parameter is set to Quaternion.

Default value is ''.

### Body rotation rates: e.g., {'p', 'q', 'r'}

Specify body rotation rate state names.

Default value is ''.

## Inputs and Outputs

| Input               | Dimension Type | Description   |
|---------------------|----------------|---|
| First               | Vector         | Contains the three applied forces.  |
| Second              | Vector         | Contains the three applied moments.   |
| Third<br>(Optional) | Vector         | Contains one or more rates of change of mass (positive if accreted, negative if ablated). |
| Fourth              | Scalar         | Contains the mass.  |
| Fifth               | 3-by-3 matrix  | Contains rate of change of inertia tensor matrix.   |

| Input              | Dimension Type       | Description  |
|--------------------|----------------------|--|
| Sixth              | 3-by-3 matrix        | Contains the inertia tensor matrix.  |
| Seventh (Optional) | Three-element vector | Contains one or more relative velocities at which the mass is accreted to or ablated from the body in body-fixed axes.                             |
| Output             | Dimension Type       | Description  |
| First              | Three-element vector | Contains the velocity in the flat Earth reference frame.   |
| Second             | Three-element vector | Contains the position in the flat Earth reference frame.   |
| Third              | Three-element vector | Contains the Euler rotation angles [roll, pitch, yaw], in radians.   |
| Fourth             | 3-by-3 matrix        | Contains the coordinate transformation from flat Earth axes to body-fixed axes.  |
| Fifth              | Three-element vector | Contains the velocity in the body-fixed frame.   |
| Sixth              | Three-element vector | Contains the angular rates in body-fixed axes, in radians per second.  |
| Seventh            | Three-element vector | Contains the angular accelerations in body-fixed axes, in radians per second squared.  |
| Eighth             | Three-element vector | Contains the accelerations in body-fixed axes with respect to body frame.  |
| Ninth (Optional)   | Three-element vector | Contains the accelerations in body-fixed axes with respect to inertial frame (flat Earth). You typically connect this signal to the accelerometer. |

## Assumptions and Limitations

The block assumes that the applied forces are acting at the center of gravity of the body.

## Reference

Stevens, Brian, and Frank Lewis, *Aircraft Control and Simulation*, Second Edition, John Wiley & Sons, 2003.

Zipfel, Peter H., *Modeling and Simulation of Aerospace Vehicle Dynamics*. Second Edition, AIAA Education Series, 2007.

## See Also

6DOF (Euler Angles)

6DOF (Quaternion)

6DOF ECEF (Quaternion)

6DOF Wind (Quaternion)

6DOF Wind (Wind Angles)

6th Order Point Mass (Coordinated Flight)

Custom Variable Mass 6DOF (Euler Angles)

Custom Variable Mass 6DOF ECEF (Quaternion)

Custom Variable Mass 6DOF Wind (Quaternion)

Custom Variable Mass 6DOF Wind (Wind Angles)

Simple Variable Mass 6DOF (Euler Angles)

Simple Variable Mass 6DOF (Quaternion)

Simple Variable Mass 6DOF ECEF (Quaternion)

Simple Variable Mass 6DOF Wind (Quaternion)

Simple Variable Mass 6DOF Wind (Wind Angles)

“About Aerospace Coordinate Systems” on page 2-8

## **Extended Capabilities**

### **C/C++ Code Generation**

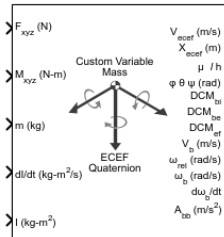
Generate C and C++ code using Simulink® Coder™.

**Introduced in R2006a**



# Custom Variable Mass 6DOF ECEF (Quaternion)

Implement quaternion representation of six-degrees-of-freedom equations of motion of custom variable mass in Earth-centered Earth-fixed (ECEF) coordinates

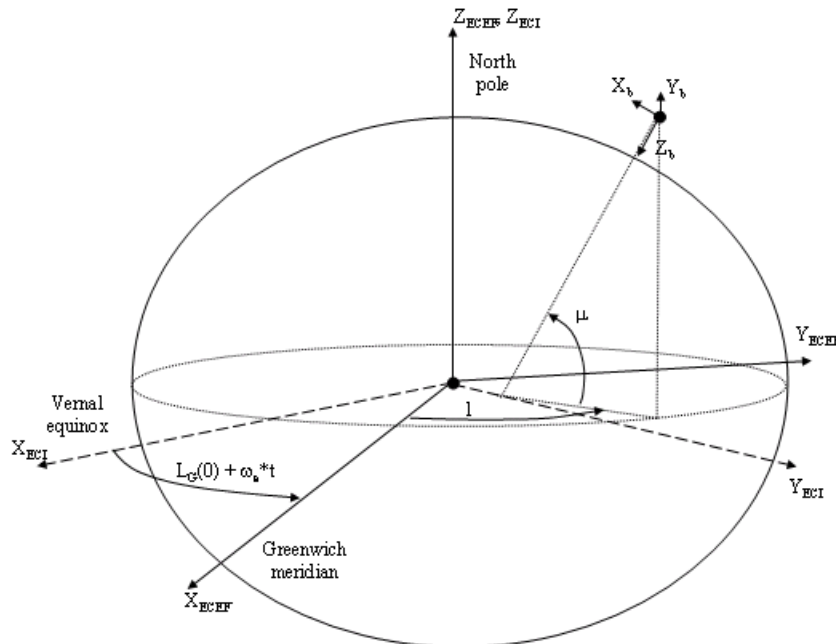


## Library

Equations of Motion/6DOF

## Description

The Custom Variable Mass 6DOF ECEF (Quaternion) block considers the rotation of a Earth-centered Earth-fixed (ECEF) coordinate frame ( $X_{ECEF}, Y_{ECEF}, Z_{ECEF}$ ) about an Earth-centered inertial (ECI) reference frame ( $X_{ECI}, Y_{ECI}, Z_{ECI}$ ). The origin of the ECEF coordinate frame is the center of the Earth, additionally the body of interest is assumed to be rigid, an assumption that eliminates the need to consider the forces acting between individual elements of mass. The representation of the rotation of ECEF frame from ECI frame is simplified to consider only the constant rotation of the ellipsoid Earth ( $\omega_e$ ) including an initial celestial longitude ( $L_G(0)$ ). This excellent approximation allows the forces due to the Earth's complex motion relative to the "fixed stars" to be neglected.



The translational motion of the ECEF coordinate frame is given below, where the applied forces  $[F_x F_y F_z]^T$  are in the body frame.  $V_{rel}$  is the relative velocity in the wind axes at which the mass flow ( $\dot{m}$ ) is ejected or added to the body in body-fixed axes.

$$\begin{aligned} \bar{F}_b &= \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} = m \left( \dot{\bar{V}}_b + \bar{\omega}_b \times \bar{V}_b + DCM_{bf} \bar{\omega}_e \times \bar{V}_b + DCM_{bf} (\bar{\omega}_e \times (\bar{\omega}_e \times \bar{X}_f)) \right) \\ &\quad + \dot{m} (\bar{V}_{rel} + DCM_{bf} (\bar{\omega}_e \times \bar{X}_f)) \\ A_{bb} &= \begin{bmatrix} \dot{u}_b \\ \dot{v}_b \\ \dot{w}_b \end{bmatrix} = \frac{\bar{F}_b - \dot{m} (\bar{V}_{rel} + DCM_{bf} (\bar{\omega}_e \times \bar{X}_f))}{m} \\ &\quad - [\bar{\omega}_b \times \bar{V}_b + DCM_{bf} \bar{\omega}_e \times \bar{V}_b + DCM_{bf} (\bar{\omega}_e (\bar{\omega}_e \times \bar{X}_f))] \\ A_{becef} &= \frac{\bar{F}_b - \dot{m} (\bar{V}_{rel} + DCM_{bf} (\bar{\omega}_e \times \bar{X}_f))}{m} \end{aligned}$$

where the change of position in ECEF  $\dot{\bar{x}}_f$  is calculated by

$$\dot{\bar{x}}_f = DCM_{fb} \bar{V}_b$$

and the velocity of the body with respect to ECEF frame, expressed in body frame ( $\bar{V}_b$ ), angular rates of the body with respect to ECI frame, expressed in body frame ( $\bar{\omega}_b$ ). Earth rotation rate ( $\bar{\omega}_e$ ), and relative angular rates of the body with respect to north-east-down (NED) frame, expressed in body frame ( $\bar{\omega}_{rel}$ ) are defined as

$$\begin{aligned} \bar{V}_b &= \begin{bmatrix} u \\ v \\ w \end{bmatrix}, \bar{\omega}_{rel} = \begin{bmatrix} p \\ q \\ r \end{bmatrix}, \bar{\omega}_e = \begin{bmatrix} 0 \\ 0 \\ \omega_e \end{bmatrix}, \bar{\omega}_b = \bar{\omega}_{rel} + DCM_{bf} \bar{\omega}_e + DCM_{be} \bar{\omega}_{ned} \\ \bar{\omega}_{ned} &= \begin{bmatrix} \dot{I} \cos \mu \\ -\dot{\mu} \\ -\dot{I} \sin \mu \end{bmatrix} = \begin{bmatrix} V_E / (N + h) \\ -V_N / (M + h) \\ -V_E \cdot \tan \mu / (N + h) \end{bmatrix} \end{aligned}$$

The rotational dynamics of the body defined in 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.

$$\begin{aligned} \bar{M}_b &= \begin{bmatrix} L \\ M \\ N \end{bmatrix} = \bar{I} \dot{\bar{\omega}}_b + \bar{\omega}_b \times (\bar{I} \bar{\omega}_b) + \dot{I} \bar{\omega}_b \\ I &= \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{yx} & I_{yy} & -I_{yz} \\ -I_{zx} & -I_{zy} & I_{zz} \end{bmatrix} \end{aligned}$$

The rate of change of the inertia tensor is defined by the following equation.

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

The integration of the rate of change of the quaternion vector is given below.

$$\begin{bmatrix} \dot{q}_0 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} = -1/2 \begin{bmatrix} 0 & \omega_b(1) & \omega_b(2) & \omega_b(3) \\ -\omega_b(1) & 0 & -\omega_b(3) & \omega_b(2) \\ -\omega_b(2) & \omega_b(3) & 0 & -\omega_b(1) \\ -\omega_b(3) & -\omega_b(2) & \omega_b(1) & 0 \end{bmatrix} \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix}$$

Aerospace Blockset uses quaternions that are defined using the scalar-first convention.

## Parameters

### Main

### Units

Specifies the input and output units:

| Units                      | Forces | Moment       | Acceleration              | Velocity          | Position | Mass     | Inertia                |
|----------------------------|--------|--------------|---------------------------|-------------------|----------|----------|------------------------|
| Metric (MKS)               | Newton | Newton meter | Meters per second squared | Meters per second | Meters   | Kilogram | Kilogram meter squared |
| English (Velocity in ft/s) | Pound  | Foot pound   | Feet per second squared   | Feet per second   | Feet     | Slug     | Slug foot squared      |
| English (Velocity in kts)  | Pound  | Foot pound   | Feet per second squared   | Knots             | Feet     | Slug     | Slug foot squared      |

### Mass type

Select the type of mass to use:

|                 |  |
|-----------------|--|
| Fixed           | Mass is constant throughout the simulation (see 6DOF ECEF (Quaternion)).                                     |
| Simple Variable | Mass and inertia vary linearly as a function of mass rate (see Simple Variable Mass 6DOF ECEF (Quaternion)). |
| Custom Variable | Mass and inertia variations are customizable.  |

The Simple Variable selection conforms to the previously described equations of motion.

### Initial position in geodetic latitude, longitude and altitude

The three-element vector for the initial location of the body in the geodetic reference frame. Latitude and longitude values can be any value. However, latitude values of +90 and -90 may return unexpected values because of singularity at the poles.

**Initial velocity in body-axis**

The three-element vector containing the initial velocity of the body with respect to ECEF frame, expressed in body frame.

**Initial Euler orientation**

The three-element vector containing the initial Euler rotation angles [roll, pitch, yaw], in radians. Euler rotation angles are those between the body and north-east-down (NED) coordinate systems.

**Initial body rotation rates**

The three-element vector for the initial angular rates of the body with respect to NED frame, expressed in body frame, in radians per second.

**Include mass flow relative velocity**

Select this check box to add a mass flow relative velocity port. This is the relative velocity at which the mass is accreted or ablated.

**Planet model**

Specifies the planet model to use, Custom or Earth (WGS84).

**Equatorial radius of planet**

Specifies the radius of the planet at its equator. The units of the equatorial radius parameter should be the same as the units for ECEF position. This option is only available when **Planet model** is set to Custom.

**Flattening**

Specifies the flattening of the planet. This option is only available when **Planet model** is set to Custom.

**Rotational rate**

Specifies the scalar rotational rate of the planet in rad/s. This option is only available when **Planet model** is set to Custom.

**Celestial longitude of Greenwich source**

Specifies the source of Greenwich meridian's initial celestial longitude:

|          |   |
|----------|---|
| Internal | Use celestial longitude value from mask dialog.   |
| External | Use external input for celestial longitude value. |

**Celestial longitude of Greenwich**

The initial angle between Greenwich meridian and the x-axis of the ECI frame.

**Include inertial acceleration**

Select this check box to enable an additional output port for the accelerations in body-fixed axes with respect to the inertial frame. You typically connect this signal to the accelerometer.

**State Attributes**

Assign unique name to each state. Use state names instead of block paths throughout the linearization process.

- To assign a name to a single state, enter a unique name between quotes, for example, 'velocity'.
- To assign names to multiple states, enter a comma-delimited list surrounded by braces, for example, {'a', 'b', 'c'}. Each name must be unique.

- If a parameter is empty ( ' '), no name assignment occurs.
- The state names apply only to the selected block with the name parameter.
- The number of states must divide evenly among the number of state names.
- You can specify fewer names than states, but you cannot specify more names than states.

For example, you can specify two names in a system with four states. The first name applies to the first two states and the second name to the last two states.

- To assign state names with a variable in the MATLAB workspace, enter the variable without quotes. A variable can be a character vector, cell array, or structure.

**Quaternion vector: e.g., {'qr', 'qi', 'qj', 'qk'}**

Specify quaternion vector name for the state.

Default value is ' '.

**Body rotation rates: e.g., {'p', 'q', 'r'}**

Specify body rotation rate name for the state.

Default value is ' '.

**Velocity: e.g., {'U', 'v', 'w'}**

Specify velocity name for the state.

Default value is ' '.

**ECEF position: e.g., {'Xecef', 'Yecef', 'Zecef'}**

Specify the ECEF position name for the state.

Default value is ' '.

**Inertial position: e.g., {'Xeci', 'Yeci', 'Zeci'}**

Specify the inertial position name for the state.

Default value is ' '.

**Celestial longitude of Greenwich: e.g., 'LG'**

Specify the Celestial longitude of Greenwich name for the state.

Default value is ' '.

## Inputs and Outputs

| Input               | Dimension Type | Description   |
|---------------------|----------------|---|
| First               | Vector         | Contains the three applied forces in body-fixed axes.                                     |
| Second              | Vector         | Contains the three applied moments in body-fixed axes.                                    |
| Third<br>(Optional) | Vector         | Contains one or more rates of change of mass (positive if accreted, negative if ablated). |
| Fourth              | Scalar         | Contains the mass.  |
| Sixth               | 3-by-3 matrix  | Applies to the inertia tensor matrix.   |

| Input                    | Dimension Type       | Description   |
|--------------------------|----------------------|---|
| Seventh<br>(Optional)    | 1-by-1-by- $m$ array | Contains one or more relative velocities at which the mass is accreted to or ablated from the body in body-fixed axes. $m$ is three times the size of the third input vector. |
| Output                   | Dimension Type       | Description   |
| First                    | Three-element vector | Contains the velocity of the body with respect to ECEF frame, expressed in ECEF frame.  |
| Second                   | Three-element vector | Contains the position in the ECEF reference frame.  |
| Third                    | Three-element vector | Contains the position in geodetic latitude, longitude and altitude, in degrees, degrees and selected units of length respectively.  |
| Fourth                   | Three-element vector | Contains the body rotation angles [roll, pitch, yaw], in radians. Euler rotation angles are those between the body and NED coordinate systems.                                |
| Fifth                    | 3-by-3 matrix        | Applies to the coordinate transformation from ECI axes to body-fixed axes.  |
| Sixth                    | 3-by-3 matrix        | Applies to the coordinate transformation from NED axes to body-fixed axes.  |
| Seventh                  | 3-by-3 matrix        | Applies to the coordinate transformation from ECEF axes to NED axes.  |
| Eighth                   | Three-element vector | Contains the velocity of the body with respect to ECEF frame, expressed in body frame.  |
| Ninth                    | Three-element vector | Contains the relative angular rates of the body with respect to NED frame, expressed in body frame, in radians per second.  |
| Tenth                    | Three-element vector | Contains the angular rates of the body with respect to ECI frame, expressed in body frame, in radians per second.   |
| Eleventh                 | Three-element vector | Contains the angular accelerations of the body with respect to ECI frame, expressed in body frame, in radians per second squared.   |
| Twelfth                  | Three-element vector | Contains the accelerations in body-fixed axes with respect to body frame.   |
| Thirteenth<br>(Optional) | Three-element vector | Contains the accelerations in body-fixed axes with respect to ECEF frame.   |

## Assumptions and Limitations

This implementation assumes that the applied forces are acting at the center of gravity of the body.

This implementation generates a geodetic latitude that lies between  $\pm 90$  degrees, and longitude that lies between  $\pm 180$  degrees. Additionally, the MSL altitude is approximate.

The Earth is assumed to be ellipsoidal. By setting flattening to 0.0, a spherical planet can be achieved. The Earth's precession, nutation, and polar motion are neglected. The celestial longitude of

Greenwich is Greenwich Mean Sidereal Time (GMST) and provides a rough approximation to the sidereal time.

The implementation of the ECEF coordinate system assumes that the origin is at the center of the planet, the x-axis intersects the Greenwich meridian and the equator, the z-axis is the mean spin axis of the planet, positive to the north, and the y-axis completes the right-handed system.

The implementation of the ECI coordinate system assumes that the origin is at the center of the planet, the x-axis is the continuation of the line from the center of the Earth toward the vernal equinox, the z-axis points in the direction of the mean equatorial plane's north pole, positive to the north, and the y-axis completes the right-handed system.

## References

Stevens, B. L., and F. L. Lewis, *Aircraft Control and Simulation, Second Edition*, John Wiley & Sons, New York, 2003.

McFarland, Richard E., *A Standard Kinematic Model for Flight simulation at NASA-Ames*, NASA CR-2497.

“Supplement to Department of Defense World Geodetic System 1984 Technical Report: Part I - Methods, Techniques and Data Used in WGS84 Development,” DMA TR8350.2-A.

## See Also

6DOF (Euler Angles)

6DOF (Quaternion)

6DOF ECEF (Quaternion)

6DOF Wind (Quaternion)

6DOF Wind (Wind Angles)

6th Order Point Mass (Coordinated Flight)

Custom Variable Mass 6DOF (Euler Angles)

Custom Variable Mass 6DOF (Quaternion)

Custom Variable Mass 6DOF Wind (Quaternion)

Custom Variable Mass 6DOF Wind (Wind Angles)

Simple Variable Mass 6DOF (Euler Angles)

Simple Variable Mass 6DOF (Quaternion)

Simple Variable Mass 6DOF ECEF (Quaternion)

Simple Variable Mass 6DOF Wind (Quaternion)

Simple Variable Mass 6DOF Wind (Wind Angles)

“About Aerospace Coordinate Systems” on page 2-8

## **Extended Capabilities**

### **C/C++ Code Generation**

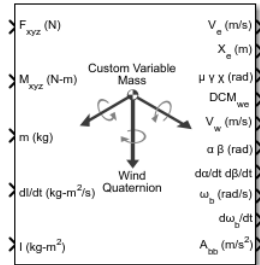
Generate C and C++ code using Simulink® Coder™.

**Introduced in R2006a**



# Custom Variable Mass 6DOF Wind (Quaternion)

Implement quaternion representation of six-degrees-of-freedom equations of motion of custom variable mass with respect to wind axes

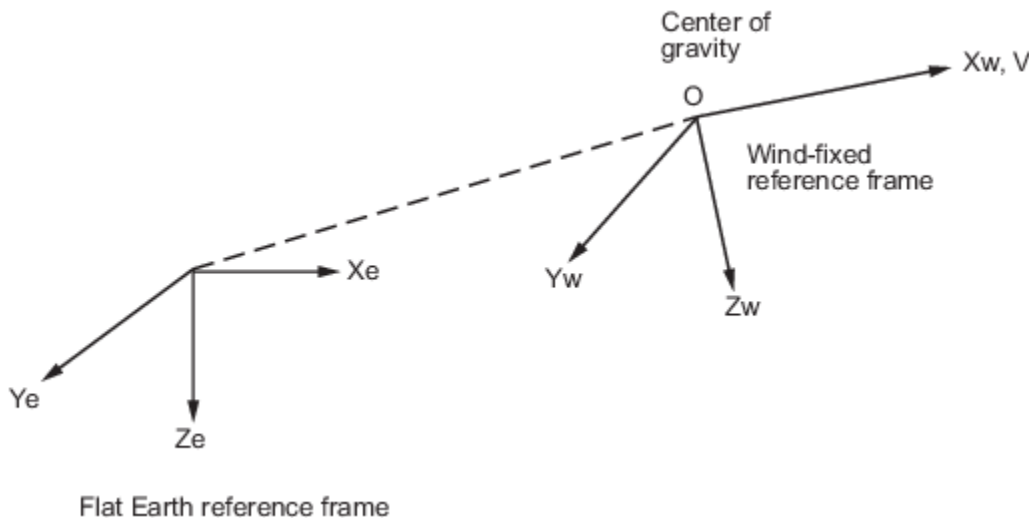


## Library

Equations of Motion/6DOF

## Description

The Custom Variable Mass 6DOF Wind (Quaternion) block considers the rotation of a wind-fixed coordinate frame ( $X_w, Y_w, Z_w$ ) about an flat Earth reference frame ( $X_e, Y_e, Z_e$ ). The origin of the wind-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 flat Earth reference frame is considered inertial, an excellent approximation that allows the forces due to the Earth's motion relative to the “fixed stars” to be neglected.



The translational motion of the wind-fixed coordinate frame is given below, where the applied forces  $[F_x, F_y, F_z]^T$  are in the wind-fixed frame.  $V_{re_w}$  is the relative velocity in the wind axes at which the mass flow ( $\dot{m}$ ) is ejected or added to the body.

$$\begin{aligned}\bar{\mathbf{F}}_w &= \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} = m(\dot{\bar{\mathbf{V}}}_w + \bar{\boldsymbol{\omega}}_w \times \bar{\mathbf{V}}_w) + \dot{m}\bar{V}_{re} \\ A_{be} &= DCM_{wb} \frac{[\bar{\mathbf{F}}_w - \dot{m}\bar{V}_{re}]}{m} \\ \bar{\mathbf{V}}_w &= \begin{bmatrix} V \\ 0 \\ 0 \end{bmatrix}, \bar{\boldsymbol{\omega}}_w = \begin{bmatrix} p_w \\ q_w \\ r_w \end{bmatrix} = DMC_{wb} \begin{bmatrix} p_b - \dot{\beta} \sin \alpha \\ q_b - \dot{\alpha} \\ r_b + \dot{\beta} \cos \alpha \end{bmatrix}, \bar{\mathbf{w}}_b = \begin{bmatrix} p_b \\ q_b \\ r_b \end{bmatrix} \\ A_{bb} &= DCM_{wb} \left[ \frac{\bar{\mathbf{F}}_w - \dot{m}\bar{V}_{re}}{m} - \bar{\boldsymbol{\omega}}_w \times \bar{\mathbf{V}}_w \right]\end{aligned}$$

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  $O$ . Inertia tensor  $I$  is much easier to define in body-fixed frame.

$$\begin{aligned}\bar{\mathbf{M}}_b &= \begin{bmatrix} L \\ M \\ N \end{bmatrix} = I\dot{\bar{\boldsymbol{\omega}}}_b + \bar{\boldsymbol{\omega}}_b \times (I\bar{\boldsymbol{\omega}}_b) + \dot{I}\bar{\boldsymbol{\omega}}_b \\ A_{bb} &= \begin{bmatrix} \dot{U}_b \\ \dot{V}_b \\ \dot{W}_b \end{bmatrix} = DCM_{wb} \left[ \frac{\bar{\mathbf{F}}_w - \dot{m}\bar{V}_{re}}{m} - \bar{\boldsymbol{\omega}}_w \times \bar{\mathbf{V}}_w \right] \\ I &= \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{yx} & I_{yy} & -I_{yz} \\ -I_{zx} & -I_{zy} & I_{zz} \end{bmatrix}\end{aligned}$$

The integration of the rate of change of the quaternion vector is given below.

$$\begin{bmatrix} \dot{q}_0 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} = -\frac{1}{2} \begin{bmatrix} 0 & p & q & r \\ -p & 0 & -r & q \\ -q & r & 0 & -p \\ -r & -q & p & 0 \end{bmatrix} \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix}$$

Aerospace Blockset uses quaternions that are defined using the scalar-first convention.

## Parameters

### Main

### Units

Specifies the input and output units:

| Units                      | Forces | Moment       | Acceleration              | Velocity          | Position | Mass     | Inertia                |
|----------------------------|--------|--------------|---------------------------|-------------------|----------|----------|------------------------|
| Metric (MKS)               | Newton | Newton meter | Meters per second squared | Meters per second | Meters   | Kilogram | Kilogram meter squared |
| English (Velocity in ft/s) | Pound  | Foot pound   | Feet per second squared   | Feet per second   | Feet     | Slug     | Slug foot squared      |
| English (Velocity in kts)  | Pound  | Foot pound   | Feet per second squared   | Knots             | Feet     | Slug     | Slug foot squared      |

**Mass Type**

Select the type of mass to use:

|                 |  |
|-----------------|--|
| Fixed           | Mass is constant throughout the simulation.                |
| Simple Variable | Mass and inertia vary linearly as a function of mass rate. |
| Custom Variable | Mass and inertia variations are customizable.              |

The Custom Variable selection conforms to the previously described equations of motion.

**Representation**

Select the representation to use:

|             |   |
|-------------|---|
| Wind Angles | Use wind angles within equations of motion. |
| Quaternion  | Use quaternions within equations of motion. |

The Quaternion selection conforms to the previously described equations of motion.

**Initial position in inertial axes**

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

**Initial airspeed, sideslip angle, and angle of attack**

The three-element vector containing the initial airspeed, initial sideslip angle and initial angle of attack.

**Initial wind orientation**

The three-element vector containing the initial wind angles [bank, flight path, and heading], in radians.

**Initial body rotation rates**

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

**Include mass flow relative velocity**

Select this check box to add a mass flow relative velocity port. This is the relative velocity at which the mass is accreted or ablated.

**Include inertial acceleration**

Select this check box to enable an additional output port for the accelerations in body-fixed axes with respect to the inertial frame. You typically connect this signal to the accelerometer.

### State Attributes

Assign unique name to each state. Use state names instead of block paths throughout the linearization process.

- To assign a name to a single state, enter a unique name between quotes, for example, 'velocity'.
- To assign names to multiple states, enter a comma-delimited list surrounded by braces, for example, {'a', 'b', 'c'}. Each name must be unique.
- If a parameter is empty (' '), no name assignment occurs.
- The state names apply only to the selected block with the name parameter.
- The number of states must divide evenly among the number of state names.
- You can specify fewer names than states, but you cannot specify more names than states.

For example, you can specify two names in a system with four states. The first name applies to the first two states and the second name to the last two states.

- To assign state names with a variable in the MATLAB workspace, enter the variable without quotes. A variable can be a character vector, cell array, or structure.

#### **Position: e.g., {'Xe', 'Ye', 'Ze'}**

Specify position name for the state.

Default value is ' '.

#### **Velocity: e.g., 'V'**

Specify velocity name for the state.

Default value is ' '.

#### **Incidence angle: e.g., 'alpha'**

Specify incidence angle name for the state.

Default value is ' '.

#### **Sideslip angle: e.g., 'beta'**

Specify sideslip angle name for the state.

Default value is ' '.

#### **Quaternion vector: e.g., {'qr', 'qi', 'qj', 'qk'}**

Specify quaternion vector name for the state.

Default value is ' '.

#### **Body rotation rates: e.g., {'p', 'q', 'r'}**

Specify body rotation rates name for the state.

Default value is ' '.

## Inputs and Outputs

| Input                 | Dimension Type       | Description  |
|-----------------------|----------------------|--|
| First                 | Vector               | Contains the three applied forces in wind-fixed axes.  |
| Second                | Vector               | Contains the three applied moments in body-fixed axes.   |
| Third<br>(Optional)   | Vector               | Contains one or more rates of change of mass, positive if accreted, negative if ablated.                         |
| Fourth                | Scalar               | Contains the mass of the body  |
| Fifth                 | 3-by-3 matrix        | Applies to the rate of change of inertia tensor matrix in body-fixed axes.                                       |
| Sixth                 | 3-by-3 matrix        | Applies to the inertia tensor matrix in body-fixed axes.   |
| Seventh<br>(Optional) | Three-element vector | Contains one or more relative velocities at which the mass is accreted to or ablated from the body in wind axes. |

| Output                 | Dimension Type       | Description  |
|------------------------|----------------------|--|
| First                  | Three-element vector | Contains the velocity in the flat Earth reference frame  |
| Second                 | Three-element vector | Contains the position in the flat Earth reference frame.   |
| Third                  | Three-element vector | Contains the wind rotation angles [bank, flight path, heading], in radians.  |
| Fourth                 | 3-by-3 matrix        | Applies to the coordinate transformation from flat Earth axes to wind-fixed axes.  |
| Fifth                  | Three-element vector | Contains to the velocity in the wind-fixed frame.  |
| Sixth                  | Two-element vector   | Contains the angle of attack and sideslip angle, in radians.   |
| Seventh                | Two-element vector   | Contains the rate of change of angle of attack and rate of change of sideslip angle, in radians per second.  |
| Eighth                 | Three-element vector | Contains the angular rates in body-fixed axes, in radians per second.  |
| Ninth                  | Three-element vector | Contains the angular accelerations in body-fixed axes, in radians per second squared.  |
| Tenth                  | Three-element vector | Contains the accelerations in body-fixed axes with respect to body frame.  |
| Eleventh<br>(Optional) | Three-element vector | Contains the accelerations in body-fixed axes with respect to inertial frame (flat Earth). You typically connect this signal to the accelerometer. |

## Assumptions and Limitations

The block assumes that the applied forces are acting at the center of gravity of the body.

## References

Stevens, Brian, and Frank Lewis, *Aircraft Control and Simulation*, Second Edition, John Wiley & Sons, 2003.

Zipfel, Peter H., *Modeling and Simulation of Aerospace Vehicle Dynamics*. Second Edition, AIAA Education Series, 2007.

## See Also

6DOF (Euler Angles)

6DOF (Quaternion)

6DOF ECEF (Quaternion)

6DOF Wind (Quaternion)

6DOF Wind (Wind Angles)

6th Order Point Mass (Coordinated Flight)

Custom Variable Mass 6DOF (Euler Angles)

Custom Variable Mass 6DOF (Quaternion)

Custom Variable Mass 6DOF ECEF (Quaternion)

Custom Variable Mass 6DOF Wind (Wind Angles)

Simple Variable Mass 6DOF (Euler Angles)

Simple Variable Mass 6DOF (Quaternion)

Simple Variable Mass 6DOF ECEF (Quaternion)

Simple Variable Mass 6DOF Wind (Quaternion)

Simple Variable Mass 6DOF Wind (Wind Angles)

“About Aerospace Coordinate Systems” on page 2-8

## Extended Capabilities

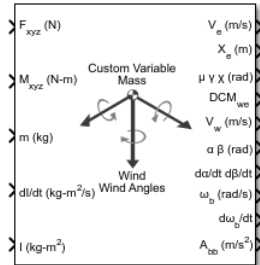
### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

### Introduced in R2006a

## Custom Variable Mass 6DOF Wind (Wind Angles)

Implement wind angle representation of six-degrees-of-freedom equations of motion of custom variable mass



### Library

Equations of Motion/6DOF

### Description

For a description of the coordinate system employed and the translational dynamics, see the block description for the Custom Variable Mass 6DOF Wind (Quaternion) block.

The relationship between the wind angles,  $[\mu \ \gamma \ \chi]^T$ , can be determined by resolving the wind rates into the wind-fixed coordinate frame.

$$\begin{bmatrix} p_w \\ q_w \\ r_w \end{bmatrix} = \begin{bmatrix} \dot{\mu} \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\mu & \sin\mu \\ 0 & -\sin\mu & \cos\mu \end{bmatrix} \begin{bmatrix} 0 \\ \dot{\gamma} \\ 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\mu & \sin\mu \\ 0 & -\sin\mu & \cos\mu \end{bmatrix} \begin{bmatrix} \cos\gamma & 0 & -\sin\gamma \\ 0 & 1 & 0 \\ \sin\gamma & 0 & \cos\gamma \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ \dot{\chi} \end{bmatrix} \equiv J^{-1} \begin{bmatrix} \dot{\mu} \\ \dot{\gamma} \\ \dot{\chi} \end{bmatrix}$$

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

$$\begin{bmatrix} \dot{\mu} \\ \dot{\gamma} \\ \dot{\chi} \end{bmatrix} = J \begin{bmatrix} p_w \\ q_w \\ r_w \end{bmatrix} = \begin{bmatrix} 1 & (\sin\mu\tan\gamma) & (\cos\mu\tan\gamma) \\ 0 & \cos\mu & -\sin\mu \\ 0 & \frac{\sin\mu}{\cos\gamma} & \frac{\cos\mu}{\cos\gamma} \end{bmatrix} \begin{bmatrix} p_w \\ q_w \\ r_w \end{bmatrix}$$

The body-fixed angular rates are related to the wind-fixed angular rate by the following equation.

$$\begin{bmatrix} p_w \\ q_w \\ r_w \end{bmatrix} = DMC_{wb} \begin{bmatrix} p_b - \dot{\beta}\sin\alpha \\ q_b - \dot{\alpha} \\ r_b + \dot{\beta}\cos\alpha \end{bmatrix}$$

Using this relationship in the wind rate vector equations, gives the relationship between the wind rate vector and the body-fixed angular rates.

$$\begin{bmatrix} \dot{\mu} \\ \dot{\gamma} \\ \dot{\chi} \end{bmatrix} = J \begin{bmatrix} p_w \\ q_w \\ r_w \end{bmatrix} = \begin{bmatrix} 1 (\sin\mu\tan\gamma) (\cos\mu\tan\gamma) \\ 0 \cos\mu & -\sin\mu \\ 0 \frac{\sin\mu}{\cos\gamma} & \frac{\cos\mu}{\cos\gamma} \end{bmatrix} DMC_{wb} \begin{bmatrix} p_b - \dot{\beta}\sin\alpha \\ q_b - \dot{\alpha} \\ r_b + \dot{\beta}\cos\alpha \end{bmatrix}$$

## Parameters

### Main

#### Units

Specifies the input and output units:

| Units                      | Forces | Moment       | Acceleration              | Velocity          | Position | Mass     | Inertia                |
|----------------------------|--------|--------------|---------------------------|-------------------|----------|----------|------------------------|
| Metric (MKS)               | Newton | Newton meter | Meters per second squared | Meters per second | Meters   | Kilogram | Kilogram meter squared |
| English (Velocity in ft/s) | Pound  | Foot pound   | Feet per second squared   | Feet per second   | Feet     | Slug     | Slug foot squared      |
| English (Velocity in kts)  | Pound  | Foot pound   | Feet per second squared   | Knots             | Feet     | Slug     | Slug foot squared      |

#### Mass Type

Select the type of mass to use:

|                 |  |
|-----------------|--|
| Fixed           | Mass is constant throughout the simulation.                |
| Simple Variable | Mass and inertia vary linearly as a function of mass rate. |
| Custom Variable | Mass and inertia variations are customizable.              |

The Custom Variable selection conforms to the previously described equations of motion.

#### Representation

Select the representation to use:

|             |   |
|-------------|---|
| Wind Angles | Use wind angles within equations of motion. |
| Quaternion  | Use quaternions within equations of motion. |

The Wind Angles selection conforms to the previously described equations of motion.

#### Initial position in inertial axes

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

#### Initial airspeed, sideslip angle, and angle of attack

The three-element vector containing the initial airspeed, initial sideslip angle and initial angle of attack.

#### Initial wind orientation

The three-element vector containing the initial wind angles [bank, flight path, and heading], in radians.



**Initial body rotation rates**

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

**Include mass flow relative velocity**

Select this check box to add a mass flow relative velocity port. This is the relative velocity at which the mass is accreted or ablated.

**Include inertial acceleration**

Select this check box to enable an additional output port for the accelerations in body-fixed axes with respect to the inertial frame. You typically connect this signal to the accelerometer.

**State Attributes**

Assign unique name to each state. Use state names instead of block paths throughout the linearization process.

- To assign a name to a single state, enter a unique name between quotes, for example, 'velocity'.
- To assign names to multiple states, enter a comma-delimited list surrounded by braces, for example, {'a', 'b', 'c'}. Each name must be unique.
- If a parameter is empty (' '), no name assignment occurs.
- The state names apply only to the selected block with the name parameter.
- The number of states must divide evenly among the number of state names.
- You can specify fewer names than states, but you cannot specify more names than states.

For example, you can specify two names in a system with four states. The first name applies to the first two states and the second name to the last two states.

- To assign state names with a variable in the MATLAB workspace, enter the variable without quotes. A variable can be a character vector, cell array, or structure.

**Position: e.g., {'Xe', 'Ye', 'Ze'}**

Specify position name for the state.

Default value is ' '.

**Velocity: e.g., 'V'**

Specify velocity name for the state.

Default value is ' '.

**Incidence angle: e.g., 'alpha'**

Specify incidence angle name for the state.

Default value is ' '.

**Sideslip angle: e.g., 'beta'**

Specify sideslip angle name for the state.

Default value is ' '.

**Wind orientation: e.g., {'mu', 'gamma', 'chi'}**

Specify wind orientation name for the state. This parameter appears if the **Representation** parameter is set to Wind Angles.

Default value is ' '.

**Body rotation rates: e.g., {'p', 'q', 'r'}**

Specify body rotation rates name for the state.

Default value is ' '.

## Inputs and Outputs

| Input              | Dimension Type       | Description   |
|--------------------|----------------------|---|
| First              | Vector               | Contains the three applied forces in wind-fixed axes.   |
| Second             | Vector               | Contains the three applied moments in body-fixed axes (+/-).  |
| Third (Optional)   | Vector               | Contains one or more rates of change of mass (positive if accreted, negative if ablated).   |
| Fourth             | Scalar               | Contains the mass.  |
| Fifth              | 3-by-3 matrix        | Applies to the rate of change of inertia tensor matrix in body-fixed axes.  |
| Sixth              | 3-by-3 matrix        | Applies to the inertia tensor matrix in body-fixed axes.  |
| Seventh (Optional) | 1-by-1-by- $m$ array | Contains one or more relative velocities at which the mass is accreted to or ablated from the body in wind axes. $m$ is three times the size of the third input vector. |

| Output  | Dimension Type       | Description   |
|---------|----------------------|---|
| First   | Three-element vector | Contains the velocity in the flat Earth reference frame.  |
| Second  | Three-element vector | Contains the position in the flat Earth reference frame.  |
| Third   | Three-element vector | Contains the wind rotation angles [bank, flight path, heading], within $\pm\pi$ , in radians.               |
| Fourth  | 3-by-3 matrix        | Applies to the coordinate transformation from flat Earth axes to wind-fixed axes.                           |
| Fifth   | Three-element vector | Contains the velocity in the wind-fixed frame.  |
| Sixth   | Two-element vector   | Contains the angle of attack and sideslip angle, in radians.  |
| Seventh | Two-element vector   | Contains the rate of change of angle of attack and rate of change of sideslip angle, in radians per second. |
| Eighth  | Three-element vector | Contains the angular rates in body-fixed axes, in radians per second.                                       |
| Ninth   | Three-element vector | Contains the angular accelerations in body-fixed axes, in radians per second squared.                       |
| Tenth   | Three-element vector | Contains the accelerations in body-fixed axes with respect to body frame.                                   |

| Output                 | Dimension Type       | Description  |
|------------------------|----------------------|--|
| Eleventh<br>(Optional) | Three-element vector | Contains the accelerations in body-fixed axes with respect to inertial frame (flat Earth). You typically connect this signal to the accelerometer. |

## Assumptions and Limitations

The block assumes that the applied forces are acting at the center of gravity of the body.

## References

Stevens, Brian, and Frank Lewis, *Aircraft Control and Simulation*, Second Edition, John Wiley & Sons, 2003.

Zipfel, Peter H., *Modeling and Simulation of Aerospace Vehicle Dynamics*. Second Edition, AIAA Education Series, 2007.

## See Also

6DOF (Euler Angles)

6DOF (Quaternion)

6DOF ECEF (Quaternion)

6DOF Wind (Quaternion)

6DOF Wind (Wind Angles)

6th Order Point Mass (Coordinated Flight)

Custom Variable Mass 6DOF (Euler Angles)

Custom Variable Mass 6DOF (Quaternion)

Custom Variable Mass 6DOF ECEF (Quaternion)

Custom Variable Mass 6DOF Wind (Quaternion)

Simple Variable Mass 6DOF (Euler Angles)

Simple Variable Mass 6DOF (Quaternion)

Simple Variable Mass 6DOF ECEF (Quaternion)

Simple Variable Mass 6DOF Wind (Quaternion)

Simple Variable Mass 6DOF Wind (Wind Angles)

“About Aerospace Coordinate Systems” on page 2-8

## **Extended Capabilities**

### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**Introduced in R2006a**

## Delta UT1

Calculate difference between principal Universal Time (UT1) and Coordinated Universal Time (UTC) according to International Astronomical Union (IAU) 2000A reference system

**Library:** Aerospace Blockset / Environment / Celestial Phenomena  
Aerospace Blockset / Utilities / Axes Transformations



### Description

The Delta UT1 block calculates the difference between principal UT1 and UTC according to the IAU 2000A reference system and Earth orientation data. By default, this block uses a prepopulated list of International Earth Rotation and Reference Systems Service (IERS) data. This list contains measured and calculated (predicted) data supplied by the IERS. The IERS measures and calculates this data for a set of predetermined dates. For dates after those listed in the prepopulated list, Delta UT1 calculates the data using this equation, limiting the values to +/- .9s:

$$UT1 - UTC = 0.5309 - 0.00123(MJD - 57808) - (UT2 - UT1)$$

### Ports

#### Input

##### UTC — UT1 for UTC

modified Julian date

UT1 for UTC, specified as a modified Julian date. Use the `mjuliandate` function to convert the UTC date to a modified Julian date.

Data Types: `double`

#### Output Arguments

##### ΔUT1 — Difference between UT1 and UTC

double

Difference between UT1 and UTC.

Data Types: `double`

### Parameters

##### IERS data file — Earth orientation data

`aeroiersdata.mat` (default) | MAT-file

Custom list of Earth orientation data, specified in a MAT-file.

#### Programmatic Use

**Block Parameter:** `FileName`

**Type:** `character vector`

**Values:** 'aeroiersdata.mat' | MAT-file

**Default:** 'aeroiersdata.mat'

#### Action for out-of-range input – Out-of-range block behavior

Warning (default) | None | Error

Out-of-range block behavior, specified as follows.

| Action          | Description   |
|-----------------|---|
| None            | No action.  |
| Warning         | Warning in the MATLAB Command Window, model simulation continues. |
| Error (default) | MATLAB returns an exception, model simulation stops.              |

#### Programmatic Use

**Block Parameter:** action

**Type:** character vector

**Values:** 'None' | 'Warning' | 'Error'

**Default:** 'Warning'

#### IERS data URL – Web site or Earth orientation data file

<https://maia.usno.navy.mil/ser7/finals2000A.data> (default) | web site address | file name

Web site or Earth orientation data file containing the Earth orientation data according to the IAU 2000A, specified as a web site address or file name.

---

**Note** If you receive an error message while accessing the default site, use one of these alternate sites:

- [https://datacenter.iers.org/data/latestVersion/10\\_FINALS.DATA\\_IAU2000\\_V2013\\_0110.txt](https://datacenter.iers.org/data/latestVersion/10_FINALS.DATA_IAU2000_V2013_0110.txt)
  - <ftp://cddis.gsfc.nasa.gov/pub/products/iers/finals2000A.data>
- 

#### Programmatic Use

**Block Parameter:** iersurl

**Type:** character vector

**Values:** 'https://maia.usno.navy.mil/ser7/finals2000A.data' | web site address | file name

**Default:** 'https://maia.usno.navy.mil/ser7/finals2000A.data'

#### Destination folder – Folder for IERS data file

Current Folder (default)

Folder for IERS data file, specified as a character array or string. Before running this function, create *foldername* with write permission.

To create the IERS data file in the destination folder, click the **Create** button.

#### Programmatic Use

**Block Parameter:** folder

**Type:** character vector

**Values:** 'Current Folder' | folder name

**Default:** 'Current Folder'

## **Extended Capabilities**

### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

### **See Also**

Direction Cosine Matrix ECI to ECEF | Earth Orientation Parameters | `aeroReadIERSData`

### **Topics**

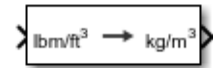
“Calculate UT1 to UTC Values” on page 2-43

### **Introduced in R2017b**

# Density Conversion

Convert from density units to desired density units

**Library:** Aerospace Blockset / Utilities / Unit Conversions



## 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 port labels change based on the input and output units selected from the **Initial unit** and the **Final unit** lists.

## Ports

### Input

#### Port\_1 – Density

scalar | array

Contains the density, specified as a scalar or array, in initial density units.

#### Dependencies

The input port label depends on the **Initial unit** setting.

Data Types: double

### Output

#### Port\_1 – Density

scalar | array

Contains the density, returned as a scalar or array, in initial density units.

#### Dependencies

The output port label depends on the **Final unit** setting.

Data Types: double

## Parameters

### Initial unit – Input units

lbm/ft<sup>3</sup> (default) | kg/m<sup>3</sup> | slug/ft<sup>3</sup> | lbm/in<sup>3</sup>

Input units, specified as:

|                     |                           |
|---------------------|---------------------------|
| lbm/ft <sup>3</sup> | Pound mass per cubic foot |
| kg/m <sup>3</sup>   | Kilograms per cubic meter |



|                      |                           |
|----------------------|---------------------------|
| slug/ft <sup>3</sup> | Slugs per cubic foot      |
| lbm/in <sup>3</sup>  | Pound mass per cubic inch |

**Dependencies**

The input port label depends on the **Initial unit** setting.

**Programmatic Use**

**Block Parameter:** IU

**Type:** character vector

**Values:** 'lbm/ft<sup>3</sup>' | 'kg/m<sup>3</sup>' | 'slug/ft<sup>3</sup>' | 'lbm/in<sup>3</sup>'

**Default:** 'lbm/ft<sup>3</sup>'

**Final unit – Output units**

kg/m<sup>3</sup> (default) | lbm/ft<sup>3</sup> | slug/ft<sup>3</sup> | lbm/in<sup>3</sup>

Output units, specified as:

|                      |                           |
|----------------------|---------------------------|
| lbm/ft <sup>3</sup>  | Pound mass per cubic foot |
| kg/m <sup>3</sup>    | Kilograms per cubic meter |
| slug/ft <sup>3</sup> | Slugs per cubic foot      |
| lbm/in <sup>3</sup>  | Pound mass per cubic inch |

**Dependencies**

The output port label depends on the **Final unit** setting.

**Programmatic Use**

**Block Parameter:** OU

**Type:** character vector

**Values:** 'lbm/ft<sup>3</sup>' | 'kg/m<sup>3</sup>' | 'slug/ft<sup>3</sup>' | 'lbm/in<sup>3</sup>'

**Default:** 'kg/m<sup>3</sup>'

**Extended Capabilities****C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**See Also**

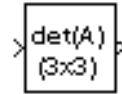
Acceleration Conversion | Angle Conversion | Angular Acceleration Conversion | Angular Velocity Conversion | Force Conversion | Length Conversion | Mass Conversion | Pressure Conversion | Temperature Conversion

**Introduced before R2006a**

## Determinant of 3x3 Matrix

Compute determinant of matrix

**Library:** Aerospace Blockset / Utilities / Math Operations



### Description

The Determinant of 3x3 Matrix block computes the determinant for the input matrix. For related equations, see “Algorithms” on page 4-220.

### Ports

#### Input

##### Port\_1 – Input matrix

3-by-3 matrix

Input matrix, specified as a 3-by-3 matrix.

Data Types: double

#### Output

##### Port\_1 – Determinant

scalar

Determinant, output as a scalar.

Data Types: double

### Algorithms

The input matrix has the form of

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix}$$

The determinant of the matrix has the form of

$$\det(A) = A_{11}(A_{22}A_{33} - A_{23}A_{32}) - A_{12}(A_{21}A_{33} - A_{23}A_{31}) + A_{13}(A_{21}A_{32} - A_{22}A_{31})$$

### Extended Capabilities

#### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

## **See Also**

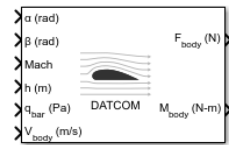
[Adjoint of 3x3 Matrix](#) | [Create 3x3 Matrix](#) | [Invert 3x3 Matrix](#)

**Introduced before R2006a**

## Digital DATCOM Forces and Moments

Compute aerodynamic forces and moments using Digital DATCOM static and dynamic stability derivatives

**Library:** Aerospace Blockset / Aerodynamics



### Description

The Digital DATCOM Forces and Moments block computes the aerodynamic forces and moments about the center of gravity using aerodynamic coefficients from Digital DATCOM.

The Digital DATCOM Forces and Moments block port labels change based on the input and output units selected from the **Units** list.

### Limitations

- The Digital DATCOM Forces and Moments block supports only Digital DATCOM, which is the 1976 version of DATCOM.
- The operational limitations of Digital DATCOM apply to the data contained in the **Digital DATCOM structure** parameter. For more information on Digital DATCOM limitations, see Section 2.4.5 of reference [1].
- The **Digital DATCOM structure** parameters `alpha`, `mach`, `alt`, `grndht`, and `delta` must be strictly monotonically increasing to be used with the Digital DATCOM Forces and Moments block.
- The **Digital DATCOM structure** coefficients must correspond to the dimensions of the breakpoints (`alpha`, `mach`, `alt`, `grndht`, and `delta`) to be used with the Digital DATCOM Forces and Moments block.

### Ports

#### Input

##### Port\_1 — Angle of attack

scalar

Angle of attack, specified as a scalar.

Data Types: `double`

##### Port\_2 — Sideslip angle

scalar

Sideslip angle, specified as a scalar, in radians.

Data Types: `double`

##### Port\_3 — Mach number

scalar

Mach number, specified as a scalar.

Data Types: double

**Port\_4 – Altitude**

scalar

Altitude, specified as a scalar, in selected length units.

Data Types: double

**Port\_5 – Dynamic pressure**

scalar

Dynamic pressure, specified as a scalar, in selected pressure units.

Data Types: double

**Port\_6 – Velocity**

three-element vector

Velocity, specified as a three-element vector, in selected velocity units and selected force axes.

Data Types: double

**Port\_7 – Angle of attack rate**

scalar

Angle of attack rate, specified as a scalar, in radians per second.

**Dependencies**

Appears when DAMP Control Card is used in input to Digital DATCOM.

Data Types: double

**Port\_8 – Body angular rates**

three-element vector

Body angular rates, specified as a three-element vector, in radians per second.

**Dependencies**

Appears when DAMP Control Card is used in input to Digital DATCOM.

Data Types: double

**Port\_9 – Ground height**

scalar

Ground height, specified as a scalar, in select units of length.

**Dependencies**

Appears when GRNDEF Namelist is used in input to Digital DATCOM.

Data Types: double

**Port\_10 – Control surface deflection**

scalar

Control surface deflection, specified as a scalar, in radians.

### Dependencies

Appears when ASYFLP or SYMFLP and GRNDEF namelists are used in input to Digital DATCOM.

Data Types: double

### Output

#### Port\_1 — Aerodynamic forces at the center of gravity

three-element vector

Aerodynamic forces at the center of gravity, returned as a three-element vector, in selected coordinate system: Body ( $F$ ,  $F_{y_x}$ , and  $F_z$ ), or Wind ( $F_D$ ,  $F_y$ , and  $F_L$ ).

Data Types: double

#### Port\_2 — Aerodynamic moments at the center of gravity

three-element vector

Aerodynamic moments at the center of gravity, returned as a three-element vector, in body coordinates ( $M_x$ ,  $M_y$ , and  $M_z$ ).

Data Types: double

## Parameters

### Units — Units

Metric (MKS) (default) | English (Velocity in ft/s) | English (Velocity in kts)

Input and output units, specified as:

| Units                      | Force  | Moment       | Length | Velocity          | Pressure              |
|----------------------------|--------|--------------|--------|-------------------|-----------------------|
| Metric (MKS)               | Newton | Newton-meter | Meters | Meters per second | Pascal                |
| English (Velocity in ft/s) | Pound  | Foot-pound   | Feet   | Feet per second   | Pound per square inch |
| English (Velocity in kts)  | Pound  | Foot-pound   | Feet   | Knots             | Pound per square inch |

### Programmatic Use

**Block Parameter:** units

**Type:** character vector

**Values:** 'Metric (MKS)' | 'English (Velocity in ft/s)' | 'English (Velocity in kts)'

**Default:** 'Metric (MKS)'

### Digital DATCOM structure — Digital DATCOM data structure

factstruct{1} (default) | structure

MATLAB structure containing the digital DATCOM data. This structure is generated by the `datcomimport` function. To include dynamic derivatives in the generated output file, call the `datomimport` function with the `damp` keyword.

For more information on creating the digital DATCOM structure, see “Importing from USAF Digital DATCOM Files” (Aerospace Toolbox). This example shows how to bring United States Air Force (USAF) Digital DATCOM files into the MATLAB environment using the Aerospace Toolbox software.

**Programmatic Use**

**Block Parameter:** `dcase`

**Type:** character vector

**Values:** `factstruct{1}` | structure

**Default:** `factstruct{1}`

**Force axes — Coordinate system for aerodynamic force**

Body (default) | Wind

Coordinate system for aerodynamic force, specified as Body or Wind.

**Programmatic Use**

**Block Parameter:** `fmode`

**Type:** character vector

**Values:** 'Body' | 'Wind'

**Default:** 'Body'

**Interpolation method — Interpolation method**

None - flat (default) | Linear

Interpolation method, specified as None (flat) or Linear. The block uses the interpolation method to interpolate the static and dynamic stability coefficients in the **Digital DATCOM structure**.

**Programmatic Use**

**Block Parameter:** `imethod`

**Type:** character vector

**Values:** 'None (flat)' | 'Linear'

**Default:** 'None (flat)'

**Extrapolation method — Extrapolation method**

None - clip (default) | Linear

Extrapolation method, specified as None (clip) or Linear. The block uses the extrapolation method to extrapolate the static and dynamic stability coefficients in the **Digital DATCOM structure**.

**Programmatic Use**

**Block Parameter:** `emethod`

**Type:** character vector

**Values:** 'None (flat)' | 'Linear'

**Default:** 'None (flat)'

**Process out-of-range input — Handle out-of-range input**

Clip to Range (default) | Linear Extrapolation

Handle out-of-range input action, Linear Extrapolation or Clip to Range.

**Programmatic Use****Block Parameter:** rmethod**Type:** character vector**Values:** 'Clip to Range' | 'Linear Extrapolation'**Default:** 'Clip to Range'**Action for out-of-range input – Out-of-range block behavior**

None (default) | Warning | Error

Out-of-range block behavior, specified as follows.

| Action          | Description   |
|-----------------|---|
| None            | No action.  |
| Warning         | Warning in the MATLAB Command Window, model simulation continues. |
| Error (default) | MATLAB returns an exception, model simulation stops.              |

**Programmatic Use****Block Parameter:** action**Type:** character vector**Values:** 'None' | 'Warning' | 'Error'**Default:** 'None'**Algorithms**

Algorithms for calculating forces and moments build up the overall aerodynamic forces and moments ( $F$  and  $M$ ) from data contained in the **Digital DATCOM structure** parameter:

$$F = F_{\text{static}} + F_{\text{dyn}} \quad (4-1)$$

$$M = M_{\text{static}} + M_{\text{dyn}} \quad (4-2)$$

$F_{\text{static}}$  and  $M_{\text{static}}$  are the static contribution, and  $F_{\text{dyn}}$  and  $M_{\text{dyn}}$  the dynamic contribution, to the aerodynamic coefficients. If the dynamic characteristics are not contained in the **Digital DATCOM structure** parameter, their contribution is set to zero.

**Static Stability Characteristics**

Static stability characteristics include the following.

| Coefficient  | Meaning   |
|--------------|---|
| $C_D$        | Matrix of drag coefficients. These coefficients are defined positive for an aft-acting load.            |
| $C_L$        | Matrix of lift coefficients. These coefficients are defined positive for an up-acting load.             |
| $C_m$        | Matrix of pitching-moment coefficients. These coefficients are defined positive for a nose-up rotation. |
| $C_{y\beta}$ | Matrix of derivatives of side-force coefficients with respect to sideslip angle                         |
| $C_{n\beta}$ | Matrix of derivatives of yawing-moment coefficients with respect to sideslip angle                      |



| Coefficient  | Meaning   |
|--------------|---|
| $C_{l\beta}$ | Matrix of derivatives of rolling-moment coefficients with respect to sideslip angle |

These are the static contributions to the aerodynamic coefficients in stability axes.

$$C_{D \text{ static}} = C_D \quad (4-3)$$

$$C_{y \text{ static}} = C_{Y\beta} \beta \quad (4-4)$$

$$C_{L \text{ static}} = C_L \quad (4-5)$$

$$C_{l \text{ static}} = C_{l\beta} \beta \quad (4-6)$$

$$C_{m \text{ static}} = C_M \quad (4-7)$$

$$C_{n \text{ static}} = C_{n\beta} \beta \quad (4-8)$$

### Dynamic Stability Characteristics

Dynamic stability characteristics include the following.

| Coefficient       | Meaning  |
|-------------------|--|
| $C_{Lq}$          | Matrix of lift force derivatives due to pitch rate                   |
| $C_{mq}$          | Matrix of pitching-moment derivatives due to pitch rate              |
| $C_{Ld\alpha/dt}$ | Matrix of lift force derivatives due to rate of angle of attack      |
| $C_{md\alpha/dt}$ | Matrix of pitching-moment derivatives due to rate of angle of attack |
| $C_{lp}$          | Matrix of rolling-moment derivatives due to roll rate                |
| $C_{Yp}$          | Matrix of lateral force derivatives due to roll rate                 |
| $C_{np}$          | Matrix of yawing-moment derivatives due to roll rate                 |
| $C_{nr}$          | Matrix of yawing-moment derivatives due to yaw rate                  |
| $C_{lr}$          | Matrix of rolling-moment derivatives due to yaw rate                 |

These are the dynamic contributions to the aerodynamic coefficients in stability axes.

$$C_{D \text{ dyn}} = 0$$

$$C_{y \text{ dyn}} = C_{yp} p (b_{\text{ref}}/2V)$$

$$C_{L \text{ dyn}} = (C_{Lq} q + C_{L\dot{\alpha}} \dot{\alpha}) (c_{\text{bar}}/2V)$$

$$C_{l \text{ dyn}} = (C_{lp} p + C_{lr} r) (b_{\text{ref}}/2V)$$

$$C_{m \text{ dyn}} = (C_{mq} q + C_{m\dot{\alpha}} \dot{\alpha}) (c_{\text{bar}}/2V)$$

$$C_{n \text{ dyn}} = (C_{np} p + C_{nr} r) (b_{\text{ref}}/2V)$$

### References

- [1] *The USAF Stability and Control Digital Datcom*, AFFDL-TR-79-3032, 1979.
- [2] Etkin, B., and L. D. Reid. *Dynamics of Flight Stability and Control*, Hoboken, NJ: John Wiley & Sons, 1996.

[3] Roskam, J. "Airplane Design Part VI: Preliminary Calculation of Aerodynamic, Thrust and Power Characteristics", Roskam Aviation and Engineering Corporation, Ottawa, Kansas: 1987.

[4] Stevens, B. L., and F. L. Lewis. *Aircraft Control and Simulation*, Hoboken, NJ: John Wiley & Sons, 1992.

## **Extended Capabilities**

### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

### **See Also**

Aerodynamic Forces and Moments | `datcomimport`

### **Topics**

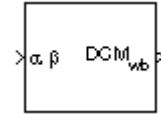
"Importing from USAF Digital DATCOM Files" (Aerospace Toolbox)

### **Introduced in R2006b**

# Direction Cosine Matrix Body to Wind

Convert angle of attack and sideslip angle to direction cosine matrix

**Library:** Aerospace Blockset / Utilities / Axes Transformations



## Description

The Direction Cosine Matrix Body to Wind block converts angle of attack and sideslip angles into a 3-by-3 direction cosine matrix (DCM). This direction cosine matrix is helpful for vector body axes to wind axes coordinate transformations. To transform the coordinates of a vector in body axes ( $ox_0$ ,  $oy_0$ ,  $oz_0$ ) to a vector in wind axes ( $ox_2$ ,  $oy_2$ ,  $oz_2$ ), multiply the block output direction cosine matrix with a vector in body axes. For information on the axis rotations for this transformation, see “Algorithms” on page 4-229.

## Ports

### Input

**$\alpha$   $\beta$  — Angle of attack and sideslip angle**

2-by-1 vector

Angle of attack and sideslip angle, specified as a 2-by-1 vector, in radians.

Data Types: double

### Output

**$DCM_{wb}$  — Direction cosine matrix**

3-by-3 direction cosine matrix

Direction cosine matrix, returned as 3-by-3 direction cosine matrix.

Data Types: double

## Algorithms

The order of the axis rotations required to bring this transformation about is:

- 1 A rotation about  $oy_0$  through the angle of attack ( $\alpha$ ) to axes ( $ox_1$ ,  $oy_1$ ,  $oz_1$ )
- 2 A rotation about  $oz_1$  through the sideslip angle ( $\beta$ ) to axes ( $ox_2$ ,  $oy_2$ ,  $oz_2$ )

$$\begin{bmatrix} ox_2 \\ oy_2 \\ oz_2 \end{bmatrix} = DCM_{wb} \begin{bmatrix} ox_0 \\ oy_0 \\ oz_0 \end{bmatrix}$$

$$\begin{bmatrix} ox_2 \\ oy_2 \\ oz_2 \end{bmatrix} = \begin{bmatrix} \cos\beta & \sin\beta & 0 \\ -\sin\beta & \cos\beta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\alpha & 0 & \sin\alpha \\ 0 & 1 & 0 \\ -\sin\alpha & 0 & \cos\alpha \end{bmatrix} \begin{bmatrix} ox_0 \\ oy_0 \\ oz_0 \end{bmatrix}$$

Combining the two axis transformation matrices defines the following DCM.

$$DCM_{wb} = \begin{bmatrix} \cos\alpha\cos\beta & \sin\beta & \sin\alpha\cos\beta \\ -\cos\alpha\sin\beta & \cos\beta & -\sin\alpha\sin\beta \\ -\sin\alpha & 0 & \cos\alpha \end{bmatrix}$$

## References

- [1] Stevens, B. L., and F. L. Lewis. *Aircraft Control and Simulation*. Hoboken, NJ: John Wiley & Sons, 1992.

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

## See Also

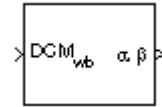
Direction Cosine Matrix Body to Wind to Alpha and Beta | Direction Cosine Matrix to Rotation Angles | Direction Cosine Matrix to Wind Angles | Rotation Angles to Direction Cosine Matrix | Wind Angles to Direction Cosine Matrix

**Introduced before R2006a**

# Direction Cosine Matrix Body to Wind to Alpha and Beta

Convert direction cosine matrix to angle of attack and sideslip angle

**Library:** Aerospace Blockset / Utilities / Axes Transformations



## Description

The Direction Cosine Matrix Body to Wind to Alpha and Beta block converts a 3-by-3 direction cosine matrix (DCM) to angle of attack and sideslip angle. The DCM performs the coordinate transformation of a vector in body axes ( $ox_0, oy_0, oz_0$ ) into a vector in wind axes ( $ox_2, oy_2, oz_2$ ). For more information on the direction cosine matrix, see “Algorithms” on page 4-232.

## Limitations

- This implementation generates angles that lie between  $\pm 90$  degrees.

## Ports

### Input

#### $DCM_{wb}$ — Direction cosine matrix

3-by-3 direction cosine matrix

Direction cosine matrix to transform body-fixed vectors to wind-fixed vectors, specified as a 3-by-3 direction cosine matrix.

Data Types: double

### Output

#### $\alpha, \beta$ — Angle of attack and sideslip angle

2-by-1 vector

Angle of attack and sideslip angle, returned as a vector, in radians.

Data Types: double

## Parameters

### Action for invalid DCM — Block behavior

None (default) | Warning | Error

Block behavior when the direction cosine matrix is invalid (not orthogonal).

- Warning — Displays warning indicating that the direction cosine matrix is invalid.
- Error — Displays error indicating that the direction cosine matrix is invalid.

- None — Does not display warning or error (default).

**Programmatic Use****Block Parameter:** action**Type:** character vector**Values:** 'None' | 'Warning' | 'Error'**Default:** 'None'

Data Types: char | string

**Tolerance for DCM validation — Tolerance**

eps(2) (default) | scalar

Tolerance of the direction cosine matrix validity, specified as a scalar. The block considers the direction cosine matrix valid if these conditions are true:

- The transpose of the direction cosine matrix times itself equals 1 within the specified tolerance ( $\text{transpose}(n)*n == 1 \pm \text{tolerance}$ ).
- The determinant of the direction cosine matrix equals 1 within the specified tolerance ( $\det(n) == 1 \pm \text{tolerance}$ ).

**Programmatic Use****Block Parameter:** tolerance**Type:** character vector**Values:** 'eps(2)' | scalar**Default:** 'eps(2)'

Data Types: double

**Algorithms**

The DCM matrix performs the coordinate transformation of a vector in body axes ( $ox_0, oy_0, oz_0$ ) into a vector in wind axes ( $ox_2, oy_2, oz_2$ ). The order of the axis rotations required to bring this about is:

- 1 A rotation about  $oy_0$  through the angle of attack ( $\alpha$ ) to axes ( $ox_1, oy_1, oz_1$ )
- 2 A rotation about  $oz_1$  through the sideslip angle ( $\beta$ ) to axes ( $ox_2, oy_2, oz_2$ )

$$\begin{bmatrix} ox_2 \\ oy_2 \\ oz_2 \end{bmatrix} = DCM_{wb} \begin{bmatrix} ox_0 \\ oy_0 \\ oz_0 \end{bmatrix}$$

$$\begin{bmatrix} ox_2 \\ oy_2 \\ oz_2 \end{bmatrix} = \begin{bmatrix} \cos\beta & \sin\beta & 0 \\ -\sin\beta & \cos\beta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\alpha & 0 & \sin\alpha \\ 0 & 1 & 0 \\ -\sin\alpha & 0 & \cos\alpha \end{bmatrix} \begin{bmatrix} ox_0 \\ oy_0 \\ oz_0 \end{bmatrix}$$

Combining the two axis transformation matrices defines the following DCM.

$$DCM_{wb} = \begin{bmatrix} \cos\alpha\cos\beta & \sin\beta & \sin\alpha\cos\beta \\ -\cos\alpha\sin\beta & \cos\beta & -\sin\alpha\sin\beta \\ -\sin\alpha & 0 & \cos\alpha \end{bmatrix}$$

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

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

$$\beta = \text{asin}(DCM(1, 2))$$

## References

- [1] Stevens, Brian L., Frank L. Lewis. *Aircraft Control and Simulation*, Second Edition. Hoboken, NJ: Wiley-Interscience.

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

## See Also

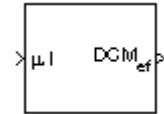
Direction Cosine Matrix Body to Wind | Direction Cosine Matrix to Rotation Angles | Direction Cosine Matrix to Wind Angles | Rotation Angles to Direction Cosine Matrix | Wind Angles to Direction Cosine Matrix

**Introduced before R2006a**

## Direction Cosine Matrix ECEF to NED

Convert geodetic latitude and longitude to direction cosine matrix

**Library:** Aerospace Blockset / Utilities / Axes Transformations



### Description

The Direction Cosine Matrix ECEF to NED block converts geodetic latitude and longitude into a 3-by-3 direction cosine matrix (DCM). The DCM matrix performs the coordinate transformation of a vector in Earth-centered Earth-fixed (ECEF) axes into a vector in north-east-down (NED) axes. For more information on the direction cosine matrix, see “Algorithms” on page 4-234.

The implementation of the ECEF coordinate system assumes that the origin is at the center of the planet, the x-axis intersects the Greenwich meridian and the equator, the z-axis is the mean spin axis of the planet, positive to the north, and the y-axis completes the right-hand system. For more information, see “About Aerospace Coordinate Systems” on page 2-8.

### Ports

#### Input

##### $\mu_l$ — Geodetic latitude and longitude

2-by-1 vector

Geodetic latitude and longitude, specified as a 2-by-1 vector, in degrees. Latitude and longitude values can be any value. However, latitude values of +90 and -90 may return unexpected values because of singularity at the poles.

Data Types: double

#### Output

##### $DCM_{ef}$ — Direction cosine matrix

3-by-3 matrix

DCM to perform coordinate transform of a vector in ECEF axes into a vector in NED axes, returned as a 3-by-3 matrix.

Data Types: double

### Algorithms

The DCM matrix performs the coordinate transformation of a vector in ECEF axes,  $(ox_0, oy_0, oz_0)$ , into a vector in NED axes,  $(ox_2, oy_2, oz_2)$ . The order of the axis rotations required to bring this about is:

- 1 A rotation about  $oz_0$  through the longitude ( $\iota$ ) to axes  $(ox_1, oy_1, oz_1)$
- 2 A rotation about  $oy_1$  through the geodetic latitude ( $\mu$ ) to axes  $(ox_2, oy_2, oz_2)$



$$\begin{bmatrix} ox_2 \\ oy_2 \\ oz_2 \end{bmatrix} = DCM_{ef} \begin{bmatrix} ox_0 \\ oy_0 \\ oz_0 \end{bmatrix}$$

$$\begin{bmatrix} ox_2 \\ oy_2 \\ oz_2 \end{bmatrix} = \begin{bmatrix} -\sin\mu & 0 & \cos\mu \\ 0 & 1 & 0 \\ -\cos\mu & 0 & -\sin\mu \end{bmatrix} \begin{bmatrix} \cos\iota & \sin\iota & 0 \\ -\sin\iota & \cos\iota & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} ox_0 \\ oy_0 \\ oz_0 \end{bmatrix}$$

Combining the two axis transformation matrices defines the following DCM.

$$DCM_{ef} = \begin{bmatrix} -\sin\mu\cos\iota & -\sin\mu\sin\iota & \cos\mu \\ -\sin\iota & \cos\iota & 0 \\ -\cos\mu\cos\iota & -\cos\mu\sin\iota & -\sin\mu \end{bmatrix}$$

## References

- [1] Stevens, B. L., and F. L. Lewis. *Aircraft Control and Simulation*, Hoboken, NJ: John Wiley & Sons, 1992.
- [2] Zipfel, Peter H., *Modeling and Simulation of Aerospace Vehicle Dynamics*. Second Edition. Reston, VA: AIAA Education Series, 2000.
- [3] *Recommended Practice for Atmospheric and Space Flight Vehicle Coordinate Systems*, R-004-1992, ANSI/AIAA, February 1992.

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

### See Also

Direction Cosine Matrix ECEF to NED to Latitude and Longitude | Direction Cosine Matrix to Rotation Angles | Direction Cosine Matrix to Wind Angles | ECEF Position to LLA | LLA to ECEF Position | Rotation Angles to Direction Cosine Matrix | Wind Angles to Direction Cosine Matrix

### Topics

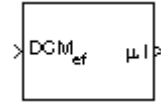
“About Aerospace Coordinate Systems” on page 2-8

### Introduced before R2006a

# Direction Cosine Matrix ECEF to NED to Latitude and Longitude

Convert direction cosine matrix to geodetic latitude and longitude

**Library:** Aerospace Blockset / Utilities / Axes Transformations



## Description

The Direction Cosine Matrix ECEF to NED to Latitude and Longitude block converts a 3-by-3 direction cosine matrix (DCM) into geodetic latitude and longitude. The DCM matrix performs the coordinate transformation of a vector in Earth-centered Earth-fixed (ECEF) axes,  $(ox_0, oy_0, oz_0)$ , into geodetic latitude and longitude. For more information on the direction cosine matrix, see “Algorithms” on page 4-237.

## Limitations

The DCM matrix performs the coordinate transformation of a vector in ECEF axes,  $(ox_0, oy_0, oz_0)$ , into geodetic latitude and longitude. The order of the axis rotations required to bring this about is:

- This implementation generates a geodetic latitude that lies between  $\pm 90$  degrees, and longitude that lies between  $\pm 180$  degrees.
- The implementation of the ECEF coordinate system assumes that the origin is at the center of the planet, the  $x$ -axis intersects the Greenwich meridian and the equator, the  $z$ -axis is the mean spin axis of the planet, positive to the north, and the  $y$ -axis completes the right-hand system. For more information, see “About Aerospace Coordinate Systems” on page 2-8.

## Ports

### Input

**DCM<sub>ef</sub> — Direction cosine matrix**

3-by-3 matrix

Direction cosine matrix from which to geodetic latitude and longitude, specified as a 3-by-3 matrix.

Example:

Data Types: double

### Output

**μ<sub>l</sub> — Geodetic latitude and longitude**

2-by-1 vector

Geodetic latitude and longitude, returned as a 2-by-1 vector in degrees.

Data Types: double

## Parameters

### Action for invalid DCM — Block behavior

None (default) | Warning | Error

Block behavior when direction cosine matrix is invalid (not orthogonal).

- Warning — Displays warning indicating that the direction cosine matrix is invalid.
- Error — Displays error indicating that the direction cosine matrix is invalid.
- None — Does not display warning or error (default).

### Programmatic Use

**Block Parameter:** action

**Type:** character vector

**Values:** 'None' | 'Warning' | 'Error'

**Default:** 'None'

Data Types: char | string

### Tolerance for DCM validation — Tolerance

eps(2) (default) | scalar

Tolerance of the direction cosine matrix validity, specified as a scalar. The block considers the direction cosine matrix valid if these conditions are true:

- The transpose of the direction cosine matrix times itself equals 1 within the specified tolerance ( $\text{transpose}(n)*n == 1 \pm \text{tolerance}$ ).
- The determinant of the direction cosine matrix equals 1 within the specified tolerance ( $\det(n) == 1 \pm \text{tolerance}$ ).

### Programmatic Use

**Block Parameter:** tolerance

**Type:** character vector

**Values:** 'eps(2)' | scalar

**Default:** 'eps(2)'

Data Types: double

## Algorithms

The DCM matrix performs the coordinate transformation of a vector in ECEF axes,  $(ox_0, oy_0, oz_0)$ , into geodetic latitude and longitude. The order of the axis rotations required to bring this about is:

- 1 A rotation about  $oz_0$  through the longitude ( $\iota$ ) to axes  $(ox_1, oy_1, oz_1)$
- 2 A rotation about  $oy_1$  through the geodetic latitude ( $\mu$ ) to axes  $(ox_2, oy_2, oz_2)$

$$\begin{bmatrix} ox_2 \\ oy_2 \\ oz_2 \end{bmatrix} = DCM_{ef} \begin{bmatrix} ox_0 \\ oy_0 \\ oz_0 \end{bmatrix}$$

$$\begin{bmatrix} ox_2 \\ oy_2 \\ oz_2 \end{bmatrix} = \begin{bmatrix} -\sin\mu & 0 & \cos\mu \\ 0 & 1 & 0 \\ -\cos\mu & 0 & -\sin\mu \end{bmatrix} \begin{bmatrix} \cos\iota & \sin\iota & 0 \\ -\sin\iota & \cos\iota & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} ox_0 \\ oy_0 \\ oz_0 \end{bmatrix}$$

Combining the two axis transformation matrices defines the following DCM.

$$DCM_{ef} = \begin{bmatrix} -\sin\mu\cos\iota & -\sin\mu\sin\iota & \cos\mu \\ -\sin\iota & \cos\iota & 0 \\ -\cos\mu\cos\iota & -\cos\mu\sin\iota & -\sin\mu \end{bmatrix}$$

To determine geodetic latitude and longitude from the DCM, the following equations are used:

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

$$\iota = \text{atan}\left(\frac{-DCM(2, 1)}{DCM(2, 2)}\right)$$

## References

- [1] Zipfel, Peter H., *Modeling and Simulation of Aerospace Vehicle Dynamics*. Second Edition. Reston, VA: AIAA Education Series, 2000.
- [2] *Recommended Practice for Atmospheric and Space Flight Vehicle Coordinate Systems*, R-004-1992, ANSI/AIAA, February 1992.
- [3] Stevens, B. L., and F. L. Lewis. *Aircraft Control and Simulation*, Hoboken, NJ: John Wiley & Sons, 1992.

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

## See Also

Direction Cosine Matrix ECEF to NED | Direction Cosine Matrix to Rotation Angles | Direction Cosine Matrix to Wind Angles | ECEF Position to LLA | LLA to ECEF Position | Rotation Angles to Direction Cosine Matrix | Wind Angles to Direction Cosine Matrix

### Topics

“Algorithms” on page 4-237

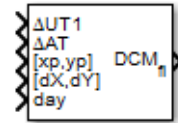
“About Aerospace Coordinate Systems” on page 2-8

**Introduced before R2006a**

# Direction Cosine Matrix ECI to ECEF

Convert Earth-centered inertial (ECI) to Earth-centered Earth-fixed (ECEF) coordinates

**Library:** Aerospace Blockset / Utilities / Axes Transformations



## Description

The Direction Cosine Matrix ECI to ECEF block calculates the position direction cosine matrix (Earth-centered inertial to Earth-centered Earth-fixed), based on the specified reduction method and Universal Coordinated Time (UTC), for the specified time and geophysical data.

## Ports

### Input

#### $\Delta$ UT1 — Difference between UTC and Universal Time

scalar

Difference between UTC and Universal Time (UT1) in seconds, specified as a scalar, for which the function calculates the direction cosine or transformation matrix.

Example: 0.234

#### Dependencies

To enable this port, select the **Higher accuracy parameters** check box.

Data Types: double

#### $\Delta$ AT — Difference between International Atomic Time and UTC

scalar

Difference between International Atomic Time (IAT) and UTC, specified as a scalar, in seconds, for which the function calculates the direction cosine or transformation matrix.

Example: 32

#### Dependencies

To enable this port, select the **Higher accuracy parameters** check box.

Data Types: double

#### [xp, yp] — Polar displacement of Earth

1-by-2 array

Polar displacement of Earth, specified as a 1-by-2 array, in radians, from the motion of the Earth crust, along the x-axis and y-axis.

Example: [-0.0682e-5 0.1616e-5]

**Dependencies**

To enable this port, select the **Higher accuracy parameters** check box.

Data Types: double

**Port\_5 — Adjustment based on reduction method**

1-by-2 array

Adjustment based on reduction method, specified as 1-by-2 array. The name of the port depends on the setting of the **Reduction** parameter:

- If the reduction method is IAU-2000/2006, this input is the adjustment to the location of the Celestial Intermediate Pole (CIP), specified in radians. This location ( $[dX, dY]$ ) is along the x-axis and y-axis.
- If the reduction method is IAU-76/FK5, this input is the adjustment to the longitude ( $[\Delta\delta\psi, \Delta\delta\epsilon]$ ), specified in radians.

For historical values, see International Earth Rotation and Reference Systems Service.

Example:  $[-0.2530e-6 \ -0.0188e-6]$

**Dependencies**

To enable this port, select the **Higher accuracy parameters** check box.

Data Types: double

**Port\_6 — Time increment source**

scalar

Time increment source, specified as a scalar, such as the Clock block.

**Dependencies**

- The port name and time increment depend on the **Time Increment** parameter.

| Time Increment Value | Port Name   |
|----------------------|-------------|
| Day                  | <b>day</b>  |
| Hour                 | <b>hour</b> |
| Min                  | <b>min</b>  |
| Sec                  | <b>sec</b>  |
| None                 | No port     |

- To disable this port, set the **Time Increment** parameter to None.

Data Types: double

**Output****DCM<sub>fi</sub> — Direction cosine matrix**

3-by-3 matrix

Direction cosine matrix ECI to ECEF.

Data Types: double

## Parameters

### Reduction — Reduction method

IAU-76/FK5 (default) | IAU-2000/2006

Reduction method to calculate the direction cosine matrix. The method can be one of the following:

- IAU-76/FK5

Reduce the calculation using the IAU-76/Fifth Fundamental Catalogue (FK5) reference system. Choose this reduction method if the reference coordinate system for the conversion is FK5.

---

**Note** This method uses the IAU 1976 precession model and the IAU 1980 theory of nutation to reduce the calculation. This model and theory are no longer current, but the software provides this reduction method for existing implementations. Because of the polar motion approximation that this reduction method uses, the block calculates the transformation matrix rather than the direction cosine matrix.

---

- IAU-2000/2006

Reduce the calculation using the International Astronomical Union (IAU)-2000/2006 reference system. Choose this reduction method if the reference coordinate system for the conversion is IAU-2000. This reduction method uses the P03 precession model to reduce the calculation.

### Programmatic Use

**Block Parameter:** red

**Type:** character vector

**Values:** 'IAU-2000/2006' | 'IAU-76/FK5'

**Default:** 'IAU-2000/2006'

### Year — Year

2013 (default) | double, whole number, greater than 1

Year to calculate the Universal Coordinated Time (UTC) date. Enter a double value that is a whole number greater than 1, such as 2013.

### Programmatic Use

**Block Parameter:** year

**Type:** character vector

**Values:** double, whole number, greater than 1

**Default:** '2013'

### Month — Month

January (default) | February | March | April | May | June | July | August | September | October | November | December

Month to calculate the UTC date.

### Programmatic Use

**Block Parameter:** month

**Type:** character vector

**Values:** 'January' | 'February' | 'March' | 'April' | 'May' | 'June' | 'July' | 'August' | 'September' | 'October' | 'November' | 'December'

**Default:** 'January'

**Day — Day**

1 (default) | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31

Day to calculate the UTC date.

**Programmatic Use**

**Block Parameter:** day

**Type:** character vector

**Values:** '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9' | '10' | '11' | '12' | '13' | '14' | '15' | '16' | '17' | '18' | '19' | '20' | '21' | '22' | '23' | '24' | '25' | '26' | '27' | '28' | '29' | '30' | '31'

**Default:** '1'

**Hour — Hour**

0 (default) | double, whole number, in the range of 0 and 24

Hour to calculate the UTC date. Enter a double value that is a whole number, from 0 to 24.

**Programmatic Use**

**Block Parameter:** hour

**Type:** character vector

**Values:** double, whole number, 0 to 24

**Default:** '0'

**Minutes — Minutes**

0 (default) | double, whole number, in the range of 0 and 60

Minutes to calculate the UTC date. Enter a double value that is a whole number, from 0 to 60.

**Programmatic Use**

**Block Parameter:** min

**Type:** character vector

**Values:** double, whole number, 0 to 60

**Default:** '0'

**Seconds — Seconds**

0 (default) | double, whole number, in the range of 0 and 60

Seconds to calculate the UTC date. Enter a double value that is a whole number, from 0 to 60.

**Programmatic Use**

**Block Parameter:** sec

**Type:** character vector

**Values:** double, whole number, 0 to 60

**Default:** '0'

**Time increment — Time increment**

Day (default) | Hour | Min | Sec | None

Time increment between the specified date and the desired model simulation time. The block adjusts the calculated direction cosine matrix to take into account the time increment from model simulation. For example, selecting Day and connecting a simulation timer to the port means that each time increment unit is one day and the block adjusts its calculation based on that simulation time.

This parameter corresponds to the time increment input, the clock source.



If you select None, the calculated Julian date does not take into account the model simulation time.

**Programmatic Use**

**Block Parameter:** deltaT

**Type:** character vector

**Values:** 'None' | 'Day' | 'Hour' | 'Min' | 'Sec'

**Default:** 'Day'

**Action for out-of-range input – Action**

Error (default) | Warning | None

Specify the block behavior when the block inputs are out of range.

| Action          | Description   |
|-----------------|---|
| None            | No action.  |
| Warning         | Warning in the MATLAB Command Window, model simulation continues. |
| Error (default) | MATLAB returns an exception, model simulation stops.              |

**Programmatic Use**

**Block Parameter:** errorflag

**Type:** character vector

**Values:** 'None' | 'Warning' | 'Error'

**Default:** 'Error'

**Higher accuracy parameters – Enable higher accuracy parameters**

on (default) | off

Select this check box to enable these inputs. These inputs let you better control the conversion result. See “Input” on page 4-239 for a description.

- $\Delta UT1$
- $\Delta AT$
- [  $x_p$  ,  $y_p$  ]
- [  $\Delta\delta\psi$ ,  $\Delta\delta\varepsilon$  ] or [ d X, d Y ]

**Programmatic Use**

**Block Parameter:** extraparamflag

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'on'

## Extended Capabilities

**C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

## See Also

Delta UT1 | ECEF Position to LLA | Earth Orientation Parameters | Geocentric to Geodetic Latitude | Geodetic to Geocentric Latitude | LLA to ECEF Position

**External Websites**

<https://www.iers.org>

**Introduced in R2013b**

# Direction Cosine Matrix to Rodrigues

Convert direction cosine matrix to Euler-Rodrigues vector

**Library:** Aerospace Blockset / Utilities / Axes Transformations



## Description

The Direction Cosine Matrix to Rodrigues block determines the 3-by-3 direction cosine matrix from a three-element Euler-Rodrigues vector. For more information on the direction cosine matrix, see “Algorithms” on page 4-246.

## Ports

### Input

#### DCM — Direction cosine matrix

3-by-3 matrix

Direction cosine matrix, specified as a 3-by-3 matrix, from which to determine the Euler-Rodrigues vector.

Data Types: double

### Output

#### rod — Euler-Rodrigues vector

three-element vector

Euler-Rodrigues vector, returned as a three-element vector.

Data Types: double

## Parameters

### Action for invalid DCM — Block behavior

None (default) | Warning | Error

Block behavior when direction cosine matrix is invalid (not orthogonal).

- Warning — Displays warning and indicates that the direction cosine matrix is invalid.
- Error — Displays error and indicates that the direction cosine matrix is invalid.
- None — Does not display warning or error (default).

### Programmatic Use

**Block Parameter:** action

**Type:** character vector

**Values:** 'None' | 'Warning' | 'Error'

**Default:** 'None'

Data Types: char | string

### Tolerance for DCM validation – Tolerance

eps(2) (default) | scalar

Tolerance of direction cosine matrix validity, specified as a scalar. The block considers the direction cosine matrix valid if these conditions are true:

- The transpose of the direction cosine matrix times itself equals 1 within the specified tolerance ( $\text{transpose}(n)*n == 1 \pm \text{tolerance}$ )
- The determinant of the direction cosine matrix equals 1 within the specified tolerance ( $\text{det}(n) == 1 \pm \text{tolerance}$ ).

#### Programmatic Use

**Block Parameter:** tolerance

**Type:** character vector

**Values:** 'eps(2)' | scalar

**Default:** 'eps(2)'

Data Types: double

## Algorithms

An Euler-Rodrigues vector  $\vec{b}$  represents a rotation by integrating a direction cosine of a rotation axis with the tangent of half the rotation angle as follows:

$$\vec{b} = [b_x \ b_y \ b_z]$$

where:

$$b_x = \tan\left(\frac{1}{2}\theta\right)s_x,$$

$$b_y = \tan\left(\frac{1}{2}\theta\right)s_y,$$

$$b_z = \tan\left(\frac{1}{2}\theta\right)s_z$$

are the Rodrigues parameters. Vector  $\vec{s}$  represents a unit vector around which the rotation is performed. Due to the tangent, the rotation vector is indeterminate when the rotation angle equals  $\pm\pi$  radians or  $\pm 180$  deg. Values can be negative or positive.

## References

- [1] Dai, J.S. "Euler-Rodrigues formula variations, quaternion conjugation and intrinsic connections." *Mechanism and Machine Theory*, 92, 144-152. Elsevier, 2015.

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

## **See Also**

Quaternions to Rodrigues | Rodrigues to Direction Cosine Matrix | Rodrigues to Quaternions |  
Rodrigues to Rotation Angles | Rotation Angles to Rodrigues

**Introduced in R2017a**

## Direction Cosine Matrix to Quaternions

Convert direction cosine matrix to quaternion vector

**Library:** Aerospace Blockset / Utilities / Axes Transformations



### Description

The Direction Cosine Matrix to Quaternions block transforms a 3-by-3 direction cosine matrix (DCM) into a four-element unit quaternion vector ( $q_0, q_1, q_2, q_3$ ). Aerospace Blockset uses quaternions that are defined using the scalar-first convention. The DCM performs the coordinate transformation of a vector in inertial axes to a vector in body axes. For more information on the direction cosine matrix, see “Algorithms” on page 4-249.

### Ports

#### Input

##### **DCM<sub>be</sub> — Direction cosine matrix**

3-by-3 matrix

Direction cosine matrix to transform the direction cosine matrix to quaternions, specified as a 3-by-3.

Data Types: double

#### Output

##### **q — Quaternion**

4-by-1 vector

Quaternion returned by transformation as a 4-by-1 vector.

Data Types: double

### Parameters

#### **Action for invalid DCM — Block behavior**

None (default) | Warning | Error

Block behavior when the direction cosine matrix is invalid (not orthogonal).

- Warning — Displays warning indicating that the direction cosine matrix is invalid.
- Error — Displays error indicating that the direction cosine matrix is invalid.
- None — Does not display warning or error (default).

#### **Programmatic Use**

**Block Parameter:** action

**Type:** character vector

**Values:** 'None' | 'Warning' | 'Error'

**Default:** 'None'

Data Types: char | string

### Tolerance for DCM validation – Tolerance

eps(2) (default) | scalar

Tolerance of the direction cosine matrix validity, specified as a scalar. The block considers the direction cosine matrix valid if these conditions are true:

- The transpose of the direction cosine matrix times itself equals 1 within the specified tolerance ( $\text{transpose}(n)*n == 1 \pm \text{tolerance}$ ).
- The determinant of the direction cosine matrix equals 1 within the specified tolerance ( $\det(n) == 1 \pm \text{tolerance}$ ).

### Programmatic Use

**Block Parameter:** tolerance

**Type:** character vector

**Values:** 'eps(2)' | scalar

**Default:** 'eps(2)'

Data Types: double

## Algorithms

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, a number of calculations arrive at the correct quaternion. The first of these is to calculate the trace of the DCM to determine which 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.

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

## See Also

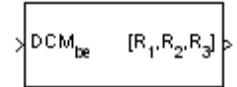
Direction Cosine Matrix to Rotation Angles | Quaternions to Direction Cosine Matrix | Quaternions to Rotation Angles | Rotation Angles to Direction Cosine Matrix | Rotation Angles to Quaternions

Introduced before R2006a

## Direction Cosine Matrix to Rotation Angles

Convert direction cosine matrix to rotation angles

**Library:** Aerospace Blockset / Utilities / Axes Transformations



### Description

The Direction Cosine Matrix to Rotation Angles block converts the first, second, and third rotation angles of a 3-by-3 direction cosine matrix (DCM) into the rotation angles R1, R2, and R3, respectively. The DCM matrix performs the coordinate transformation of a vector in inertial axes into a vector in body axes. The block **Rotation Order** parameter specifies the order of the block output rotations. For example, if **Rotation Order** has a value of ZYX, the block outputs are in the rotation order z-y-x (psi theta phi).

### Ports

#### Input

**DCM<sub>be</sub> — Direction cosine matrix**

3-by-3 matrix

Direction cosine matrix from which to determine the rotation angles, specified as a 3-by-3 matrix.

Data Types: double

#### Output

**[R<sub>1</sub>, R<sub>2</sub>, R<sub>3</sub>] — Rotation angles**

3-by-1 vector

Rotation angles, returned as a 3-by-1 vector, in radians.

Data Types: double

### Parameters

**Rotation Order — Block output rotation order**

ZYX (default) | ZYZ | ZXY | ZXZ | YXZ | YXY | YZX | YZY | XYZ | YXX | XZY | XZX

Rotation order for three wind rotation angles.

For the ZYX, ZXY, YXZ, YZX, XYZ, and XZY rotations, the block generates an R2 angle that lies between  $\pm\pi/2$  radians, and R1 and R3 angles that lie between  $\pm\pi$  radians.

For the 'ZYZ', 'ZXZ', 'YXY', 'YZY', 'YXX', and 'XZX' rotations, the block generates an R2 angle that lies between 0 and pi radians, and R1 and R3 angles that lie between  $\pm\pi$  radians. However, in the latter case, R3 is set to 0 radians.



**Programmatic Use****Block Parameter:** rotationOrder**Type:** character vector**Values:** 'ZYX' | 'ZYZ' | 'ZXY' | 'ZXZ' | 'YXZ' | 'YXY' | 'YZX' | 'YZY' | 'XYZ' | 'XYX' | 'XZY' | 'XZX'**Default:** 'ZYX'**Action for invalid DCM — Block behavior**

None (default) | Warning | Error

Block behavior when the direction cosine matrix is invalid (not orthogonal).

- Warning — Displays warning indicating that the direction cosine matrix is invalid.
- Error — Displays error indicating that the direction cosine matrix is invalid.
- None — Does not display warning or error (default).

**Programmatic Use****Block Parameter:** action**Type:** character vector**Values:** 'None' | 'Warning' | 'Error'**Default:** 'None'

Data Types: char | string

**Tolerance for DCM validation — Tolerance**

eps(2) (default) | scalar

Tolerance of the direction cosine matrix validity, specified as a scalar. The block considers the direction cosine matrix valid if these conditions are true:

- The transpose of the direction cosine matrix times itself equals 1 within the specified tolerance ( $\text{transpose}(n) * n == 1 \pm \text{tolerance}$ ).
- The determinant of the direction cosine matrix equals 1 within the specified tolerance ( $\det(n) == 1 \pm \text{tolerance}$ ).

**Programmatic Use****Block Parameter:** tolerance**Type:** character vector**Values:** 'eps(2)' | scalar**Default:** 'eps(2)'

Data Types: double

**Extended Capabilities****C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**See Also**

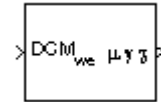
Direction Cosine Matrix to Quaternions | Quaternions to Direction Cosine Matrix | Rotation Angles to Direction Cosine Matrix

**Introduced in R2007b**

# Direction Cosine Matrix to Wind Angles

Convert direction cosine matrix to wind angles

**Library:** Aerospace Blockset / Utilities / Axes Transformations



## Description

The Direction Cosine Matrix to Wind Angles block converts a 3-by-3 direction cosine matrix (DCM) into three wind rotation angles. The DCM matrix performs the coordinate transformation of a vector in earth axes ( $ox_0, oy_0, oz_0$ ) into a vector in wind axes ( $ox_3, oy_3, oz_3$ ). For more information on the direction cosine matrix, see “Algorithms” on page 4-254.

This implementation generates a flight path angle that lies between  $\pm 90$  degrees, and bank and heading angles that lie between  $\pm 180$  degrees.

## Ports

### Input

#### **DCM<sub>we</sub> — Direction cosine matrix**

3-by-3 matrix

Direction cosine matrix, specified as a 3-by-3 direction cosine matrix, to transform Earth-fixed vectors to wind-fixed vectors.

Data Types: double

### Output

#### **$\mu \ \gamma \ x$ — Wind angles**

3-by-1 vector

Wind angles (bank, flight path, heading), returned as a 3-by-1 vector, in radians.

Data Types: double

## Parameters

#### **Action for invalid DCM — Block behavior**

None (default) | Warning | Error

Block behavior when the direction cosine matrix is invalid (not orthogonal).

- **Warning** — Displays warning indicating that the direction cosine matrix is invalid.
- **Error** — Displays error indicating that the direction cosine matrix is invalid.
- **None** — Does not display warning or error (default).

**Programmatic Use****Block Parameter:** action**Type:** character vector**Values:** 'None' | 'Warning' | 'Error'**Default:** 'None'

Data Types: char | string

**Tolerance for DCM validation – Tolerance**

eps(2) (default) | scalar

Tolerance of the direction cosine matrix validity, specified as a scalar. The block considers the direction cosine matrix valid if these conditions are true:

- The transpose of the direction cosine matrix times itself equals 1 within the specified tolerance ( $\text{transpose}(n)*n == 1 \pm \text{tolerance}$ ).
- The determinant of the direction cosine matrix equals 1 within the specified tolerance ( $\det(n) == 1 \pm \text{tolerance}$ ).

**Programmatic Use****Block Parameter:** tolerance**Type:** character vector**Values:** 'eps(2)' | scalar**Default:** 'eps(2)'

Data Types: double

**Algorithms**

The DCM matrix performs the coordinate transformation of a vector in earth axes ( $ox_0, oy_0, oz_0$ ) into a vector in wind axes ( $ox_3, oy_3, oz_3$ ). The order of the axis rotations required to bring this about is:

- 1 A rotation about  $oz_0$  through the heading angle ( $\chi$ ) to axes ( $ox_1, oy_1, oz_1$ )
- 2 A rotation about  $oy_1$  through the flight path angle ( $\gamma$ ) to axes ( $ox_2, oy_2, oz_2$ )
- 3 A rotation about  $ox_2$  through the bank angle ( $\mu$ ) to axes ( $ox_3, oy_3, oz_3$ )

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

$$\begin{bmatrix} ox_3 \\ oy_3 \\ oz_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\mu & \sin\mu \\ 0 & -\sin\mu & \cos\mu \end{bmatrix} \begin{bmatrix} \cos\gamma & 0 & -\sin\gamma \\ 0 & 1 & 0 \\ \sin\gamma & 0 & \cos\gamma \end{bmatrix} \begin{bmatrix} \cos\chi & \sin\chi & 0 \\ -\sin\chi & \cos\chi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} ox_0 \\ oy_0 \\ oz_0 \end{bmatrix}$$

Combining the three axis transformation matrices defines the following DCM.

$$DCM_{we} = \begin{bmatrix} \cos\gamma\cos\chi & \cos\gamma\sin\chi & -\sin\gamma \\ (\sin\mu\sin\gamma\cos\chi - \cos\mu\sin\chi) & (\sin\mu\sin\gamma\sin\chi + \cos\mu\cos\chi) & \sin\mu\cos\gamma \\ (\cos\mu\sin\gamma\cos\chi + \sin\mu\sin\chi) & (\cos\mu\sin\gamma\sin\chi - \sin\mu\cos\chi) & \cos\mu\cos\gamma \end{bmatrix}$$

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

$$\mu = \operatorname{atan}\left(\frac{DCM(2, 3)}{DCM(3, 3)}\right)$$

$$\gamma = \operatorname{asin}(-DCM(1, 3))$$

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

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

### See Also

[Direction Cosine Matrix Body to Wind](#) | [Direction Cosine Matrix Body to Wind to Alpha and Beta](#) | [Direction Cosine Matrix to Rotation Angles](#) | [Rotation Angles to Direction Cosine Matrix](#) | [Wind Angles to Direction Cosine Matrix](#)

**Introduced before R2006a**

## Discrete Wind Gust Model

Generate discrete wind gust



## Library

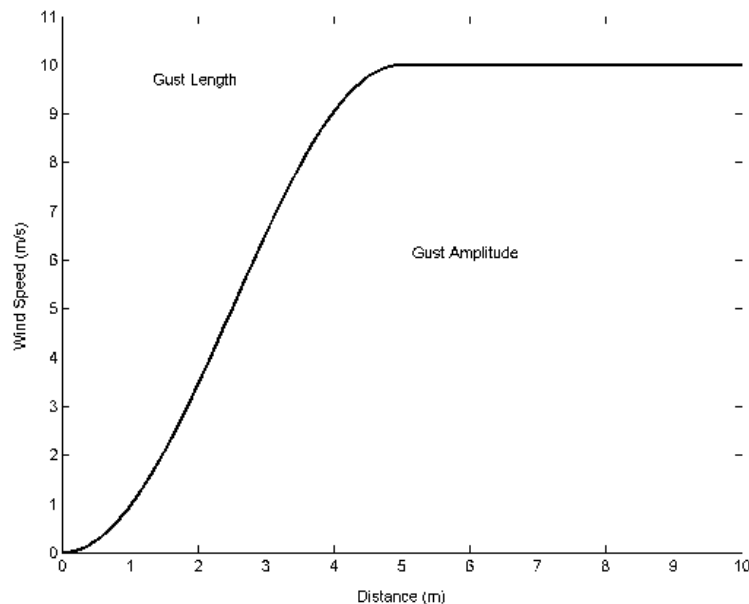
Environment/Wind

## 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. You specify 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.

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.

## Parameters

### Units

Define the units of wind gust.

| Units                      | Wind          | Altitude |
|----------------------------|---------------|----------|
| Metric (MKS)               | Meters/second | Meters   |
| English (Velocity in ft/s) | Feet/second   | Feet     |
| English (Velocity in kts)  | Knots         | Feet     |

### Gust in u-axis

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

### Gust in v-axis

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

### Gust in w-axis

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

### Gust start time (sec)

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

### Gust length [dx dy dz] (m or f)

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

### Gust amplitude [ug vg wg] (m/s, f/s, or knots)

The magnitude of the increase in wind speed caused by the gust in each axis. These values may be positive or negative.

## Inputs and Outputs

| Input  | Dimension Type | Description                                |
|--------|----------------|--|
| First  |                | Contains the airspeed in units selected.   |
| Output | Dimension Type | Description                                |
| First  |                | Contains the wind speed in units selected. |

## **Examples**

See Discrete Wind Gust Model in aeroblk\_HL20 for an example of this block.

## **Reference**

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

## **See Also**

Dryden Wind Turbulence Model (Continuous)

Dryden Wind Turbulence Model (Discrete)

Von Karman Wind Turbulence Model (Continuous)

Wind Shear Model

## **Extended Capabilities**

### **C/C++ Code Generation**

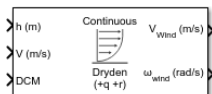
Generate C and C++ code using Simulink® Coder™.

**Introduced before R2006a**



# Dryden Wind Turbulence Model (Continuous)

Generate continuous wind turbulence with Dryden velocity spectra



## Library

Environment/Wind

## Description

The Dryden Wind Turbulence Model (Continuous) 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, Military Handbook MIL-HDBK-1797, Military Handbook MIL-HDBK-1797B.

Turbulence is 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  $\Omega$  radians per meter, the circular frequency  $\omega$  is calculated by multiplying  $V$  by  $\Omega$ . MIL-F-8785C and MIL-HDBK-1797/1797B provide these definitions of longitudinal, lateral, and vertical component spectra functions:

|                      | MIL-F-8785C  | MIL-HDBK-1797 and MIL-HDBK-1797B   |
|----------------------|--|--|
| <b>Longitudinal</b>  |  |  |
| $\Phi_u(\omega)$     | $\frac{2\sigma_u^2 L_u}{\pi V} \cdot \frac{1}{1 + (L_u \frac{\omega}{V})^2}$   | $\frac{2\sigma_u^2 L_u}{\pi V} \cdot \frac{1}{1 + (L_u \frac{\omega}{V})^2}$   |
| $\Phi_{p_g}(\omega)$ | $\frac{\sigma_w^2}{V L_w} \cdot \frac{0.8 \left( \frac{\pi L_w}{4b} \right)^{1/3}}{1 + \left( \frac{4b\omega}{\pi V} \right)^2}$ | $\frac{\sigma_w^2}{2V L_w} \cdot \frac{0.8 \left( \frac{2\pi L_w}{4b} \right)^{1/3}}{1 + \left( \frac{4b\omega}{\pi V} \right)^2}$ |
| <b>Lateral</b>       |  |  |
| $\Phi_v(\omega)$     | $\frac{\sigma_v^2 L_v}{\pi V} \cdot \frac{1 + 3(L_v \frac{\omega}{V})^2}{\left[ 1 + (L_v \frac{\omega}{V})^2 \right]^2}$         | $\frac{2\sigma_v^2 L_v}{\pi V} \cdot \frac{1 + 12(L_v \frac{\omega}{V})^2}{\left[ 1 + 4(L_v \frac{\omega}{V})^2 \right]^2}$        |
| $\Phi_r(\omega)$     | $\frac{\mp \left( \frac{\omega}{V} \right)^2}{1 + \left( \frac{3b\omega}{\pi V} \right)^2} \cdot \Phi_v(\omega)$                 | $\frac{\mp \left( \frac{\omega}{V} \right)^2}{1 + \left( \frac{3b\omega}{\pi V} \right)^2} \cdot \Phi_v(\omega)$                   |
| <b>Vertical</b>      |  |  |
| $\Phi_w(\omega)$     | $\frac{\sigma_w^2 L_w}{\pi V} \cdot \frac{1 + 3(L_w \frac{\omega}{V})^2}{\left[ 1 + (L_w \frac{\omega}{V})^2 \right]^2}$         | $\frac{2\sigma_w^2 L_w}{\pi V} \cdot \frac{1 + 12(L_w \frac{\omega}{V})^2}{\left[ 1 + 4(L_w \frac{\omega}{V})^2 \right]^2}$        |

|                  | MIL-F-8785C  | MIL-HDBK-1797 and MIL-HDBK-1797B   |
|------------------|--|--|
| $\Phi_q(\omega)$ | $\frac{\pm \left(\frac{\omega}{V}\right)^2}{1 + \left(\frac{4b\omega}{\pi V}\right)^2} \cdot \Phi_w(\omega)$ | $\frac{\pm \left(\frac{\omega}{V}\right)^2}{1 + \left(\frac{4b\omega}{\pi V}\right)^2} \cdot \Phi_w(\omega)$ |

where:

- $b$  represents the aircraft wingspan.
- $L_u, L_v, L_w$  represent the turbulence scale lengths.
- $\sigma_u, \sigma_v, \sigma_w$  represent the turbulence intensities.

The spectral density definitions of turbulence angular rates are defined in the specifications as three variations:

$$\begin{array}{lll}
 p_g = \frac{\partial w_g}{\partial y} & q_g = \frac{\partial w_g}{\partial x} & r_g = -\frac{\partial v_g}{\partial x} \\
 p_g = \frac{\partial w_g}{\partial y} & q_g = \frac{\partial w_g}{\partial x} & r_g = \frac{\partial v_g}{\partial x} \\
 p_g = -\frac{\partial w_g}{\partial y} & q_g = -\frac{\partial w_g}{\partial x} & r_g = \frac{\partial v_g}{\partial x}
 \end{array}$$

The variations affect only the vertical ( $q_g$ ) and lateral ( $r_g$ ) turbulence angular rates.

The longitudinal turbulence angular rate spectrum,

$$\Phi_{p_g}(\omega)$$

is a rational function. The rational function is derived from curve-fitting a complex algebraic function, not the vertical turbulence velocity spectrum,  $\Phi_w(\omega)$ , multiplied by a scale factor. The variations exist because the turbulence angular rate spectra contribute less to the aircraft gust response than the turbulence velocity.

The variations result in these combinations of vertical and lateral turbulence angular rate spectra.

| Vertical          | Lateral           |
|-------------------|-------------------|
| $\Phi_q(\omega)$  | $-\Phi_r(\omega)$ |
| $\Phi_q(\omega)$  | $\Phi_r(\omega)$  |
| $-\Phi_q(\omega)$ | $\Phi_r(\omega)$  |

To generate a signal with correct characteristics, a band-limited white noise signal is passed through forming filters. The forming filters are derived from the spectral square roots of the spectrum equations.

MIL-F-8785C and MIL-HDBK-1797/1797B provide these transfer functions:

|                     | MIL-F-8785C | MIL-HDBK-1797 and MIL-HDBK-1797B |
|---------------------|-------------|----------------------------------|
| <b>Longitudinal</b> |             |                                  |

|                 | MIL-F-8785C  | MIL-HDBK-1797 and MIL-HDBK-1797B  |
|-----------------|--|---|
| $H_u(s)$        | $\sigma_u \sqrt{\frac{2L_u}{\pi V}} \cdot \frac{1}{1 + \frac{L_u}{V}s}$                                  | $\sigma_u \sqrt{\frac{2L_u}{\pi V}} \cdot \frac{1}{1 + \frac{L_u}{V}s}$                                     |
| $H_p(s)$        | $\sigma_w \sqrt{\frac{0.8}{V}} \cdot \frac{(\frac{\pi}{4b})^{1/6}}{L_w^{1/3} (1 + (\frac{4b}{\pi V})s)}$ | $\sigma_w \sqrt{\frac{0.8}{V}} \cdot \frac{(\frac{\pi}{4b})^{1/6}}{(2L_w)^{1/3} (1 + (\frac{4b}{\pi V})s)}$ |
| <b>Lateral</b>  |  |   |
| $H_v(s)$        | $\sigma_v \sqrt{\frac{L_v}{\pi V}} \cdot \frac{1 + \frac{\sqrt{3}L_v}{V}s}{(1 + \frac{L_v}{V}s)^2}$      | $\sigma_v \sqrt{\frac{2L_v}{\pi V}} \cdot \frac{1 + \frac{2\sqrt{3}L_v}{V}s}{(1 + \frac{2L_v}{V}s)^2}$      |
| $H_r(s)$        | $\frac{\mp \frac{s}{V}}{(1 + (\frac{3b}{\pi V})s)} \cdot H_v(s)$   | $\frac{\mp \frac{s}{V}}{(1 + (\frac{3b}{\pi V})s)} \cdot H_v(s)$  |
| <b>Vertical</b> |  |   |
| $H_w(s)$        | $\sigma_w \sqrt{\frac{L_w}{\pi V}} \cdot \frac{1 + \frac{\sqrt{3}L_w}{V}s}{(1 + \frac{L_w}{V}s)^2}$      | $\sigma_w \sqrt{\frac{2L_w}{\pi V}} \cdot \frac{1 + \frac{2\sqrt{3}L_w}{V}s}{(1 + \frac{2L_w}{V}s)^2}$      |
| $H_q(s)$        | $\frac{\pm \frac{s}{V}}{(1 + (\frac{4b}{\pi V})s)} \cdot H_w(s)$   | $\frac{\pm \frac{s}{V}}{(1 + (\frac{4b}{\pi V})s)} \cdot H_w(s)$  |

Divided into two distinct regions, the turbulence scale lengths and intensities are functions of altitude.

**Note** The military specifications result in the same transfer function after evaluating the turbulence scale lengths. The differences in turbulence scale lengths and turbulence transfer functions balance offset.

### Low-Altitude Model (Altitude Under 1000 Feet)

According to the military references, the turbulence scale lengths at low altitudes, where  $h$  is the altitude in feet, are represented in the following table:

| MIL-F-8785C                                       | MIL-HDBK-1797 and MIL-HDBK-1797B                   |
|---|--|
| $L_w = h$   | $2L_w = h$   |
| $L_u = L_v = \frac{h}{(0.177 + 0.000823h)^{1.2}}$ | $L_u = 2L_v = \frac{h}{(0.177 + 0.000823h)^{1.2}}$ |

Typically, at 20 feet (6 meters) the wind speed is 15 knots in light turbulence, 30 knots in moderate turbulence, and 45 knots for severe turbulence. See these turbulence intensities, where  $W_{20}$  is the wind speed at 20 feet (6 meters).

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

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

The turbulence axes orientation in this region is defined:

- Longitudinal turbulence velocity,  $u_g$ , aligned along the horizontal relative mean wind vector
- Vertical turbulence velocity,  $w_g$ , aligned with vertical

At this altitude range, the output of the block is transformed into body coordinates.

**Medium/High Altitudes (Altitude Above 2000 Feet)**

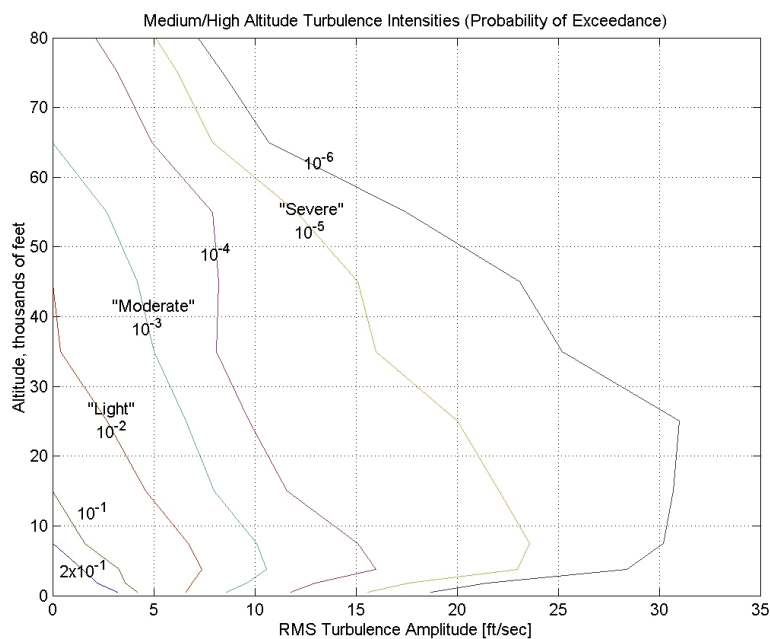
Turbulence scale lengths and intensities for medium-to-high altitudes the are based on the assumption that the turbulence is isotropic. MIL-F-8785C and MIL-HDBK-1797/1797B provide these representations of scale lengths:

| MIL-F-8785C                 | MIL-HDBK-1797 and MIL-HDBK-1797B |
|-----------------------------|----------------------------------|
| $L_u = L_v = L_w = 1750$ ft | $L_u = 2L_v = 2L_w = 1750$ ft    |

The turbulence intensities are determined from a lookup table that provides the turbulence intensity as a function of altitude and the probability of the turbulence intensity being exceeded. The relationship of the turbulence intensities is represented in the following equation:

$$\sigma_u = \sigma_v = \sigma_w.$$

The turbulence axes orientation in this region is defined as being aligned with the body coordinates.



### Between Low and Medium/High Altitudes (Between 1000 and 2000 Feet)

At altitudes between 1000 and 2000, the turbulence velocities and turbulence angular rates are determined by linearly interpolating between the value from the low-altitude model at 1000 feet transformed from mean horizontal wind coordinates to body coordinates and the value from the high-altitude model at 2000 feet in body coordinates.

## Parameters

### Units

Define the units of wind speed due to the turbulence.

| Units                      | Wind Velocity | Altitude | Airspeed      |
|----------------------------|---------------|----------|---------------|
| Metric (MKS)               | Meters/second | Meters   | Meters/second |
| English (Velocity in ft/s) | Feet/second   | Feet     | Feet/second   |
| English (Velocity in kts)  | Knots         | Feet     | Knots         |

### Specification

Define which military reference to use. This affects the application of turbulence scale lengths in the lateral and vertical directions.

### Model type

Select the wind turbulence model to use.

|                               |   |
|-------------------------------|---|
| Continuous Von Karman (+q -r) | Use continuous representation of Von Kármán velocity spectra with positive vertical and negative lateral angular rates spectra. |
| Continuous Von Karman (+q +r) | Use continuous representation of Von Kármán velocity spectra with positive vertical and lateral angular rates spectra.          |
| Continuous Von Karman (-q +r) | Use continuous representation of Von Kármán velocity spectra with negative vertical and positive lateral angular rates spectra. |
| Continuous Dryden (+q -r)     | Use continuous representation of Dryden velocity spectra with positive vertical and negative lateral angular rates spectra.     |
| Continuous Dryden (+q +r)     | Use continuous representation of Dryden velocity spectra with positive vertical and lateral angular rates spectra.              |
| Continuous Dryden (-q +r)     | Use continuous representation of Dryden velocity spectra with negative vertical and positive lateral angular rates spectra.     |
| Discrete Dryden (+q -r)       | Use discrete representation of Dryden velocity spectra with positive vertical and negative lateral angular rates spectra.       |

|                         |   |
|-------------------------|---|
| Discrete Dryden (+q +r) | Use discrete representation of Dryden velocity spectra with positive vertical and lateral angular rates spectra.          |
| Discrete Dryden (-q +r) | Use discrete representation of Dryden velocity spectra with negative vertical and positive lateral angular rates spectra. |

The Continuous Dryden selections conform to the transfer function descriptions.

#### **Wind speed at 6 m defines the low altitude intensity**

Measured wind speed at a height of 6 meters (20 feet) provides the intensity for the low-altitude turbulence model.

#### **Wind direction at 6 m (degrees clockwise from north)**

Measured wind direction at a height of 6 meters (20 feet) is an angle to aid in transforming the low-altitude turbulence model into a body coordinates.

#### **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 exceeding the turbulence intensity.

#### **Scale length at medium/high altitudes (m)**

Turbulence scale length above 2000 feet, assumed constant. MIL-F-8785C and MIL-HDBK-1797/1797B recommend 1750 feet for the longitudinal turbulence scale length of the Dryden spectra.

---

**Note** An alternative scale length value changes the power spectral density asymptote and gust load.

---

#### **Wingspan**

Wingspan required in the calculation of the turbulence on the angular rates.

#### **Band-limited noise sample time (sec)**

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

#### **Noise seeds**

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 this parameter generates the turbulence signals.

#### **Ports**

| Input  | Dimension Type | Description                                     |
|--------|----------------|---|
| First  | scalar         | Contains the altitude, in units selected.       |
| Second | scalar         | Contains the aircraft speed, in units selected. |

| Input  | Dimension Type       | Description   |
|--------|----------------------|---|
| Third  | 3-by-3 matrix        | Contains the direction cosine matrix.                         |
| Output | Dimension Type       | Description   |
| First  | Three-element signal | Contains the turbulence velocities, in the selected units.    |
| Second | Three-element signal | Contains the turbulence angular rates, in radians per second. |

## Limitations

The frozen turbulence field assumption is valid for the cases of mean-wind velocity and the root-mean-square turbulence velocity, or intensity, is small relative to the aircraft ground speed.

The turbulence model describes an average of all conditions for clear air turbulence. These factors are not incorporated into the model:

- Terrain roughness
- Lapse rate
- Wind shears
- Mean wind magnitude
- Other meteorological factors

## Examples

See Airframe/Environment Models/Wind Models in `aeroblk_HL20` for an example of this block.

## References

Chalk, Charles, T.P. Neal, T.M. Harris, Francis E. Pritchard, and Robert J. Woodcock. *Background Information and User Guide for MIL-F-8785B(ASG), "Military Specification-Flying Qualities of Piloted Airplanes."* AD869856. Buffalo, NY: Cornell Aeronautical Laboratory, 1969.

*Flying Qualities of Piloted Aircraft.* Department of Defense Handbook. MIL-HDBK-1797. Washington, DC: U.S. Department of Defense, 1997.

*Flying Qualities of Piloted Aircraft.* Department of Defense Handbook. MIL-HDBK-1797B. Washington, DC: U.S. Department of Defense, 2012.

*Flying Qualities of Piloted Airplanes.* U.S. Military Specification MIL-F-8785C. Washington, D.C.: U.S. Department of Defense, 1980.

Hoblit, F., *Gust Loads on Aircraft: Concepts and Applications*, AIAA Education Series, 1988.

Ly, U. and Y. Chan. "Time-Domain Computation of Aircraft Gust Covariance Matrices," AIAA Paper 80-1615, presented at the 6th Atmospheric Flight Mechanics Conference, Danvers, Massachusetts, August 1980.

McFarland, Richard E, A Standard Kinematic Model for Flight Simulation at NASA-AMES. NASA CR-2497. Mountain view, CA: Computer Sciences Corporation, 1975.

McRuer, Duane, Dunstan Graham, and Irving Ashkenas. *Aircraft Dynamics and Automatic Control* Princeton University Press, 1974, R1990.

Moorhouse, David J. and Robert J. Woodcock. *Background Information and User Guide for MIL-F-8785C, "Military Specification—Flying Qualities of Piloted Airplanes."* ADA119421. Wright-Patterson AFB, OH: Air Force Wright Aeronautical Labs, 1982.

Tatom, Frank B., George H. Fichtl, and Stephen R. Smith. "Simulation of Atmospheric Turbulent Gusts and Gust Gradients," AIAA Paper 81-0300, presented at the 19th Aerospace Sciences Meeting, St. Louis, Missouri, January 1981.

Yeager, Jessie, *Implementation and Testing of Turbulence Models for the F18-HARV Simulation* NASA CR-1998-206937. Hampton, VA: Lockheed Martin Engineering & Sciences, 1998.

## **See Also**

Dryden Wind Turbulence Model (Discrete)

Discrete Wind Gust Model

Wind Shear Model

Von Karman Wind Turbulence Model (Continuous)

## **Extended Capabilities**

### **C/C++ Code Generation**

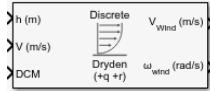
Generate C and C++ code using Simulink® Coder™.

**Introduced before R2006a**



# Dryden Wind Turbulence Model (Discrete)

Generate discrete wind turbulence with Dryden velocity spectra



## Library

Environment/Wind

## Description

The Dryden Wind Turbulence Model (Discrete) block uses the Dryden spectral representation to add turbulence to the aerospace model by using band-limited white noise with appropriate digital filter finite difference equations. This block implements the mathematical representation in the Military Specification MIL-F-8785C, Military Handbook MIL-HDBK-1797, and Military Handbook MIL-HDBK-1797B.

According to the military references, turbulence is 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  $\Omega$  radians per meter, the circular frequency  $\omega$  is calculated by multiplying  $V$  by  $\Omega$ . The following table displays the component spectra functions:

|                     | MIL-F-8785C  | MIL-HDBK-1797 and MIL-HDBK-1797B   |
|---------------------|--|--|
| <b>Longitudinal</b> |  |  |
| $\Phi_u(\omega)$    | $\frac{2\sigma_u^2 L_u}{\pi V} \cdot \frac{1}{1 + (L_u \frac{\omega}{V})^2}$   | $\frac{2\sigma_u^2 L_u}{\pi V} \cdot \frac{1}{1 + (L_u \frac{\omega}{V})^2}$   |
| $\Phi_p(\omega)$    | $\frac{\sigma_w^2}{V L_w} \cdot \frac{0.8 \left( \frac{\pi L_w}{4b} \right)^{1/3}}{1 + \left( \frac{4b\omega}{\pi V} \right)^2}$ | $\frac{\sigma_w^2}{2V L_w} \cdot \frac{0.8 \left( \frac{2\pi L_w}{4b} \right)^{1/3}}{1 + \left( \frac{4b\omega}{\pi V} \right)^2}$ |
| <b>Lateral</b>      |  |  |
| $\Phi_v(\omega)$    | $\frac{\sigma_v^2 L_v}{\pi V} \cdot \frac{1 + 3(L_v \frac{\omega}{V})^2}{\left[ 1 + (L_v \frac{\omega}{V})^2 \right]^2}$         | $\frac{2\sigma_v^2 L_v}{\pi V} \cdot \frac{1 + 12(L_v \frac{\omega}{V})^2}{\left[ 1 + 4(L_v \frac{\omega}{V})^2 \right]^2}$        |
| $\Phi_r(\omega)$    | $\frac{\mp \left( \frac{\omega}{V} \right)^2}{1 + \left( \frac{3b\omega}{\pi V} \right)^2} \cdot \Phi_v(\omega)$                 | $\frac{\mp \left( \frac{\omega}{V} \right)^2}{1 + \left( \frac{3b\omega}{\pi V} \right)^2} \cdot \Phi_v(\omega)$                   |
| <b>Vertical</b>     |  |  |

|                  | MIL-F-8785C   | MIL-HDBK-1797 and MIL-HDBK-1797B   |
|------------------|---|--|
| $\Phi_w(\omega)$ | $\frac{\sigma_w^2 L_w}{\pi V} \cdot \frac{1 + 3(L_w \frac{\omega}{V})^2}{[1 + (L_w \frac{\omega}{V})^2]^2}$ | $\frac{2\sigma_w^2 L_w}{\pi V} \cdot \frac{1 + 12(L_w \frac{\omega}{V})^2}{[1 + 4(L_w \frac{\omega}{V})^2]^2}$ |
| $\Phi_q(\omega)$ | $\frac{\pm (\frac{\omega}{V})^2}{1 + (\frac{4b\omega}{\pi V})^2} \cdot \Phi_w(\omega)$                      | $\frac{\pm (\frac{\omega}{V})^2}{1 + (\frac{4b\omega}{\pi V})^2} \cdot \Phi_w(\omega)$                         |

The variable  $b$  represents the aircraft wingspan. The variables  $L_u, L_v, L_w$  represent the turbulence scale lengths. The variables  $\sigma_u, \sigma_v, \sigma_w$  represent the turbulence intensities.

The spectral density definitions of turbulence angular rates are defined in the references as three variations, which are displayed in the following table:

$$\begin{array}{lll}
 p_g = \frac{\partial w_g}{\partial y} & q_g = \frac{\partial w_g}{\partial x} & r_g = -\frac{\partial v_g}{\partial x} \\
 p_g = \frac{\partial w_g}{\partial y} & q_g = \frac{\partial w_g}{\partial x} & r_g = \frac{\partial v_g}{\partial x} \\
 p_g = -\frac{\partial w_g}{\partial y} & q_g = -\frac{\partial w_g}{\partial x} & r_g = \frac{\partial v_g}{\partial x}
 \end{array}$$

The variations affect only the vertical ( $q_g$ ) and lateral ( $r_g$ ) turbulence angular rates.

Keep in mind that the longitudinal turbulence angular rate spectrum,  $\Phi_p(\omega)$ , is a rational function. The rational function is derived from curve-fitting a complex algebraic function, not the vertical turbulence velocity spectrum,  $\Phi_w(\omega)$ , multiplied by a scale factor. Because the turbulence angular rate spectra contribute less to the aircraft gust response than the turbulence velocity spectra, it may explain the variations in their definitions.

The variations lead to the following combinations of vertical and lateral turbulence angular rate spectra:

| Vertical          | Lateral           |
|-------------------|-------------------|
| $\Phi_q(\omega)$  | $-\Phi_r(\omega)$ |
| $\Phi_q(\omega)$  | $\Phi_r(\omega)$  |
| $-\Phi_q(\omega)$ | $\Phi_r(\omega)$  |

To generate a signal with the correct characteristics, a unit variance, band-limited white noise signal is used in the digital filter finite difference equations.

The following table displays the digital filter finite difference equations:

|                     | MIL-F-8785C | MIL-HDBK-1797 and MIL-HDBK-1797B |
|---------------------|-------------|----------------------------------|
| <b>Longitudinal</b> |             |                                  |

|                 | MIL-F-8785C   | MIL-HDBK-1797 and MIL-HDBK-1797B  |
|-----------------|---|---|
| $u_g$           | $\left(1 - \frac{V}{L_u}T\right)u_g + \sqrt{2\frac{V}{L_u}T\frac{\sigma_u}{\sigma_\eta}}\eta_1$   | $\left(1 - \frac{V}{L_u}T\right)u_g + \sqrt{2\frac{V}{L_u}T\frac{\sigma_u}{\sigma_\eta}}\eta_1$   |
| $p_g$           | $\left(1 - \frac{2.6}{\sqrt{L_w b}}T\right)p_g +$<br>$\left(\sqrt{2\frac{2.6}{\sqrt{L_w b}}T}\right)\left(\frac{0.95}{\sqrt[3]{2L_w b^2}}\frac{\sigma_w}{\sigma_\eta}\eta_4\right)\sigma_w$ | MIL-HDBK-1797<br>$\left(1 - \frac{2.6}{\sqrt{2L_w b}}T\right)p_g +$<br>$\left(\sqrt{2\frac{2.6}{\sqrt{2L_w b}}T}\right)\left(\frac{1.9}{\sqrt{2L_w b}}\frac{\sigma_w}{\sigma_\eta}\eta_4\right)\sigma_w$    |
|                 |   | MIL-HDBK-1797B<br>$\left(1 - \frac{2.6V}{\sqrt{2L_w b}}T\right)p_g +$<br>$\left(\sqrt{2\frac{2.6V}{\sqrt{2L_w b}}T}\right)\left(\frac{1.9}{\sqrt{2L_w b}}\frac{\sigma_w}{\sigma_\eta}\eta_4\right)\sigma_w$ |
| <b>Lateral</b>  |   |   |
| $v_g$           | $\left(1 - \frac{V}{L_u}T\right)v_g + \sqrt{2\frac{V}{L_u}T\frac{\sigma_v}{\sigma_\eta}}\eta_2$   | $\left(1 - \frac{V}{L_u}T\right)v_g + \sqrt{2\frac{V}{L_u}T\frac{\sigma_v}{\sigma_\eta}}\eta_2$   |
| $r_g$           | $\left(1 - \frac{\pi V}{3b}T\right)r_g \mp \frac{\pi}{3b}(v_g - v_{g\text{past}})$  | $\left(1 - \frac{\pi V}{3b}T\right)r_g \mp \frac{\pi}{3b}(v_g - v_{g\text{past}})$  |
| <b>Vertical</b> |   |   |
| $w_g$           | $\left(1 - \frac{V}{L_u}T\right)w_g + \sqrt{2\frac{V}{L_u}T\frac{\sigma_w}{\sigma_\eta}}\eta_3$   | $\left(1 - \frac{V}{L_u}T\right)w_g + \sqrt{2\frac{V}{L_u}T\frac{\sigma_w}{\sigma_\eta}}\eta_3$   |
| $q_g$           | $\left(1 - \frac{\pi V}{4b}T\right)q_g \pm \frac{\pi}{4b}(w_g - w_{g\text{past}})$  | $\left(1 - \frac{\pi V}{4b}T\right)q_g \pm \frac{\pi}{4b}(w_g - w_{g\text{past}})$  |

Divided into two distinct regions, the turbulence scale lengths and intensities are functions of altitude.

**Low-Altitude Model (Altitude < 1000 feet)**

According to the military references, the turbulence scale lengths at low altitudes, where  $h$  is the altitude in feet, are represented in the following table:

| MIL-F-8785C                                       | MIL-HDBK-1797 and MIL-HDBK-1797B                   |
|---|--|
| $L_w = h$   | $2L_w = h$   |
| $L_u = L_v = \frac{h}{(0.177 + 0.000823h)^{1.2}}$ | $L_u = 2L_v = \frac{h}{(0.177 + 0.000823h)^{1.2}}$ |

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.

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

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

The turbulence axes orientation in this region is defined as follows:

- Longitudinal turbulence velocity,  $u_g$ , aligned along the horizontal relative mean wind vector
- Vertical turbulence velocity,  $w_g$ , aligned with vertical.

At this altitude range, the output of the block is transformed into body coordinates.

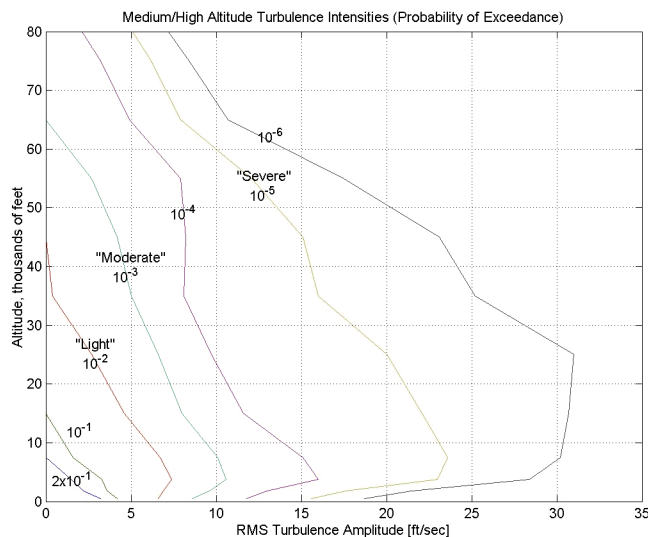
### Medium/High Altitudes (Altitude > 2000 feet)

For medium to high altitudes the turbulence scale lengths and intensities are based on the assumption that the turbulence is isotropic. In the military references, the scale lengths are represented by the following equations:

| MIL-F-8785C                 | MIL-HDBK-1797 and MIL-HDBK-1797B |
|-----------------------------|----------------------------------|
| $L_u = L_v = L_w = 1750$ ft | $L_u = 2L_v = 2L_w = 1750$ ft    |

The turbulence intensities are determined from a lookup table that provides the turbulence intensity as a function of altitude and the probability of the turbulence intensity being exceeded. The relationship of the turbulence intensities is represented in the following equation:  $\sigma_u = \sigma_v = \sigma_w$ .

The turbulence axes orientation in this region is defined as being aligned with the body coordinates.



### Between Low and Medium/High Altitudes (1000 feet < Altitude < 2000 feet)

At altitudes between 1000 feet and 2000 feet, the turbulence velocities and turbulence angular rates are determined by linearly interpolating between the value from the low altitude model at 1000 feet transformed from mean horizontal wind coordinates to body coordinates and the value from the high altitude model at 2000 feet in body coordinates.

## Parameters

### Units

Define the units of wind speed due to the turbulence.

| Units                      | Wind Velocity | Altitude | Airspeed      |
|----------------------------|---------------|----------|---------------|
| Metric (MKS)               | Meters/second | Meters   | Meters/second |
| English (Velocity in ft/s) | Feet/second   | Feet     | Feet/second   |
| English (Velocity in kts)  | Knots         | Feet     | Knots         |

### Specification

Define which military reference to use. This affects the application of turbulence scale lengths in the lateral and vertical directions

### Model type

Select the wind turbulence model to use:

|                               |   |
|-------------------------------|---|
| Continuous Von Karman (+q -r) | Use continuous representation of Von Kármán velocity spectra with positive vertical and negative lateral angular rates spectra. |
| Continuous Von Karman (+q +r) | Use continuous representation of Von Kármán velocity spectra with positive vertical and lateral angular rates spectra.          |
| Continuous Von Karman (-q +r) | Use continuous representation of Von Kármán velocity spectra with negative vertical and positive lateral angular rates spectra. |
| Continuous Dryden (+q -r)     | Use continuous representation of Dryden velocity spectra with positive vertical and negative lateral angular rates spectra.     |
| Continuous Dryden (+q +r)     | Use continuous representation of Dryden velocity spectra with positive vertical and lateral angular rates spectra.              |
| Continuous Dryden (-q +r)     | Use continuous representation of Dryden velocity spectra with negative vertical and positive lateral angular rates spectra.     |
| Discrete Dryden (+q -r)       | Use discrete representation of Dryden velocity spectra with positive vertical and negative lateral angular rates spectra.       |
| Discrete Dryden (+q +r)       | Use discrete representation of Dryden velocity spectra with positive vertical and lateral angular rates spectra.                |
| Discrete Dryden (-q +r)       | Use discrete representation of Dryden velocity spectra with negative vertical and positive lateral angular rates spectra.       |

The Discrete Dryden selections conform to the transfer function descriptions.

**Wind speed at 6 m defines the low altitude intensity**

The measured wind speed at a height of 6 meters (20 feet) provides the intensity for the low-altitude turbulence model.

**Wind direction at 6 m (degrees clockwise from north)**

The measured wind direction at a height of 6 meters (20 feet) is an angle to aid in transforming the low-altitude turbulence model into a body coordinates.

**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 the military references, a figure of 1750 feet is recommended for the longitudinal turbulence scale length of the Dryden spectra.

---

**Note** An alternate scale length value changes the power spectral density asymptote and gust load.

---

**Wingspan**

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

**Band-limited noise and discrete filter sample time (sec)**

The sample time at which the unit variance white noise signal is generated and at which the discrete filters are updated.

**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**

| Input  | Dimension Type       | Description   |
|--------|----------------------|---|
| First  | scalar               | Contains the altitude, in units selected.                     |
| Second | scalar               | Contains the aircraft speed, in units selected.               |
| Third  | 3-by-3 matrix        | Contains the NED direction cosine matrix.                     |
| Output | Dimension Type       | Description   |
| First  | Three-element signal | Contains the turbulence velocities, in the selected units.    |
| Second | Three-element signal | Contains the turbulence angular rates, in radians per second. |

## Assumptions and Limitations

The “frozen turbulence field” assumption is valid for the cases of mean-wind velocity and the root-mean-square turbulence velocity, or intensity, is small relative to the aircraft's ground speed.

The turbulence model describes an average of all conditions for clear air turbulence because the following factors are not incorporated into the model:

- Terrain roughness
- Lapse rate
- Wind shears
- Mean wind magnitude
- Other meteorological factors (except altitude)

## References

U.S. Military Handbook MIL-HDBK-1797B, 9 April 2012.

U.S. Military Handbook MIL-HDBK-1797, 19 December 1997.

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

Chalk, C., Neal, P., Harris, T., Pritchard, F., Woodcock, R., “Background Information and User Guide for MIL-F-8785B(ASG), `Military Specification-Flying Qualities of Piloted Airplanes',” AD869856, Cornell Aeronautical Laboratory, August 1969.

Hoblitt, F., *Gust Loads on Aircraft: Concepts and Applications*, AIAA Education Series, 1988.

Ly, U., Chan, Y., “Time-Domain Computation of Aircraft Gust Covariance Matrices,” AIAA Paper 80-1615, Atmospheric Flight Mechanics Conference, Danvers, Massachusetts, August 11-13, 1980.

McRuer, D., Ashkenas, I., Graham, D., *Aircraft Dynamics and Automatic Control*, Princeton University Press, July 1990.

Moorhouse, D., Woodcock, R., “Background Information and User Guide for MIL-F-8785C, `Military Specification-Flying Qualities of Piloted Airplanes',” ADA119421, Flight Dynamic Laboratory, July 1982.

McFarland, R., “A Standard Kinematic Model for Flight Simulation at NASA-Ames,” NASA CR-2497, Computer Sciences Corporation, January 1975.

Tatom, F., Smith, R., Fichtl, G., “Simulation of Atmospheric Turbulent Gusts and Gust Gradients,” AIAA Paper 81-0300, Aerospace Sciences Meeting, St. Louis, Missouri, January 12-15, 1981.

Yeager, J., “Implementation and Testing of Turbulence Models for the F18-HARV Simulation,” NASA CR-1998-206937, Lockheed Martin Engineering & Sciences, March 1998.

## See Also

Dryden Wind Turbulence Model (Continuous)

Von Karman Wind Turbulence Model (Continuous)

Discrete Wind Gust Model

Wind Shear Model

## **Extended Capabilities**

### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

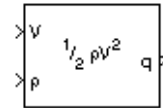
**Introduced before R2006a**



# Dynamic Pressure

Compute dynamic pressure using velocity and air density

**Library:** Aerospace Blockset / Flight Parameters



## Description

The Dynamic Pressure block computes dynamic pressure.

Dynamic pressure is defined as:

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

where  $\rho$  is air density and  $V$  is velocity.

## Ports

### Input

#### **V — Velocity**

three-element vector

Velocity, specified as a three-element vector.

Data Types: double

#### **$\rho$ — Air density**

scalar

Air density, specified as a scalar.

Data Types: double

### Output

#### **Output 1 — Dynamic pressure**

scalar

Dynamic pressure, returned as a scalar.

Data Types: double

## Extended Capabilities

### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**See Also**

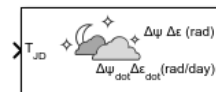
Aerodynamic Forces and Moments | Mach Number

**Introduced before R2006a**

# Earth Nutation

Implement Earth nutation

**Library:** Aerospace Blockset / Environment / Celestial Phenomena



## Description

The Earth Nutation block implements the International Astronomical Union (IAU) 1980 nutation series for a given Julian date. The block uses the Chebyshev coefficients that the NASA Jet Propulsion Laboratory provides.

The **Epoch** parameter controls the number of block inputs. If you select `Julian date`, the block has one input port, if you select `T0` and `elapsed Julian time`, the block has two input ports.

---

**Tip** For  $T_{JD}$ , Julian date input for the block:

- Calculate the date using the Julian Date Conversion block or the Aerospace Toolbox `juliandate` function.
  - Calculate the Julian date using some other means and input it using the Constant block.
- 

## Ports

### Input

#### $T_{JD}$ — Julian date

scalar | positive | between minimum and maximum Julian dates

Julian date, specified as a positive scalar between minimum and maximum Julian dates.

See the **Ephemeris model** parameter for the minimum and maximum Julian dates.

#### Dependencies

This port displays if the **Epoch** parameter is set to `Julian date`.

Data Types: `double`

#### $T0_{JD}$ — Fixed Julian date

scalar | positive

Fixed Julian date for a specific epoch that is the most recent midnight at or before the interpolation epoch, specified as a positive scalar. The sum of  $T0_{JD}$  and  $\Delta T_{JD}$  must fall between the minimum and maximum Julian dates.

See the **Ephemeris model** parameter for the minimum and maximum Julian dates.

#### Dependencies

This port displays if the **Epoch** parameter is set to `T0` and `elapsed Julian time`.

Data Types: double

### $\Delta T_{JD}$ — Elapsed Julian time

scalar | positive

Elapsed Julian time between the fixed Julian date and the ephemeris time, specified as a positive scalar. The sum of  $T0_{JD}$  and  $\Delta T_{JD}$  must fall between the minimum and maximum Julian date.

See the **Ephemeris model** parameter for the minimum and maximum Julian dates.

#### Dependencies

This port displays if the **Epoch** parameter is set to T0 and elapsed Julian time.

Data Types: double

#### Output

### $\Delta\psi \ \Delta\epsilon$ (rad) — Earth nutation

vector

Earth nutation, output as a vector of longitude ( $\Delta\psi$ ) and obliquity ( $\Delta\epsilon$ ), in rad.

Data Types: double

### $\Delta\psi_{\text{dot}} \ \Delta\epsilon_{\text{dot}}$ (rad/day) — Earth nutation angular rate

scalar

Earth nutation angular rate for the longitude ( $\Delta\psi_{\text{dot}}$ ) and obliquity ( $\Delta\epsilon_{\text{dot}}$ ), specified as a scalar in rad/day.

#### Dependencies

This port displays if the **Calculate rates** parameter is selected.

Data Types: double

## Parameters

### Epoch — Epoch

Julian date (default) | T0 and elapsed Julian time

Epoch, specified as:

- Julian date

Julian date to calculate the Earth nutation. When this option is selected, the block has one input port,  $T_{JD}$ .

- T0 and elapsed Julian time

Julian date, specified by two block inputs:

- Fixed Julian date representing a starting epoch.
- Elapsed Julian time between the  $T0_{JD}$  and the desired model simulation time. The sum of  $T0_{JD}$  and  $\Delta T_{JD}$  must fall between the minimum and maximum Julian dates.

**Programmatic Use****Block Parameter:** epochflag**Type:** character vector**Values:** Julian date | T0 and elapsed Julian time**Default:** 'Julian date'**Ephemeris model — Ephemeris model**

DE405 (default) | DE421 | DE423 | DE430

Select an Ephemeris model from the list defined by the Jet Propulsion Laboratory:

| Ephemeris Model | Description   |
|-----------------|---|
| DE405           | Released in 1998. This ephemeris takes into account the Julian date range 2305424.50 (December 9, 1599) to 2525008.50 (February 20, 2201).<br><br>This block implements these ephemerides with respect to the International Celestial Reference Frame version 1.0, adopted in 1998. |
| DE421           | Released in 2008. This ephemeris takes into account the Julian date range 2414992.5 (December 4, 1899) to 2469808.5 (January 2, 2050).<br><br>This block implements these ephemerides with respect to the International Celestial Reference Frame version 1.0, adopted in 1998.     |
| DE423           | Released in 2010. This ephemeris takes into account the Julian date range 2378480.5 (December 16, 1799) to 2524624.5 (February 1, 2200).<br><br>This block implements these ephemerides with respect to the International Celestial Reference Frame version 2.0, adopted in 2010.   |
| DE430           | Released in 2013. This ephemeris takes into account the Julian date range 2287184.5 (December 21, 1549) to 2688976.5 (January 25, 2650).<br><br>This block implements these ephemerides with respect to the International Celestial Reference Frame version 2.0, adopted in 2010.   |

**Note** This block requires that you download ephemeris data using the Add-On Explorer. To start the Add-On Explorer, in the MATLAB Command Window, type `aeroDataPackage`. on the MATLAB desktop toolstrip, click the **Add-Ons** button.

**Programmatic Use****Block Parameter:** de**Type:** character vector**Values:** DE405 | DE421 | DE423 | DE430**Default:** 'DE405'**Action for out-of-range input — Out-of-range block behavior**

None (default) | Warning | Error

Out-of-range block behavior, specified as follows.

| Action | Description |
|--------|-------------|
| None   | No action.  |

| Action          | Description   |
|-----------------|---|
| Warning         | Warning in the MATLAB Command Window, model simulation continues. |
| Error (default) | MATLAB returns an exception, model simulation stops.              |

**Programmatic Use****Block Parameter:** errorflag**Type:** character vector**Values:** 'None' | 'Warning' | 'Error'**Default:** 'Error'**Calculate rates – Calculate rate of Earth nutation**

on (default) | off

Calculate the rate of the Earth nutation by selecting this check box.

**Dependencies**

Select this check box to display the  $\Delta\psi_{dot} \Delta\epsilon_{dot}$  port.

**Programmatic Use****Block Parameter:** velflag**Type:** character vector**Values:** 'off' | 'on' |**Default:** 'on'**References**

[1] Folkner, W. M., J. G. Williams, D. H. Boggs. "The Planetary and Lunar Ephemeris DE 421." *IPN Progress Report 42-178*, 2009.

[2] Vallado, D. A., *Fundamentals of Astrodynamics and Applications*, McGraw-Hill, New York, 1997.

**Extended Capabilities****C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**See Also**

Moon Libration | Planetary Ephemeris | aeroDataPackage

**Introduced in R2013a**

# Earth Orientation Parameters

Calculate Earth orientation parameters (EOP)

**Library:** Aerospace Blockset / Environment / Celestial Phenomena



## Description

The Earth Orientation Parameters block calculates these parameters:

- Difference between the UTC and Universal Time (UT1)
- Movement of the rotation axis with respect to the crust of the Earth
- Adjustment to the location of the Celestial Intermediate Pole (CIP)

By default, this block uses a prepopulated list of International Earth Rotation and Reference Systems Service (IERS) data. This list contains measured and calculated (predicted) data supplied by the IERS. The IERS measures and calculates this data for a set of predetermined dates. For dates after those listed in the prepopulated list, Earth Orientation Parameters calculates the  $\Delta\text{UT1}$  using this equation, limiting the values to  $\pm 0.9$  s:

$$\text{UT1} - \text{UTC} = 0.5309 - 0.00123(\text{MJD} - 57808) - (\text{UT2} - \text{UT1})$$

Use this block when your application uses Earth Centered Inertial to Earth Centered Earth Fixed transformations, such as for high altitude applications.

## Ports

### Input

#### **UTC<sub>MJD</sub> — UT1 for UTC**

scalar

UT1 for UTC, specified as a scalar modified Julian date. Use the Julian Date Conversion block to convert the UTC date to a modified Julian date.

Data Types: double

### Output

#### **ΔUT1 — Difference between UT1 and UTC**

scalar

Difference between UT1 and UTC, specified as a scalar, in seconds.

Data Types: double

#### **[xp, yp] — Polar displacement of Earth**

vector

Polar displacement of the Earth,  $[xp,yp]$ , specified as a vector, in radians, from the motion of the Earth crust, along the  $x$ - and  $y$ -axes.

Data Types: `double`

#### **$[dX, dY]$ — Adjustment to location of Celestial Intermediate Pole (CIP)**

vector

Adjustment to the location of the Celestial Intermediate Pole (CIP), specified as a vector, in radians. This location ( $[dX,dY]$ ) is along the  $x$ - and  $y$ -axes.

Data Types: `double`

#### **$\Delta UT1_{err}$ — Return errors for the measured and predicted values in the IERS data**

vector

Return errors for the measured and predicted values in the IERS data for the difference between UT1 and UTC, specified as a vector, in seconds.

##### **Dependencies**

This port is enabled when the **Output parameter error** is selected.

#### **$[xp, yp]_{err}$ — Return errors for the measured and predicted values in the IERS data**

vector

Return errors for the measured and predicted values in the IERS data for the polar displacement of Earth, specified as a vector, in radians.

##### **Dependencies**

This port is enabled when the **Output parameter error** is selected.

#### **$[dX, dY]_{err}$ — Return errors for the measured and predicted values in the IERS data**

vector

Return errors for the measured and predicted values in the IERS data for the adjustment to location of Celestial Intermediate Pole (CIP), specified as a vector, in radians.

##### **Dependencies**

This port is enabled when the **Output parameter error** is selected.

## **Parameters**

#### **IERS data file — Earth orientation data**

`aeroiersdata.mat` (default) | MAT-file

Custom list of Earth orientation data, specified in a MAT-file.

##### **Programmatic Use**

**Block Parameter:** `FileName`

**Type:** character vector

**Values:** scalar

**Default:** `'aeroiersdata.mat'`



**Output parameter error — Enable output ports to return errors**

off (default) | on

Select this parameter to enable output ports to return errors for the measured and predicted values in the IERS data file:

- Difference between UT1 and UTC
- Polar displacement of Earth
- Adjustment to location of Celestial Intermediate Pole (CIP)

**Dependencies**

Selecting this check box enables these ports:

- $\Delta UT1_{err}$
- $[xp,yp]_{err}$
- $[dX,dY]_{err}$

**Programmatic Use****Block Parameter:** OutputError**Type:** character vector**Values:** scalar**Default:** 'off'**Action for out-of-range input — Out-of-range block behavior**

Warning (default) | None | Error

Out-of-range block behavior, specified as follows.

| Action          | Description   |
|-----------------|---|
| None            | No action.  |
| Warning         | Warning in the MATLAB Command Window, model simulation continues. |
| Error (default) | MATLAB returns an exception, model simulation stops.              |

**Programmatic Use****Block Parameter:** action**Type:** character vector**Values:** 'None' | 'Warning' | 'Error'**Default:** 'Warning'**IERS data URL — Web site or Earth orientation data file**

<http://maia.usno.navy.mil/ser7/finals2000A.data> (default) | web site address | file name

Web site or Earth orientation data file containing the Earth orientation data according to the IAU 2000A, specified as a web site address or file name.

**Note** If you receive an error message while accessing the default site, use one of these alternate sites:

- [https://datacenter.iers.org/data/latestVersion/10\\_FINALS.DATA\\_IAU2000\\_V2013\\_0110.txt](https://datacenter.iers.org/data/latestVersion/10_FINALS.DATA_IAU2000_V2013_0110.txt)

- <ftp://cddis.gsfc.nasa.gov/pub/products/iers/finals2000A.data>
- 

**Programmatic Use****Block Parameter:** FileName**Type:** character vector**Values:** scalar**Default:** 'aeroiersdata.mat'**Destination folder — Folder for IERS data file**

current Folder (default)

Folder for IERS data file, specified as a character array or string. Before running this function, create *foldername* with write permission.

To create the IERS data file in the destination folder, click the **Create** button.

**Programmatic Use****Block Parameter:** FileName**Type:** character vector**Values:** scalar**Default:** 'aeroiersdata.mat'**Extended Capabilities****C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**See Also**

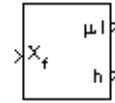
Delta UT1 | Direction Cosine Matrix ECI to ECEF | [aeroReadIERSData](#)

**Introduced in R2018b**

## ECEF Position to LLA

Calculate geodetic latitude, longitude, and altitude above planetary ellipsoid from Earth-centered Earth-fixed (ECEF) position

**Library:** Aerospace Blockset / Utilities / Axes Transformations



### Description

The ECEF Position to LLA block converts a 3-by-1 vector of ECEF position ( $\bar{p}$ ) into geodetic latitude ( $\bar{\mu}$ ), longitude ( $\bar{l}$ ), and altitude ( $\bar{h}$ ) above the planetary ellipsoid. For more information on the ECEF position, see “Algorithms” on page 4-287.

### Limitations

- This implementation generates a geodetic latitude that lies between  $\pm 90$  degrees, and longitude that lies between  $\pm 180$  degrees. The planet is assumed to be ellipsoidal. By setting the flattening to 0, you model a spherical planet.
- The implementation of the ECEF coordinate system assumes that its origin lies at the center of the planet, the x-axis intersects the prime (Greenwich) meridian and the equator, the z-axis is the mean spin axis of the planet (positive to the north), and the y-axis completes the right-handed system.

### Ports

#### Input

##### $X_f$ — Position

3-by-1 vector

Position in ECEF frame, specified as a 3-by-1 vector.

Data Types: double

#### Output

##### $\mu \ l$ — Geodetic latitude and longitude

2-by-1 vector

Geodetic latitude and longitude, returned as a 2-by-1 vector, in degrees.

Data Types: double

##### $h$ — Altitude

scalar

Altitude above the planetary ellipsoid, returned as a scalar, in the same units as the ECEF position.

Data Types: double

## Parameters

### Units — Output units

Metric (MKS) (default) | English

Output units, specified as:

| Units        | Position | Equatorial Radius | Altitude |
|--------------|----------|-------------------|----------|
| Metric (MKS) | Meters   | Meters            | Meters   |
| English      | Feet     | Feet              | Feet     |

### Dependencies

To enable this parameter, set **Planet model** to Earth (WGS84).

#### Programmatic Use

**Block Parameter:** units

**Type:** character vector

**Values:** 'Metric (MKS)' | 'English'

**Default:** 'Metric (MKS)'

### Planet model — Planet model

Earth (WGS84) (default) | Custom

Planet model to use, Custom or Earth (WGS84).

#### Programmatic Use

**Block Parameter:** ptype

**Type:** character vector

**Values:** 'Earth (WGS84)' | 'Custom'

**Default:** 'Earth (WGS84)'

### Flattening — Flattening of planet

1/298.257223563 (default) | scalar

Flattening of the planet, specified as a double scalar.

### Dependencies

To enable this parameter, set **Planet model** to Custom.

#### Programmatic Use

**Block Parameter:** F

**Type:** character vector

**Values:** double scalar

**Default:** 1/298.257223563

### Equatorial radius of planet — Radius of planet at equator

6378137 (default) | scalar

Radius of the planet at its equator, in the same units as the desired units for the ECEF position.

### Dependencies

To enable this parameter, set **Planet model** to Custom.

**Programmatic Use****Block Parameter:** R**Type:** character vector**Values:** double scalar**Default:** 6378137**Algorithms**

The ECEF position is defined as:

$$\bar{p} = \begin{bmatrix} \bar{p}_x \\ \bar{p}_y \\ \bar{p}_z \end{bmatrix}.$$

Longitude is calculated from the ECEF position by

$$l = \text{atan}\left(\frac{p_y}{p_x}\right).$$

Geodetic latitude ( $\bar{\mu}$ ) is calculated from the ECEF position using Bowring's method, which typically converges after two or three iterations. The method begins with an initial guess for geodetic latitude ( $\bar{\mu}$ ) and reduced latitude ( $\bar{\beta}$ ). An initial guess takes the form:

$$\bar{\beta} = \text{atan}\left(\frac{p_z}{(1-f)s}\right)$$

$$\bar{\mu} = \text{atan}\left(\frac{p_z + \frac{e^2(1-f)R(\sin\beta)^3}{(1-e^2)}}{s - e^2R(\cos\beta)^3}\right)$$

where  $R$  is the equatorial radius,  $f$  is the flattening of the planet,  $e^2 = 1 - (1-f)^2$ , the square of first eccentricity, and:

$$s = \sqrt{p_x^2 + p_y^2}.$$

After the initial guesses are calculated, the reduced latitude ( $\bar{\beta}$ ) is recalculated using

$$\beta = \text{atan}\left(\frac{(1-f)\sin\mu}{\cos\mu}\right)$$

and geodetic latitude ( $\bar{\mu}$ ) is reevaluated. This last step is repeated until  $\bar{\mu}$  converges.

The altitude ( $\bar{h}$ ) above the planetary ellipsoid is calculated with

$$h = s\cos\mu + (p_z + e^2N\sin\mu)\sin\mu - N,$$

where the radius of curvature in the vertical prime ( $\bar{N}$ ) is given by

$$N = \frac{R}{\sqrt{1 - e^2(\sin\mu)^2}}.$$

## References

- [1] Stevens, B. L., and F. L. Lewis. *Aircraft Control and Simulation*, Hoboken, NJ: John Wiley & Sons, 1992.
- [2] Zipfel, Peter H., *Modeling and Simulation of Aerospace Vehicle Dynamics*. Second Edition. Reston, VA: AIAA Education Series, 2000.
- [3] *Recommended Practice for Atmospheric and Space Flight Vehicle Coordinate Systems*, R-004-1992, ANSI/AIAA, February 1992.

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

### See Also

Direction Cosine Matrix ECEF to NED | Direction Cosine Matrix ECEF to NED to Latitude and Longitude | Geocentric to Geodetic Latitude | LLA to ECEF Position | Radius at Geocentric Latitude

### Topics

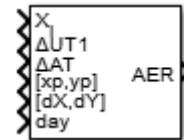
“About Aerospace Coordinate Systems” on page 2-8

### Introduced before R2006a

## ECI Position to AER

Convert Earth-centered inertial (ECI) coordinates to azimuth coordinates

**Library:** Aerospace Blockset / Utilities / Axes Transformations



### Description

The ECI Position to AER block converts Earth-centered inertial (ECI) position coordinates to azimuth, elevation, and slant-range coordinates (AER), based on the geodetic position (latitude, longitude, and altitude).

- Azimuth (A) — Angle measured clockwise from true north. It ranges from 0 to 360 degrees.
- Elevation (E) — Angle between a plane perpendicular to the ellipsoid and the line that goes from the local reference to the object position. It ranges from  $-90$  to  $90$  degrees.
- Slant range (R) — Straight line distance between the local reference and the object.

### Ports

#### Input

##### $X_i$ — Position

3-by-1 element vector

Position, specified as a 3-by-1 element vector, in ECI coordinates.

Data Types: `double`

##### $\Delta UT1$ — Difference between UTC and Universal Time

scalar

Difference between UTC and Universal Time (UT1) in seconds, specified as a scalar, for which the block calculates the direction cosine or transformation matrix.

Example: `0.234`

#### Dependencies

To enable this port, select **Higher accuracy parameters**.

Data Types: `double`

##### $\Delta AT$ — Difference between International Atomic Time and UTC

scalar

Difference between International Atomic Time (IAT) and UTC, specified as a scalar, in seconds, for which the function calculates the direction cosine or transformation matrix.

Example: `32`

**Dependencies**

This port is disabled if the **Higher accuracy parameters** check box is cleared.

Data Types: double

**[xp, yp] — Polar displacement of Earth**

1-by-2 array

Polar displacement of Earth, specified as a 1-by-2 array, in radians, from the motion of the Earth crust, along the x-axis and y-axis.

Example: [-0.0682e-5 0.1616e-5]

**Dependencies**

To enable this port, select **Higher accuracy parameters**.

Data Types: double

**Port\_5 — Adjustment based on reduction method**

1-by-2 array

Adjustment based on reduction method, specified as 1-by-2 array. The name of the port depends on the setting of the **Reduction** parameter:

- If the reduction method is IAU-2000/2006, this input is the adjustment to the location of the Celestial Intermediate Pole (CIP), specified in radians. This location ([dX,dY]) is along the x-axis and y-axis, for example, [-0.2530e-6 -0.0188e-6].
- If the reduction method is IAU-76/FK5, this input is the adjustment to the longitude ( $[\Delta\delta\psi, \Delta\delta\epsilon]$ ), specified in radians.

For historical values, see the International Earth Rotation and Reference Systems Service website (<https://www.iers.org>) and navigate to the Earth Orientation Data Data/Products page.

Example: [-0.2530e-6 -0.0188e-6]

**Dependencies**

To enable this port, select **Higher accuracy parameters**.

Data Types: double

**Port\_6 — Time increment source**

scalar

Time increment source, specified as a scalar, such as the Clock block.

**Dependencies**

- The port name and time increment depend on the **Time Increment** parameter.

| Time Increment Value | Port Name   |
|----------------------|-------------|
| Day                  | <b>day</b>  |
| Hour                 | <b>hour</b> |
| Min                  | <b>min</b>  |



| Time Increment Value | Port Name |
|----------------------|-----------|
| Sec                  | sec       |
| None                 | No port   |

- To disable this port, set the **Time Increment** parameter to None.

Data Types: double

## Output

### AER — Azimuth, elevation, and slant range

3-by-1 element vector

Local reference coordinates azimuth (degrees), elevation (degrees), and slant range (meters), specified as a 3-by-1 element vector.

Data Types: double

## Parameters

### Reduction — Reduction method

IAU-76/FK5 (default) | IAU-2000/2006

Reduction method to convert the coordinates. Method can be one of:

- IAU-76/FK5

Reduce the calculation using the International Astronomical Union 76/Fifth Fundamental Catalogue (IAU-76/FK5) reference system. Choose this reduction method if the reference coordinate system for the conversion is FK5.

---

**Note** This method uses the IAU 1976 precession model and the IAU 1980 theory of nutation to reduce the calculation. This model and theory are no longer current, but the software provides this reduction method for existing implementations. Because of the polar motion approximation that this reduction method uses, the block calculates the transformation matrix rather than the direction cosine matrix.

---

- IAU-2000/2006

Reduce the calculation using the International Astronomical Union 2000/2006 reference system. Choose this reduction method if the reference coordinate system for the conversion is IAU-2000. This reduction method uses the P03 precession model to reduce the calculation.

### Programmatic Use

**Block Parameter:** red

**Type:** character vector

**Values:** 'IAU-2000/2006' | 'IAU-76/FK5'

**Default:** 'IAU-2000/2006'

### Year — Year

2014 (default) | double, whole number, greater than 1

Year to calculate the Universal Coordinated Time (UTC) date. Enter a double value that is a whole number greater than 1, such as 2014.

**Programmatic Use****Block Parameter:** year**Type:** character vector**Values:** double, whole number, greater than 1**Default:** '2013'**Month – Month**January (default) | February | March | April | May | June | July | August | September |  
October | November | December

Month to calculate the UTC date.

**Programmatic Use****Block Parameter:** month**Type:** character vector**Values:** 'January' | 'February' | 'March' | 'April' | 'May' | 'June' | 'July' | 'August' |  
'September' | 'October' | 'November' | 'December'**Default:** 'January'**Day – Day**1 (default) | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |  
24 | 25 | 26 | 27 | 28 | 29 | 30 | 31

Day to calculate the UTC date.

**Programmatic Use****Block Parameter:** day**Type:** character vector**Values:** '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9' | '10' | '11' | '12' | '13' | '14' |  
'15' | '16' | '17' | '18' | '19' | '20' | '21' | '22' | '23' | '24' | '25' | '26' | '27' | '28' |  
'29' | '30' | '31'**Default:** '1'**Hour – Hour**

0 (default) | double, whole number, 0 to 24

Hour to calculate the UTC date. Enter a double value that is a whole number, from 0 to 24.

**Programmatic Use****Block Parameter:** hour**Type:** character vector**Values:** double, whole number, 0 to 24**Default:** '0'**Minutes – Minutes**

0 (default) | double, whole number, 0 to 60

Minutes to calculate the UTC date. Enter a double value that is a whole number, from 0 to 60.

**Programmatic Use****Block Parameter:** min**Type:** character vector**Values:** double, whole number, 0 to 60**Default:** '0'

**Seconds — Seconds**

0 (default)

Seconds to calculate the UTC date. Enter a double value that is a whole number, from 0 to 60.

**Programmatic Use**

**Block Parameter:** sec

**Type:** character vector

**Values:** double, whole number, 0 to 60

**Default:** '0'

**Time increment — Time increment**

None (default) | Day | Hour | Min | Sec

Time increment between the specified date and the desired model simulation time. The block adjusts the calculated direction cosine matrix to take into account the time increment from model simulation. For example, selecting Day and connecting a simulation timer to the port means that each time increment unit is one day and the block adjusts its calculation based on that simulation time.

This parameter corresponds to the time increment input, the clock source.

If you select None, the calculated Julian date does not take into account the model simulation time.

**Programmatic Use**

**Block Parameter:** deltaT

**Type:** character vector

**Values:** 'None' | 'Day' | 'Hour' | 'Min' | 'Sec'

**Default:** 'Day'

**Action for out-of-range input — Action taken when input are out of range**

None (default) | Warning | Error

Specify the block behavior when the block inputs are out of range.

| Action          | Description   |
|-----------------|---|
| None            | No action.  |
| Warning         | Warning in the MATLAB Command Window, model simulation continues. |
| Error (default) | MATLAB returns an exception, model simulation stops.              |

**Programmatic Use**

**Block Parameter:** errorflag

**Type:** character vector

**Values:** 'None' | 'Warning' | 'Error'

**Default:** 'Error'

**Higher accuracy parameters — Enable higher accuracy parameters**

on (default) | off

Select this check box to allow the following as block inputs. These inputs let you better control the conversion result. See “Input” on page 4-289 for a description.

- $\Delta UT1$

- $\Delta AT$
- [  $x_p$  ,  $y_p$  ]
- [  $\Delta\delta\psi$ ,  $\Delta\delta\varepsilon$  ] or [  $d X$  ,  $d Y$  ]

**Programmatic Use****Block Parameter:** extraparamflag**Type:** character vector**Values:** 'on' | 'off'**Default:** 'on'**Units – Units**

Metric (MKS) (default) | English

Specifies the parameter and output units.

| Units        | Position | Equatorial Radius | Altitude |
|--------------|----------|-------------------|----------|
| Metric (MKS) | Meters   | Meters            | Meters   |
| English      | Feet     | Feet              | Feet     |

**Dependencies**To enable this option, set **Earth model** to WGS84.**Programmatic Use****Block Parameter:** eunits**Type:** character vector**Values:** 'Metric (MKS)' | 'English'**Default:** 'Metric (MKS)'**Earth model – Earth model**

Custom (default) | WGS84

Earth model to use, Custom or Earth (WGS84).

**Programmatic Use****Block Parameter:** earthmodel**Type:** character vector**Values:** 'Earth (WGS84)' | 'Custom'**Default:** 'Earth (WGS84)'**Flattening – Flattening of planet**

1/298.257223563 (default) | scalar

Flattening of the planet, specified as a double scalar.

**Dependencies**To enable this parameter, set **Earth model** to Custom.**Programmatic Use****Block Parameter:** flat**Type:** character vector**Values:** double scalar**Default:** 1/298.257223563

**Equatorial radius — Radius of planet at equator**

6378137 (default) | double scalar

Radius of the planet at its equator.

**Dependencies**

To enable this parameter, set **Earth model** to Custom.

**Programmatic Use****Block Parameter:** eqradius**Type:** character vector**Values:** double scalar**Default:** 6378137**Initial geodetic latitude and longitude [deg] — Initial geodetic latitude and longitude**

[0 0] (default) | 2-by-1 vector

Reference location in latitude and longitude, specified as 2-by-1 vector, in degrees.

**Programmatic Use****Block Parameter:** latlon0**Type:** character vector**Values:** 2-by-1 vector**Default:** [0 0]**Angular direction of the local reference system (degrees clockwise from north) — Angular direction**

0 (default) | scalar

Specifies angle for converting the flat Earth x and y coordinates to north and east coordinates, respectively. An example is the angle between the vessel and the true geodetic north.

**Programmatic Use****Block Parameter:** psi0**Type:** character vector**Values:** double scalar**Default:** 0**Reference height — Reference height**

0 (default) | scalar

Specifies the reference height measured from the surface of the Earth to the flat Earth frame. It uses the same units as the ECI position. Estimate the reference height relative to the Earth frame.

**Programmatic Use****Block Parameter:** href**Type:** character vector**Values:** double scalar**Default:** 0**Extended Capabilities****C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**See Also**

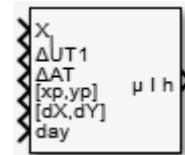
Direction Cosine Matrix ECI to ECEF | ECI Position to LLA | LLA to ECI Position

**Introduced in R2015a**

## ECI Position to LLA

Convert Earth-centered inertial (ECI) coordinates to geodetic latitude, longitude, altitude (LLA) coordinates

**Library:** Aerospace Blockset / Utilities / Axes Transformations



### Description

The ECI Position to LLA block converts Earth-centered inertial (ECI) position coordinates to geodetic latitude, longitude, altitude (LLA) coordinates, based on the specified reduction method and Universal Coordinated Time (UTC), for the specified time and geophysical data.

### Ports

#### Input

##### $X_i$ — Original position

3-by-1 element vector

Original position vector with respect to the ECI reference system, specified as a 3-by-1 element vector.

Data Types: double

##### $\Delta UT1$ — Difference between UTC and Universal Time

scalar

Difference between UTC and Universal Time (UT1) in seconds, specified as a scalar, for which the block calculates the direction cosine or transformation matrix.

Example: 0.234

#### Dependencies

To enable this port, select the **Higher accuracy parameters** check box.

Data Types: double

##### $\Delta AT$ — Difference between International Atomic Time and UTC

scalar

Difference between International Atomic Time (IAT) and UTC, specified as a scalar, in seconds, for which the block calculates the direction cosine or transformation matrix.

Example: 32

#### Dependencies

To enable this port, select the **Higher accuracy parameters** check box.

Data Types: double

**[xp, yp] — Polar displacement of Earth**

1-by-2 array

Polar displacement of Earth, specified as a 1-by-2 array, in radians, from the motion of the Earth crust, along the x-axis and y-axis.

Example: [-0.00682e-5 0.1616e-5]

**Dependencies**

To enable this port, select the **Higher accuracy parameters** check box.

Data Types: double

**Port\_5 — Adjustment based on reduction method**

1-by-2 array

Adjustment based on reduction method, specified as 1-by-2 array. The name of the port depends on the setting of the **Reduction** parameter:

- If reduction method is IAU-2000/2006, this input is the adjustment to the location of the Celestial Intermediate Pole (CIP), specified in radians. This location ([dX,dY]) is along the x-axis and y-axis, for example, [-0.2530e-6 -0.0188e-6].
- If reduction method is IAU-76/FK5, this input is the adjustment to the longitude ( $[\Delta\delta\psi, \Delta\delta\epsilon]$ ), specified in radians.

For historical values, see the International Earth Rotation and Reference Systems Service website (<https://www.iers.org>) and navigate to the Earth Orientation Data Data/Products page.

Example: [-0.2530e-6 -0.0188e-6]

**Dependencies**

To enable this port, select **Higher accuracy parameters**.

Data Types: double

**Port\_6 — Time increment source**

scalar

Time increment source, specified as a scalar, such as the Clock block.

**Dependencies**

- The port name and time increment depend on the **Time Increment** parameter.

| Time Increment Value | Port Name   |
|----------------------|-------------|
| Day                  | <b>day</b>  |
| Hour                 | <b>hour</b> |
| Min                  | <b>min</b>  |
| Sec                  | <b>sec</b>  |
| None                 | No port     |

- To disable this port, set the **Time Increment** parameter to None.

Data Types: double



## Output

### $\mu \ l \ h$ — Original position vector

3-by-1 element vector

Original position vector in geodetic LLA coordinates, returned as a 3-by-1 element vector, in degrees.

Data Types: double

## Parameters

### Reduction — Reduction method

IAU-76/FK5 (default) | IAU-2000/2006

Reduction method to convert the coordinates. Method can be one of:

- IAU-76/FK5

Reduce the calculation using the International Astronomical Union 76/Fifth Fundamental Catalogue (IAU-76/FK5) reference system. Choose this reduction method if the reference coordinate system for the conversion is FK5.

---

**Note** This method uses the IAU 1976 precession model and the IAU 1980 theory of nutation to reduce the calculation. This model and theory are no longer current, but the software provides this reduction method for existing implementations. Because of the polar motion approximation that this reduction method uses, the block calculates the transformation matrix rather than the direction cosine matrix.

---

- IAU-2000/2006

Reduce the calculation using the International Astronomical Union 2000/2006 reference system. Choose this reduction method if the reference coordinate system for the conversion is IAU-2000. This reduction method uses the P03 precession model to reduce the calculation.

### Programmatic Use

**Block Parameter:** red

**Type:** character vector

**Values:** 'IAU-2000/2006' | 'IAU-76/FK5'

**Default:** 'IAU-2000/2006'

### Year — Year

2014 (default) | double, whole number, greater than 1

Year to calculate the Universal Coordinated Time (UTC) date. Enter a double value that is a whole number greater than 1, such as 2014.

### Programmatic Use

**Block Parameter:** year

**Type:** character vector

**Values:** double, whole number, greater than 1

**Default:** '2013'

### Month — Month

January (default) | February | March | April | May | June | July | August | September | October | November | December

Month to calculate the UTC date.

**Programmatic Use****Block Parameter:** month**Type:** character vector**Values:** 'January' | 'February' | 'March' | 'April' | 'May' | 'June' | 'July' | 'August' | 'September' | 'October' | 'November' | 'December'**Default:** 'January'**Day — Day**

1 (default) | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31

Day to calculate the UTC date.

**Programmatic Use****Block Parameter:** day**Type:** character vector**Values:** '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9' | '10' | '11' | '12' | '13' | '14' | '15' | '16' | '17' | '18' | '19' | '20' | '21' | '22' | '23' | '24' | '25' | '26' | '27' | '28' | '29' | '30' | '31'**Default:** '1'**Hour — Hour**

0 (default) | double, whole number, 0 to 24

Hour to calculate the UTC date. Enter a double value that is a whole number, from 0 to 24.

**Programmatic Use****Block Parameter:** hour**Type:** character vector**Values:** double, whole number, 0 to 24**Default:** '0'**Minutes — Minutes**

0 (default) | double, whole number, 0 to 60

Minutes to calculate the UTC date. Enter a double value that is a whole number, from 0 to 60.

**Programmatic Use****Block Parameter:** min**Type:** character vector**Values:** double, whole number, 0 to 60**Default:** '0'**Seconds — Seconds**

0 (default)

Seconds to calculate the UTC date. Enter a double value that is a whole number, from 0 to 60.

**Programmatic Use****Block Parameter:** sec**Type:** character vector**Values:** double, whole number, 0 to 60**Default:** '0'

**Time increment – Time increment**

None (default) | Day | Hour | Min | Sec

Time increment between the specified date and the desired model simulation time. The block adjusts the calculated direction cosine matrix to take into account the time increment from model simulation. For example, selecting Day and connecting a simulation timer to the port means that each time increment unit is one day and the block adjusts its calculation based on that simulation time.

This parameter corresponds to the time increment input, the clock source.

If you select None, the calculated Julian date does not take into account the model simulation time.

**Programmatic Use****Block Parameter:** deltaT**Type:** character vector**Values:** 'None' | 'Day' | 'Hour' | 'Min' | 'Sec'**Default:** 'Day'**Action for out-of-range input – Action taken when inputs are out of range**

None (default) | Warning | Error

Specify the block behavior when the block inputs are out of range.

| Action          | Description   |
|-----------------|---|
| None            | No action.  |
| Warning         | Warning in the MATLAB Command Window, model simulation continues. |
| Error (default) | MATLAB returns an exception, model simulation stops.              |

**Programmatic Use****Block Parameter:** errorflag**Type:** character vector**Values:** 'None' | 'Warning' | 'Error'**Default:** 'Error'**Higher accuracy parameters – Enable higher accuracy parameters**

on (default) | off

Select this check box to allow the following as block inputs. These inputs let you better control the conversion result. See “Input” on page 4-297 for a description.

- $\Delta UT1$
- $\Delta AT$
- [  $x_p$  ,  $y_p$  ]
- [  $\Delta\delta\psi$  ,  $\Delta\delta\varepsilon$  ] or [  $d X$  ,  $d Y$  ]

**Programmatic Use****Block Parameter:** extraparamflag**Type:** character vector**Values:** 'on' | 'off'**Default:** 'on'

**Units – Output units**

Metric (MKS) (default) | English

Specifies the parameter and output units.

| Units        | Position | Equatorial Radius | Altitude |
|--------------|----------|-------------------|----------|
| Metric (MKS) | Meters   | Meters            | Meters   |
| English      | Feet     | Feet              | Feet     |

**Dependencies**To enable this parameter, set **Earth model** to Earth (WGS84).**Programmatic Use****Block Parameter:** eunits**Type:** character vector**Values:** 'Metric (MKS)' | 'English'**Default:** 'Metric (MKS)'**Earth model – Earth model**

Custom (default) | WGS84

Earth model to use, Custom or Earth (WGS84).

**Programmatic Use****Block Parameter:** earthmodel**Type:** character vector**Values:** 'Earth (WGS84)' | 'Custom'**Default:** 'Earth (WGS84)'**Flattening – Flattening of the planet**

1/298.257223563 (default) | scalar

Flattening of the planet, specified as a double scalar.

**Dependencies**To enable this parameter, set **Earth model** to Custom.**Programmatic Use****Block Parameter:** flat**Type:** character vector**Values:** double scalar**Default:** 1/298.257223563**Equatorial radius – Radius**

6378137 (default) | scalar

Radius of the planet at its equator.

**Dependencies**To enable this parameter, set **Earth model** to Custom.**Programmatic Use****Block Parameter:** eqradius

**Type:** character vector

**Values:** double scalar

**Default:** 6378137

## **Extended Capabilities**

### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

### **See Also**

LLA to ECI Position

### **External Websites**

<https://www.iers.org>

**Introduced in R2014a**

## EGM96 Geoid

Calculate geoid height as determined from EGM96 Geopotential Model

### Library

Environment/Gravity

### Description

The EGM96 Geoid block calculates the geoid height as determined from the EGM96 Geopotential Model. The block interpolates the geoid heights from a 15 minute grid of point values in the tide-free system. It uses the EGM96 Geopotential Model to degree and order 360. The geoid undulations are with respect to the WGS84 ellipsoid.

The interpolation scheme wraps over the poles to allow for geoid height calculations at and near these locations.

### Parameters

#### Units

Specifies the parameter and output units:

| Units        | Height |
|--------------|--------|
| Metric (MKS) | Meters |
| English      | Feet   |

#### Data type

Specify the data type of the input and output signals. From the list, select **double** or **single**.

### Inputs and Outputs

| Input  | Dimension Type | Description  |
|--------|----------------|--|
| First  | Scalar         | Contains geocentric latitude in degrees, where north latitude is positive, and south latitude is negative. Input latitude must be of type single or double. If latitude is not in the range from -90 to 90, the block wraps it to be within the range. |
| Second | Scalar         | Contains geocentric longitude in degrees, where east longitude is positive in the range from 0 to 360. Input longitude must be of type single or double. If longitude is not in the range from 0 to 360, the block wraps it to be within the range.    |
| Output | Dimension Type | Description  |
| First  | Scalar         | Contains the geoid height, in meters.  |

## Limitations

This block has the limitations of the 1996 Earth Geopotential Model. For more information, see <http://earth-info.nga.mil/GandG/wgs84/gravitymod/egm96/egm96.html>.

The WGS84 EGM96 geoid undulations have an error range of +/-0.5 to +/-1.0 meters worldwide.

## References

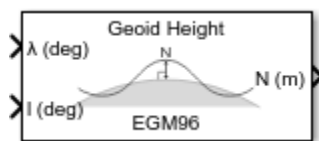
“Department of Defense World Geodetic System 1984, Its Definition and Relationship with Local Geodetic Systems”, NIMA TR8350.2.

“The Development of the Joint NASA GSFC and NIMA Geopotential Model EGM96”, NASA/TP-1998-206861.

National Geospatial-Intelligence Agency Web site: <http://earth-info.nga.mil/GandG/wgs84/gravitymod/egm96/egm96.html>

## See Also

WGS84 Gravity Model



## Compatibility

---

**Note** EGM96 Geoid will be removed in a future version. Use Geoid Height instead.

---

**Introduced in R2007b**

# Exhaust Gas Temperature (EGT) Indicator

Display measurements for engine exhaust gas temperature (EGT)

**Library:** Aerospace Blockset / Flight Instruments



## Description

The EGT Indicator block displays temperature measurements for engine exhaust gas temperature (EGT) in Celsius.

This block displays values using both:

- A needle on a gauge. A major tick is **(Maximum-Minimum)/1,000** degrees, a minor tick is **(Maximum-Minimum)/200** degrees Celsius.
- A numeric indicator. The operating range for the indicator goes from **Minimum** to **Maximum** degrees Celsius.

If the value of the signal is under **Minimum**, the needle displays 5 degrees under the **Minimum** value, the numeric display shows the **Minimum** value. If the value exceeds the **Maximum** value, the needle displays 5 degrees over the maximum tick, and the numeric displays the **Maximum** value.

---

**Tip** To facilitate understanding and debugging your model, you can modify instrument block connections in your model during normal and accelerator mode simulations.

---

## Parameters

### Connection — Connect to signal

signal name

Connect to signal for display, selected from list of signal names.

To view the data from a signal, select a signal in the model. The signal appears in the **Connection** table. Select the option button next to the signal you want to display. Click **Apply** to connect the signal.

The table has a row for the signal connected to the block. If there are no signals selected in the model, or the block is not connected to any signals, the table is empty.

### Minimum — Minimum tick mark value

0 (default) | finite | double | scalar

Minimum tick mark value, specified as a finite double, or scalar value, in ft/min.



**Dependencies**

The **Minimum** tick value must be less than the **Minimum** tick value.

**Programmatic Use**

**Block Parameter:** Limits

**Type:** double

**Values:** double scalar

**Default:** [0 1000], where 0 is the minimum value

**Maximum — Maximum tick mark value**

1000 (default) | finite | double | scalar

Specify the maximum tick mark value, specified as a finite double, or scalar value, in ft/min..

**Dependencies**

The **Maximum** tick value must be greater than the **Maximum** tick value.

**Programmatic Use**

**Block Parameter:** Limits

**Type:** double

**Values:** double scalar

**Default:** [0 1000], where 1000 is the maximum value

**Scale Colors — Ranges of color bands**

0 (default) | double | scalar

Ranges of color bands on the outside of the scale, specified as a finite double, or scalar value. Specify the minimum and maximum color range to display on the gauge.

To add a new color, click +. To remove a color, click -.

**Programmatic Use**

**Block Parameter:** ScaleColors

**Type:** *n*-by-1 struct array

**Values:** struct array with elements Min, Max, and Color

**Label — Block label location**

Top (default) | Bottom | Hide

Block label, displayed at the top or bottom of the block, or hidden.

- Top

Show label at the top of the block.

- Bottom

Show label at the bottom of the block.

- Hide

Do not show the label or instructional text when the block is not connected.

**Programmatic Use**

**Block Parameter:** LabelPosition

**Type:** character vector

**Values:** 'Top' | 'Bottom' | 'Hide'

**Default:** 'Top'

## **Extended Capabilities**

### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

This block is ignored for code generation.

## **See Also**

Indicator | Airspeed Indicator | Altimeter | Artificial Horizon | Climb Rate Indicator | Heading Indicator | Revolutions Per Minute (RPM) Indicator | Turn Coordinator

### **Topics**

“Display Measurements with Cockpit Instruments” on page 2-39

“Programmatically Interact with Gauge Band Colors” on page 2-41

“Flight Instrument Gauges” on page 2-38

**Introduced in R2016a**

# Estimate Center of Gravity

Calculate center of gravity location

**Library:** Aerospace Blockset / Mass Properties



## Description

The Estimate Center of Gravity block calculates the center of gravity location and the rate of change of the center of gravity.

Linear interpolation is used to estimate the location of the center of gravity as a function of mass. The rate of change of the center of gravity is a linear function of the rate of change of mass.

## Ports

### Input

#### mass — Mass

scalar

Mass, specified as a scalar.

Data Types: double

#### dm/dt — Rate of change

scalar | 3-element vector

Rate of change of mass, specified as a scalar or three-element vector.

Data Types: | double

### Output

#### CG — Center of gravity location

3-element vector

Center of gravity location, returned as a three-element vector.

Data Types: double

#### dCG/dt — Rate of change

3-element vector

Rate of the change of center of gravity location, returned as a three-element vector.

Data Types: double

## Parameters

#### Full mass — Mass

2 (default) | scalar

Gross mass of the vehicle, specified as a double scalar.

**Programmatic Use**

**Block Parameter:** fmass

**Type:** character vector

**Values:** double scalar

**Default:** '2'

**Empty mass — Empty mass**

1 (default) | scalar

Empty mass of the vehicle, specified as double scalar.

**Programmatic Use**

**Block Parameter:** emass

**Type:** character vector

**Values:** double scalar

**Default:** '1'

**Full center of gravity — Full center of gravity**

[1 1 1]' (default) | 3-element vector

Center of gravity at the gross mass of the vehicle, specified as a three-element vector.

**Programmatic Use**

**Block Parameter:** fcg

**Type:** character vector

**Values:** 3-element vector

**Default:** [1 1 1]'

**Empty center of gravity — Empty center of gravity**

[0.5 0.5 0.5]' (default) | 3-element vector

Center of gravity at the empty mass of the vehicle, specified as a three-element vector.

**Programmatic Use**

**Block Parameter:** ecg

**Type:** character vector

**Values:** 3-element vector

**Default:** [0.5 0.5 0.5]'

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

## See Also

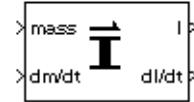
Aerodynamic Forces and Moments | Estimate Inertia Tensor | Moments about CG due to Forces

**Introduced before R2006a**

# Estimate Inertia Tensor

Calculate inertia tensor

**Library:** Aerospace Blockset / Mass Properties



## Description

The Estimate Inertia Tensor block calculates the inertia tensor and the rate of change of the inertia tensor.

Linear interpolation is used to estimate the inertia tensor as a function of mass. The rate of change of the inertia tensor is a linear function of rate of change of mass.

## Ports

### Input

#### **mass — Mass**

scalar

Mass, specified as a scalar.

Data Types: double

#### **dm/dt — Rate of change**

scalar

Rate of change of mass, specified as a scalar.

Data Types: double

### Output

#### **I — Inertia tensor**

3-by-3 matrix

Inertia tensor, returned as a 3-by-3 matrix.

Data Types: double

#### **dI/dt — Rate of change**

3-by-3 matrix

Rate of change of inertia tensor, returned as a 3-by-3 matrix.

Data Types: double

## Parameters

### Full mass — Mass

2 (default) | scalar

Gross mass of the vehicle, specified as a double scalar.

#### Programmatic Use

**Block Parameter:** fmass

**Type:** character vector

**Values:** double scalar

**Default:** '2'

### Empty mass — Empty mass

1 (default) | scalar

Empty mass of the vehicle, specified as a double scalar.

#### Programmatic Use

**Block Parameter:** emass

**Type:** character vector

**Values:** double scalar

**Default:** '1'

### Full inertia matrix — Full inertia matrix

eye(3) (default) | 3-element matrix

Inertia tensor at gross mass of the vehicle, specified as a three-element matrix.

#### Programmatic Use

**Block Parameter:** fI

**Type:** character vector

**Values:** double scalar

**Default:** 'eye(3)'

### Empty inertia matrix — Empty inertia matrix

eye(3)/2 (default) | scalar

Inertia tensor at empty mass of the vehicle, specified as a scalar.

#### Programmatic Use

**Block Parameter:** eI

**Type:** character vector

**Values:** scalar

**Default:** 'eye(3)/2'

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

## See Also

Estimate Center of Gravity | Symmetric Inertia Tensor

**Introduced before R2006a**

## Rodrigues to Direction Cosine Matrix

Convert Euler-Rodrigues vector to direction cosine matrix

**Library:** Aerospace Blockset / Utilities / Axes Transformations



### Description

The Rodrigues to Direction Cosine Matrix block determines the 3-by-3 direction cosine matrix from a three-element Euler-Rodrigues vector. For more information on Euler-Rodrigues vectors, see “Algorithms” on page 4-314.

### Ports

#### Input

**rod — Euler-Rodrigues vector**

three-element vector

Euler-Rodrigues vector from which to determine the direction cosine matrix.

Data Types: double

#### Output

**DCM — Direction cosine matrix**

3-by-3 matrix

Direction cosine matrix determined from the Euler-Rodrigues vector.

Data Types: double

### Algorithms

An Euler-Rodrigues vector  $\vec{b}$  represents a rotation by integrating a direction cosine of a rotation axis with the tangent of half the rotation angle as follows:

$$\vec{b} = [b_x \ b_y \ b_z]$$

where:

$$b_x = \tan\left(\frac{1}{2}\theta\right)s_x,$$

$$b_y = \tan\left(\frac{1}{2}\theta\right)s_y,$$

$$b_z = \tan\left(\frac{1}{2}\theta\right)s_z$$



are the Rodrigues parameters. Vector  $\vec{s}$  represents a unit vector around which the rotation is performed. Due to the tangent, the rotation vector is indeterminate when the rotation angle equals  $\pm\pi$  radians or  $\pm 180$  deg. Values can be negative or positive.

## References

- [1] Dai, J.S. "Euler-Rodrigues formula variations, quaternion conjugation and intrinsic connections." *Mechanism and Machine Theory*, 92, 144-152. Elsevier, 2015.

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

## See Also

Direction Cosine Matrix to Rodrigues | Quaternions to Rodrigues | Rodrigues to Quaternions | Rodrigues to Rotation Angles | Rotation Angles to Rodrigues

**Introduced in R2017a**

## Rodrigues to Quaternions

Convert Euler-Rodrigues vector to quaternion

**Library:** Aerospace Blockset / Utilities / Axes Transformations



### Description

The Rodrigues to Quaternions block determines the 4-by-1 quaternion from a three-element Euler-Rodrigues vector. Aerospace Blockset uses quaternions that are defined using the scalar-first convention. For more information on Euler-Rodrigues vectors, see “Algorithms” on page 4-316.

### Ports

#### Input

##### **rod — Euler-Rodrigues vector**

three-element vector

Euler-Rodrigues vector from which to determine the quaternion.

Data Types: double

#### Output

##### **q — Quaternion**

4-by-1 matrix

Quaternion determined from the Euler-Rodrigues vector.

Data Types: double

### Algorithms

An Euler-Rodrigues vector  $\vec{b}$  represents a rotation by integrating a direction cosine of a rotation axis with the tangent of half the rotation angle as follows:

$$\vec{b} = [b_x \ b_y \ b_z]$$

where:

$$b_x = \tan\left(\frac{1}{2}\theta\right)s_x,$$

$$b_y = \tan\left(\frac{1}{2}\theta\right)s_y,$$

$$b_z = \tan\left(\frac{1}{2}\theta\right)s_z$$

are the Rodrigues parameters. Vector  $\vec{s}$  represents a unit vector around which the rotation is performed. Due to the tangent, the rotation vector is indeterminate when the rotation angle equals  $\pm\pi$  radians or  $\pm 180$  deg. Values can be negative or positive.

## References

- [1] Dai, J.S. "Euler-Rodrigues formula variations, quaternion conjugation and intrinsic connections." *Mechanism and Machine Theory*, 92, 144-152. Elsevier, 2015.

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

## See Also

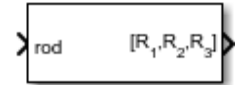
Direction Cosine Matrix to Rodrigues | Quaternions to Rodrigues | Rodrigues to Direction Cosine Matrix | Rodrigues to Rotation Angles | Rotation Angles to Rodrigues

**Introduced in R2017a**

## Rodrigues to Rotation Angles

Convert Euler-Rodrigues vector to rotation angles

**Library:** Aerospace Blockset / Utilities / Axes Transformations



### Description

The Rodrigues to Rotation Angles block converts the three-element Euler-Rodrigues vector into rotation angles. For more information on Euler-Rodrigues vectors, see “Algorithms” on page 4-319.

### Ports

#### Input

##### **rod — Euler-Rodrigues vector**

three-element vector

Euler-Rodrigues vector determined from rotation angles.

Data Types: double

#### Output

##### **R1, R2, R3 — Rotation angles**

three-element vector

Rotation angles, in radians, from which to determine the Euler-Rodrigues vector. Quaternion scalar is the first element.

Data Types: double

### Parameters

#### **Rotation order — Rotation order**

ZYX (default) | ZYZ | ZXY | ZXZ | YXZ | YXY | YZX | YZY | XYZ | YXY | XZY | XZX

Rotation order for three wind rotation angles.

For the 'ZYX', 'ZXY', 'YXZ', 'YZX', 'XYZ', and 'XZY' rotations, the block generates an R2 angle that lies between  $\pm\pi/2$  radians ( $\pm 90$  degrees), and R1 and R3 angles that lie between  $\pm\pi$  radians ( $\pm 180$  degrees).

For the 'YZZ', 'ZXZ', 'YXY', 'YZY', 'YXY', and 'XZX' rotations, the block generates an R2 angle that lies between 0 and  $\pi$  radians (180 degrees), and R1 and R3 angles that lie between  $\pm\pi$  ( $\pm 180$  degrees). However, in the latter case, when R2 is 0, R3 is set to 0 radians.

#### **Programmatic Use**

**Block Parameter:** rotationOrder

**Type:** character vector

**Values:** 'ZYX' | 'YZY' | 'ZXY' | 'ZXZ' | 'YXZ' | 'YXY' | 'YZX' | 'YZY' | 'XYZ' | 'XYX' | 'XZY' | 'XZX'

**Default:** 'ZYX'

## Algorithms

An Euler-Rodrigues vector  $\vec{b}$  represents a rotation by integrating a direction cosine of a rotation axis with the tangent of half the rotation angle as follows:

$$\vec{b} = [b_x \ b_y \ b_z]$$

where:

$$b_x = \tan\left(\frac{1}{2}\theta\right)s_x,$$

$$b_y = \tan\left(\frac{1}{2}\theta\right)s_y,$$

$$b_z = \tan\left(\frac{1}{2}\theta\right)s_z$$

are the Rodrigues parameters. Vector  $\vec{s}$  represents a unit vector around which the rotation is performed. Due to the tangent, the rotation vector is indeterminate when the rotation angle equals  $\pm\pi$  radians or  $\pm 180$  deg. Values can be negative or positive.

## References

- [1] Dai, J.S. "Euler-Rodrigues formula variations, quaternion conjugation and intrinsic connections." *Mechanism and Machine Theory*, 92, 144-152. Elsevier, 2015.

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

## See Also

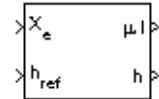
Direction Cosine Matrix to Rodrigues | Quaternions to Rodrigues | Rodrigues to Direction Cosine Matrix | Rodrigues to Quaternions | Rotation Angles to Rodrigues

**Introduced in R2017a**

## Flat Earth to LLA

Estimate geodetic latitude, longitude, and altitude from flat Earth position

**Library:** Aerospace Blockset / Utilities / Axes Transformations



### Description

The Flat Earth to LLA block converts a 3-by-1 vector of Flat Earth position ( $\bar{p}$ ) into geodetic latitude ( $\bar{\mu}$ ), longitude ( $\bar{l}$ ), and altitude ( $h$ ). For more information on the flat Earth coordinate system, see “Algorithms” on page 4-323.

### Limitations

- This estimation method assumes the flight path and bank angle are zero.
- This estimation method assumes the flat Earth z-axis is normal to the Earth at the initial geodetic latitude and longitude only. This method has higher accuracy over small distances from the initial geodetic latitude and longitude, and nearer to the equator. The longitude will have higher accuracy when there are smaller the variations in latitude. Additionally, longitude is singular at the poles.

### Ports

#### Input

##### $X_e$ — Position in flat Earth frame

3-by-1 vector

Position in flat Earth frame, specified as a 3-by-1 vector.

Data Types: double

##### $h_{ref}$ — Reference height

scalar

Reference height from surface of Earth to flat Earth frame with regard to Earth frame, specified as a scalar in the same units as the flat Earth position.

Data Types: double

##### $\mu_{ref}$ $l_{ref}$ — Reference location

2-by-1 vector

Reference location, specified as a 2-by-1 vector, in degrees of latitude and longitude, for the origin of the estimation and the origin of the flat Earth coordinate system. Use this port if you want to specify the reference location as a dynamic value.

#### Dependencies

This port is enabled if the **Input reference position and orientation** check box is selected.

Data Types: double

### $\Psi_{\text{ref}}$ — Direction of flat Earth x-axis

scalar

Angle, specified as a scalar, for converting flat Earth x and y coordinates to North and East coordinates. Use this port if you want to specify the angle as a dynamic value.

#### Dependencies

This port is enabled if the **Input reference position and orientation** check box is selected.

Data Types: double

### Output

#### $\mu \ \lambda$ — Geodetic latitude and longitude

2-by-1 vector

Geodetic latitude and longitude, returned as a 2-by-1 vector, in degrees.

Data Types: double

#### $h$ — Altitude

scalar

Altitude above the input reference altitude, returned as a scalar, in the same units as the flat Earth position.

Data Types: double

## Parameters

### Units — Units

Metric (MKS) (default) | English

Parameter and output units.

| Units        | Position | Equatorial Radius | Altitude |
|--------------|----------|-------------------|----------|
| Metric (MKS) | Meters   | Meters            | Meters   |
| English      | Feet     | Feet              | Feet     |

#### Dependencies

To enable this parameter, set **Planet model** to Earth (WGS84).

#### Programmatic Use

**Block Parameter:** units

**Type:** character vector

**Values:** 'Metric (MKS)' | 'English'

**Default:** 'Metric (MKS)'

#### **Planet model** — Planet model

Earth (WGS84) (default) | Custom

Planet model to use, specified as either Custom or Earth (WGS84).

**Dependencies**

Selecting the Custom option enables these parameters:

- **Flattening**
- **Equatorial radius of planet**

**Programmatic Use**

**Block Parameter:** ptype

**Type:** character vector

**Values:** 'Earth (WGS84)' | 'Custom'

**Default:** 'Earth (WGS84)'

**Flattening – Flattening of planet**

1/298.257223563 (default) | scalar

Flattening of the planet, specified as a double scalar.

**Dependencies**

To enable this parameter, set **Planet model** to Custom.

**Programmatic Use**

**Block Parameter:** F

**Type:** character vector

**Values:** double scalar

**Default:** 1/298.257223563

**Equatorial radius of planet – Radius of planet at equator**

6378137 (default) | scalar

Radius of the planet at its equator, specified as a double scalar, in the same units as the **Units** parameter.

**Dependencies**

This parameter is enabled when **Planet model** is set to Custom.

**Programmatic Use**

**Block Parameter:** R

**Type:** character vector

**Values:** double scalar

**Default:** 6378137

**Input reference position and orientation – Input reference position and orientation as ports**

off (default) | on

Select this check box to enable ports for reference position and angle to convert flat Earth. Select this check box if you want to specify the reference positions and angle as dynamic values.

**Dependencies**

Selecting this check box replaces these parameters:

- **Reference geodetic latitude and longitude [deg]**



- **Direction of flat Earth x-axis (degrees clockwise from north)**

with these input ports:

- $\mu_{\text{ref}}$   $l_{\text{ref}}$
- $\psi_{\text{ref}}$  input ports.

**Programmatic Use**

**Block Parameter:** refPosPort

**Type:** character vector

**Values:** 'off' | 'on'

**Default:** 'off'

**Reference geodetic latitude and longitude [deg] – Initial geodetic latitude and longitude**

[0 10] (default) | 2-by-1 vector

Reference location in latitude and longitude, specified as 2-by-1 vector, in degrees.

**Dependencies**

To enable this parameter, clear the **Input reference position and orientation** check box.

**Programmatic Use**

**Block Parameter:** LL0

**Type:** character vector

**Values:** 2-by-1 vector

**Default:** [0 10]

**Direction of flat Earth x-axis (degrees clockwise from north) – Angle**

0 (default) | scalar

Angle to convert flat Earth x and y coordinates to North and East coordinates, specified as a scalar double, in degrees.

**Dependencies**

This parameter is disabled if the **Input reference position and orientation** check box is selected.

**Programmatic Use**

**Block Parameter:** psi

**Type:** character vector

**Values:** double scalar

**Default:** 0

## Algorithms

The flat Earth coordinate system assumes the z-axis is downward positive. The estimation begins by transforming the flat Earth x and y coordinates to North and East coordinates. The transformation has the form of:

$$\begin{bmatrix} N \\ E \end{bmatrix} = \begin{bmatrix} \cos\psi & -\sin\psi \\ \sin\psi & \cos\psi \end{bmatrix} \begin{bmatrix} p_x \\ p_y \end{bmatrix},$$

where( $\bar{\psi}$ ) is the angle in degrees clockwise between the x-axis and north.

To convert the North and East coordinates to geodetic latitude and longitude, the radius of curvature in the prime vertical ( $R_N$ ) and the radius of curvature in the meridian ( $R_M$ ) are used.

( $R_N$ ) and ( $R_M$ ) are defined by the following relationships:

$$R_N = \frac{R}{\sqrt{1 - (2f - f^2)\sin^2\mu_0}}$$

$$R_M = R_N \frac{1 - (2f - f^2)}{1 - (2f - f^2)\sin^2\mu_0}$$

where ( $R$ ) is the equatorial radius of the planet and ( $\bar{f}$ ) is the flattening of the planet.

Small changes in the in latitude and longitude are approximated from small changes in the North and East positions by:

$$d\mu = \text{atan}\left(\frac{1}{R_M}\right)dN$$

$$dt = \text{atan}\left(\frac{1}{R_N\cos\mu}\right)dE$$

The output latitude and longitude are simply the initial latitude and longitude plus the small changes in latitude and longitude:

$$\mu = \mu_0 + d\mu$$

$$t = t_0 + dt$$

The altitude is the negative flat Earth  $z$ -axis value minus the reference height ( $h_{ref}$ ):

$$h = -p_z - h_{ref}.$$

## References

- [1] Stevens, B. L., and F. L. Lewis. *Aircraft Control and Simulation*, Hoboken, NJ: John Wiley & Sons, 2003.
- [2] Etkin, B. *Dynamics of Atmospheric Flight* Hoboken, NJ: John Wiley & Sons, 1972.

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

### See Also

Direction Cosine Matrix ECEF to NED | Direction Cosine Matrix ECEF to NED to Latitude and Longitude | ECEF Position to LLA | Geocentric to Geodetic Latitude | LLA to ECEF Position | Radius at Geocentric Latitude

**Introduced before R2006a**

# FlightGear Preconfigured 6DoF Animation

Connect model to FlightGear flight simulator

**Library:** Aerospace Blockset / Animation / Flight Simulator Interfaces



## Description

The FlightGear Preconfigured 6DoF Animation block lets you drive position and attitude values to a FlightGear flight simulator vehicle given double-precision values for longitude ( $l$ ), latitude ( $\mu$ ), altitude ( $h$ ), roll ( $\phi$ ), pitch ( $\theta$ ), and yaw ( $\psi$ ), respectively.

The block is configured as a sim viewing device. If you generate code for your model using Simulink Coder and connect to the running target code using external mode simulation, Simulink software can obtain the data from the target on the fly and transmit position and attitude data to FlightGear. For more information, see “Use C/C++ S-Functions as Sim Viewing Devices in External Mode” (Simulink).

## Ports

### Input

**$l, \mu, h, \phi, \theta, \psi$  — Longitude, latitude, altitude, roll, pitch, and yaw**  
vector

Longitude, latitude, altitude, roll, pitch, and yaw, in double-precision, specified as a vector. Units are degrees west/north for longitude and latitude, meters above mean sea level for altitude, and radians for attitude values.

Data Types: single | double | int8 | int16 | int32 | uint8 | uint16 | uint32 | Boolean | fixed point | enumerated | bus

## Parameters

### FlightGear version — FlightGear software version

v2019.1 (default) | v2018.3 | v2018.2 | v2018.1 | v2017.3 | v2017.1 | v2016.3 | v2016.1 | v3.4 | v3.2 | v3.0 | v2.12 | v2.10 | v2.8 | v2.6 | v2.4 | v2.0

Select your FlightGear software version from the list.

---

**Note** If you are using a FlightGear version older than 2.0, the model displays a notification from the Simulink Upgrade Advisor. Consider upgrading your FlightGear version using the Upgrade Advisor. For more information, see “Supported FlightGear Versions” on page 2-16.

---

### Programmatic Use

**Block Parameter:** xFlightGearVersion

**Type:** character vector

**Values:** v2019.1 | v2018.3 | v2018.2 | v2018.1 | v2017.3 | v2017.1 | v2016.3 | v2016.1 | v3.4 | v3.2 | v3.0 | v2.12 | v2.10 | v2.8 | v2.6 | v2.4 | v2.0

**Default:** 'v2019.1'

### Destination IP address — Destination IP address

127.0.0.1 (default) | scalar

Destination IP address of the machine running FlightGear software, specified as a scalar.

#### Programmatic Use

**Block Parameter:** DestinationIpAddress

**Type:** character vector

**Values:** scalar

**Default:** '127.0.0.1'

### Destination port — Destination port

scalar

Destination port of the machine running FlightGear software, specified as a scalar.

#### Programmatic Use

**Block Parameter:** DestinationPort

**Type:** character vector

**Values:** scalar

**Default:** '5502'

### Sample time — Sample time

1/30 (default) | scalar

Sample time specified as a scalar (-1 for inherited).

#### Programmatic Use

**Block Parameter:** SampleTime

**Type:** character vector

**Values:** scalar

**Default:** '1/30'

## Algorithms

The block is a masked subsystem containing principally a Pack net\_fdm Packet for FlightGear block set for 6DoF inputs, a Send net\_fdm Packet to FlightGear block, and a Simulation Pace block. To access the full capabilities of these blocks, use the individual corresponding blocks from the Aerospace Blockset library.

## References

- [1] Bowditch, N., *American Practical Navigator, An Epitome of Navigation*. US Navy Hydrographic Office, 1802.

## See Also

Generate Run Script | Pack net\_fdm Packet for FlightGear | Receive net\_ctrl Packet from FlightGear | Send net\_fdm Packet to FlightGear | Unpack net\_ctrl Packet from FlightGear

**Topics**

“Flight Simulator Interface” on page 2-16

“Work with the Flight Simulator Interface” on page 2-20

**Introduced before R2006a**

## Force Conversion

Convert from force units to desired force units

**Library:** Aerospace Blockset / Utilities / Unit Conversions



### 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 port labels change based on the input and output units selected from the **Initial unit** and the **Final unit** lists.

### Ports

#### Input

##### Port\_1 — Force

scalar | array

Force, specified as a scalar or array, in initial force units.

#### Dependencies

The input port label depends on the **Initial unit** setting.

Data Types: double

#### Output

##### Port\_1 — Force

scalar | array

Force, returned as a scalar or array, in final force units.

#### Dependencies

The output port label depends on the **Final unit** setting.

Data Types: double

### Parameters

#### Initial unit — Input units

lbf (default) | N

Input units, specified as:

|     |             |
|-----|-------------|
| lbf | Pound force |
| N   | Newtons     |

**Dependencies**

The input port label depends on the **Initial unit** setting.

**Programmatic Use**

**Block Parameter:** IU

**Type:** character vector

**Values:** lbf | N

**Default:** lbf

**Final unit – Output units**

N (default) | lbf

Output units, specified as:

|     |             |
|-----|-------------|
| lbf | Pound force |
| N   | Newtons     |

**Dependencies**

The output port label depends on the **Final unit** setting.

**Programmatic Use**

**Block Parameter:** OU

**Type:** character vector

**Values:** lbf | N

**Default:** N

**Extended Capabilities****C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**See Also**

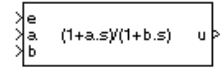
Acceleration Conversion | Angle Conversion | Angular Acceleration Conversion | Angular Velocity Conversion | Density Conversion | Length Conversion | Mass Conversion | Pressure Conversion | Temperature Conversion | Velocity Conversion

**Introduced before R2006a**

## Gain Scheduled Lead-Lag

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

**Library:** Aerospace Blockset / GNC / Control



### 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  $u$  is the filter output.

The coefficients  $a$  and  $b$  are inputs to the block. These values can depend on the flight condition or operating point. For example, you can produce them from the Lookup Table (n-D) Simulink block.

### Ports

#### Input

##### **e** – Filter input

scalar

Filter input, specified as a scalar.

Data Types: single | double | int8 | int16 | int32 | uint8 | uint16 | uint32 | Boolean | fixed point | enumerated | bus

##### **a** – Numerator coefficient

scalar

Numerator coefficient, specified as a scalar.

Data Types: single | double | int8 | int16 | int32 | uint8 | uint16 | uint32 | Boolean | fixed point | enumerated | bus

##### **b** – Denominator coefficient

positive scalar

Denominator coefficient, specified as a positive scalar.

Data Types: single | double | int8 | int16 | int32 | uint8 | uint16 | uint32 | Boolean | fixed point | enumerated | bus

#### Output

##### **u** – Filter output

scalar

Filter output, specified as a scalar.



Data Types: single | double | int8 | int16 | int32 | uint8 | uint16 | uint32 | Boolean | fixed point | enumerated | bus

## Parameters

### Initial state, `x_initial` – Initial internal state

0 (default) | vector

Initial internal state, specified as a vector, for the filter `x_initial`. Given this initial state, the initial output is given by

$$u|_{t=0} = \frac{x\_initial + ae}{b}$$

#### Programmatic Use

**Block Parameter:** initial state, `x_initial`

**Type:** character vector

**Values:** vector

**Default:** '0'

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

## See Also

Lookup Table (n-D)

Introduced before R2006a

## Generate Run Script

Generate FlightGear run script on current platform

**Library:** Aerospace Blockset / Animation / Flight Simulator Interfaces



### Description

The Generate Run Script block generates a customized FlightGear run script on the current platform.

To generate the run script, fill in the required information in the Parameters fields, then click **Generate Script**.

In the dialog box, fields marked with an asterisk (\*) are evaluated as MATLAB expressions. The other fields are treated as literal text.

### Parameters

#### Select target architecture – Target platform to run script

Default (default) | Win64 | Linux | Mac

From the list, select the target platform on which you want to execute the run script. This platform can differ from the platform on which you create the run script. Select **Default** if you want to generate a run script to run on the platform from which you create the run script.

- Win64
- Linux
- Mac

#### Programmatic Use

**Block Parameter:** Architecture

**Type:** character vector

**Values:** 'Win64' | 'Linux' | 'Mac'

**Default:** 'Default'

#### Select FlightGear data flow – FlightGear data flow

Send (default) | Receive | Send-Receive

From the list, select the direction of the data flow:

- Send

Creates the run script to set up the sending of the `net_fdm` control model from Simulink to FlightGear.

- Receive

Creates the run script to set up the receiving of the `net_ctrl` control model from FlightGear to Simulink.

- Send-Receive

Creates the run script to set up FlightGear to receive and broadcast data to and from Simulink.

---

**Note** Selecting the Send-Receive option does not mean that you receive the same data that you sent (for example, you might not see control surface position data). With this option, you see primarily user input (such as data input via joystick) and environmental data.

---

**Programmatic Use**

**Block Parameter:** dataFlow

**Type:** character vector

**Values:** 'Receive' | 'Send-Receive'

**Default:** 'Send'

**FlightGear geometry model name — Folder containing FlightGear geometry**

HL20 (default)

Specify the name of the folder containing the model geometry that you want in the *FlightGear* \data\Aircraft folder.

**Programmatic Use**

**Block Parameter:** GeometryModelName

**Type:** character vector

**Values:** 'HL20'

**Default:** 'HL20'

**Airport ID — ID of supported airport**

KSFO (default)

ID of supported airport, selected from a list of supported airports available in the FlightGear interface, under **Location**.

**Programmatic Use**

**Block Parameter:** 'AirportId'

**Type:** character vector

**Values:** 'KSFO'

**Default:** 'KSFO'

**Runway ID — ID of supported runway**

10L (default)

Specify the runway ID.

**Programmatic Use**

**Block Parameter:** RunwayId

**Type:** character vector

**Values:** '10L'

**Default:** '10L'

**Initial altitude (ft)\* — Initial aircraft altitude**

7224 | MATLAB expression

Initial altitude of the aircraft, in feet. The block evaluates the value as a MATLAB expression.

**Programmatic Use****Block Parameter:** InitialAltitude**Type:** character vector**Values:** '7224'**Default:** '7224'**Initial heading (deg)\* – Initial aircraft heading**

113 | MATLAB expression.

Initial heading of the aircraft, in degrees. The block evaluates the value as a MATLAB expression.

**Programmatic Use****Block Parameter:** InitialHeading**Type:** character vector**Values:** '113'**Default:** '113'**Offset distance (miles)\* – Offset distance**

4.72 | MATLAB expression.

Offset distance of the aircraft from the airport, in miles. The block evaluates the value as a MATLAB expression.

**Programmatic Use****Block Parameter:** OffsetDistance**Type:** character vector**Values:** '4.72'**Default:** '4.72'**Offset azimuth (deg)\* – Aircraft offset azimuth**

0 | MATLAB expression.

Offset azimuth of the aircraft, in degrees. The block evaluates the value as a MATLAB expression.

**Programmatic Use****Block Parameter:** OffsetAzimuth**Type:** character vector**Values:** '0'**Default:** '0'**Install FlightGear scenery during simulation (requires Internet connection) – Install FlightGear scenery**

off (default) | on

Select this check box to direct FlightGear to automatically install required scenery while the simulator is running. Selecting this check box requires a stable Internet connection. For Windows systems, you may encounter an error message while launching FlightGear with this option enabled. For more information, see “Install Additional FlightGear Scenery” on page 2-18.

**Programmatic Use****Block Parameter:** InstallScenery**Type:** character vector**Values:** 'off' | 'on'**Default:** 'off'

**Disable FlightGear shader options – Disable FlightGear shader**

off (default) | on

Select this check box to disable FlightGear shader options. Your computer built-in video card, such as NVIDIA cards, can conflict with FlightGear shaders. Consider selecting this check box if you have this conflict.

**Programmatic Use****Block Parameter:** DisableShaders**Type:** character vector**Values:** 'off' | 'on'**Default:** 'off'**Destination/Origin IP address – Network IP address of FlightGear machine**

127.0.0.1

Network IP address of the machine on which the FlightGear software runs. This value is read-only.

**Programmatic Use****Block Parameter:** OriginAddress**Type:** character vector**Values:** '127.0.0.1'**Default:** '127.0.0.1'**Destination port – Destination port of FlightGear machine**

5502

Network flight dynamics model (fdm) port. For more information, see the Send net\_fdm Packet to FlightGear block reference.

**Programmatic Use****Block Parameter:** DestinationPort**Type:** character vector**Values:** '5502'**Default:** '5502'**Origin port – Origin port of FlightGear machine**

5505

Network control (ctrl) port. For more information, see the Receive net\_ctrl Packet from FlightGear block.

**Programmatic Use****Block Parameter:** OriginPort**Type:** character vector**Values:** '5505'**Default:** '5505'**Network IP address – Network IP address of FlightGear machine**

127.0.0.1

Network IP address of the machine on which the MATLAB software runs.

**Programmatic Use****Block Parameter:** LocalAddress**Type:** character vector

**Values:** '127.0.0.1'  
**Default:** '127.0.0.1'

### Output file name — Output file

runfg.bat

Output file name. The file name is the name of the command that you use to start FlightGear with these initial parameters.

---

**Note** The run script file name must be composed of ASCII characters.

---

Use these file extensions:

| Platform        | Extension |
|-----------------|-----------|
| Windows         | .bat      |
| Linux and macOS | .sh       |

### Programmatic Use

**Block Parameter:** OutputFileName

**Type:** character vector

**Values:** 'runfg.bat'

**Default:** 'runfg.bat'

### FlightGear base directory — FlightGear base directory

C:\Program Files\FlightGear

Specify the name of the FlightGear installation folder.

---

**Note** FlightGear must be installed in a folder path name composed of ASCII characters.

---

### Programmatic Use

**Block Parameter:** FlightGearBaseDirectory

**Type:** character vector

**Values:** 'C:\Program Files\FlightGear'

**Default:** 'C:\Program Files\FlightGear'

### Generate Script — Generate Script button

button

Click **Generate Script** to generate a run script for FlightGear. Do not click this button until you have entered the correct information in the dialog box parameters.

### See Also

FlightGear Preconfigured 6DoF Animation | Pack net\_fdm Packet for FlightGear | Receive net\_ctrl Packet from FlightGear | Send net\_fdm Packet to FlightGear | Unpack net\_ctrl Packet from FlightGear

### Topics

“Flight Simulator Interface” on page 2-16

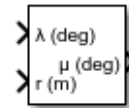
“Work with the Flight Simulator Interface” on page 2-20

**Introduced before R2006a**

# Geocentric to Geodetic Latitude

Convert geocentric latitude to geodetic latitude

**Library:** Aerospace Blockset / Utilities / Axes Transformations



## Description

The Geocentric to Geodetic Latitude block converts a geocentric latitude ( $\lambda$ ) into geodetic latitude ( $\mu$ ). A number of geometric relationships are used to calculate the geodetic latitude in this noniterative method. For more information, see “Algorithms” on page 4-340.

## Limitations

This implementation generates a geodetic latitude that lies between  $\pm 90$  degrees.

## Ports

### Input

#### $\lambda$ — Geocentric latitude

scalar

Geocentric latitude, specified as a scalar, in degrees. Latitude values can be any value. However, values of +90 and -90 may return unexpected values because of singularity at the poles.

Data Types: double

#### $r$ — Radius

scalar

Radius from center of the planet to the center of gravity, specified as a scalar.

Data Types: double

### Output

#### $\mu$ — Geodetic latitude

scalar

Geodetic latitude, specified as a scalar, in degrees.

Data Types: double

## Parameters

### Units — Units

Metric (MKS) (default) | English

Parameter and output units:



| <b>Units</b> | <b>Radius from CG to Center of Planet</b> | <b>Equatorial Radius</b> |
|--------------|---|--------------------------|
| Metric (MKS) | Meters                                    | Meters                   |
| English      | Feet                                      | Feet                     |

**Programmatic Use****Block Parameter:** units**Type:** character vector**Values:** 'Metric (MKS)' | 'English'**Default:** 'Metric (MKS)'**Planet model – Planet model**

Earth (WGS84) (default) | Custom

Planet model to use, Custom or Earth (WGS84).

**Programmatic Use****Block Parameter:** ptype**Type:** character vector**Values:** 'Earth (WGS84)' | 'Custom'**Default:** 'Earth (WGS84)'**Flattening – Flattening**

1/298.257223563 (default) | scalar

Flattening of the planet, specified as a double scalar.

**Dependencies**This parameter is enabled when **Planet model** is set to Custom.**Programmatic Use****Block Parameter:** F**Type:** character vector**Values:** double scalar**Default:** 1/298.257223563**Equatorial radius of planet – Radius**

6378137.0 (default) | scalar

Radius of the planet at its equator, in the same units as the **Units** parameter.**Dependencies**This parameter is enabled when **Planet model** is set to Custom.**Programmatic Use****Block Parameter:** R**Type:** character vector**Values:** double scalar**Default:** 6378137

## Algorithms

The Geocentric to Geodetic Latitude block converts a geocentric latitude ( $\lambda$ ) into geodetic latitude ( $\mu$ ), where:

- $\lambda$  — Geocentric latitude
- $\mu$  — Geodetic latitude
- $r$  — Radius from the center of the planet
- $f$  — Flattening
- $a$  — Equatorial radius of the planet (semi-major axis)

Given geocentric latitude ( $\lambda$ ) and the radius ( $r$ ) from the center of the planet, this block first converts the desired points into the distance from the polar axis ( $\rho$ ) and the distance from the equatorial axis ( $z$ ).

$$\begin{aligned}\rho &= r(\cos(\lambda)) \\ z &= r(\sin(\lambda)).\end{aligned}$$

It then calculates the geometric properties of the planet:

$$\begin{aligned}b &= a(1 - f) \\ e^2 &= f(2 - f) \\ e'^2 &= \frac{e^2}{(1 - e^2)}.\end{aligned}$$

And then uses the fixed-point iteration of Bowring's formula to calculate  $\mu$ . This formula typically converges in three iterations.

$$\begin{aligned}\beta &= \tan^{-1}\left(\frac{(1 - f)\sin(\mu)}{\cos(\mu)}\right) \\ \mu &= \tan^{-1}\left(\frac{z + be'^2 \sin(\beta)^3}{\rho - ae^2 \cos(\beta)^3}\right).\end{aligned}$$

## References

- [1] Jackson, E. B., *Manual for a Workstation-based Generic Flight Simulation Program (LaRCsim) Version 1.4*, NASA TM 110164, April, 1995.
- [2] Hedgley, D. R., Jr. "An Exact Transformation from Geocentric to Geodetic Coordinates for Nonzero Altitudes." NASA TR R-458, March, 1976.
- [3] Clynych, J. R. "Radius of the Earth - Radii Used in Geodesy." Naval Postgraduate School, Monterey, California, 2002.
- [4] Stevens, B. L., and F. L. Lewis. *Aircraft Control and Simulation*, Hoboken, NJ: John Wiley & Sons, 1992.
- [5] Edwards, C. H., and D. E. Penny. *Calculus and Analytical Geometry 2nd Edition*, Prentice-Hall, Englewood Cliffs, New Jersey, 1986.

## **Extended Capabilities**

### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

### **See Also**

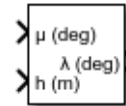
[ECEF Position to LLA](#) | [Flat Earth to LLA](#) | [Geodetic to Geocentric Latitude](#) | [LLA to ECEF Position](#)

**Introduced before R2006a**

# Geodetic to Geocentric Latitude

Convert geodetic latitude to geocentric latitude

**Library:** Aerospace Blockset / Utilities / Axes Transformations



## Description

The Geodetic to Geocentric Latitude block converts a geodetic latitude ( $\mu$ ) into geocentric latitude ( $\lambda$ ). For more information on the geocentric latitude, see “Algorithms” on page 4-344.

## Limitations

This block implementation generates a geocentric latitude that lies between  $\pm 90$  degrees.

## Ports

### Input

#### $\mu$ — Geodetic latitude

scalar

Geodetic latitude, specified as a scalar, in degrees. Latitude values can be any value. However, values of +90 and -90 may return unexpected values because of singularity at the poles.

Data Types: double

#### $h$ — Mean sea-level altitude

scalar

Mean sea-level altitude (MSL), specified as a scalar.

Data Types: double

### Output

#### $\lambda$ — Geocentric latitude

scalar

Contains the geocentric latitude, specified as a scalar, in degrees.

Data Types: double

## Parameters

### Units — Units

Metric (MKS) (default) | English

Parameter and output units:

| <b>Units</b> | <b>Radius from CG to Center of Planet</b> | <b>Equatorial Radius</b> |
|--------------|---|--------------------------|
| Metric (MKS) | Meters                                    | Meters                   |
| English      | Feet                                      | Feet                     |

**Programmatic Use****Block Parameter:** units**Type:** character vector**Values:** 'Metric (MKS)' | 'English'**Default:** 'Metric (MKS)'**Planet model – Planet model**

Earth (WGS84) (default) | Custom

Planet model to use, Custom or Earth (WGS84).

**Dependencies**

Selecting the Custom option enables these parameters:

- **Flattening**
- **Equatorial radius of planet**

**Programmatic Use****Block Parameter:** ptype**Type:** character vector**Values:** 'Earth (WGS84)' | 'Custom'**Default:** 'Earth (WGS84)'**Flattening – Flattening**

1/298.257223563 (default) | scalar

Flattening of the planet, specified as a double scalar.

**Dependencies**This parameter is enabled when **Planet model** is set to Custom.**Programmatic Use****Block Parameter:** F**Type:** character vector**Values:** double scalar**Default:** 1/298.257223563**Equatorial radius of planet – Radius**

6378137.0 (default) | scalar

Radius of the planet at its equator, in the same units as the **Units** parameter.**Dependencies**This parameter is enabled when **Planet model** is set to Custom.**Programmatic Use****Block Parameter:** R

**Type:** character vector

**Values:** double scalar

**Default:** 6378137

## Algorithms

The Geodetic to Geocentric Latitude block converts a geodetic latitude ( $\mu$ ) into geocentric latitude ( $\lambda$ ), where:

- $\lambda$  — Geocentric latitude
- $\mu$  — Geodetic latitude
- $h$  — Height from the surface of the planet
- $f$  — Flattening
- $a$  — Equatorial radius of the planet (semi-major axis)

Given the geodetic latitude ( $\mu$ ) and the height from the surface of the planet ( $h$ ), this block first calculates the geometric properties of the planet.

$$e^2 = \frac{f}{(2-f)}$$

$$N = \frac{a}{\sqrt{1 - e^2 \sin^2(\mu)}}$$

It then calculates the geocentric latitude from the point's distance from the polar axis ( $\rho$ ) and distance from the equatorial axis ( $z$ ).

$$\rho = (N + h)\sin(\mu)$$

$$z = (N(1 - e^2) + h)\sin(\mu)$$

$$\lambda = \tan^{-1}\left(\frac{z}{\rho}\right).$$

## References

- [1] Stevens, B. L., and F. L. Lewis. *Aircraft Control and Simulation*, Hoboken, NJ: John Wiley & Sons, 1992.

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

## See Also

### Topics

ECEF Position to LLA  
 Flat Earth to LLA  
 Geocentric to Geodetic Latitude  
 LLA to ECEF Position  
 Radius at Geocentric Latitude

**Introduced before R2006a**

# Heading Indicator

Display measurements for aircraft heading

**Library:** Aerospace Blockset / Flight Instruments



## Description

The Heading Indicator block displays measurements for aircraft heading in degrees.

The block represents values between 0 and 360 degrees.

---

**Tip** To facilitate understanding and debugging your model, you can modify instrument block connections in your model during normal and accelerator mode simulations.

---

## Parameters

### Connection — Connect to signal

signal name

Connect to signal for display, selected from list of signal names.

To view the data from a signal, select a signal in the model. The signal appears in the **Connection** table. Select the option button next to the signal you want to display. Click **Apply** to connect the signal.

The table has a row for the signal connected to the block. If there are no signals selected in the model, or the block is not connected to any signals, the table is empty.

### Label — Block label location

Top (default) | Bottom | Hide

Block label, displayed at the top or bottom of the block, or hidden.

- Top

Show label at the top of the block.

- Bottom

Show label at the bottom of the block.

- Hide

Do not show the label or instructional text when the block is not connected.



**Programmatic Use****Block Parameter:** LabelPosition**Type:** character vector**Values:** 'Top' | 'Bottom' | 'Hide'**Default:** 'Top'**Extended Capabilities****C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

This block is ignored for code generation.

**See Also**

Airspeed Indicator | Altimeter | Artificial Horizon | Climb Rate Indicator | Exhaust Gas Temperature (EGT) Indicator | Revolutions Per Minute (RPM) Indicator | Turn Coordinator

**Topics**

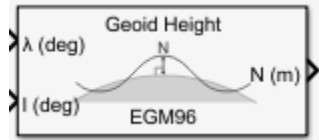
“Display Measurements with Cockpit Instruments” on page 2-39

“Flight Instrument Gauges” on page 2-38

**Introduced in R2016a**

## Geoid Height

Calculate undulations/height



## Library

Environment/Gravity

## Description

The Geoid Height block calculates the geoid height using the **Geopotential model** parameter. The block interpolates the geoid heights from a grid of point values in the tide-free system. It uses the specified geopotential model to degree and order 360. The geoid undulations are relative to the WGS84 ellipsoid.

The interpolation scheme wraps over the poles to allow for geoid height calculations at and near these locations.

## Parameters

### Units

Specifies the parameter and output units:

| Units        | Height |
|--------------|--------|
| Metric (MKS) | Meters |
| English      | Feet   |

### Geopotential model

From the list, select the geopotential model.

| Geopotential Model | Description   |
|--------------------|---|
| EGM96 (Earth)      | Default. EGM96 Geopotential Model to degree and order 360. This model uses a 15-minute grid of point values in the tide-free system. This block calculates geoid heights to an accuracy of 0.01 m for this model. |

| Geopotential Model | Description  |
|--------------------|--|
| EGM2008 (Earth)    | EGM2008 Geopotential Model to degree and order 2159. This model uses a 2.5-minute grid of point values in the tide-free system. This block calculates geoid heights to an accuracy of 0.001 m for this model.<br><br><b>Note</b> This block requires that you download geoid data for the EGM2008 Geopotential Model with the Add-On Explorer. Click the <b>Get data</b> button to start the Add-On Explorer. For more information, see <code>aeroDataPackage</code> . If the data is installed, the <b>Get data</b> button does not appear. |
| Custom             | Custom geopotential model that you define in <b>Geopotential mat-file</b> . This block calculates geoid heights to an accuracy of 0.01 m for custom models. Selecting <code>Custom</code> enables the <b>Geopotential mat-file</b> parameter.  |

### Geopotential mat-file

Specifies the MAT-file that defines your custom geopotential model. Selecting **Geopotential model** > **Custom** enables this parameter.

### Data type

Specifies the data type of the input and output signals. From the list, select `double` or `single`.

### Action for out-of-range input

Defines action for out-of-range input. Specify one:

- 'Error'
- 'Warning' (default)
- 'None'

## Inputs and Outputs

| Input  | Dimension Type | Description   |
|--------|----------------|---|
| First  | Scalar         | Contains geocentric latitude in degrees, where north latitude is positive, and south latitude is negative. Input latitude must be of type <code>single</code> or <code>double</code> . If latitude is not in the range from -90 to 90, the block wraps it to be within the range. |
| Second | Scalar         | Contains geocentric longitude in degrees, where east longitude is positive in the range from 0 to 360. Input longitude must be of type <code>single</code> or <code>double</code> . If longitude is not in the range from 0 to 360, the block wraps it to be within the range.    |
| Output | Dimension Type | Description   |
| N      | Scalar         | Contains the geoid height in selected length units. The data type is the same as the latitude in the first input.   |

## **Limitations**

This block has the limitations of the selected geopotential model.

## **References**

Vallado, D. A. "Fundamentals of Astrodynamics and Applications." McGraw-Hill, New York, 1997.

NIMA TR8350.2: "Department of Defense World Geodetic System 1984, Its Definition, and Relationship with Local Geodetic Systems."

National Geospatial-Intelligence Agency website: <http://earth-info.nga.mil/GandG/publications/vertdatum.html>

## **See Also**

WGS84 Gravity Model, Spherical Harmonic Gravity Model

## **Extended Capabilities**

### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

### **Introduced in R2010b**

# Horizontal Wind Model

Transform horizontal wind into body-axes coordinates



## Library

Environment/Wind

## Description

The Horizontal Wind Model block computes the wind velocity in body-axes coordinates.

The wind is specified by wind speed and wind direction in Earth axes. The speed and direction can be constant or variable over time. The direction of the wind is 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. Using the direction cosine matrix (DCM), the wind velocities are transformed into body-axes coordinates.

## Parameters

### Units

Specifies the input and output units:

| Units                      | Wind Speed        | Wind Velocity     |
|----------------------------|-------------------|-------------------|
| Metric (MKS)               | Meters per second | Meters per second |
| English (Velocity in ft/s) | Feet per second   | Feet per second   |
| English (Velocity in kts)  | Knots             | Knots             |

### Wind speed source

Specify source of wind speed:

|          |                                       |
|----------|---------------------------------------|
| External | Variable wind speed input to block    |
| Internal | Constant wind speed specified in mask |

### Wind speed at altitude (m/s)

Constant wind speed used if internal wind speed source is selected.

### Wind direction source

Specify source of wind direction:

|          |   |
|----------|---|
| External | Variable wind direction input to block    |
| Internal | Constant wind direction specified in mask |

**Wind direction at altitude (degrees clockwise from north)**

Constant wind direction used if internal wind direction source is selected. The direction of the wind is 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.

**Inputs and Outputs**

| Input                | Dimension Type | Description                                |
|----------------------|----------------|--|
| First                |                | Contains the direction cosine matrix.      |
| Second<br>(Optional) |                | Contains the wind speed in selected units. |
| Third<br>(Optional)  |                | Contains the wind direction in degrees.    |

| Output | Dimension Type | Description   |
|--------|----------------|---|
| First  |                | Contains the wind velocity in body-axes, in selected units. |

**See Also**

Dryden Wind Turbulence Model (Continuous)

Dryden Wind Turbulence Model (Discrete)

Discrete Wind Gust Model

Horizontal Wind Model 07

Von Karman Wind Turbulence Model (Continuous)

Wind Shear Model

**Extended Capabilities****C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**Introduced before R2006a**

# Horizontal Wind Model 07

Implement Horizontal Wind Model 07



## Library

Environment/Wind

## Description

The Horizontal Wind Model 07 block implements the U.S. Naval Research Laboratory HWM™ routine to calculate the meridional and zonal components of the wind for a set of geophysical data: latitude, longitude, and altitude.

## Parameters

### Units

Specify the input and output units. The units you select determine the input and output wind speed and velocity, as shown in the table.

| Units                      | Wind Speed        | Wind Velocity     |
|----------------------------|-------------------|-------------------|
| Metric (MKS)               | Meters per second | Meters per second |
| English (Velocity in ft/s) | Feet per second   | Feet per second   |
| English (Velocity in kts)  | Knots             | Knots             |

### Model

Select the horizontal wind model type for which to calculate the wind components.

- **Disturbance**

Calculate the effect of only magnetic disturbances in the wind. For this model type, input Ap index values greater than or equal to 0.

- **Quiet**

Calculate the horizontal wind model without the magnetic disturbances. For this model type, do not input an Ap index value.

- **Total**

Calculate the combined effect of the quiet and magnetic disturbances. For this model type, input Ap index values greater than or equal to 0.

### Action for out-of-range input

Specify the block behavior when the block inputs are out of range.

| Value           | Description   |
|-----------------|---|
| None            | No action. The block imposes upper and lower limits on an input signal.   |
| Warning         | Warning in the Diagnostic Viewer, model simulation continues. For Accelerator and Rapid Accelerator modes, setting the action to Warning has no effect and the model behaves as though the action is set to None. |
| Error (default) | MATLAB returns an exception, model simulation stops. For Accelerator and Rapid Accelerator modes, setting the action to Error has no effect and the model behaves as though the action is set to None.            |

## Inputs and Outputs

| Input             | Dimension Type                  | Description   |
|-------------------|---------------------------------|---|
| First             | Three-element vector of doubles | <p>The input specifies the geodetic latitude (<math>\mu</math>), longitude (<math>\mathcal{L}</math>), and geopotential altitude (<math>h</math>) where the block implements the model.</p> <p>Latitude and longitude values are in degrees.</p> <p>The altitude value is in the units you selected in the <b>Units</b> parameter. Specify the altitude element as a value between 0 and 500 km.</p>  |
| Second            | Scalar double                   | The input specifies the day of year in Universal Coordinated Time (UTC). The input specifies the day as a value between 1 and 366 (for a leap year).  |
| Third             | Scalar double                   | Contains elapsed seconds since midnight for the selected day, in UTC.   |
| Fourth (Optional) | Scalar double                   | <p>Contains the Ap index for the Universal Time (UT) when the block evaluates the model. Select the index from the NOAA National Geophysical Data Center, which contains 3 hour interval geomagnetic disturbance index values. If the Ap index value is greater than zero, the software takes into account magnetic effects during model evaluation.</p> <p>This input appears when you select the total or disturbance wind model type. Specify the Ap index as a value between 0 and 400. This input disappears when you select the quiet model type in the <b>Model</b> parameter.</p> |



| Output | Dimension Type           | Description   |
|--------|--------------------------|---|
| First  | 1-by-2 vector of doubles | The wind velocity vector contains the meridional and zonal wind components in that order. |

## Limitation

For code generation, use this block only for targets whose type is int 32 or higher.

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

## See Also

Discrete Wind Gust Model | Dryden Wind Turbulence Model (Discrete) | Dryden Wind Turbulence Model (Continuous) | Horizontal Wind Model | Von Karman Wind Turbulence Model (Continuous) | Wind Shear Model

### External Websites

NOAA National Geophysical Data Center

### Introduced in R2014b

# Horizontal Wind Model 14

Implement Horizontal Wind Model 14

**Library:** Aerospace Blockset / Environment / Wind



## Description

The Horizontal Wind Model 14 block implements the U.S. Naval Research Laboratory (HWM) routine to calculate the meridional and zonal components of the wind for a set of geophysical data: latitude, longitude, and altitude.

## Limitation

For code generation, use this block only for targets whose type is int 32 or higher.

## Ports

### Input

#### First — geodetic latitude ( $\mu$ ), longitude ( $l$ ), and geopotential altitude ( $h$ )

three-element vector of doubles

The input specifies the geodetic latitude ( $\mu$ ), longitude ( $l$ ), and geopotential altitude ( $h$ ) where the block implements the model.

Latitude and longitude values are in degrees.

The altitude value is in the units you selected in the **Units** parameter. Specify the altitude element as a value between 0 and 500 km.

#### Second — day of year

scalar double

The input specifies the day of year in Universal Coordinated Time (UTC). The input specifies the day as a value between 1 and 366 (for a leap year).

#### Third — elapsed seconds

scalar double

Contains elapsed seconds since midnight for the selected day, in UTC.

#### Fourth (Optional) — Ap index

scalar double

Contains the Ap index for the Universal Time (UT) when the block evaluates the model. Select the index from the NOAA National Geophysical Data Center, which contains 3 hour interval geomagnetic disturbance index values. If the Ap index value is greater than zero, the software takes into account magnetic effects during model evaluation.

## Output

### First — wind velocity vector

1-by-2 vector of doubles

The wind velocity vector contains the meridional and zonal wind components in that order.

## Parameters

### Units — input and output units

Metric (MKS) (default) | English (Velocity in ft/s) | English (Velocity in kts)

Specify the input and output units. The units you select determine the input and output wind speed and velocity, as shown in the table.

| Units                      | Wind Speed        | Wind Velocity     |
|----------------------------|-------------------|-------------------|
| Metric (MKS)               | Meters per second | Meters per second |
| English (Velocity in ft/s) | Feet per second   | Feet per second   |
| English (Velocity in kts)  | Knots             | Knots             |

### Programmatic Use

**Block Parameter:** units

**Type:** character vector

**Values:** 'Metric (MKS)' | 'English (Velocity in ft/s)' | 'English (Velocity in kts)'

**Default:** 'Metric (MKS)'

### Model — horizontal wind model

Quiet (default) | Total | Disturbance

Select the horizontal wind model type for which to calculate the wind components.

- Quiet

Calculate the horizontal wind model without the magnetic disturbances. For this model type, do not input an Ap index value.

- Total

Calculate the combined effect of the quiet and magnetic disturbances. For this model type, input Ap index values greater than or equal to zero.

- Disturbance

Calculate the effect of magnetic disturbances in the wind. For this model type, input Ap index values greater than or equal to zero.

### Programmatic Use

**Block Parameter:** model

**Type:** character vector

**Values:** 'Quiet' | 'Total' | 'Disturbance'

**Default:** 'Quiet'

### Action for out-of-range input — block behavior

Error (default) | Warning | None

Specify the block behavior when the block inputs are out of range.

| Value           | Description   |
|-----------------|---|
| Error (default) | MATLAB returns an exception, and model simulation stops. For Accelerator and Rapid Accelerator modes, setting the action to Error has no effect and the model behaves as though the action is set to None.            |
| Warning         | Warning in the Diagnostic Viewer, and model simulation continues. For Accelerator and Rapid Accelerator modes, setting the action to Warning has no effect and the model behaves as though the action is set to None. |
| None            | No action. The block imposes upper and lower limits on an input signal.   |

**Programmatic Use**

**Block Parameter:** action

**Type:** character vector

**Values:** 'None' | 'Warning' | 'Error'

**Default:** 'Error'

**Extended Capabilities****C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**See Also****External Websites**

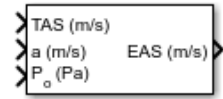
NOAA National Geophysical Data Center

**Introduced in R2016b**

# Ideal Airspeed Correction

Calculate equivalent airspeed (EAS), calibrated airspeed (CAS), or true airspeed (TAS) from each other

**Library:** Aerospace Blockset / Flight Parameters



## Description

The Ideal Airspeed Correction block calculates one of these airspeeds from one of the other two airspeeds:

- Equivalent airspeed (EAS)
- Calibrated airspeed (CAS)
- True airspeed (TAS)

## Limitations

This block assumes that the air flow is compressible dry air with constant specific heat ratio,  $\gamma$

## Ports

### Input

#### TAS — True input airspeed

scalar

True input airspeed, specified as a scalar, in the units specified by the **Units** parameter.

### Dependencies

To enable this port, set **Airspeed input** to TAS.

Data Types: double

#### EAS — Equivalent input airspeed

scalar

Equivalent input airspeed, specified as a scalar, in the units specified by the **Units** parameter.

### Dependencies

To enable this port, set **Airspeed input** to EAS.

Data Types: double

#### CAS — Calibrated input airspeed

scalar

Calibrated input airspeed, specified as a scalar, in the units specified by the **Units** parameter.

**Dependencies**

To enable this port, set **Airspeed input** to EAS.

Data Types: double

**a — Speed of sound**

scalar

Speed of sound, specified as a scalar, in the units specified by the **Units** parameter.

Data Types: double

 **$P_0$  — Static pressure**

scalar

Static pressure, specified as a scalar, in the units specified by the **Units** parameter.

Data Types: double

**Output****EAS — Equivalent output airspeed**

scalar

Equivalent output airspeed, returned as a scalar, in the units specified by the **Units** parameter.

**Dependencies**

To enable this port, set **Airspeed input** to TAS or CAS and **Airspeed output** to EAS.

Data Types: double

**CAS — Calibrated output airspeed**

scalar

Calibrated output airspeed, returned as a scalar, in the units specified by the **Units** parameter.

**Dependencies**

To enable this port, set **Airspeed input** to TAS or EAS and **Airspeed output** to CAS.

Data Types: double

**TAS — True output airspeed**

scalar

True output airspeed, returned as a scalar, in the units specified by the **Units** parameter.

**Dependencies**

To enable this port, set **Airspeed input** to CAS or EAS and **Airspeed output** to TAS.

Data Types: double

**Parameters****Units — Units**

Metric (MKS) (default) | English (Velocity in ft/s) | English (Velocity in kts)

Input and output units, specified as:

| Units                      | Airspeed Input    | Speed of Sound    | Air Pressure                | Airspeed Output   |
|----------------------------|-------------------|-------------------|-----------------------------|-------------------|
| Metric (MKS)               | Meters per second | Meters per second | Pascal                      | Meters per second |
| English (Velocity in ft/s) | Feet per second   | Feet per second   | Pound force per square inch | Feet per second   |
| English (Velocity in kts)  | Knots             | Knots             | Pound force per square inch | Knots             |

#### Programmatic Use

**Block Parameter:** units

**Type:** character vector

**Values:** 'Metric (MKS)' | 'English'

**Default:** 'Metric (MKS)'

#### Airspeed input – Airspeed input type

TAS (default) | EAS | CAS

Airspeed input type, specified as:

|     |                     |
|-----|---------------------|
| TAS | True airspeed       |
| EAS | Equivalent airspeed |
| CAS | Calibrated airspeed |

#### Programmatic Use

**Block Parameter:** vel\_in

**Type:** character vector

**Values:** 'TAS' | 'EAS' | 'CAS'

**Default:** 'TAS'

#### Airspeed output – Airspeed output type

EAS (default) | CAS | TAS

Airspeed output type, specified as:

| Airspeed Input | Airspeed Output           |
|----------------|---------------------------|
| TAS            | EAS (equivalent airspeed) |
|                | CAS (calibrated airspeed) |
| EAS            | TAS (true airspeed)       |
|                | CAS (calibrated airspeed) |
| CAS            | TAS (true airspeed)       |
|                | EAS (equivalent airspeed) |

**Programmatic Use**

**Block Parameter:** `vel_out_tas`, `vel_out_cas`, `vel_out_eas`, depending on the input velocity type, `vel_in`. For more information, see the airspeed output type table.

**Type:** character vector

**Values:** 'EAS' | 'CAS' 'TAS'

**Default:** 'EAS'

**Method — Method for computing conversion factor**

Table Lookup (default) | Equation

Method for computing the conversion factor, specified as:

|              |  |
|--------------|--|
| Table Lookup | <p>(Default) Generate output airspeed by looking up or estimating table values based on block inputs.</p> <p>If the <b>Subsonic airspeeds only</b> check box is selected, the Ideal Airspeed Correction block generates code that includes subsonic (Mach &lt; 1) lookup table data.</p> <p>If the <b>Subsonic airspeeds only</b> check box is cleared, the Ideal Airspeed Correction block generates code that includes all (Mach &lt; 5) lookup table data. Beyond Mach 5, the block uses the <code>equation</code> method.</p> <p>The Table Lookup method is not recommended for either of these instances:</p> <ul style="list-style-type: none"> <li>• Speed of sound less than 200 m/s or greater than 350 m/s.</li> <li>• Static pressure less than 1000 Pa or greater than 106,500 Pa.</li> </ul> <p>Using the Table Lookup method in these instances causes inaccuracies.</p> |
| Equation     | <p>Compute output airspeed directly using block input values.</p> <p>Calculations involving supersonic airspeeds (greater than Mach 1) require an iterative computation. If the function does not find a solution within 30 iterations, it displays an error message.</p> <p>The block does not include lookup table data in generated code.</p>   |

The Ideal Airspeed Correction block automatically uses the Equation method for any of these instances:

- Conversion with **Airspeed input** set to TAS and **Airspeed output** set to EAS.
- Conversion with **Airspeed input** set to EAS and **Airspeed output** set to TAS.
- Conversion when block input airspeed is greater than five times the speed of sound at sea level (approximately 1700 m/s).

**Programmatic Use**

**Block Parameter:** method

**Type:** character vector

**Values:** 'Table Lookup' | 'Equation'

**Default:** 'Table Lookup'



**Subsonic airspeeds only – Use with subsonic airspeed**

off (default) | on

Select this check box to use this block only with subsonic airspeed (airspeeds less than Mach 1) applications. Selecting this check box may improve performance.

The block generates code as follows:

- If this check box is selected, the Ideal Airspeed Correction block generates code that includes subsonic (Mach < 1) lookup table data if **Method** is set to Table Lookup.

Selecting this check box displays the **Action for out-of-range input** parameter.

- If this check box is cleared, the Ideal Airspeed Correction block generates code that includes all (Mach < 5) lookup table data if **Method** is set to Table Lookup.

**Programmatic Use****Block Parameter:** SubOnly**Type:** character vector**Values:** 'off' | 'on'**Default:** 'off'**Action for out-of-range input – Out-of-range block behavior**

None (default) | Warning | Error

Out-of-range block behavior, where airspeed is greater than Mach 1, specified as follows.

| Value   | Description  |
|---------|--|
| None    | Does not display warning or error.                                       |
| Warning | Displays warning and indicates that the airspeed is greater than Mach 1. |
| Error   | Displays error and indicates that the airspeed is greater than Mach 1.   |

**Dependencies**

This parameter is enabled only if the **Subsonic airspeeds only** check box is selected.

**Programmatic Use****Block Parameter:** action**Type:** character vector**Values:** 'None' | 'Warning' | 'Error'**Default:** 'None'**References**

[1] Lowry, J. T., *Performance of Light Aircraft*, AIAA Education Series, Washington, DC, 1999.

[2] *Aeronautical Vestpocket Handbook*, United Technologies Pratt & Whitney, August, 1986.

[3] Gracey, William, *Measurement of Aircraft Speed and Altitude*, NASA Reference Publication 1046, 1980.

## **Extended Capabilities**

### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

### **See Also**

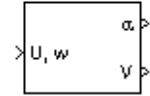
COESA Atmosphere Model | ISA Atmosphere Model | Lapse Rate Model | Non-Standard Day 210C | Non-Standard Day 310

**Introduced before R2006a**

# Incidence & Airspeed

Calculate incidence and airspeed

**Library:** Aerospace Blockset / Flight Parameters



## 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 airspeed from the velocity components in the body-fixed coordinate frame.

$$\alpha = \text{atan}\left(\frac{w}{u}\right)$$

$$V = \sqrt{u^2 + w^2}$$

## Ports

### Input

#### **U, w – Velocity**

two-element vector

Velocity of the body, specified as a two-element vector, resolved into the body-fixed coordinate frame.

Data Types: double

### Output

#### **alpha – Incidence angle**

scalar

Incidence angle, returned as a scalar, in radians.

Data Types: double

#### **V – Airspeed**

scalar

Airspeed of the body, returned as a scalar.

Data Types: double

## Extended Capabilities

### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

## See Also

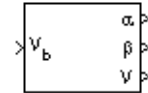
Incidence, Sideslip & Airspeed

**Introduced before R2006a**

# Incidence, Sideslip, & Airspeed

Calculate incidence, sideslip, and airspeed

**Library:** Aerospace Blockset / Flight Parameters



## Description

The Incidence, Sideslip, & Airspeed block calculates the angles between the velocity vector and the body, and also the total airspeed from the velocity components in the body-fixed coordinate frame. For the equations used in the calculation, see “Algorithms” on page 4-367.

## Ports

### Input

#### $V_b$ — Velocity of body

three-element vector

Velocity of the body, specified as a three-element vector, resolved into the body-fixed coordinate frame.

Data Types: double

### Output

#### $\alpha$ — Incidence angle

scalar

Incidence angle, returned as a scalar, in radians.

Data Types: double

#### $\beta$ — Sideslip angle

scalar

Sideslip angle, returned as a scalar, in radians.

Data Types: double

#### $V$ — Airspeed

scalar

Airspeed of the body, returned as a scalar.

Data Types: double

## Algorithms

To calculate the angles between the velocity vector and the body, and the total airspeed, the block uses these equations:

$$\alpha = \text{atan}\left(\frac{w}{u}\right)$$

$$\beta = \text{asin}\left(\frac{v}{V}\right)$$

$$V = \sqrt{u^2 + v^2 + w^2}$$

$$\alpha = \text{atan}\left(\frac{w}{u}\right)$$

$$\beta = \text{asin}\left(\frac{v}{V}\right)$$

$$V = \sqrt{u^2 + v^2 + w^2}$$

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

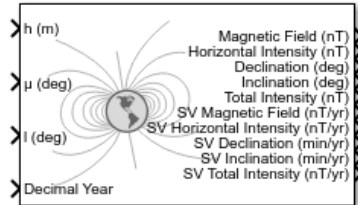
### See Also

Incidence & Airspeed

**Introduced before R2006a**

# International Geomagnetic Reference Field 12

Calculate Earth magnetic field and secular variation using 12th generation International Geomagnetic Reference Field



## Library

Environment/Gravity

## Description

The International Geomagnetic Reference Field 12 block calculates the Earth magnetic field and secular variation using the 12th generation International Geomagnetic Reference Field. It calculates these values at a location and time that you define.

## Parameters

### Units

Specifies the parameter and output units.

| Units        | Height |
|--------------|--------|
| Metric (MKS) | Meters |
| English      | Feet   |

### Input decimal year

When you select this check box, the decimal year is an input for the International Geomagnetic Reference Field 12 block. Otherwise, specify a date using the **Month**, **Day**, and **Year** parameters.

### Month

Specifies the month to calculate decimal year.

### Day

Specifies the day to calculate decimal year.

### Year

Specifies the year to calculate decimal year. From the list, select from 1900 to 2020.

### Action for out-of-range input

Specifies whether out-of-range input causes a warning, error, or no action.

### Output secular variance

Select this check box to enable the output of secular variances (annual rate of change) with nonsecular variances.

| Secular Variance        | Description  |
|-------------------------|--|
| Magnetic Field          | Magnetic field vector, in nanotesla (nT). Z is the vertical component (+ve down)   |
| Horizontal Intensity    | Horizontal intensity, in nanotesla (nT)  |
| Declination             | Declination, in degrees (+ve east)   |
| Inclination             | Inclination, in degrees (+ve down)   |
| Total Intensity         | Total intensity, in nanotesla (nT)   |
| SV Magnetic Field       | Secular variation of magnetic field  |
| SV Horizontal Intensity | Secular variation of horizontal intensity  |
| SV Declination          | Secular variation of declination, the angle between true north and the magnetic field vector (positive eastward)           |
| SV Inclination          | Secular variation of inclination, the angle between the horizontal plane and the magnetic field vector (positive downward) |
| SV Total Intensity      | Secular variation of total intensity   |

Clear this check box to enable just the nonsecular variances:

- Magnetic Field
- Horizontal Intensity
- Declination
- Inclination
- Total Intensity

### Inputs and Outputs

| Input                | Dimension Type | Description  |
|----------------------|----------------|--|
| First                | Scalar         | Contains the height, in selected units.  |
| Second               | Scalar         | Contains the latitude, in degrees.   |
| Third                | Scalar         | Contains the longitude, in degrees.  |
| Fourth<br>(Optional) | Scalar         | Contains the desired year in a decimal format to include any fraction of the year that has already passed. The value is the current year plus the number of days that have passed in this year divided by 365. |
|                      |                | <p>This code shows how to calculate the decimal year, <code>dyear</code>, for March 21, 2015:</p> <pre>dyear = decyear('21-March-2015', 'dd-mmm-yyyy')</pre>   |



| Output                | Dimension Type | Description   |
|-----------------------|----------------|---|
| First                 |                | Contains the magnetic field vector, in selected units.                                |
| Second                |                | Contains the horizontal intensity, in selected units.                                 |
| Third                 |                | Contains the declination, in degrees.   |
| Fourth                |                | Contains the inclination, in degrees.   |
| Fifth                 |                | Contains the total intensity, in selected units.                                      |
| Sixth<br>(Optional)   |                | Contains the secular variation of magnetic field vector, in selected units per years. |
| Seventh<br>(Optional) |                | Contains the secular variation of horizontal intensity, in selected units per year.   |
| Eight<br>(Optional)   |                | Contains the secular variation of declination, in minutes per year.                   |
| Ninth<br>(Optional)   |                | Contains the secular variation of inclination, in minutes per year.                   |
| Tenth<br>(Optional)   |                | Contains the secular variation of total intensity, in selected units per year.        |

## Limitations

This block is valid between the heights of -1000 m and 600,000 m.

This block is valid between the years 1900 and 2020.

This site shows additional limitations:

<https://www.ngdc.noaa.gov/IAGA/vmod/igrfhw.html>

## References

International Association of Geomagnetism and Aeronomy. 12th Generation International Geomagnetic Reference Field: <https://www.ngdc.noaa.gov/IAGA/vmod/igrf.html>.

Blakely, R. J., *Potential Theory in Gravity & Magnetic Applications*. Cambridge, UK: Cambridge University Press, 1996.

Lowes, F. J. "The International Geomagnetic Reference Field: A 'Health' Warning." January, 2010. <https://www.ngdc.noaa.gov/IAGA/vmod/igrfhw.html>.

## Extended Capabilities

### C/C++ Code Generation

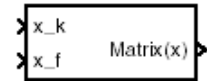
Generate C and C++ code using Simulink® Coder™.

### Introduced in R2015b

## Interpolate Matrix(x)

Return interpolated matrix for given input

**Library:** Aerospace Blockset / GNC / Control



### Description

The Interpolate Matrix(x) block interpolates a one-dimensional array of matrices. The block assumes a one-dimensional array as defined in “Algorithms” on page 4-373.

The matrix to be interpolated must 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;
```

### Limitations

This block must be driven from the Prelookup block.

### Ports

#### Input

##### **x\_k — Interpolation index $i$**

scalar

Interpolation index  $i$ , specified as a scalar.

Data Types: single | double | int8 | int16 | int32 | uint8 | uint16 | uint32 | Boolean | fixed point | enumerated | bus

##### **x\_f — Interpolation fraction**

scalar

Interpolation fraction  $\lambda$ , specified as a scalar.

Data Types: single | double | int8 | int16 | int32 | uint8 | uint16 | uint32 | Boolean | fixed point | enumerated | bus

#### Output

##### **Matrix(x) — Interpolated matrix**

matrix

Interpolated matrix, specified as a matrix.

Data Types: single | double | int8 | int16 | int32 | uint8 | uint16 | uint32 | Boolean | fixed point | enumerated | bus

## Parameters

### Matrix to interpolate – Matrix

matrix (default)

Matrix to be interpolated, with three indices and the third index labeling the interpolating values of  $x$ .

#### Programmatic Use

**Block Parameter:** matrix

**Type:** character vector

**Values:** matrix

**Default:** 'matrix'

## Algorithms

This one-dimensional case assumes a matrix  $M$  is defined at a discrete number of values of an independent variable

$$x = [x_1 x_2 x_3 \dots x_i x_{i+1} \dots x_n].$$

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

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

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

## See Also

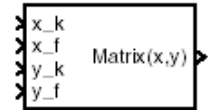
on page 4-2 | 1D Observer Form [A(v),B(v),C(v),F(v),H(v)] | 1D Self-Conditioned [A(v),B(v),C(v),D(v)] | Interpolate Matrix(x,y) | Interpolate Matrix(x,y,z)

**Introduced before R2006a**

## Interpolate Matrix(x,y)

Return interpolated matrix for given inputs

**Library:** Aerospace Blockset / GNC / Control



### Description

The Interpolate Matrix(x,y) block interpolates a two-dimensional array of matrices. In two-dimensional cases, the interpolation is carried out first on  $x$  and then  $y$ . For more information, see “Algorithms” on page 4-375.

The matrix to be interpolated must 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;
```

### Limitations

This block must be driven from the Prelookup block.

### Ports

#### Input

##### **x\_k — First interpolation index**

scalar

First interpolation index  $i$ , specified as a scalar and vector.

Data Types: double

##### **x\_f — First interpolation fraction**

scalar

First interpolation fraction  $\lambda_x$ , specified as a scalar

Data Types: double

##### **y\_k — Second interpolation index**

scalar

Second interpolation index  $j$ , specified as a scalar.

Data Types: double

### **y\_f — Second interpolation fraction**

scalar

Second interpolation fraction  $\lambda_y$ , specified as a scalar.

Data Types: double

### **Output**

#### **Matrix(x,y) — Interpolated matrix**

matrix

Interpolated matrix, specified as a matrix.

Data Types: double

### **Parameters**

#### **Matrix to interpolate — Matrix**

matrix (default)

Matrix to be interpolated, with four indices and the third and fourth indices labeling the interpolating values of  $x$  and  $y$ .

#### **Programmatic Use**

**Block Parameter:** matrix

**Type:** character vector

**Values:** matrix

**Default:** 'matrix'

### **Algorithms**

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

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

### **Extended Capabilities**

#### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**See Also**

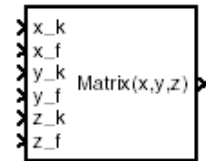
2D Controller [A(v),B(v),C(v),D(v)] | 2D Observer Form [A(v),B(v),C(v),F(v),H(v)] | 2D Self-Conditioned [A(v),B(v),C(v),D(v)] | Interpolate Matrix(x) | Interpolate Matrix(x,y,z) | Prelookup

**Introduced before R2006a**

# Interpolate Matrix(x,y,z)

Return interpolated matrix for given inputs

**Library:** Aerospace Blockset / GNC / Control



## Description

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:

$$\begin{aligned}
 x &= [x_1 \quad x_2 \quad x_3 \quad \dots \quad x_i \quad x_{i+1} \quad \dots \quad x_n] \\
 y &= [y_1 \quad y_2 \quad y_3 \quad \dots \quad y_j \quad y_{j+1} \quad \dots \quad y_m] \\
 z &= [z_1 \quad z_2 \quad z_3 \quad \dots \quad z_k \quad z_{k+1} \quad \dots \quad z_p]
 \end{aligned}$$

For given values of  $x$ ,  $y$ , and  $z$ , eight matrices are interpolated. Then for

$$\begin{aligned}
 x & \quad i & < & x & < & x & \quad i+1 \\
 y & \quad j & < & y & < & y & \quad j+1 \\
 z & \quad k & < & z & < & z & \quad k+1
 \end{aligned}$$

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

$$\begin{aligned}
 \lambda_x &= (x - x_i) / (x_{i+1} - x_i) \\
 \lambda_y &= (y - y_j) / (y_{j+1} - y_j) \\
 \lambda_z &= (z - z_k) / (z_{k+1} - z_k)
 \end{aligned}$$

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

The matrix to be interpolated 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; |

## Limitations

This block must be driven from the Prelookup block.

## Ports

### Input

#### **x\_k — First interpolation index**

scalar

First interpolation index  $i$ , specified as a scalar.

Data Types: double

#### **x\_f — First interpolation fraction**

scalar

First interpolation fraction  $\lambda_x$ , specified as a scalar.

Data Types: double

#### **y\_k — Second interpolation index**

scalar

Second interpolation index  $j$ , specified as a scalar.

Data Types: double

#### **y\_f — Second interpolation fraction**

scalar

Second interpolation fraction  $\lambda_y$ , specified as a scalar.

Data Types: double

#### **z\_k — Third interpolation index**

scalar

Third interpolation index  $k$ , specified as a scalar.

Data Types: double



**z\_f – Third interpolation fraction**

scalar

Third interpolation fraction  $\lambda_z$ , specified as a scalar.

Data Types: double

**Output****Matrix(x,y,z) – Interpolated matrix**

matrix

Interpolated matrix, specified as a matrix.

Data Types: double

**Parameters****Matrix to interpolate – Matrix to interpolate**

matrix (default)

Matrix to be interpolated, with five indices and the third, fourth, and fifth indices labeling the interpolating values of  $x$ ,  $y$ , and  $z$ .**Programmatic Use****Block Parameter:** matrix**Type:** character vector**Values:** matrix**Default:** 'matrix'**Extended Capabilities****C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**See Also**

3D 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)] | Interpolate Matrix(x) | Interpolate Matrix(x,y) | Prelookup

**Introduced before R2006a**

# Invert 3x3 Matrix

Compute inverse of 3-by-3 matrix

**Library:** Aerospace Blockset / Utilities / Math Operations



## Description

The Invert 3x3 Matrix block computes the inverse of 3-by-3 matrix.

If  $\det(A) = 0$ , an error occurs and the simulation stops.

## Ports

### Input

#### Port\_1 — Input matrix

3-by-3 matrix

Input matrix to be inverted, specified as a 3-by-3 matrix.

Data Types: double

### Output

#### Port\_1 — Matrix inverse

3-by-3 matrix

Matrix inverse of input matrix, returned as a 3-by-3 matrix.

Data Types: double

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

## See Also

Adjoint of 3x3 Matrix | Create 3x3 Matrix | Determinant of 3x3 Matrix

**Introduced before R2006a**

# ISA Atmosphere Model

Implement International Standard Atmosphere (ISA)

**Library:** Aerospace Blockset / Environment / Atmosphere



## Description

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

The ISA Atmosphere Model and Lapse Rate Model blocks are identical blocks. When configured for ISA Atmosphere Model, the block implements ISA values. When configured for Lapse Rate Model, the block implements lapse rate values.

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

## Limitations

- Below the geopotential altitude of 0 km and above the geopotential altitude of the tropopause, temperature and pressure values are held.
- Density and speed of sound are calculated using a perfect gas relationship.

## Ports

### Input

#### **h (m) — Geopotential height**

scalar | array

Geopotential height, specified as a scalar or array.

Data Types: double

### Output

#### **T (K) — Temperature**

scalar | array

Temperature, returned as a scalar or array, in K.

Data Types: double

#### **a (m/s) — Speed of sound**

scalar | array

Speed of sound, returned as a scalar or array, in m/s.

Data Types: double

**P (Pa) – Air pressure**

scalar | array

Air pressure, returned as a scalar or array, in Pa.

Data Types: double

 **$\rho$  (kg/m<sup>3</sup>) – Air density**

scalar | array

Air density, returned as scalar or array, in kg/m<sup>3</sup>.

Data Types: double

**Parameters****Change atmospheric parameters – Customize parameters**

off (default) | on

Customize various atmospheric parameters to be different from the ISA values. Selecting this check box converts the block from ISA Atmosphere Model to Lapse Rate Model.

**Dependencies**

Selecting this check box enables the parameters:

- **Acceleration due to gravity (m/s<sup>2</sup>)**
- **Ratio of specific heats**
- **Characteristic gas constant (J/Kg/K)**
- **Lapse rate (K/m)**
- **Height of troposphere (m)**
- **Height of tropopause (m)**
- **Air density at mean sea level (Kg/m<sup>3</sup>)**
- **Ambient pressure at mean sea level (N/m<sup>2</sup>)**
- **Lowest altitude (m)**

**Programmatic Use****Block Parameter:** custom**Type:** character vector**Values:** 'off' | 'on'**Default:** 'off'**Acceleration due to gravity (m/s<sup>2</sup>) – Acceleration**

9.80665 (default) | scalar

Acceleration from gravity (*g*). in m/s<sup>2</sup>, specified as double scalar.**Dependencies**

This parameter is enabled when the **Change atmospheric parameters** check box is selected.

**Programmatic Use****Block Parameter:** g

**Type:** character vector

**Values:** double scalar

**Default:** 9.80665

### Ratio of specific heats — Ratio of heats

1.4 (default) | scalar

Ratio of specific heats  $\gamma$ , specified as a double value.

#### Dependencies

This parameter is enabled when the **Change atmospheric parameters** check box is selected.

#### Programmatic Use

**Block Parameter:** gamma

**Type:** character vector

**Values:** double scalar

**Default:** 1.4

### Characteristic gas constant (J/Kg/K) — Gas constant

287.0531 (default) | scalar

Characteristic gas constant (R), specified as double scalar, in J/Kg/K.

#### Dependencies

This parameter is enabled when the **Change atmospheric parameters** check box is selected.

#### Programmatic Use

**Block Parameter:** R

**Type:** character vector

**Values:** double scalar

**Default:** 287.0531

### Lapse rate (K/m) — Lapse rate

0.0065 (default) | scalar

Lapse rate of the troposphere, specified as double scalar, in K/m.

#### Dependencies

This parameter is enabled when the **Change atmospheric parameters** check box is selected.

#### Programmatic Use

**Block Parameter:** L

**Type:** character vector

**Values:** double scalar

**Default:** 0.0065

### Height of troposphere (m) — Troposphere height

11000 (default) | scalar

Height of the troposphere (range of decreasing temperatures), specified as double scalar, in m.

#### Dependencies

This parameter is enabled when the **Change atmospheric parameters** check box is selected.

**Programmatic Use****Block Parameter:** h\_trop**Type:** character vector**Values:** double scalar**Default:** 11000**Height of tropopause (m) – Tropopause height**

20000 (default) | scalar

Height of the tropopause (range of constant temperature), specified as double scalar, in m.

**Dependencies**

This parameter is enabled when the **Change atmospheric parameters** check box is selected.

**Programmatic Use****Block Parameter:** h\_strat**Type:** character vector**Values:** double scalar**Default:** 20000**Air density at mean sea level (Kg/m<sup>3</sup>) – Air density**

1.225 (default) | scalar

Air density at mean sea level, specified as double scalar, in Kg/m<sup>3</sup>.

**Dependencies**

This parameter is enabled when the **Change atmospheric parameters** check box is selected.

**Programmatic Use****Block Parameter:** rho0**Type:** character vector**Values:** double scalar**Default:** 1.225**Ambient pressure at mean sea level (N/m<sup>2</sup>) – Ambient pressure**

101325 (default) | scalar

Ambient pressure at mean sea level, specified as double scalar, in N/m<sup>2</sup>.

**Dependencies**

This parameter is enabled when the **Change atmospheric parameters** check box is selected.

**Programmatic Use****Block Parameter:** P0**Type:** character vector**Values:** double scalar**Default:** 101325**Ambient temperature at mean sea level (K) – Ambient temperature**

288.15 (default) | scalar

Ambient temperature at mean sea level ( $T_0$ ), specified as double scalar, in K.

### Dependencies

This parameter is enabled when the **Change atmospheric parameters** check box is selected.

#### Programmatic Use

**Block Parameter:** K

**Type:** character vector

**Values:** double scalar

**Default:** 101325

### Lowest altitude (m) – Lowest altitude

0 (default) | scalar

Lowest altitude above which temperature and pressure lapse, specified as double scalar, in m.

**Lowest altitude (m)** must be below **Height of tropopause**.

### Dependencies

This parameter is enabled when the **Change atmospheric parameters** check box is selected.

#### Programmatic Use

**Block Parameter:** h0

**Type:** character vector

**Values:** double scalar

**Default:** 0

## References

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

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

## See Also

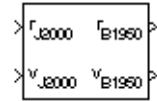
CIRA-86 Atmosphere Model | COESA Atmosphere Model | Lapse Rate Model

**Introduced before R2006a**

## Julian Epoch to Besselian Epoch

Transform position and velocity components from Standard Julian Epoch (J2000) to discontinued Standard Besselian Epoch (B1950)

**Library:** Aerospace Blockset / Utilities / Axes Transformations



### Description

The Julian Epoch to Besselian Epoch block transforms two 3-by-1 vectors of Julian Epoch position ( $\bar{r}_{J2000}$ ), and Julian Epoch velocity ( $\bar{v}_{J2000}$ ) into Besselian Epoch position ( $\bar{r}_{B1950}$ ), and Besselian Epoch velocity ( $\bar{v}_{B1950}$ ). For more information on the transformation, see “Algorithms” on page 4-386.

### Ports

#### Input

##### $r_{J2000}$ — Position

3-by-1 vector

Position in Standard Julian Epoch (J2000), specified as a 3-by-1 vector.

Data Types: double

##### $v_{J2000}$ — Velocity

3-by-1 vector

Velocity in Standard Julian Epoch (J2000), specified as a 3-by-1 vector.

Data Types: double

#### Output

##### $r_{B1950}$ — Position

3-by-1 vector

Position in Standard Besselian Epoch (B1950), returned as a 3-by-1 vector.

Data Types: double

##### $v_{B1950}$ — Velocity

3-by-1 vector

Velocity in Standard Besselian Epoch (B1950), returned as a 3-by-1 vector.

Data Types: double

### Algorithms

The transformation is calculated using:



$$\begin{bmatrix} \bar{r}_{B1950} \\ \bar{v}_{B1950} \end{bmatrix} = \begin{bmatrix} \bar{M}_{rr} & \bar{M}_{vr} \\ \bar{M}_{rv} & \bar{M}_{vv} \end{bmatrix}^T \begin{bmatrix} \bar{r}_{J2000} \\ \bar{v}_{J2000} \end{bmatrix},$$

where

$$(\bar{M}_{rr}, \bar{M}_{vr}, \bar{M}_{rv}, \bar{M}_{vv})$$

are defined as:

$$\bar{M}_{rr} = \begin{bmatrix} 0.9999256782 & -0.0111820611 & -0.0048579477 \\ 0.0111820610 & 0.9999374784 & -0.0000271765 \\ 0.0048579479 & -0.0000271474 & 0.9999881997 \end{bmatrix}$$

$$\bar{M}_{vr} = \begin{bmatrix} 0.00000242395018 & -0.00000002710663 & -0.00000001177656 \\ 0.00000002710663 & 0.00000242397878 & -0.00000000006587 \\ 0.00000001177656 & -0.00000000006582 & 0.00000242410173 \end{bmatrix}$$

$$\bar{M}_{rv} = \begin{bmatrix} -0.000551 & -0.238565 & 0.435739 \\ 0.238514 & -0.002667 & -0.008541 \\ -0.435623 & 0.012254 & 0.002117 \end{bmatrix}$$

$$\bar{M}_{vv} = \begin{bmatrix} 0.99994704 & -0.01118251 & -0.00485767 \\ 0.01118251 & 0.99995883 & -0.00002718 \\ 0.00485767 & -0.00002714 & 1.00000956 \end{bmatrix}$$

## References

- [1] "Supplement to Department of Defense World Geodetic System 1984 Technical Report: Part I - Methods, Techniques and Data Used in WGS84 Development," DMA TR8350.2-A.

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

## See Also

Besselian Epoch to Julian Epoch

Introduced before R2006a

# Julian Date Conversion

Calculate Julian date or modified Julian date

**Library:** Aerospace Blockset / Utilities / Unit Conversions



## Description

The Julian Date Conversion block converts the specified date to the Julian date or modified Julian date.

## Limitations

- This block is valid for all common era (CE) dates in the Gregorian calendar.
- The calculation of Julian date does not take into account leap seconds.

## Ports

### Input

#### day — Clock source

scalar | array

Clock source for model simulation, specified as a scalar or array.

### Dependencies

The presence and label of this port depends on the **Time increment** parameter.

| Port           | Time increment Setting |
|----------------|------------------------|
| day            | Day                    |
| hour           | Hour                   |
| min            | Min                    |
| sec            | Sec                    |
| No inport port | None                   |

Data Types: double

### Output

#### JD — Julian date

scalar | array

Julian date, returned as a scalar or array.

### Dependencies

Data Types: double

## Parameters

### Year — Year

2013 (default) | double, whole number, greater than 1

Year, specified as a scalar, to calculate the Julian date.

#### Programmatic Use

**Block Parameter:** year

**Type:** character vector

**Values:** double, greater than 1

**Default:** '2013'

### Month — Month

January (default) | February | March | April | May | June | July | August | September |  
October | November | December

Month to calculate the Julian date. From the list, select the month from January to December.

#### Programmatic Use

**Block Parameter:** month

**Type:** character vector

**Values:** 'January' | 'February' | 'March' | 'April' | 'May' | 'June' | 'July' | 'August' |  
'September' | 'October' | 'November' | 'December'

**Default:** 'January'

### Day — Day

1 (default) | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |  
24 | 25 | 26 | 27 | 28 | 29 | 30 | 31

Day to calculate the Julian date. From the list, select the day from 1 to 31.

#### Programmatic Use

**Block Parameter:** day

**Type:** character vector

**Values:** '1' | '2' | '3' | '4' | '5' | '5' | '6' | '7' | '8' | '9' | '10' | '11' | '12' | '13' | '14' |  
'15' | '16' | '17' | '18' | '19' | '20' | '21' | '22' | '23' | '24' | '25' | '26' | '27' | '28' |  
'29' | '30' | '31'

**Default:** '1'

### Hour — Hour

0 (default) | double, whole number, 0 to 24

Hour used to calculate the Julian date. Enter a value from 0 to 24.

#### Programmatic Use

**Block Parameter:** hour

**Type:** character vector

**Values:** double, whole number, 0 to 24

**Default:** '0'

### Minutes — Minutes

0 (default) | double, whole number, 0 to 60

Minutes to calculate the Julian date. Enter a number from 0 to 60.

**Programmatic Use****Block Parameter:** min**Type:** character vector**Values:** double, whole number, 0 to 60**Default:** '0'**Seconds — Seconds**

0 (default) | double, whole number, 0 to 60

Specify the seconds used to calculate the Julian date. Enter a number from 0 to 60.

**Programmatic Use****Block Parameter:** sec**Type:** character vector**Values:** double, whole number, 0 to 60**Default:** '0'**Calculate modified Julian date — Modified Julian data**

off (default) | on

Select this check box to calculate the modified Julian date (MJD) for corresponding elements of the year, month, day, hour, minute, and second.

**Dependencies**

Selecting this check box changes the output port label to MJD. Clearing this check box changes the output port label to JD.

**Programmatic Use****Block Parameter:** modflag**Type:** character vector**Values:** 'on' | 'off'**Default:** 'off'**Time increment — Time increment**

Day (default) | Hour | Min | Sec | None

Time increment between the specified date and the desired model simulation time. The block adjusts the calculated Julian date to take into account the time increment from model simulation. For example, selecting Day and connecting a simulation timer to the port means that each time increment unit is one day and the block adjusts its calculation based on that simulation time.

If you select None, the calculated Julian date does not take into account the model simulation time. Selecting this option removes the first block input.

**Dependencies**

This parameter controls the presence and label of output port.

| Time increment Setting | Port |
|------------------------|------|
| Day                    | day  |
| Hour                   | hour |
| Min                    | min  |
| Sec                    | sec  |

| Time increment Setting | Port           |
|------------------------|----------------|
| None                   | No inport port |

**Programmatic Use****Block Parameter:** deltaT**Type:** character vector**Values:** 'Day' | 'Hour' | 'Min' | 'Sec' | 'None'**Default:** 'Day'**Action for out-of-range input – Out-of-range block behavior**

None (default) | Warning | Error

Out-of-range block behavior, specified as follows.

| Action          | Description   |
|-----------------|---|
| None            | No action.  |
| Warning         | Warning in the MATLAB Command Window, model simulation continues. |
| Error (default) | MATLAB returns an exception, model simulation stops.              |

**Programmatic Use****Block Parameter:** errorflag**Type:** character vector**Values:** 'None' | 'Warning' | 'Error'**Default:** 'Error'**Extended Capabilities****C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**See Also**

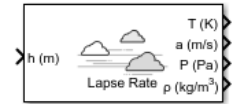
juliandate

**Introduced in R2013b**

# Lapse Rate Model

Implement lapse rate model for atmosphere

**Library:** Aerospace Blockset / Environment / Atmosphere



## Description

The Lapse Rate Model block implements the mathematical representation of the lapse rate atmospheric equations for ambient temperature, pressure, density, and speed of sound for the input geopotential altitude. You can customize this atmospheric model by specifying atmospheric properties.

The ISA Atmosphere Model and Lapse Rate Model blocks are identical blocks. When configured for ISA Atmosphere Model, the block implements ISA values. When configured for Lapse Rate Model, the block implements the mathematical representation of lapse rate atmospheric equations.

The Lapse Rate Model block icon displays the input and output metric units.

## Limitations

- Below the geopotential altitude of 0 km and above the geopotential altitude of the tropopause, temperature and pressure values are held.
- Density and speed of sound are calculated using a perfect gas relationship.

## Ports

### Input

#### **h (m) — Geopotential height**

scalar | array

Geopotential height, specified as a scalar or array.

Data Types: double

### Output

#### **T (K) — Temperature**

scalar | array

Temperature, returned as a scalar or array, in K.

Data Types: double

#### **a (m/s) — Speed of sound**

scalar | array

Speed of sound, returned as a scalar or array, in m/s.

Data Types: double

**P (Pa) – Air pressure**

scalar | array

Air pressure, returned as a scalar or array, in Pa.

Data Types: double

 **$\rho$  (kg/m<sup>3</sup>) – Air density**

scalar | array

Air density, returned as scalar or array, in kg/m<sup>3</sup>.

Data Types: double

**Parameters****Change atmospheric parameters – Customize parameters**

off (default) | on

Customize various atmospheric parameters to be different from the lapse rate values. Selecting this check box converts the block from Lapse Rate Model to ISA Atmosphere Model.

**Dependencies**

Selecting this check box enables the parameters:

- **Acceleration due to gravity (m/s<sup>2</sup>)**
- **Ratio of specific heats**
- **Characteristic gas constant (J/Kg/K)**
- **Lapse rate (K/m)**
- **Height of troposphere (m)**
- **Height of tropopause (m)**
- **Air density at mean sea level (Kg/m<sup>3</sup>)**
- **Ambient pressure at mean sea level (N/m<sup>2</sup>)**
- **Lowest altitude (m)**

**Programmatic Use****Block Parameter:** custom**Type:** character vector**Values:** 'off' | 'on'**Default:** 'off'**Acceleration due to gravity (m/s<sup>2</sup>) – Acceleration**

9.80665 (default) | scalar

Acceleration from gravity ( $g$ ). in m/s<sup>2</sup>, specified as double scalar.**Dependencies**

This parameter is enabled when the **Change atmospheric parameters** check box is selected.

**Programmatic Use****Block Parameter:** g

**Type:** character vector

**Values:** double scalar

**Default:** 9.80665

### **Ratio of specific heats — Ratio of heats**

1.4 (default) | scalar

Ratio of specific heats  $\gamma$ , specified as a double value.

#### **Dependencies**

This parameter is enabled when the **Change atmospheric parameters** check box is selected.

#### **Programmatic Use**

**Block Parameter:** gamma

**Type:** character vector

**Values:** double scalar

**Default:** 1.4

### **Characteristic gas constant (J/Kg/K) — Gas constant**

287.0531 (default) | scalar

Characteristic gas constant (R), specified as double scalar, in J/Kg/K.

#### **Dependencies**

This parameter is enabled when the **Change atmospheric parameters** check box is selected.

#### **Programmatic Use**

**Block Parameter:** R

**Type:** character vector

**Values:** double scalar

**Default:** 287.0531

### **Lapse rate (K/m) — Lapse rate**

0.0065 (default) | scalar

Lapse rate of the troposphere, specified as double scalar, in K/m.

#### **Dependencies**

This parameter is enabled when the **Change atmospheric parameters** check box is selected.

#### **Programmatic Use**

**Block Parameter:** L

**Type:** character vector

**Values:** double scalar

**Default:** 0.0065

### **Height of troposphere (m) — Troposphere height**

11000 (default) | scalar

Height of the troposphere (range of decreasing temperatures), specified as double scalar, in m.

#### **Dependencies**

This parameter is enabled when the **Change atmospheric parameters** check box is selected.



**Programmatic Use****Block Parameter:** h\_trop**Type:** character vector**Values:** double scalar**Default:** 11000**Height of tropopause (m) – Tropopause height**

20000 (default) | scalar

Height of the tropopause (range of constant temperature), specified as double scalar, in m.

**Dependencies**

This parameter is enabled when the **Change atmospheric parameters** check box is selected.

**Programmatic Use****Block Parameter:** h\_strat**Type:** character vector**Values:** double scalar**Default:** 20000**Air density at mean sea level (Kg/m<sup>3</sup>) – Air density**

1.225 (default) | scalar

Air density at mean sea level, specified as double scalar, in Kg/m<sup>3</sup>.

**Dependencies**

This parameter is enabled when the **Change atmospheric parameters** check box is selected.

**Programmatic Use****Block Parameter:** rho0**Type:** character vector**Values:** double scalar**Default:** 1.225**Ambient pressure at mean sea level (N/m<sup>2</sup>) – Ambient pressure**

101325 (default) | scalar

Ambient pressure at mean sea level, specified as double scalar, in N/m<sup>2</sup>.

**Dependencies**

This parameter is enabled when the **Change atmospheric parameters** check box is selected.

**Programmatic Use****Block Parameter:** P0**Type:** character vector**Values:** double scalar**Default:** 101325**Ambient temperature at mean sea level (K) – Ambient temperature**

288.15 (default) | scalar

Ambient temperature at mean sea level ( $T_0$ ), specified as double scalar, in K.

**Dependencies**

This parameter is enabled when the **Change atmospheric parameters** check box is selected.

**Programmatic Use**

**Block Parameter:** K

**Type:** character vector

**Values:** double scalar

**Default:** 101325

**Lowest altitude (m) – Lowest altitude**

0 (default) | scalar

Lowest altitude above which temperature and pressure lapse, specified as double scalar, in m.

**Lowest altitude (m)** must be below **Height of tropopause**.

**Dependencies**

This parameter is enabled when the **Change atmospheric parameters** check box is selected.

**Programmatic Use**

**Block Parameter:** h0

**Type:** character vector

**Values:** double scalar

**Default:** 0

**Algorithms**

These equations define the troposphere:

$$T = T_0 - Lh$$

$$P = P_0 \left( \frac{T}{T_0} \right)^{\frac{g}{LR}}$$

$$\rho = \rho_0 \left( \frac{T}{T_0} \right)^{\frac{g}{LR} - 1}$$

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

These equations define the tropopause (lower stratosphere):

$$T = T_0 - Lh_{ts}$$

$$P = P_0 \left( \frac{T}{T_0} \right)^{\frac{g}{LR}} e^{\frac{g}{RT}(h_{ts} - h)}$$

$$\rho = \rho_0 \left( \frac{T}{T_0} \right)^{\frac{g}{LR} - 1} e^{\frac{g}{RT}(h_{ts} - h)}$$

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

where:

|       |  |
|-------|--|
| $T_0$ | Absolute temperature at mean sea level in kelvin (K) |
|-------|--|

|          |   |
|----------|---|
| $\rho_0$ | Air density at mean sea level in $\text{kg/m}^3$    |
| $P_0$    | Static pressure at mean sea level in $\text{N/m}^2$ |
| $h$      | Altitude in m                                       |
| $h_{ts}$ | Height of the troposphere in m                      |
| $T$      | Absolute temperature at altitude $h$ in kelvin (K)  |
| $\rho$   | Air density at altitude $h$ in $\text{kg/m}^3$      |
| $P$      | Static pressure at altitude $h$ in $\text{N/m}^2$   |
| $a$      | Speed of sound at altitude $h$ in $\text{m/s}^2$    |
| $L$      | Lapse rate in K/m                                   |
| $R$      | Characteristic gas constant J/kg-K                  |
| $\gamma$ | Specific heat ratio                                 |
| $g$      | Acceleration due to gravity in $\text{m/s}^2$       |

## References

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

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

### See Also

COESA Atmosphere Model | ISA Atmosphere Model

**Introduced before R2006a**

# Length Conversion

Convert from length units to desired length units

**Library:** Aerospace Blockset / Utilities / Unit Conversions



## 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 port labels change based on the input and output units selected from the **Initial unit** and the **Final unit** lists.

## Ports

### Input

#### Port\_1 — Length

scalar | array

Length, specified as a scalar or array, in initial length units.

### Dependencies

The input port label depends on the **Initial unit** setting.

Data Types: double

### Output

#### Port\_1 — Length

scalar | array

Length, returned as a scalar or array, in final length units.

### Dependencies

The output port label depends on the **Final unit** setting.

Data Types: double

## Parameters

### Initial unit — Input units

ft (default) | m | km | in | mi | naut mi

Input units, specified as:

|    |        |
|----|--------|
| m  | Meters |
| ft | Feet   |

|         |                |
|---------|----------------|
| km      | Kilometers     |
| in      | Inches         |
| mi      | Miles          |
| naut mi | Nautical miles |

**Dependencies**

The input port label depends on the **Initial unit** setting.

**Programmatic Use**

**Block Parameter:** IU

**Type:** character vector

**Values:** 'm' | 'ft' | 'km' | 'in' | 'mi' | 'naut mi'

**Default:** 'ft'

**Final unit – Input units**

m (default) | ft | km | in | mi | naut mi

Output units, specified as:

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

**Dependencies**

The output port label depends on the **Final unit** setting.

**Programmatic Use**

**Block Parameter:** OU

**Type:** character vector

**Values:** 'm' | 'ft' | 'km' | 'in' | 'mi' | 'naut mi'

**Default:** 'm'

**Extended Capabilities****C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**See Also**

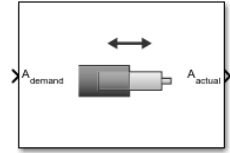
Acceleration Conversion | Angle Conversion | Angular Acceleration Conversion | Angular Velocity Conversion | Density Conversion | Force Conversion | Mass Conversion | Pressure Conversion | Temperature Conversion | Velocity Conversion

**Introduced before R2006a**

# Linear Second-Order Actuator

Implement second-order linear actuator

**Library:** Aerospace Blockset / Actuators



## Description

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

## Ports

### Input

**$A_{\text{demand}}$  — Demanded actuator position**

scalar | array

Demanded actuator position, specified as a scalar or array.

Data Types: double

### Output

**$A_{\text{actual}}$  — Actual actuator position**

scalar | array

Actual actuator position, returned as a scalar or array.

Data Types: double

## Parameters

**Natural frequency — Natural frequency**

1 (default) | scalar

Natural frequency of the actuator, specified as a scalar double, in radians per second.

### Programmatic Use

**Block Parameter:** `wn_fin`

**Type:** character vector

**Values:** scalar | double

**Default:** '1'

**Damping ratio — Damping ratio**

0.3 (default) | scalar

Damping ratio of the actuator, specified as a scalar double.

**Programmatic Use****Block Parameter:** z\_fin**Type:** character vector**Values:** scalar | double**Default:** '0.3'**Initial position — Initial position**

0 (default) | scalar

Initial position of the actuator, specified as a scalar double. The units of initial position must be the same as the  $A_{\text{demand}}$  input.

**Programmatic Use****Block Parameter:** fin\_act\_0**Type:** character vector**Values:** scalar | double**Default:** '0'**Initial velocity — Initial velocity**

0 (default) | scalar

Initial velocity of the actuator, specified as a scalar double. The units of initial velocity must be the same as the  $A_{\text{demand}}$  input..

**Programmatic Use****Block Parameter:** fin\_act\_vel**Type:** character vector**Values:** scalar | double**Default:** '0'**Extended Capabilities****C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**See Also**

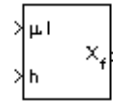
Nonlinear Second-Order Actuator

**Introduced in R2012a**

## LLA to ECEF Position

Calculate Earth-centered Earth-fixed (ECEF) position from geodetic latitude, longitude, and altitude above planetary ellipsoid

**Library:** Aerospace Blockset / Utilities / Axes Transformations



### Description

The LLA to ECEF Position block converts geodetic latitude ( $\bar{\mu}$ ), longitude ( $\bar{l}$ ), and altitude ( $\bar{h}$ ) above the planetary ellipsoid into a 3-by-1 vector of ECEF position ( $\bar{p}$ ). Latitude and longitude values can be any value. However, latitude values of +90 and -90 may return unexpected values because of singularity at the poles. For more information the ECEF position calculation, see “Algorithms” on page 4-404.

### Limitations

- The planet is assumed to be ellipsoidal. To use a spherical planet, set the **Flattening** parameter to zero.
- The implementation of the ECEF coordinate system assumes that the origin is at the center of the planet, the x-axis intersects the Greenwich meridian and the equator, the z-axis is the mean spin axis of the planet, positive to the north, and the y-axis completes the right-handed system.

### Ports

#### Input

##### $\bar{\mu} \ \bar{l}$ — Geodetic latitude and longitude

2-by-1 vector

Geodetic latitude and longitude, specified as a 2-by-1 vector, in degrees.

Data Types: double

##### $h$ — Altitude

scalar

Altitude above the planetary ellipsoid, specified as a scalar.

Data Types: double

#### Output

##### $X_f$ — Position

3-by-1 vector

Position in ECEF frame, returned as a 3-by-1 vector, in the same units as the input at the  $h$  port.

Data Types: double



## Parameters

### Units – Units

Metric (MKS) (default) | English

Parameter and output units:

| Units        | Radius from CG to Center of Planet | Equatorial Radius |
|--------------|------------------------------------|-------------------|
| Metric (MKS) | Meters                             | Meters            |
| English      | Feet                               | Feet              |

### Dependencies

To enable this, set **Planet model** to Earth (WGS84).

#### Programmatic Use

**Block Parameter:** units

**Type:** character vector

**Values:** 'Metric (MKS)' | 'English'

**Default:** 'Metric (MKS)'

### Planet model – Planet model

Earth (WGS84) (default) | Custom

Planet model to use, Custom or Earth (WGS84).

#### Programmatic Use

**Block Parameter:** ptype

**Type:** character vector

**Values:** 'Earth (WGS84)' | 'Custom'

**Default:** 'Earth (WGS84)'

### Flattening – Flattening of planet

1/298.257223563 (default) | scalar

Flattening of the planet, specified as a double scalar.

### Dependencies

To enable this parameter, set **Planet model** to Custom.

#### Programmatic Use

**Block Parameter:** F

**Type:** character vector

**Values:** double scalar

**Default:** 1/298.257223563

### Equatorial radius of planet – Radius of planet at equator

6378137 (default) | scalar

Radius of the planet at its equator, in the same units as the desired units for ECEF position.

### Dependencies

To enable this parameter, set **Planet model** to Custom.

**Programmatic Use****Block Parameter:** R**Type:** character vector**Values:** double scalar**Default:** 6378137**Algorithms**

The ECEF position is calculated from the geocentric latitude at mean sea-level ( $\lambda_s$ ) and longitude using:

$$\bar{p} = \begin{bmatrix} \bar{p}_x \\ \bar{p}_y \\ \bar{p}_z \end{bmatrix} = \begin{bmatrix} r_s \cos \lambda_s \cos l + h \cos \mu \cos l \\ r_s \cos \lambda_s \sin l + h \cos \mu \sin l \\ r_s \sin \lambda_s + h \sin \mu \end{bmatrix},$$

where geocentric latitude at mean sea-level and the radius at a surface point ( $r_s$ ) are defined by flattening ( $\bar{f}$ ), and equatorial radius ( $\bar{R}$ ) in the following relationships:

$$\lambda_s = \text{atan}\left((1 - f)^2 \tan \mu\right)$$

$$r_s = \sqrt{\frac{\bar{R}^2}{1 + \left(1/(1 - f)^2 - 1\right) \sin^2 \lambda_s}}$$

**References**

- [1] Stevens, B. L., and F. L. Lewis. *Aircraft Control and Simulation*, Hoboken, NJ: John Wiley & Sons, 1992.
- [2] Zipfel, Peter H., *Modeling and Simulation of Aerospace Vehicle Dynamics*. Second Edition. Reston, VA: AIAA Education Series, 2000.
- [3] *Recommended Practice for Atmospheric and Space Flight Vehicle Coordinate Systems*, R-004-1992, ANSI/AIAA, February 1992.

**Extended Capabilities****C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**See Also**

Direction Cosine Matrix ECEF to NED | Direction Cosine Matrix ECEF to NED to Latitude and Longitude | ECEF Position to LLA | Flat Earth to LLA | Radius at Geocentric Latitude

**Topics**

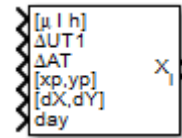
“About Aerospace Coordinate Systems” on page 2-8

**Introduced before R2006a**

# LLA to ECI Position

Convert latitude, longitude, altitude (LLA) coordinates to Earth-centered inertial (ECI) coordinates

**Library:** Aerospace Blockset / Utilities / Axes Transformations



## Description

The LLA to ECI Position block converts latitude, longitude, and altitude (LLA) coordinates to Earth-centered inertial (ECI) position coordinates, based on the specified reduction method and Universal Coordinated Time (UTC), for the specified time and geophysical data. The latitude and longitude values can be any value. However, latitude values of +90 and -90 may return unexpected values because of singularity at the poles.

## Ports

### Input

**[ $\mu$  l h]** — Latitude, longitude, and altitude

three-element vector

Latitude, longitude, and altitude values of coordinates to convert, specified as a three-element vector, in degrees.

Data Types: double

**$\Delta$ UT1** — Difference between UTC and Universal Time

scalar

Difference between UTC and Universal Time (UT1) in seconds, specified as a scalar, for which the block calculates the direction cosine or transformation matrix.

Example: 0.234

### Dependencies

To enable this, select **Higher accuracy parameters**.

Data Types: double

**$\Delta$ AT** — Difference between International Atomic Time and UTC

scalar

Difference between International Atomic Time (IAT) and UTC, specified as a scalar, in seconds, for which the block calculates the direction cosine or transformation matrix.

Example: 32

### Dependencies

To enable this port, select **Higher accuracy parameters**.

Data Types: double

### [xp, yp] — Polar displacement of Earth

1-by-2 array

Polar displacement of Earth, specified as a 1-by-2 array, in radians, from the motion of the Earth crust, along the x- and y-axes.

Example: [-0.0682e-5 0.1616e-5]

#### Dependencies

To enable this port, select **Higher accuracy parameters**.

Data Types: double

### Port\_5 — Adjustment based on reduction method

1-by-2 array

Adjustment based on reduction method, specified as 1-by-2 array. The name of the port depends on the setting of the **Reduction** parameter:

- If reduction method is IAU-2000/2006, this input is the adjustment to the location of the Celestial Intermediate Pole (CIP), specified in radians. This location ([dX,dY]) is along the x-axis and y-axis.
- If reduction method is IAU-76/FK5, this input is the adjustment to the longitude ( $[\Delta\delta\psi, \Delta\delta\varepsilon]$ ), specified in radians.

For historical values, see International Earth Rotation and Reference Systems Service and navigate to the Earth Orientation Data Data/Products page.

Example: [-0.2530e-6 -0.0188e-6]

#### Dependencies

To enable this port, select **Higher accuracy parameters**.

Data Types: double

### Port\_6 — Time increment source

scalar

Time increment source, specified as a scalar, such as the Clock block.

#### Dependencies

- The port name and time increment depend on the **Time Increment** parameter.

| Time Increment Value | Port Name |
|----------------------|-----------|
| Day                  | day       |
| Hour                 | hour      |
| Min                  | min       |
| Sec                  | sec       |
| None                 | No port   |

- To disable this port, set the **Time Increment** parameter to None.

Data Types: double

## Output

### $X_i$ — Original position

3-by-1 element vector

Original position vector with respect to the ECI reference system, returned as a 3-by-1 element vector.

Data Types: double

## Parameters

### Reduction — Reduction method

IAU-76/FK5 (default) | IAU-2000/2006

Reduction method to convert the coordinates. Method can be one of:

- IAU-76/FK5

Reduce the calculation using the International Astronomical Union 76/Fifth Fundamental Catalogue (IAU-76/FK5) reference system. Choose this reduction method if the reference coordinate system for the conversion is FK5.

---

**Note** This method uses the IAU 1976 precession model and the IAU 1980 theory of nutation to reduce the calculation. This model and theory are no longer current, but the software provides this reduction method for existing implementations. Because of the polar motion approximation that this reduction method uses, the block calculates the transformation matrix rather than the direction cosine matrix.

---

- IAU-2000/2006

Reduce the calculation using the International Astronomical Union 2000/2006 reference system. Choose this reduction method if the reference coordinate system for the conversion is IAU-2000. This reduction method uses the P03 precession model to reduce the calculation.

### Programmatic Use

**Block Parameter:** red

**Type:** character vector

**Values:** 'IAU-2000/2006' | 'IAU-76/FK5'

**Default:** 'IAU-2000/2006'

### Year — Year

2013 (default) | double, whole number, greater than 1

Year to calculate the Universal Coordinated Time (UTC) date. Enter a double value that is a whole number greater than 1, such as 2013.

### Programmatic Use

**Block Parameter:** year

**Type:** character vector

**Values:** double, whole number, greater than 1

**Default:** '2013'

**Month — Month**

January (default) | February | March | April | May | June | July | August | September | October | November | December

Month to calculate the UTC date.

**Programmatic Use**

**Block Parameter:** month

**Type:** character vector

**Values:** 'January' | 'February' | 'March' | 'April' | 'May' | 'June' | 'July' | 'August' | 'September' | 'October' | 'November' | 'December'

**Default:** 'January'

**Day — Day**

1 (default) | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31

Day to calculate the UTC date.

**Programmatic Use**

**Block Parameter:** day

**Type:** character vector

**Values:** '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9' | '10' | '11' | '12' | '13' | '14' | '15' | '16' | '17' | '18' | '19' | '20' | '21' | '22' | '23' | '24' | '25' | '26' | '27' | '28' | '29' | '30' | '31'

**Default:** '1'

**Hour — Hour**

0 (default) | double, whole number, 0 to 24

Hour to calculate the UTC date. Enter a double value that is a whole number, from 0 to 24.

**Programmatic Use**

**Block Parameter:** hour

**Type:** character vector

**Values:** double, whole number, 0 to 24

**Default:** '0'

**Minutes — Minutes**

0 (default) | double, whole number, 0 to 60

Minutes to calculate the UTC date. Enter a double value that is a whole number, from 0 to 60.

**Programmatic Use**

**Block Parameter:** min

**Type:** character vector

**Values:** double, whole number, 0 to 60

**Default:** '0'

**Seconds — Seconds**

0 (default)

Seconds to calculate the UTC date. Enter a double value that is a whole number, from 0 to 60.

**Programmatic Use**

**Block Parameter:** sec

**Type:** character vector

**Values:** double, whole number, 0 to 60

**Default:** '0'

### Time increment — Time increment

Day (default) | None | Hour | Min | Sec

Time increment between the specified date and the desired model simulation time. The block adjusts the calculated direction cosine matrix to take into account the time increment from model simulation. For example, selecting Day and connecting a simulation timer to the port means that each time increment unit is one day and the block adjusts its calculation based on that simulation time.

This parameter corresponds to the time increment input, the clock source.

If you select None, the calculated Julian date does not take into account the model simulation time.

#### Programmatic Use

**Block Parameter:** deltaT

**Type:** character vector

**Values:** 'None' | 'Day' | 'Hour' | 'Min' | 'Sec'

**Default:** 'Day'

### Action for out-of-range input — Action taken when inputs are out of range

Error (default) | Warning | None

Specify the block behavior when the block inputs are out of range.

| Action          | Description   |
|-----------------|---|
| None            | No action.  |
| Warning         | Warning in the MATLAB Command Window, model simulation continues. |
| Error (default) | MATLAB returns an exception, model simulation stops.              |

#### Programmatic Use

**Block Parameter:** errorflag

**Type:** character vector

**Values:** 'None' | 'Warning' | 'Error'

**Default:** 'Error'

### Higher accuracy parameters — Enable higher accuracy parameters

on (default) | off

Select this check box to allow the following as block inputs. These inputs let you better control the conversion result. See "Input" on page 4-405 for a description.

- $\Delta UT1$
- $\Delta AT$
- $[x_p, y_p]$
- $[\Delta\delta\psi, \Delta\delta\varepsilon]$  or  $[dX, dY]$

#### Programmatic Use

**Block Parameter:** extraparamflag

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'on'

### Units — Units

Metric (MKS) (default) | English

Specifies the parameter and output units.

| Units        | Position | Equatorial Radius | Altitude |
|--------------|----------|-------------------|----------|
| Metric (MKS) | Meters   | Meters            | Meters   |
| English      | Feet     | Feet              | Feet     |

### Dependencies

To enable this parameter, set **Earth model** to Earth (WGS84).

#### Programmatic Use

**Block Parameter:** eunits

**Type:** character vector

**Values:** 'Metric (MKS)' | 'English'

**Default:** 'Metric (MKS)'

### Earth model — Earth model

WGS84 (default) | Custom

Earth model to use, Custom or Earth (WGS84).

#### Programmatic Use

**Block Parameter:** earthmodel

**Type:** character vector

**Values:** 'Earth (WGS84)' | 'Custom'

**Default:** 'Earth (WGS84)'

### Flattening — Flattening of planet

1/298.257223563 (default) | scalar

Flattening of the planet, specified as a double scalar.

### Dependencies

To enable this parameter, set **Earth model** to Custom.

#### Programmatic Use

**Block Parameter:** flat

**Type:** character vector

**Values:** double scalar

**Default:** 1/298.257223563

### Equatorial radius — Radius of planet at equator

6378137 (default) | scalar

Radius of the planet at its equator.

### Dependencies

To enable this parameter, set **Earth model** to Custom.



**Programmatic Use****Block Parameter:** eqradius**Type:** character vector**Values:** double scalar**Default:** 6378137**Extended Capabilities****C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**See Also**

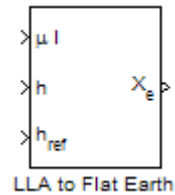
ECI Position to LLA

**External Websites**<https://www.iers.org>**Introduced in R2014a**

## LLA to Flat Earth

Estimate flat Earth position from geodetic latitude, longitude, and altitude

**Library:** Aerospace Blockset / Utilities / Axes Transformations



### Description

The LLA to Flat Earth block converts a geodetic latitude ( $\bar{\mu}$ ), longitude ( $\bar{l}$ ), and altitude ( $h$ ) into a 3-by-1 vector of Flat Earth position ( $\bar{p}$ ). Latitude and longitude values can be any value. However, latitude values of +90 and -90 may return unexpected values because of singularity at the poles. For more information on the flat Earth coordinate system, see “Algorithms” on page 4-415.

### Limitations

- This estimation method assumes that the flight path and bank angle are zero.
- This estimation method assumes the flat Earth z-axis is normal to the Earth at the initial geodetic latitude and longitude only. This method has higher accuracy over small distances from the initial geodetic latitude and longitude, and nearer to the equator. The longitude has higher accuracy with smaller variations in latitude. Additionally, longitude is singular at the poles.

### Ports

#### Input

##### $\mu l$ — Geodetic latitude and longitude

2-by-1 vector

Geodetic latitude and longitude, specified as a 2-by-1 vector, in degrees.

Data Types: double

##### $h$ — Altitude

scalar

Altitude above the input reference altitude, specified as a scalar, in the same units as the flat Earth position.

Data Types: double

##### $h_{ref}$ — Reference height

scalar

Reference height from the surface of the Earth to the flat Earth frame, specified as a scalar, in the same units as the flat Earth position. The reference height is estimated with regard to Earth frame.

Data Types: double

**$\mu_{\text{ref}}$   $\mathbf{l}_{\text{ref}}$  — Reference location**

2-by-1 vector

Reference location, specified as a 2-by-1 vector, in degrees of latitude and longitude, for the origin of the estimation and the origin of the flat Earth coordinate system. Use this port if you want to specify the reference location as a dynamic value.

**Dependencies**

To enable this port, select **Input reference position and orientation**.

Data Types: double

 **$\psi_{\text{ref}}$  — Direction of flat Earth x-axis**

scalar

Angle, specified as a scalar, for converting flat Earth x and y coordinates to North and East coordinates. Use this port if you want to specify the angle as a dynamic value.

**Dependencies**

To enable this port, select **Input reference position and orientation**.

Data Types: double

**Output** **$\mathbf{X}_e$  — Position**

3-by-1 vector | 4-by-1 vector

Position in flat Earth frame, returned as a vector.

Data Types: double

**Parameters****Units — Units**

Metric (MKS) (default) | English

Parameter and output units:

| Units        | Position | Equatorial Radius | Altitude |
|--------------|----------|-------------------|----------|
| Metric (MKS) | Meters   | Meters            | Meters   |
| English      | Feet     | Feet              | Feet     |

**Programmatic Use****Block Parameter:** units**Type:** character vector**Values:** 'Metric (MKS)' | 'English'**Default:** 'Metric (MKS)'**Planet model — Planet model**

Earth (WGS84) (default) | Custom

Planet model to use, Custom or Earth (WGS84).

**Dependencies**

Selecting the Custom option disables the **Units** parameter and enables these parameters:

- **Flattening**
- **Equatorial radius of planet**

**Programmatic Use**

**Block Parameter:** ptype

**Type:** character vector

**Values:** 'Earth (WGS84)' | 'Custom'

**Default:** 'Earth (WGS84)'

**Flattening — Flattening of Earth**

1/298.257223563 (default) | scalar

Flattening of the planet, specified as a double scalar.

**Dependencies**

To enable this parameter, set **Planet model** to Custom.

**Programmatic Use**

**Block Parameter:** F

**Type:** character vector

**Values:** double scalar

**Default:** 1/298.257223563

**Equatorial radius of planet — Radius of planet at equator**

6378137 (default) | scalar

Radius of the planet at its equator, in the same units as the desired units for ECEF position.

**Dependencies**

To enable this parameter, set **Planet model** to Custom.

**Programmatic Use**

**Block Parameter:** R

**Type:** character vector

**Values:** double scalar

**Default:** 6378137

**Input reference position and orientation — Input reference position and orientation as ports**

off (default) | on

Select this check box to enable ports for reference position and angle to convert flat Earth. Select this check box if you want to specify the reference positions and angle as dynamic values.

**Programmatic Use**

**Block Parameter:** refPosPort

**Type:** character vector

**Values:** 'off' | 'on'

**Default:** 'off'

**Reference geodetic latitude and longitude [deg] – Initial geodetic latitude and longitude**

[0 10] (default) | 2-by-1 vector

Reference location in latitude and longitude, specified as 2-by-1 vector, in degrees.

**Dependencies**

To enable this parameter, clear **Input reference position and orientation**.

**Programmatic Use****Block Parameter:** LL0**Type:** character vector**Values:** 2-by-1 vector**Default:** [0 10]**Direction of flat Earth x-axis (degrees clockwise from north) – Flat Earth x and y coordinates**

0 (default) | scalar

Angle to convert flat Earth x and y coordinates to North and East coordinates, specified as a scalar double, in degrees.

**Dependencies**

To enable this parameter, clear **Input reference position and orientation**.

**Programmatic Use****Block Parameter:** psi**Type:** character vector**Values:** double scalar**Default:** 0**Algorithms**

The flat Earth coordinate system assumes the z-axis is downward positive. The estimation begins by finding the small changes in latitude and longitude from the output latitude and longitude minus the initial latitude and longitude.

$$d\mu = \mu - \mu_0$$

$$dt = t - t_0$$

To convert geodetic latitude and longitude to the North and East coordinates, the estimation uses the radius of curvature in the prime vertical ( $R_N$ ) and the radius of curvature in the meridian ( $R_M$ ).  $R_N$  and  $R_M$  are defined by the following relationships:

$$R_N = \frac{R}{\sqrt{1 - (2f - f^2)\sin^2\mu_0}}$$

$$R_M = R_N \frac{1 - (2f - f^2)}{1 - (2f - f^2)\sin^2\mu_0}$$

where ( $R$ ) is the equatorial radius of the planet and  $f$  is the flattening of the planet.

Small changes in the North (dN) and East (dE) positions are approximated from small changes in the North and East positions by

$$dN = \frac{d\mu}{\text{atan}\left(\frac{1}{R_M}\right)}$$

$$dE = \frac{d\mu}{\text{atan}\left(\frac{1}{R_N \cos \mu_0}\right)}$$

With the conversion of the North and East coordinates to the flat Earth x and y coordinates, the transformation has the form of

$$\begin{bmatrix} p_x \\ p_y \end{bmatrix} = \begin{bmatrix} \cos \psi & \sin \psi \\ -\sin \psi & \cos \psi \end{bmatrix} \begin{bmatrix} N \\ E \end{bmatrix},$$

where

$$(\psi)$$

is the angle in degrees clockwise between the x-axis and north.

The flat Earth z-axis value is the negative altitude minus the reference height ( $h_{ref}$ ):

$$p_z = -h - h_{ref}.$$

## References

- [1] Stevens, B. L., and F. L. Lewis. *Aircraft Control and Simulation*, Hoboken, NJ: John Wiley & Sons, 2003.
- [2] Etkin, B. *Dynamics of Atmospheric Flight* Hoboken, NJ: John Wiley & Sons, 1972.

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

## See Also

Direction Cosine Matrix ECEF to NED | Direction Cosine Matrix ECEF to NED to Latitude and Longitude | ECEF Position to LLA | Flat Earth to LLA | Geocentric to Geodetic Latitude | LLA to ECEF Position | Radius at Geocentric Latitude

### Introduced in R2011a

# Mach Number

Compute Mach number using velocity and speed of sound

**Library:** Aerospace Blockset / Flight Parameters



## Description

The Mach Number block computes the Mach number. The Mach number is defined as

$$Mach = \frac{\sqrt{V \cdot V}}{a},$$

where  $a$  is the speed of sound and  $V$  is the velocity vector.

## Ports

### Input

#### **V — Velocity**

$m$ -by-3 vector

Velocity vector, specified as an  $m$ -by-3 vector.

Data Types: double

#### **a — Speed of sound**

1-by-1 array

Speed of sound, specified as a 1-by-1 array.

Data Types: double

### Output

#### **Mach — Mach number**

$m$ -by-1 vector

Mach number, returned as an  $m$ -by-1 vector.

Data Types: double

## Extended Capabilities

### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

## See Also

Aerodynamic Forces and Moments | Dynamic Pressure

**Introduced before R2006a**



# Mass Conversion

Convert from mass units to desired mass units

**Library:** Aerospace Blockset / Utilities / Unit Conversions



## 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 port block labels change based on the input and output units selected from the **Initial unit** and the **Final unit** lists.

## Ports

### Input

#### Port\_1 – Mass

scalar | array

Mass, specified as a scalar or array, in initial mass units.

#### Dependencies

The input port label depends on the **Initial unit** setting.

Data Types: double

### Output

#### Port\_1 – Mass

scalar | array

Mass, returned as a scalar or array, in final mass units.

#### Dependencies

The output port label depends on the **Final unit** setting.

Data Types: double

## Parameters

### Initial unit – Input units

lbn (default) | kg | slug

Input units, specified as.

|     |            |
|-----|------------|
| lbn | Pound mass |
| kg  | Kilograms  |

|      |       |
|------|-------|
| slug | Slugs |
|------|-------|

**Dependencies**

The input port label depends on the **Initial unit** setting.

**Programmatic Use**

**Block Parameter:** IU

**Type:** character vector

**Values:** 'l<sub>bm</sub>' | 'kg' | 'slug'

**Default:** 'l<sub>bm</sub>'

**Final unit – Output units**

kg (default) | l<sub>bm</sub> | slug

Output units, specified as:

|                 |            |
|-----------------|------------|
| l <sub>bm</sub> | Pound mass |
| kg              | Kilograms  |
| slug            | Slugs      |

**Dependencies**

The output port label depends on the **Final unit** setting.

**Programmatic Use**

**Block Parameter:** OU

**Type:** character vector

**Values:** 'l<sub>bm</sub>' | 'kg' | 'slug'

**Default:** 'kg'

**Extended Capabilities****C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**See Also**

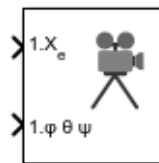
Acceleration Conversion | Angle Conversion | Angular Acceleration Conversion | Angular Velocity Conversion | Density Conversion | Force Conversion | Length Conversion | Pressure Conversion | Temperature Conversion | Velocity Conversion

**Introduced before R2006a**

# MATLAB Animation

Create six-degrees-of-freedom multibody custom geometry block

**Library:** Aerospace Blockset / Animation / MATLAB-Based Animation



## Description

The MATLAB Animation block creates a six-degrees-of-freedom multibody custom geometry block based on the `Aero.Animation` object. This block animates one or more vehicle geometries with  $x$ - $y$ - $z$  position and Euler angles through the specified bounding box, camera offset, and field of view. This block expects the rotation order  $z$ - $y$ - $x$  (psi, theta, phi).

To update the camera parameters in the animation, first set the parameters then close and double-click the block to reopen the MATLAB Animation window.

To access the parameters for this block, do one of:

- Right-click the block, then select **Mask > Mask Parameters**.
- Double-click the block to display the MATLAB Animation window, then click the **Block Parameters** icon.

---

**Note** The underlying graphics system stores values in single precision. As a result, you might notice that motion at coordinate positions greater than approximately  $1e6$  appear unstable. This is because a single-precision number has approximately six digits of precision. The instability is due to quantization at the local value of the `eps` MATLAB function. To visualize more stable motion for coordinates beyond  $1e6$ , either offset the input data to a local zero, or scale down the coordinate values feeding the visualization.

---

## Ports

### Input

#### 1.X<sub>e</sub> — Downrange position, crossrange position, and altitude of vehicle

three-element vector

Downrange position, crossrange position, and altitude of the vehicle in Earth coordinates, specified as a three-element vector. The number on the port indicates the vehicle number.

Data Types: `double`

#### 1.φ θ ψ — Euler angles

three-element vector

Euler angles (roll, pitch, and yaw) of the vehicle, specified as a three-element vector. The number on the port indicates the vehicle number.

Data Types: double

### **Port\_N – Downrange position, crossrange position, and altitude of vehicle**

three-element vector

Nth input downrange position, crossrange position, and altitude of the vehicle in Earth coordinates, specified as a three-element vector. The number on the port indicates the vehicle number.

#### **Dependencies**

To enable this port, select a **Vehicles** number from 2 to 10.

Data Types: double

### **Port\_N – Euler angles**

three-element vector

Nth input Euler angles (roll, pitch, and yaw) of the vehicle, specified as a three-element vector. The number on the port indicates the vehicle number.

#### **Dependencies**

To enable this port, select a **Vehicles** number from 2 to 10.

Data Types: double

## **Parameters**

### **Vehicles – Vehicle to animate**

1 (default) | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10

Vehicle to animate, specified from 1 to 10.

#### **Dependencies**

Selecting a vehicle number from 2 to 10 adds corresponding input ports. Each vehicle has its own set of input ports, denoted by the number at the beginning of the input port label.

#### **Programmatic Use**

**Block Parameter:** Vehicles

**Type:** character vector

**Values:** 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10

**Default:** '1'

### **Geometries (use 'quotes' on filenames) – Vehicle geometries**

'astredwedge.mat' (default) | MAT-file

Vehicle geometries, specified in a MAT-file. You can specify these geometries using:

- Variable name, for example geomVar
- Cell array of variable names, for example {geomVar, AltGeomVar}
- Character vector with single quotes, for example, 'astredwedge.mat'
- Mixed cell array of variable names and character vectors, for example {'file1.mat', 'file2.mat', 'file3.ac', geomVar}

---

**Note** All specified geometries specified must exist in the MATLAB workspace and file names must exist in the current folder or be on the MATLAB path.

---

**Programmatic Use****Block Parameter:** Geometries**Type:** character vector**Values:** MAT-file**Default:** 'astredwedge.mat'**Bounding box coordinates – Boundary coordinates**

[-50,50,-50,50,-50,50] (default) | six-element vector

Boundary coordinates for the vehicle, specified as a six-element vector.

This parameter is not tunable during simulation. A change to this parameter takes effect after simulation stops.

**Programmatic Use****Block Parameter:** BoundingBoxCoordinates**Type:** character vector**Values:** six-element vector**Default:** '[-50,50,-50,50,-50,50]'**Camera offset – Distance from camera aim point to camera**

[-150,-50,0] (default) | three-element vector

Distance from the camera aim point to the camera itself, specified as a three-element vector.

This parameter is not tunable during simulation. A change to this parameter takes effect after simulation stops.

**Programmatic Use****Block Parameter:** CameraOffset**Type:** character vector**Values:** three-element vector**Default:** '[-150,-50,0]'**Camera view angle – Camera view angle**

3 (default) | scalar

Camera view angle, specified as a double scalar. By default, the camera aim point is the position of the first body lagged dynamically to indicate motion.

This parameter is not tunable during simulation. A change to this parameter takes effect after simulation stops.

**Programmatic Use****Block Parameter:** CameraViewAngle**Type:** character vector**Values:** double scalar**Default:** '3'**Sample time – Sample time**

0.2 (default) | scalar

Sample time (-1 for inherited), specified as a double scalar.

**Programmatic Use**

**Block Parameter:** SampleTime

**Type:** character vector

**Values:** double scalar

**Default:** '0.2'

## **Extended Capabilities**

### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

### **See Also**

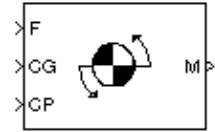
Aero.Animation

**Introduced in R2007a**

# Moments about CG due to Forces

Compute moments about center of gravity due to forces applied at a point, not center of gravity

**Library:** Aerospace Blockset / Mass Properties



## Description

The Moments about CG due to Forces block computes moments about center of gravity due to forces that are applied at point CP, not at the center of gravity.

## Ports

### Input

#### **F — Applied forces**

3-element vector

Forces applied at point CP, specified as a three-element vector.

Data Types: `double`

#### **CG — Center of gravity**

3-element vector

Center of gravity, specified as three-element vector.

Data Types: `double`

#### **CP — Application point of forces**

3-element vector

Application point of forces, specified as a three-element vector.

Data Types: `double` | `bus`

### Output

#### **M — Moments at the center of gravity**

3-element vector

Moments at the center of gravity in x-axis, y-axis and z-axis, returned as a three-element vector.

Data Types: `double`

## Extended Capabilities

### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**See Also**

Aerodynamic Forces and Moments | Estimate Center of Gravity

**Introduced before R2006a**



# Moon Libration

Implement Moon librations

**Library:** Aerospace Blockset / Environment / Celestial Phenomena



## Description

The Moon Libration block implements the Moon librations using Chebyshev coefficients or a given Julian date. The block uses the Chebyshev coefficients that the NASA Jet Propulsion Laboratory provides.

---

**Tip** For  $T_{JD}$ , Julian date input for the block:

- Calculate the date using the Julian Date Conversion block or the Aerospace Toolbox juliandate function.
  - Calculate the Julian date using some other means and input it using the Constant block.
- 

## Ports

### Input

#### $T_{JD}$ — Julian date

scalar | positive | between minimum and maximum Julian dates

Julian date, specified as a positive scalar between minimum and maximum Julian dates.

See the **Ephemeris model** parameter for the minimum and maximum Julian dates.

#### Dependencies

This port displays if the **Epoch** parameter is set to Julian date.

Data Types: double

#### $T_{0JD}$ — Fixed Julian date

scalar | positive

Fixed Julian date for a specific epoch that is the most recent midnight at or before the interpolation epoch, specified as a positive scalar. The sum of  $T_{0JD}$  and  $\Delta T_{JD}$  must fall between the minimum and maximum Julian dates.

See the **Ephemeris model** parameter for the minimum and maximum Julian dates.

#### Dependencies

This port displays if the **Epoch** parameter is set to  $T_0$  and elapsed Julian time.

Data Types: double

**$\Delta T_{JD}$  — Elapsed Julian time**

scalar | positive

Elapsed Julian time between the fixed Julian date and the ephemeris time, specified as a positive scalar. The sum of  $T0_{JD}$  and  $\Delta T_{JD}$  must fall between the minimum and maximum Julian date.

See the **Ephemeris model** parameter for the minimum and maximum Julian dates.

**Dependencies**

This port displays if the **Epoch** parameter is set to T0 and elapsed Julian time.

Data Types: double

**Output** **$\varphi \ \theta \ \psi$  (rad) — Euler angles**

vector

Euler angles ( $\varphi \ \theta \ \psi$ ) for Moon attitude, in rad.

Data Types: double

 **$\omega$  (rad/day) — Moon libration Euler angular rate**

vector

Moon libration Euler angular rates ( $\omega$ ), in rad/day.

Data Types: double

**Parameters****Epoch — Epoch**

Julian date (default) | T0 and elapsed Julian time

Epoch, specified as:

- Julian date

Julian date to calculate the Moon libration. When this option is selected, the block has one input port,  $T_{JD}$ .

- T0 and elapsed Julian time

Julian date, specified by two block inputs:

- Fixed Julian date representing a starting epoch.
- Elapsed Julian time between the fixed Julian date ( $T0_{JD}$ ) and the desired model simulation time. The sum of  $T0_{JD}$  and  $\Delta T_{JD}$  must fall between the minimum and maximum Julian dates.

**Programmatic Use**

**Block Parameter:** epochflag

**Type:** character vector

**Values:** Julian date | T0 and elapsed Julian time

**Default:** 'Julian date'

**Ephemeris model – Ephemeris model**

DE405 (default) | DE421 | DE423 | DE430 | DE432t

Select one of the following ephemeris models defined by the Jet Propulsion Laboratory.

| <b>Ephemeris Model</b> | <b>Description</b>   |
|------------------------|--|
| DE405                  | Released in 1998. This ephemeris takes into account the Julian date range 2305424.50 (December 9, 1599) to 2525008.50 (February 20, 2201).<br><br>This block implements these ephemerides with respect to the International Celestial Reference Frame version 1.0, adopted in 1998.        |
| DE421                  | Released in 2008. This ephemeris takes into account the Julian date range 2414992.5 (December 4, 1899) to 2469808.5 (January 2, 2050).<br><br>This block implements these ephemerides with respect to the International Celestial Reference Frame version 1.0, adopted in 1998.            |
| DE423                  | Released in 2010. This ephemeris takes into account the Julian date range 2378480.5 (December 16, 1799) to 2524624.5 (February 1, 2200).<br><br>This block implements these ephemerides with respect to the International Celestial Reference Frame version 2.0, adopted in 2010.          |
| DE430                  | Released in 2013. This ephemeris takes into account the Julian date range 2287184.5 (December 21, 1549) to 2688976.5 (January 25, 2650).<br><br>This block implements these ephemerides with respect to the International Celestial Reference Frame version 2.0, adopted in 2010.          |
| DE432t                 | Released in April 2014. This ephemeris takes into account the Julian date range 2287184.5, (December 21, 1549 ) to 2688976.5, (January 25, 2650).<br><br>This block implements these ephemerides with respect to the International Celestial Reference Frame version 2.0, adopted in 2010. |

**Note** This block requires that you download ephemeris data using the Add-On Explorer. To start the Add-On Explorer, in the MATLAB Command Window, type `aeroDataPackage`. on the MATLAB desktop toolstrip, click the **Add-Ons** button.

**Programmatic Use****Block Parameter:** de**Type:** character vector**Values:** DE405 | DE421 | DE423 | DE430**Default:** 'DE405'**Action for out-of-range input – Out-of-range block behavior**

None (default) | Warning | Error

Out-of-range block behavior, specified as follows.

| <b>Action</b> | <b>Description</b> |
|---------------|--------------------|
| None          | No action.         |

| Action          | Description   |
|-----------------|---|
| Warning         | Warning in the MATLAB Command Window, model simulation continues. |
| Error (default) | MATLAB returns an exception, model simulation stops.              |

**Programmatic Use****Block Parameter:** errorflag**Type:** character vector**Values:** 'None' | 'Warning' | 'Error'**Default:** 'Error'**Calculate rates — Calculate rate of Moon libration**

on (default) | off

Select to calculate the rate of the Moon libration.

**Dependencies**

Select this check box to display the  $\omega$  port.

**Programmatic Use****Block Parameter:** velflag**Type:** character vector**Values:** 'off' | 'on' |**Default:** 'on'**References**

[1] Folkner, W. M., J. G. Williams, D. H. Boggs. "The Planetary and Lunar Ephemeris DE 421." *IPN Progress Report 42-178*, 2009.

[2] Vallado, D. A. *Fundamentals of Astrodynamics and Applications*. New York: McGraw-Hill, 1997.

**Extended Capabilities****C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**See Also**

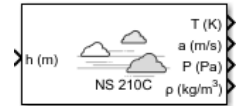
Earth Nutation | Planetary Ephemeris | aeroDataPackage

**Introduced in R2013a**

# Non-Standard Day 210C

Implement MIL-STD-210C climatic data

**Library:** Aerospace Blockset / Environment / Atmosphere



## Description

The **Non-Standard Day 210C** block implements a portion of the climatic data of the MIL-STD-210C worldwide air environment to 80 km (geometric or approximately 262,467 feet geometric) for absolute temperature, pressure, density, and speed of sound for the input geopotential altitude.

The COESA Atmosphere Model, Non-Standard Day 210C, and Non-Standard Day 310 blocks are identical blocks. When configured for COESA Atmosphere Model, the block implements the COESA mathematical representation. When configured for Non-Standard Day 210C, the block implements MIL-STD-210C climatic data. When configured for Non-Standard Day 310, the block implements MIL-HDBK-310 climatic data.

The COESA Atmosphere Model block port labels change based on the input and output units selected from the **Units** list.

## Limitations

All values are held below the geometric altitude of 0 m (0 feet) and above the geometric altitude of 80,000 meters (approximately 262,467 feet). The envelope atmospheric model has a few exceptions where values are held below the geometric altitude of 1 kilometer (approximately 3,281 feet) and above the geometric altitude of 30,000 meters (approximately 98,425 feet). These exceptions arise from lack of data in MIL-STD-210C for these conditions.

In general, temperature values are interpolated linearly, and density values are interpolated logarithmically. Pressure and speed of sound are calculated using a perfect gas law. The envelope atmospheric model has a few exceptions where the extreme value is the only value provided as an output. Pressure in these cases is interpolated logarithmically. These envelope atmospheric model exceptions apply to all cases of high and low pressure, high and low temperature, and high and low density, excluding the extreme values and 1% frequency of occurrence. These exceptions arise from lack of data in MIL-STD-210C for these conditions.

Another limitation is that climatic data for the region south of 60°S latitude is excluded from consideration in MIL-STD-210C.

This block uses the metric version of data from the MIL-STD-210C specifications. Certain data within the envelope are inconsistent between metric and English versions for low density, low temperature, high temperature, low pressure, and high pressure. The most significant differences occur in the following values:

- For low density envelope data with 5% frequency, the density values in metric units are inconsistent at 4 km and 18 km and the density values in English units are inconsistent at 14 km.
- For low density envelope data with 10% frequency,

- The density values in metric units are inconsistent at 18 km.
- The density values in English units are inconsistent at 14 km.
- For low density envelope data with 20% frequency, the density values in English units are inconsistent at 14 km.
- For low temperature envelope data with 20% frequency, the temperature values at 20 km are inconsistent.
- For high pressure envelope data with 10% frequency, the pressure values in metric units at 8 km are inconsistent.

## Ports

### Input

#### **Port\_1 — Geopotential height**

scalar | array

Geopotential height, specified as a scalar or array, in specified units.

Data Types: double

### Output

#### **Port\_2 — Temperature**

scalar | array

Temperature, specified as a scalar or array, in specified units.

Data Types: double

#### **Port\_2 — Speed of sound**

scalar | array

Speed of sound, specified as a scalar or array, in specified units.

Data Types: double

#### **Port\_3 — Air pressure**

scalar | array

Air pressure, specified as a scalar or array, in specified units.

Data Types: double

#### **Port\_4 — Air density**

scalar | array

Air density, specified as a scalar or array, in specified units.

Data Types: double

## Parameters

### **Units — Input and output units**

Metric (MKS) (default) | English (Velocity in ft/s) | English (Velocity in kts)

Input and output units, specified as:

| Units                      | Height | Temperature     | Speed of Sound    | Air Pressure                | Air Density               |
|----------------------------|--------|-----------------|-------------------|-----------------------------|---------------------------|
| Metric (MKS)               | Meters | Kelvin          | Meters per second | Pascal                      | Kilograms per cubic meter |
| English (Velocity in ft/s) | Feet   | Degrees Rankine | Feet per second   | Pound-force per square inch | Slug per cubic foot       |
| English (Velocity in kts)  | Feet   | Degrees Rankine | Knots             | Pound-force per square inch | Slug per cubic foot       |

#### Programmatic Use

**Block Parameter:** units

**Type:** character vector

**Values:** 'Metric (MKS)' | 'English (Velocity in ft/s)' | 'English (Velocity in kts)'

**Default:** 'Metric (MKS)'

#### Specification — Atmosphere model type

1976 COESA-extended U.S. Standard Atmosphere (default) | MIL-HDBK-310 | MIL-STD-210C

Atmosphere model type, specified as 1976 COESA-extended U.S. Standard Atmosphere, MIL-HDBK-310, or MIL-STD-210C. For the MIL-HDBK-310 and MIL-STD-210C options:

|              |   |
|--------------|---|
| MIL-HDBK-310 | This selection is linked to the Non-Standard Day 310 block. See the block reference for more information. Selecting MIL-HDBK-310 enables the parameters <b>Atmospheric model type</b> , <b>Extreme parameter</b> , <b>Frequency of occurrence</b> , and <b>Altitude of extreme value</b> .  |
| MIL-STD-210C | This selection is linked to the Non-Standard Day 210C block. See the block reference for more information. Selecting MIL-HDBK-310 enables the parameters <b>Atmospheric model type</b> , <b>Extreme parameter</b> , <b>Frequency of occurrence</b> , and <b>Altitude of extreme value</b> . |

#### Dependencies

Selecting MIL-HDBK-310 or MIL-STD-210C enables these parameters:

- **Atmospheric model type**
- **Extreme parameter**
- **Frequency of occurrence**
- **Altitude of extreme value**

#### Programmatic Use

**Block Parameter:** spec

**Type:** character vector

**Values:** '1976 COESA-extended U.S. Standard Atmosphere' | 'MIL-HDBK-310' | 'MIL-STD-210C'

**Default:** '1976 COESA-extended U.S. Standard Atmosphere'

### Atmospheric model type — Model type

Profile (default) | Envelope

Representation of atmospheric model type, specified as:

|          |  |
|----------|--|
| Profile  | Realistic atmospheric profiles associated with extremes at specified altitudes. Recommended for simulation of vehicles vertically traversing the atmosphere or when the total influence of the atmosphere is needed. |
| Envelope | Uses extreme atmospheric values at each altitude. Recommended for vehicles only horizontally traversing the atmosphere without much change in altitude.  |

#### Dependencies

- Selecting MIL-HDBK-310 or MIL-STD-210C for the **Specification** parameter enables this parameter.
- Selecting Profile enables the **Attitude of extreme value** parameter.

#### Programmatic Use

**Block Parameter:** model

**Type:** character vector

**Values:** 'Profile' | 'Envelope'

**Default:** 'Profile'

### Extreme parameter — Model type

High temperature (default) | Low temperature | High density | Low density | High pressure | Low pressure

Atmospheric parameter that is the extreme value.

#### Dependencies

- Selecting MIL-HDBK-310 or MIL-STD-210C for the **Specification** parameter enables this parameter.
- The High pressure and Low pressure options appear only when **Atmospheric model type** is set to Envelope.

#### Programmatic Use

**Block Parameter:** profile\_var

**Type:** character vector

**Values:** 'High temperature' | 'Low temperature' | 'High density' | 'Low density' | 'High pressure' | 'Low pressure'

**Default:** 'High temperature'

### Frequency of occurrence — Model type

1% (default) | Extreme values | 5% | 10% | 20%

Percent of time the values would occur.



**Dependencies**

- Selecting MIL-HDBK-310 or MIL-STD-210C for the **Specification** parameter enables this parameter.
- Extreme values, 5%, and 20% are available only when Envelope is selected for **Atmospheric model type**.
- 1% and 10% are always available.

**Programmatic Use****Block Parameter:** profile\_percent**Type:** character vector**Values:** 'Extreme values' | '1%' | '5%' | '10%' | '20%'**Default:** '1%'**Altitude of extreme value — Geometric altitude**

5 km (16404 ft) (default) | 10 km (32808 ft) | 20 km (65617 ft) | 30 km (98425 ft) | 40 km (131234 ft)

Geometric altitude at which the extreme values occur, specified as 5 km (16404 ft), 10 km (32808 ft), 20 km (65617 ft), 30 km (98425 ft), or 40 km (131234 ft).

**Dependencies**

This parameter appears if the **Atmospheric model type** is set to Profile.

**Programmatic Use****Block Parameter:** profile\_alt**Type:** character vector**Values:** 5 km (16404 ft) | 10 km (32808 ft) | 20 km (65617 ft) | 30 km (98425 ft) | 40 km (131234 ft)**Default:** 40 km (131234 ft)**Action for out-of-range input — Out-of-range block behavior**

Warning (default) | None | Error

Out-of-range block behavior, specified as follows.

| Action          | Description   |
|-----------------|---|
| None            | No action.  |
| Warning         | Warning in the MATLAB Command Window, model simulation continues. |
| Error (default) | MATLAB returns an exception, model simulation stops.              |

**Programmatic Use****Block Parameter:** action**Type:** character vector**Values:** 'None' | 'Warning' | 'Error'**Default:** 'Warning'**References**

- [1] *Global Climatic Data for Developing Military Products*. MIL-STD-210C, Washington, D.C.: Department of Defense, 1987.

## **Extended Capabilities**

### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

### **See Also**

CIRA-86 Atmosphere Model | COESA Atmosphere Model | ISA Atmosphere Model | Non-Standard Day  
310

**Introduced before R2006a**

# Non-Standard Day 310

Implement MIL-HDBK-310 climatic data

**Library:** Aerospace Blockset / Environment / Atmosphere



## Description

The Non-Standard Day 310 block implements a portion of the climatic data of the MIL-HDBK-310 worldwide air environment to 80 km (geometric or approximately 262,467 feet geometric) for absolute temperature, pressure, density, and speed of sound for the input geopotential altitude.

The COESA Atmosphere Model, Non-Standard Day 210C, and Non-Standard Day 310 blocks are identical blocks. When configured for COESA Atmosphere Model, the block implements the COESA mathematical representation. When configured for Non-Standard Day 210C, the block implements MIL-STD-210C climatic data. When configured for Non-Standard Day 310, the block implements MIL-HDBK-310 climatic data.

The COESA Atmosphere Model block port labels change based on the input and output units selected from the **Units** list.

## Limitations

All values are held below the geometric altitude of 0 m (0 feet) and above the geometric altitude of 80,000 meters (approximately 262,467 feet). The envelope atmospheric model has a few exceptions where values are held below the geometric altitude of 1 kilometer (approximately 3,281 feet) and above the geometric altitude of 30,000 meters (approximately 98,425 feet). These exceptions arise from lack of data in MIL-HDBK-310 for these conditions.

In general, temperature values are interpolated linearly, and density values are interpolated logarithmically. Pressure and speed of sound are calculated using a perfect gas law. The envelope atmospheric model has a few exceptions where the extreme value is the only value provided as an output. Pressure in these cases is interpolated logarithmically. These envelope atmospheric model exceptions apply to all cases of high and low pressure, high and low temperature, and high and low density, excluding the extreme values and 1% frequency of occurrence. These exceptions arise from lack of data in MIL-HDBK-310 for these conditions.

Another limitation is that climatic data for the region south of 60°S latitude is excluded from consideration in MIL-HDBK-310.

This block uses the metric version of data from the MIL-STD-310 specifications. Certain data within the envelope are inconsistent between metric and English versions for low density, low temperature, high temperature, low pressure, and high pressure. The most significant differences occur in the following values:

- For low density envelope data with 5% frequency, the density values in metric units are inconsistent at 4 km and 18 km and the density values in English units are inconsistent at 14 km.
- For low density envelope data with 10% frequency,

- The density values in metric units are inconsistent at 18 km.
- The density values in English units are inconsistent at 14 km.
- For low density envelope data with 20% frequency, the density values in English units are inconsistent at 14 km.
- For low temperature envelope data with 20% frequency, the temperature values at 20 km are inconsistent.
- For high pressure envelope data with 10% frequency, the pressure values in metric units at 8 km are inconsistent.

## Ports

### Input

#### **Port\_1 — Geopotential height**

scalar | array

Geopotential height, specified as a scalar or array, in specified units.

Data Types: double

### Output

#### **Port\_1 — Temperature**

scalar | array

Temperature, specified as a scalar or array, in specified units.

Data Types: double

#### **Port\_2 — Speed of sound**

scalar | array

Speed of sound, specified as a scalar or array, in specified units.

Data Types: double

#### **Port\_3 — Air pressure**

scalar | array

Air pressure, specified as a scalar or array, in specified units.

Data Types: double

#### **Port\_4 — Air density**

scalar | array

Air density, specified as a scalar or array, in specified units.

Data Types: double

## Parameters

### **Units — Input and output units**

Metric (MKS) (default) | English (Velocity in ft/s) | English (Velocity in kts)

Input and output units, specified as:

| Units                      | Height | Temperature     | Speed of Sound    | Air Pressure                | Air Density               |
|----------------------------|--------|-----------------|-------------------|-----------------------------|---------------------------|
| Metric (MKS)               | Meters | Kelvin          | Meters per second | Pascal                      | Kilograms per cubic meter |
| English (Velocity in ft/s) | Feet   | Degrees Rankine | Feet per second   | Pound-force per square inch | Slug per cubic foot       |
| English (Velocity in kts)  | Feet   | Degrees Rankine | Knots             | Pound-force per square inch | Slug per cubic foot       |

#### Programmatic Use

**Block Parameter:** units

**Type:** character vector

**Values:** 'Metric (MKS)' | 'English (Velocity in ft/s)' | 'English (Velocity in kts)'

**Default:** 'Metric (MKS)'

#### Specification — Atmosphere model type

1976 COESA-extended U.S. Standard Atmosphere (default) | MIL-HDBK-310 | MIL-STD-210C

Atmosphere model type, specified as 1976 COESA-extended U.S. Standard Atmosphere, MIL-HDBK-310, or MIL-STD-210C. For the MIL-HDBK-310 and MIL-STD-210C options:

|              |   |
|--------------|---|
| MIL-HDBK-310 | This selection is linked to the Non-Standard Day 310 block. See the block reference for more information. Selecting MIL-HDBK-310 enables the parameters <b>Atmospheric model type</b> , <b>Extreme parameter</b> , <b>Frequency of occurrence</b> , and <b>Altitude of extreme value</b> .  |
| MIL-STD-210C | This selection is linked to the Non-Standard Day 210C block. See the block reference for more information. Selecting MIL-HDBK-310 enables the parameters <b>Atmospheric model type</b> , <b>Extreme parameter</b> , <b>Frequency of occurrence</b> , and <b>Altitude of extreme value</b> . |

#### Dependencies

Selecting MIL-HDBK-310 or MIL-STD-210C enables these parameters:

- **Atmospheric model type**
- **Extreme parameter**
- **Frequency of occurrence**
- **Altitude of extreme value**

#### Programmatic Use

**Block Parameter:** spec

**Type:** character vector

**Values:** '1976 COESA-extended U.S. Standard Atmosphere' | 'MIL-HDBK-310' | 'MIL-STD-210C'

**Default:** '1976 COESA-extended U.S. Standard Atmosphere'

### Atmospheric model type — Model type

Profile (default) | Envelope

Representation of atmospheric model type, specified as:

|          |  |
|----------|--|
| Profile  | Realistic atmospheric profiles associated with extremes at specified altitudes. Recommended for simulation of vehicles vertically traversing the atmosphere or when the total influence of the atmosphere is needed. |
| Envelope | Uses extreme atmospheric values at each altitude. Recommended for vehicles only horizontally traversing the atmosphere without much change in altitude.  |

#### Dependencies

- Selecting MIL-HDBK-310 or MIL-STD-210C for the **Specification** parameter enables this parameter.
- Selecting Profile enables the **Attitude of extreme value** parameter.

#### Programmatic Use

**Block Parameter:** model

**Type:** character vector

**Values:** 'Profile' | 'Envelope'

**Default:** 'Profile'

### Extreme parameter — Model type

High temperature (default) | Low temperature | High density | Low density | High pressure | Low pressure

Atmospheric parameter that is the extreme value.

#### Dependencies

- Selecting MIL-HDBK-310 or MIL-STD-210C for the **Specification** parameter enables this parameter.
- The High pressure and Low pressure options appear only when **Atmospheric model type** is set to Envelope.

#### Programmatic Use

**Block Parameter:** profile\_var

**Type:** character vector

**Values:** 'High temperature' | 'Low temperature' | 'High density' | 'Low density' | 'High pressure' | 'Low pressure'

**Default:** 'High temperature'

### Frequency of occurrence — Model type

1% (default) | Extreme values | 5% | 10% | 20%

Percent of time the values would occur.

**Dependencies**

- Selecting MIL-HDBK-310 or MIL-STD-210C for the **Specification** parameter enables this parameter.
- Extreme values, 5%, and 20% are available only when Envelope is selected for **Atmospheric model type**.
- 1% and 10% are always available.

**Programmatic Use****Block Parameter:** profile\_percent**Type:** character vector**Values:** 'Extreme values' | '1%' | '5%' | '10%' | '20%'**Default:** '1%'**Altitude of extreme value — Geometric altitude**

5 km (16404 ft) (default) | 10 km (32808 ft) | 20 km (65617 ft) | 30 km (98425 ft) | 40 km (131234 ft)

Geometric altitude at which the extreme values occur, specified as 5 km (16404 ft), 10 km (32808 ft), 20 km (65617 ft), 30 km (98425 ft), or 40 km (131234 ft).

**Dependencies**

This parameter appears if the **Atmospheric model type** is set to Profile.

**Programmatic Use****Block Parameter:** profile\_alt**Type:** character vector**Values:** 5 km (16404 ft) | 10 km (32808 ft) | 20 km (65617 ft) | 30 km (98425 ft) | 40 km (131234 ft)**Default:** 40 km (131234 ft)**Action for out-of-range input — Out-of-range block behavior**

Warning (default) | None | Error

Out-of-range block behavior, specified as follows.

| Action          | Description   |
|-----------------|---|
| None            | No action.  |
| Warning         | Warning in the MATLAB Command Window, model simulation continues. |
| Error (default) | MATLAB returns an exception, model simulation stops.              |

**Programmatic Use****Block Parameter:** action**Type:** character vector**Values:** 'None' | 'Warning' | 'Error'**Default:** 'Warning'**References**

- [1] *Global Climatic Data for Developing Military Products*. MIL-HDBK-310, Washington, D.C.: Department of Defense, 1987.

## **Extended Capabilities**

### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

### **See Also**

CIRA-86 Atmosphere Model | COESA Atmosphere Model | ISA Atmosphere Model | Non-Standard Day 210C

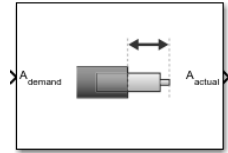
**Introduced before R2006a**



# Nonlinear Second-Order Actuator

Implement second-order actuator with rate and deflection limits

**Library:** Aerospace Blockset / Actuators



## Description

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

## Ports

### Input

**A<sub>demand</sub>** — Demanded actuator position

scalar | array

Demanded actuator position, specified as a scalar or array.

Data Types: double

### Output

**A<sub>actual</sub>** — Actual actuator position

scalar | array

Actual actuator position, returned as a scalar or array.

Data Types: double

## Parameters

**Natural frequency** — Natural frequency

1 (default) | scalar

Natural frequency of actuator, specified as a scalar double, in radians per second.

### Programmatic Use

**Block Parameter:** wn\_fin

**Type:** character vector

**Values:** scalar | double

**Default:** '1'

**Damping ratio** — Damping ratio

0.3 (default) | scalar

Damping ratio of actuator, specified as a scalar double.

**Programmatic Use****Block Parameter:** z\_fin**Type:** character vector**Values:** scalar | double**Default:** '0.3'**Maximum deflection — Largest actuator position allowable** $20 \cdot \pi / 180$  (default) | scalar

Largest actuator position allowable, specified as a scalar double, in the same units as demanded actuator position.

**Programmatic Use****Block Parameter:** fin\_max**Type:** character vector**Values:** scalar | double**Default:** ' $20 \cdot \pi / 180$ '**Minimum deflection — Smallest actuator position allowable** $-20 \cdot \pi / 180$  (default) | scalar

Smallest actuator position allowable, specified as a scalar double, in the same units as demanded actuator position.

**Programmatic Use****Block Parameter:** fin\_min**Type:** character vector**Values:** scalar | double**Default:** ' $-20 \cdot \pi / 180$ '**Rate limit — Fastest speed allowable** $500 \cdot \pi / 180$  (default) | scalar

Fastest speed allowable for actuator motion, specified as a scalar double, in the units of demanded actuator position per second.

**Programmatic Use****Block Parameter:** fin\_maxrate**Type:** character vector**Values:** scalar | double**Default:** ' $500 \cdot \pi / 180$ '**Initial position — Initial position**

0 (default) | scalar

Initial position of actuator, specified as a scalar double, in the same units as demanded actuator position.

- If the specified value is less than the value of **Minimum deflection**, the block sets the value of **Minimum deflection** as the initial position value.
- If the specified value is greater than the value of **Maximum deflection**, the block sets the value of **Maximum deflection** as the initial position value.

**Programmatic Use****Block Parameter:** fin\_act\_0

**Type:** character vector

**Values:** scalar | double

**Default:** '0'

### **Initial velocity – Initial velocity**

0 (default) | scalar

Initial velocity of actuator, specified as a scalar double, in the units of demanded actuator position per second.

If the absolute value of the specified value is greater than the absolute value of **Rate Limit**, this block sets the value of **Rate Limit** as the initial velocity value.

#### **Programmatic Use**

**Block Parameter:** fin\_act\_vel

**Type:** character vector

**Values:** scalar | double

**Default:** '0'

## **Extended Capabilities**

### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

## **See Also**

Linear Second-Order Actuator

### **Topics**

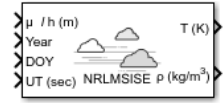
“Explore the NASA HL-20 Model” on page 1-5

**Introduced in R2012a**

## NRLMSISE-00 Atmosphere Model

Implement mathematical representation of 2001 United States Naval Research Laboratory Mass Spectrometer and Incoherent Scatter Radar Exosphere

**Library:** Aerospace Blockset / Environment / Atmosphere



### Description

The NRLMSISE-00 Atmosphere Model block implements the mathematical representation of the 2001 United States Naval Research Laboratory Mass Spectrometer and Incoherent Scatter Radar Exosphere (NRLMSISE-00) of the MSIS<sup>®</sup> class model. This block calculates the neutral atmosphere empirical model from the surface to lower exosphere (0 to 1,000,000 meters). When configuring the block for this calculation, you can also take into account the anomalous oxygen, which can affect the satellite drag above 500,000 meters.

### Limitations

- This block has the limitations of the NRLMSISE-00 model. For more information, see <https://ccmc.gsfc.nasa.gov/modelweb/atmos/nrlmsise00.html>.
- This block is valid only for altitudes between 0 and 1,000,000 meters (1,000 kilometers).
- The F107 and F107A values used to generate the model correspond to the 10.7 cm radio flux at the actual distance of the Earth from the Sun rather than the radio flux at 1 AU. These sites provide both classes of values:
  - [ftp://ftp.ngdc.noaa.gov/STP/GEOMAGNETIC\\_DATA/INDICES/KP\\_AP/](ftp://ftp.ngdc.noaa.gov/STP/GEOMAGNETIC_DATA/INDICES/KP_AP/)
  - <ftp://ftp.ngdc.noaa.gov/STP/space-weather/solar-data/solar-features/solar-radio/noontime-flux/penticton/>

The format for the data indices for these values are located here:

[ftp://ftp.ngdc.noaa.gov/STP/GEOMAGNETIC\\_DATA/INDICES/KP\\_AP/kp\\_ap.fmt](ftp://ftp.ngdc.noaa.gov/STP/GEOMAGNETIC_DATA/INDICES/KP_AP/kp_ap.fmt)

### Ports

#### Input

##### Port\_1 — Geodetic latitudes, longitude, and altitude

three-element matrix

Geodetic latitudes, in degrees, longitude, in degrees, and altitude, in selected length units, specified as three-element matrix.

Data Types: double

##### Port\_2 — Years

array

$N$  years, specified as an array.

Data Types: double

### **Port\_3 – Days**

array

$N$  days of a year (1 to 365 (or 366)), specified as an array.

Data Types: double

### **Port\_4 – Seconds**

array

$N$  seconds in a day, specified as an array, in universal time (UT).

Data Types: double

### **Port\_5 – Local apparent solar time**

array

$N$  local apparent solar time, specified as an array, in hours.

Data Types: double

### **Port\_6 – 81-day average F10.7 flux**

array

$N$  81-day averages of F10.7 flux, centered on day of year (doy), specified as an array.

Data Types: double

### **Port\_7 – Daily average F10.7 flux**

array

$N$  daily F10.7 fluxes for previous days, specified as an array.

Data Types: double

### **Port\_8 – Magnetic index information**

$N$ -by-7 array

Magnetic index information, specified as an  $N$ -by-7. If you specify *magneticIndex*, you must also specify *f107Average* and *f107Daily*. The magnetic index information consists of:

Daily magnetic index (AP)

3 hour AP for current time

3 hour AP for 3 hours before current time

3 hour AP for 6 hours before current time

3 hour AP for 9 hours before current time

Average of eight 3 hour AP indices from 12 to 33 hours before current time

Average of eight 3 hour AP indices from 36 to 57 hours before current time

The effects of daily magnetic index are not large or established below 80,000 m. As a result, the block sets the default value to 4. See the limitations in “Limitations” on page 4-446 for more information.

Data Types: double

### **Port\_9 – Flags**

array of 23

Flags, returned as an array of 21, to enable or disable particular variations for the outputs.

| Field      | Description  |
|------------|--|
| Flags (1)  | F10.7 effect on mean   |
| Flags (2)  | Independent of time  |
| Flags (3)  | Symmetrical annual   |
| Flags (4)  | Symmetrical semiannual   |
| Flags (5)  | Asymmetrical annual  |
| Flags (6)  | Asymmetrical semiannual  |
| Flags (7)  | Diurnal  |
| Flags (8)  | Semidiurnal  |
| Flags (9)  | Daily AP. If you set this field to -1, the block uses the entire matrix of magnetic index information (APH) instead of APH (: , 1) |
| Flags (10) | All UT, longitudinal effects   |
| Flags (11) | Longitudinal   |
| Flags (12) | UT and mixed UT, longitudinal  |
| Flags (13) | Mixed AP, UT, longitudinal   |
| Flags (14) | Terdiurnal   |
| Flags (15) | Departures from diffusive equilibrium  |
| Flags (16) | All exospheric temperature variations  |
| Flags (17) | All variations from 120,000 meter temperature (TLB)  |
| Flags (18) | All lower thermosphere (TN1) temperature variations  |
| Flags (19) | All 120,000 meter gradient (S) variations  |
| Flags (20) | All upper stratosphere (TN2) temperature variations  |
| Flags (21) | All variations from 120,000 meter values (ZLB)   |
| Flags (22) | All lower mesosphere temperature (TN3) variations  |
| Flags (23) | Turbopause scale height variations   |

Data Types: double

### Output

#### Port\_1 – Temperature

*N*-by-2 array

Temperature values, returned in a *N*-by-2 array of values, in selected temperature units. The first column contains exospheric temperatures, the second column contains temperature at altitude.

Data Types: double

#### Port\_2 – Densities

*N*-by-9 array

Density values, returned in a *N*-by-9 array, in selected density units.

| Density    | Description   |
|------------|---|
| Density(1) | Density of He   |
| Density(2) | Density of O  |
| Density(3) | Density of N2   |
| Density(4) | Density of O2   |
| Density(5) | Density of Ar   |
| Density(6) | Total mass density<br><br>Density(6), total mass density, is defined as the sum of the mass densities of He, O, N2, O2, Ar, H, and N. Optionally, Density(6) can include the mass density of anomalous oxygen making Density(6), the effective total mass density for drag. |
| Density(7) | Density of H  |
| Density(8) | Density of N  |
| Density(9) | Anomalous oxygen number density   |

Data Types: double

## Parameters

### Units – Input and output units

Metric (MKS) (default) | English

Input and output units, specified as:

| Units        | Temperature | Height | Density  |
|--------------|-------------|--------|--|
| Metric (MKS) | Kelvin      | Meters | kg/m <sup>3</sup> , some density outputs 1/m <sup>3</sup>    |
| English      | Rankine     | Feet   | lbm/ft <sup>3</sup> , some density outputs 1/ft <sup>3</sup> |

### Programmatic Use

**Block Parameter:** units

**Type:** character vector

**Values:** 'Metric (MKS)' | 'English'

**Default:** 'Metric (MKS)'

### Input local apparent solar time – Apparent solar time

off (default) | on

Select this check box to input the local apparent solar time, in hours. Otherwise, the block inputs the default value.

### Programmatic Use

**Block Parameter:** 1st input

**Type:** character vector

**Values:** 'off' | 'on'

**Default:** 'off'

**Input flux and magnetic index information — Daily F10.7 flux for previous day and magnetic index information**

off (default) | on

Select this check box to input the 81-day average of F10.7, the daily F10.7 flux for the previous day, and the array of 7 magnetic index information (see the `aph` argument in the `atmosnrmsise00` function). Otherwise, the block inputs the default value.

**Programmatic Use****Block Parameter:** flux\_ap\_input**Type:** character vector**Values:** 'off' | 'on'**Default:** 'off'**Source for flags — Variation flag source**

Internal (default) | External

Variation flag source, specified as `Internal` or `External`. If you specify `External`, specify the variation flag as an array of 23. If you specify `Internal`, the flag source is internal to the block.

**Dependencies**

Setting **Source for flags** to `Internal` enables the **Flags** parameter.

**Programmatic Use****Block Parameter:** flags\_input**Type:** character vector**Values:** 'Internal' | 'External'**Default:** 'Internal'**Flags — Variation flags**

ones(1,23) (default)

Variation flag, specified as an array of 23 (`ones(1,23)`). This parameter applies only when **Source for flags** has a value of `Internal`. You can specify one of the following values for a field. The default value for each field is 1.

- 0.0  
Removes that value's effect on the output.
- 1.0  
Applies the main and the cross-term effects of that value on the output.
- 2.0  
Applies only the cross-term effect of that value on the output.

The array has the following fields.

| Field    | Description          |
|----------|----------------------|
| Flags(1) | F10.7 effect on mean |
| Flags(2) | Independent of time  |
| Flags(3) | Symmetrical annual   |



| Field      | Description  |
|------------|--|
| Flags (4)  | Symmetrical semiannual   |
| Flags (5)  | Asymmetrical annual  |
| Flags (6)  | Asymmetrical semiannual  |
| Flags (7)  | Diurnal  |
| Flags (8)  | Semidiurnal  |
| Flags (9)  | Daily AP. If you set this field to -1, the block uses the entire matrix of magnetic index information (APH) instead of APH( : , 1) |
| Flags (10) | All UT, longitudinal effects   |
| Flags (11) | Longitudinal   |
| Flags (12) | UT and mixed UT, longitudinal  |
| Flags (13) | Mixed AP, UT, longitudinal   |
| Flags (14) | Terdiurnal   |
| Flags (15) | Departures from diffusive equilibrium  |
| Flags (16) | All exospheric temperature variations  |
| Flags (17) | All variations from 120,000 meter temperature (TLB)  |
| Flags (18) | All lower thermosphere (TN1) temperature variations  |
| Flags (19) | All 120,000 meter gradient (S) variations  |
| Flags (20) | All upper stratosphere (TN2) temperature variations  |
| Flags (21) | All variations from 120,000 meter values (ZLB)   |
| Flags (22) | All lower mesosphere temperature (TN3) variations  |
| Flags (23) | Turbopause scale height variations   |

### Dependencies

Setting **Source for flags** to Internal enables the **Flags** parameter.

### Programmatic Use

**Block Parameter:** flags

**Type:** character vector

**Values:** 'ones(1,23)'

**Default:** 'ones(1,23)'

### Include anomalous oxygen number density in total mass density – Anomalous oxygen

off (default) | on

Select this check box to take into account the anomalous oxygen when calculating the neutral atmosphere empirical model from the surface to lower exosphere (0 to 1,000,000 meters). Taking into account this number can affect the satellite drag above 500,000 meters.

### Programmatic Use

**Block Parameter:** oxygen\_in

**Type:** character vector

**Values:** 'off' | 'on'

**Default:** 'off'

**Action for out-of-range input – Out-of-range block behavior**

Warning (default) | None | Error

Out-of-range block behavior, specified as follows.

| Action          | Description   |
|-----------------|---|
| None            | No action.  |
| Warning         | Warning in the MATLAB Command Window, model simulation continues. |
| Error (default) | MATLAB returns an exception, model simulation stops.              |

**Programmatic Use****Block Parameter:** action**Type:** character vector**Values:** 'None' | 'Warning' | 'Error'**Default:** 'Warning'**Extended Capabilities****C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**See Also**

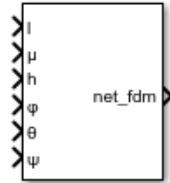
CIRA-86 Atmosphere Model | COESA Atmosphere Model | ISA Atmosphere Model

**External Websites**<https://ccmc.gsfc.nasa.gov/modelweb/atmos/nrlmsise00.html>[ftp://ftp.ngdc.noaa.gov/STP/GEOMAGNETIC\\_DATA/INDICES/KP\\_AP/kp\\_ap.fmt](ftp://ftp.ngdc.noaa.gov/STP/GEOMAGNETIC_DATA/INDICES/KP_AP/kp_ap.fmt)<ftp://ftp.ngdc.noaa.gov/STP/space-weather/solar-data/solar-features/solar-radio/noontime-flux/penticton/>[ftp://ftp.ngdc.noaa.gov/STP/GEOMAGNETIC\\_DATA/INDICES/KP\\_AP/](ftp://ftp.ngdc.noaa.gov/STP/GEOMAGNETIC_DATA/INDICES/KP_AP/)**Introduced in R2007b**

# Pack net\_fdm Packet for FlightGear

Generate net\_fdm packet for FlightGear

**Library:** Aerospace Blockset / Animation / Flight Simulator Interfaces



## Description

The Pack net\_fdm Packet for FlightGear block creates, from separate inputs, a FlightGear net\_fdm data packet compatible with a particular version of FlightGear flight simulator. This block accepts all signals supported by the FlightGear net\_fdm data packet. These signals are arranged into six groups:

- Position/attitude inputs
- Velocity/acceleration inputs
- Control surface position inputs
- Engine/fuel inputs
- Landing gear inputs
- Environment inputs

To enable or disable the inputs for these groups, select the associated block parameter. The block input ports change depending on the requested signal groups. The block inserts zeros for packet values that are part of inactive signal groups.

## Ports

### Input

#### Position/Attitude Inputs

##### λ — Longitude

scalar

Longitude, specified as a scalar, in rad.

#### Dependencies

To enable this port, select the **Show position/attitude inputs** check box.

Data Types: double

##### μ — Latitude

scalar

Latitude, specified as a scalar, in rad.

**Dependencies**

To enable this port, select the **Show position/attitude inputs** check box.

Data Types: double

**h — Altitude**

scalar

Altitude, specified as a scalar, in m.

**Dependencies**

To enable this port, select the **Show position/attitude inputs** check box.

Data Types: double

 **$\phi$  — Roll**

scalar

Roll, specified as a scalar, in rad.

**Dependencies**

To enable this port, select the **Show position/attitude inputs** check box.

Data Types: single

 **$\theta$  — Pitch**

scalar

Pitch, specified as a scalar, in rad.

**Dependencies**

To enable this port, select the **Show position/attitude inputs** check box.

Data Types: single

 **$\psi$  — Yaw**

scalar

Yaw, specified as a scalar, in rad.

**Dependencies**

To enable this port, select the **Show position/attitude inputs** check box.

Data Types: single

**Velocity/Acceleration Inputs** **$\alpha$  — Angle of attack**

scalar

Angle of attack, specified as a scalar, in rad.

**Dependencies**

To enable this port, select the **Show velocity/acceleration inputs** check box.

Data Types: single

 **$\beta$  – Sideslip angle**

scalar

Sideslip angle, specified as a scalar, in rad.

**Dependencies**

To enable this port, select the **Show velocity/acceleration inputs** check box.

Data Types: double

 **$d\phi/dt$  – Roll rate**

scalar

Roll rate, specified as a scalar, in rad/sec.

**Dependencies**

To enable this port, select the **Show velocity/acceleration inputs** check box.

Data Types: double

 **$d\theta/dt$  – Pitch rate**

scalar

Pitch rate, specified as a scalar, in rad/sec.

**Dependencies**

To enable this port, select the **Show velocity/acceleration inputs** check box.

Data Types: single

 **$d\psi/dt$  – Yaw rate**

scalar

Yaw rate, specified as a scalar, in rad/sec.

**Dependencies**

To enable this port, select the **Show velocity/acceleration inputs** check box.

Data Types: single

 **$V_{cas}$  – Calibrated airspeed**

scalar

Calibrated airspeed, specified as a scalar, in knots.

**Dependencies**

To enable this port, select the **Show velocity/acceleration inputs** check box.

Data Types: single

**climb\_rate – Rate of climb**

scalar

Rate of climb, specified as a scalar, in feet/sec.

**Dependencies**

To enable this port, select the **Show velocity/acceleration inputs** check box.

Data Types: `single`

**$v_{north}$  — North velocity in body frame**

scalar

North velocity in body frame, specified as a scalar, in ft/sec.

**Dependencies**

To enable this port, select the **Show velocity/acceleration inputs** check box.

Data Types: `single`

**$v_{east}$  — East velocity in body frame**

scalar

East velocity in body frame, specified as a scalar, in feet/sec.

**Dependencies**

To enable this port, select the **Show velocity/acceleration inputs** check box.

Data Types: `single`

**$v_{down}$  — Down velocity**

scalar

Down velocity, specified as a scalar, in feet/sec.

**Dependencies**

To enable this port, select the **Show velocity/acceleration inputs** check box.

Data Types: `single`

**$v_{wind\ body\ north}$  — North velocity in body frame relative to local airmass**

scalar

North velocity in body frame relative to local airmass, specified as a scalar, in ft/sec.

**Dependencies**

To enable this port, select the **Show velocity/acceleration inputs** check box.

Data Types: `single`

**$v_{wind\ body\ east}$  — East velocity in body frame relative to local airmass**

scalar

East velocity in body frame relative to local airmass, specified as a scalar, in ft/sec.

**Dependencies**

To enable this port, select the **Show velocity/acceleration inputs** check box.

Data Types: single

**$V_{\text{wind body down}}$  — Down velocity in body frame relative to airmass**  
scalar

Down velocity in body frame relative to airmass, specified as a scalar, in ft/sec.

**Dependencies**

To enable this port, select the **Show velocity/acceleration inputs** check box.

Data Types: single

**$A_{X \text{ pilot}}$  — X acceleration in body frame**  
scalar

X acceleration in body frame, specified as a scalar, in ft/sec<sup>2</sup>.

**Dependencies**

To enable this port, select the **Show velocity/acceleration inputs** check box.

Data Types: single

**$A_{Y \text{ pilot}}$  — Y acceleration in body frame**  
scalar

Y acceleration in body frame, specified as a scalar, in ft/sec<sup>2</sup>.

**Dependencies**

To enable this port, select the **Show velocity/acceleration inputs** check box.

Data Types: single

**$A_{Z \text{ pilot}}$  — Z acceleration in body frame**  
scalar

Z acceleration in body frame, specified as a scalar, in ft/sec<sup>2</sup>.

**Dependencies**

To enable this port, select the **Show velocity/acceleration inputs** check box.

Data Types: single

**stall\_warning — Amount of stall**  
scalar

Amount of stall [0-1], specified as a scalar, in ftsec<sup>2</sup>.

**Dependencies**

To enable this port, select the **Show velocity/acceleration inputs** check box.

Data Types: single

**slip\_degree — Slip ball deflection**  
scalar

Slip ball deflection. specified as a scalar, in deg.

**Dependencies**

To enable this port, select the **Show velocity/acceleration inputs** check box.

Data Types: `single`

**Control Surface Position Inputs****elevator — Normalized elevator position**

scalar

Normalized elevator position, specified as a scalar.

**Dependencies**

To enable this port, select the **Show control surface position inputs** check box.

Data Types: `double`

**elevator\_trim\_tab — Normalized elevator trim tab position**

scalar

Normalized elevator trim tab position, specified as a scalar.

**Dependencies**

To enable this port, select the **Show control surface position inputs** check box.

Data Types: `double`

**left\_flap — Normalized left flap position**

scalar

Normalized left flap position, specified as a scalar.

**Dependencies**

To enable this port, select the **Show control surface position inputs** check box.

Data Types: `double`

**right\_flap — Normalized right flap position**

scalar

Normalized right flap position, specified as a scalar.

**Dependencies**

To enable this port, select the **Show control surface position inputs** check box.

Data Types: `single`

**left\_aileron — Normalized left aileron position**

scalar

Normalized left aileron position. specified as a scalar.



**Dependencies**

To enable this port, select the **Show control surface position inputs** check box.

Data Types: `single`

**right\_aileron — Normalized right aileron position**

scalar

Normalized right aileron position, specified as a scalar.

**Dependencies**

To enable this port, select the **Show control surface position inputs** check box.

Data Types: `single`

**rudder — Normalized rudder position**

scalar

Normalized rudder position, specified as a scalar.

**Dependencies**

To enable this port, select the **Show control surface position inputs** check box.

Data Types: `single`

**nose\_wheel — Normalized nose wheel position**

scalar

Normalized nose wheel position, specified as a scalar.

**Dependencies**

To enable this port, select the **Show control surface position inputs** check box.

Data Types: `single`

**speedbrake — Normalized speedbrake position**

scalar

Normalized speedbrake position, specified as a scalar.

**Dependencies**

To enable this port, select the **Show control surface position inputs** check box.

Data Types: `single`

**spoilers — Normalized spoilers position**

scalar

Normalized spoilers position, specified as a scalar.

**Dependencies**

To enable this port, select the **Show control surface position inputs** check box.

Data Types: `single`

**Engine/Fuel Inputs****num\_engines — Number of engines**

scalar

Number of engines, specified as a scalar.

**Dependencies**To enable this port, select the **Show engine/fuel inputs** check box.

Data Types: uint32

**eng\_state — Engine state**

vector

Engine state (off, cranking, running), specified as a vector.

**Dependencies**To enable this port, select the **Show engine/fuel inputs** check box.

Data Types: uint32

**rpm — Engine RPM**

vector

Engine RPM, specified as a vector, in rev/min.

**Dependencies**To enable this port, select the **Show engine/fuel inputs** check box.

Data Types: double

**fuel\_flow — Fuel flow**

vector

Fuel flow, specified as a vector, in gal/hr.

**Dependencies**To enable this port, select the **Show engine/fuel inputs** check box.

Data Types: single

**fuel\_px — Fuel pressure**

vector

Fuel pressure, specified as a vector, in gal/hour.

**Dependencies**To enable this port, select the **Show engine/fuel inputs** check box.

Data Types: single

**egt — Exhaust gas temperature**

vector

Exhaust gas temperature, specified as a vector, in deg F.

**Dependencies**

To enable this port, select the **Show engine/fuel inputs** check box.

Data Types: `single`

**cht — Cylinder head temperature**

scalar

Cylinder head temperature, specified as a vector, in deg F.

**Dependencies**

To enable this port, select the **Show engine/fuel inputs** check box.

Data Types: `single`

**mp\_osi — Manifold pressure**

vector

Manifold pressure, specified as a vector, in psi.

**Dependencies**

To enable this port, select the **Show engine/fuel inputs** check box.

Data Types: `single`

**tit — Turbine inlet temperature**

vector

Turbine inlet temperature, specified as a vector, in deg F.

**Dependencies**

To enable this port, select the **Show engine/fuel inputs** check box.

Data Types: `single`

**oil\_temp — Oil temperature**

vector

Oil temperature, specified as a vector, in deg F.

**Dependencies**

To enable this port, select the **Show engine/fuel inputs** check box.

Data Types: `single`

**oil\_px — Oil pressure**

vector

Oil pressure, specified as a vector, in psi.

**Dependencies**

To enable this port, select the **Show engine/fuel inputs** check box.

Data Types: `single`

**num\_tanks — Number of fuel tanks**

scalar

Number of fuel tanks, specified as a scalar.

**Dependencies**To enable this port, select the **Show engine/fuel inputs** check box.Data Types: `uint32`**fuel\_quantity — Fuel quantity per tank**

vector

Fuel quantity per tank, specified as a vector, in gal.

**Dependencies**To enable this port, select the **Show engine/fuel inputs** check box.Data Types: `single`**Landing Gear Inputs****num\_wheels — Number of wheels**

uint32

Number of wheels, specified as a scalar.

**Dependencies**To enable this port, select the **Show landing gear inputs** check box.Data Types: `uint32`**wow — Weight on wheels switch**

vector

Weight on wheels switch, specified as a vector.

**Dependencies**To enable this port, select the **Show landing gear inputs** check box.Data Types: `uint32`**gear\_pos — Landing gear normalized position**

vector

Landing gear normalized position, specified as a vector.

**Dependencies**To enable this port, select the **Show landing gear inputs** check box.Data Types: `single`**gear\_steer — Landing gear normalized steering**

vector

Landing gear normalized steering, specified as a vector.

**Dependencies**

To enable this port, select the **Show landing gear inputs** check box.

Data Types: `single`

**gear\_compression — Landing gear normalized compression**  
vector

Landing gear normalized compression, specified as a vector.

**Dependencies**

To enable this port, select the **Show landing gear inputs** check box.

Data Types: `single`

**Environment Inputs****agl — Above ground level**  
scalar

Above ground level, specified as a scalar, in m.

**Dependencies**

To enable this port, select the **Show environment inputs** check box.

Data Types: `single`

**cur\_time — Current UNIX® time**  
scalar

Current UNIX time, specified as a scalar, in sec.

**Dependencies**

To enable this port, select the **Show environment inputs** check box.

Data Types: `uint32`

**warp — Offset in seconds to UNIX time**  
scalar

Offset in seconds to UNIX time, specified as a scalar, in sec.

**Dependencies**

To enable this port, select the **Show environment inputs** check box.

Data Types: `double`

**visibility — Visibility**  
scalar

Visibility (for visual effects), specified as a scalar, in m.

**Dependencies**

To enable this port, select the **Show environment inputs** check box.

Data Types: single

## Output

### net\_fdm — Packet generated for FlightGear

array

Packet generated for FlightGear, specified as an array.

Data Types: single | double | uint32

## Parameters

### FlightGear version — FlightGear version

v2019.1 (default) | v2018.3 | v2018.2 | v2018.1 | v2017.3 | v2017.1 | v2016.3 | v2016.1 | v3.4 | v3.2 | v3.0 | v2.12 | v2.10 | v2.8 | v2.6 | v2.4 | v2.0

FlightGear software version, selected from the list.

---

**Note** If you are using a FlightGear version older than 2.0, the model displays a notification from the Simulink Upgrade Advisor. Consider using the Upgrade Advisor to upgrade your FlightGear version. For more information, see “Supported FlightGear Versions” on page 2-16.

---

### Programmatic Use

**Block Parameter:** FlightGearVersion

**Type:** character vector

**Values:** v2019.1 | v2018.3 | v2018.2 | v2018.1 | v2017.3 | v2017.1 | v2016.3 | v2016.1 | v3.4 | v3.2 | v3.0 | v2.12 | v2.10 | v2.8 | v2.6 | v2.4 | v2.0

**Default:** 'v2019.1'

### Show position/altitude inputs — Position and altitude inputs

on (default) | off

Select this check box to include the position and altitude inputs in the FlightGear net\_fdm data packet.

### Dependencies

Select this check box to enable these input ports.

## Signal Group 1: Position/Altitude Inputs

| Name             | Units | Type   | Width | Description              |
|------------------|-------|--------|-------|--------------------------|
| <i>longitude</i> | rad   | double | 1     | Geodetic longitude       |
| <i>latitude</i>  | rad   | double | 1     | Geodetic latitude        |
| <i>altitude</i>  | m     | double | 1     | Altitude above sea level |
| <i>theta</i>     | rad   | single | 1     | Pitch                    |
| <i>phi</i>       | rad   | single | 1     | Roll                     |
| <i>psi</i>       | rad   | single | 1     | Yaw                      |

**Programmatic Use****Block Parameter:** ShowPositionAttitudeInputs**Type:** character vector**Values:** 'off' | 'on'**Default:** 'on'**Show velocity/acceleration inputs – Velocity and acceleration inputs**

off (default) | on

Select this check box to include the velocity and acceleration inputs in the FlightGear net\_fdm data packet.

**Dependencies**

Select this check box to enable these input ports.

**Signal Group 2: Velocity/Acceleration Inputs**

| Name                     | Units                 | Type   | Width | Description  |
|--------------------------|-----------------------|--------|-------|--|
| <i>alpha</i>             | rad                   | single | 1     | Angle of attack  |
| <i>beta</i>              | rad                   | single | 1     | Sideslip angle   |
| <i>dphi/dt</i>           | rad/sec               | single | 1     | Roll rate  |
| <i>dtheta/dt</i>         | rad/sec               | single | 1     | Pitch rate   |
| <i>dpsi/dt</i>           | rad/sec               | single | 1     | Yaw rate   |
| <i>Vcas</i>              | knot                  | single | 1     | Calibrated airspeed                                    |
| <i>climb_rate</i>        | feet/sec              | single | 1     | Rate of climb  |
| <i>v_north</i>           | feet/sec              | single | 1     | North velocity in body frame                           |
| <i>v_east</i>            | feet/sec              | single | 1     | East velocity in body frame                            |
| <i>v_down</i>            | feet/sec              | single | 1     | Down velocity  |
| <i>v_wind_body_north</i> | feet/sec              | single | 1     | North velocity in body frame relative to local airmass |
| <i>v_wind_body_east</i>  | feet/sec              | single | 1     | East velocity in body frame relative to local airmass  |
| <i>v_wind_body_down</i>  | feet/sec              | single | 1     | Down velocity in body frame relative to airmass        |
| <i>Axpilot</i>           | feet/sec <sup>2</sup> | single | 1     | X acceleration in body frame                           |
| <i>Aypilot</i>           | feet/sec <sup>2</sup> | single | 1     | Y acceleration in body frame                           |
| <i>Azpilot</i>           | feet/sec <sup>2</sup> | single | 1     | Z acceleration in body frame                           |
| <i>stall_warning</i>     | —                     | single | 1     | Amount of stall [0-1]                                  |
| <i>slip_deg</i>          | deb                   | single | 1     | Slip ball deflection                                   |

**Programmatic Use****Block Parameter:** ShowVelocityAccelerationInputs**Type:** character vector**Values:** 'off' | 'on'**Default:** 'off'**Show control surface position inputs – Control surface position inputs**

off (default) | on

Select this check box to include the control surface position inputs in the FlightGear net\_fdm data packet.

**Dependencies**

Select this check box to enable these input ports.

**Signal Group 3: Control Surface Position Inputs**

| Name                     | Units             | Type   | Width | Description                           |
|--------------------------|-------------------|--------|-------|---------------------------------------|
| <i>elevator</i>          | 1 (dimensionless) | single | 1     | Normalized elevator position          |
| <i>elevator_trim_tab</i> | 1 (dimensionless) | single | 1     | Normalized elevator trim tab position |
| <i>left_flap</i>         | 1 (dimensionless) | single | 1     | Normalized left flap position         |
| <i>right_flap</i>        | 1 (dimensionless) | single | 1     | Normalized right flap position        |
| <i>left_aileron</i>      | 1 (dimensionless) | single | 1     | Normalized left aileron position      |
| <i>right_aileron</i>     | 1 (dimensionless) | single | 1     | Normalized right aileron position     |
| <i>rudder</i>            | 1 (dimensionless) | single | 1     | Normalized rudder position            |
| <i>nose_wheel</i>        | 1 (dimensionless) | single | 1     | Normalized nose wheel position        |
| <i>speedbrake</i>        | 1 (dimensionless) | single | 1     | Normalized speedbrake position        |
| <i>spoilers</i>          | 1 (dimensionless) | single | 1     | Normalized spoilers position          |

**Programmatic Use****Block Parameter:** ShowControlSurfacePositionInputs**Type:** character vector**Values:** 'off' | 'on'**Default:** 'off'**Show engine/fuel inputs – Engine and fuel inputs**

off (default) | on

Select this check box to include the engine and fuel inputs in the FlightGear net\_fdm data packet.



**Dependencies**

Select this check box to enable these input ports.

**Signal Group 4: Engine/Fuel Inputs**

| Name                 | Units    | Type   | Width | Description                           |
|----------------------|----------|--------|-------|---------------------------------------|
| <i>num_engines</i>   | —        | uint32 | 1     | Number of engines                     |
| <i>eng_state</i>     | —        | uint32 | 4     | Engine state (off, cranking, running) |
| <i>rpm</i>           | rev/min  | single | 4     | Engine RPM                            |
| <i>fuel_flow</i>     | gal/hour | single | 4     | Fuel flow                             |
| <i>fuel_px</i>       | psi      | single | 4     | Fuel pressure                         |
| <i>egt</i>           | deg F    | single | 4     | Exhaust gas temperature               |
| <i>cht</i>           | deg F    | single | 4     | Cylinder head temperature             |
| <i>mp_osi</i>        | psi      | single | 4     | Manifold pressure                     |
| <i>tit</i>           | deg F    | single | 4     | Turbine inlet temperature             |
| <i>oil_temp</i>      | deg F    | single | 4     | Oil temperature                       |
| <i>oil_px</i>        | psi      | single | 4     | Oil pressure                          |
| <i>num_tanks</i>     | —        | uint32 | 1     | Number of fuel tanks                  |
| <i>fuel_quantity</i> | gal      | single | 4     | Fuel quantity per tank                |

**Programmatic Use**

**Block Parameter:** ShowEngineFuelInputs

**Type:** character vector

**Values:** 'off' | 'on'

**Default:** 'off'

**Show landing gear inputs — Landing gear inputs**

off (default) | on

Select this check box to include the landing gear inputs in the FlightGear net\_fdm data packet.

**Dependencies**

Select this check box to enable these input ports.

**Signal Group 5: Landing Gear Inputs**

| Name                    | Units | Type   | Width | Description                         |
|-------------------------|-------|--------|-------|-------------------------------------|
| <i>num_wheels</i>       | —     | uint32 | 3     | Number of wheels                    |
| <i>wow</i>              | —     | uint32 | 1     | Weight on wheels switch             |
| <i>gear_pos</i>         | —     | single | 3     | Landing gear normalized position    |
| <i>gear_steer</i>       | —     | single | 3     | Landing gear normalized steering    |
| <i>gear_compression</i> | —     | single | 3     | Landing gear normalized compression |

**Programmatic Use****Block Parameter:** ShowLandingGearInputs**Type:** character vector**Values:** 'off' | 'on'**Default:** 'off'**Show environment inputs – Environment inputs**

off (default) | on

Select this check box to include the environment inputs in the FlightGear net\_fdm data packet.

**Dependencies**

Select this check box to enable these input ports.

**Signal Group 6: Environment Inputs**

| Name              | Units | Type   | Width | Description                               |
|-------------------|-------|--------|-------|---|
| <i>agl</i>        | m     | single | 1     | Above ground level                        |
| <i>cur_time</i>   | sec   | uint32 | 1     | Current UNIX time                         |
| <i>warp</i>       | sec   | uint32 | 1     | Offset in seconds to UNIX time            |
| <i>visibility</i> | m     | single | 1     | Visibility in meters (for visual effects) |

**Programmatic Use****Block Parameter:** ShowEnvironmentInputs**Type:** character vector**Values:** 'off' | 'on'**Default:** 'off'**Sample time – Sample time**

1/30 (default) | scalar

Specify the sample time (-1 for inherited).

**Programmatic Use****Block Parameter:** SampleTime**Type:** character vector

**Values:** scalar  
**Default:** '1/30'

## **See Also**

FlightGear Preconfigured 6DoF Animation | Generate Run Script | Receive net\_ctrl Packet from FlightGear | Send net\_fdm Packet to FlightGear | Unpack net\_ctrl Packet from FlightGear

## **Topics**

“Flight Simulator Interface” on page 2-16

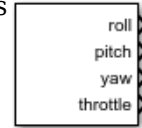
“Work with the Flight Simulator Interface” on page 2-20

**Introduced before R2006a**

# Pilot Joystick

Provide joystick interface on Windows platform

**Library:** Aerospace Blockset / Animation / Animation Support Utilities



## Description

The Pilot Joystick block provides a pilot joystick interface for a Windows platform. Roll, pitch, yaw, and throttle are mapped to the joystick *X*, *Y*, *R*, and *Z* channels respectively.

You can also configure the block to output all channels by setting the **Output configuration** parameter to `AllOutputs`. For more information, see Pilot Joystick All. The Pilot Joystick and Pilot Joystick All blocks are identical blocks with different **Output configuration** default settings.

This block does not produce deployable code.

## Limitations

- If the joystick does not support an *R* (rudder or twist) channel, yaw output is set to zero. Outputs are of type double, except when **Joystick ID** is set to `AllOutputs` mode, which is a uint32 flagword of bits. On non Microsoft platforms, this block outputs zeros.
- Pitch value has the opposite sense as that delivered by the FlightGear joystick interface.

## Ports

### Output

#### **roll** — Roll

range [-1, 1]

Roll command, specified in the range [-1, 1], that corresponds to the joystick left and right directions.

#### **Dependencies**

To enable this port, set **Output configuration** to `FourAxis`.

Data Types: double

#### **pitch** — Pitch

range [-1, 1]

Pitch command, specified in the range [-1, 1], that corresponds to the joystick forward or down and back and up directions.

#### **Dependencies**

This output port is enabled when the **Output configuration** parameter is set to `FourAxis`.

Data Types: double

**yaw — Yaw**

range [-1, 1]

Yaw command, specified in the range [-1, 1], that corresponds to the joystick twist left and twist right directions.

**Dependencies**

To enable this port, set **Output configuration** to FourAxis.

Data Types: double

**throttle — Throttle**

range [ 0, 1]

Throttle command, specified in the range [0, 1], that corresponds to the joystick min and max position.

**Dependencies**

to enable this port, set **Output configuration** to FourAxis.

Data Types: double

**Parameters****Joystick ID — Joystick ID**

Joystick1 (default) | Joystick2 | None

Specify the joystick ID as Joystick1, Joystick2, or None.

**Programmatic Use****Block Parameter:** JoystickID**Type:** character vector**Values:** Joystick1 | Joystick2 | None**Default:** 'Joystick1'**Output configuration — Joystick output configuration**

FourAxis (default) | AllOutputs

Joystick output configuration, specified as FourAxis or AllOutputs. For more information on the AllOutputs configuration, see Pilot Joystick All.

**Programmatic Use****Block Parameter:** OutputConfiguration**Type:** character vector**Values:** FourAxis | AllOutputs**Default:** 'FourAxis'**Sample time — Sample time**

1/30 (default) | -1 | scalar

Specify the sample time (-1 for inherited), specified as a scalar.

**Programmatic Use****Block Parameter:** SampleTime

**Type:** character vector

**Values:** scalar

**Default:** '1/30'

### **See Also**

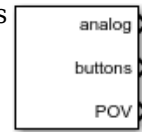
Pilot Joystick All | Simulation Pace

**Introduced before R2006a**

# Pilot Joystick All

Provide joystick interface in All Outputs configuration on Windows platform

**Library:** Aerospace Blockset / Animation / Animation Support Utilities



## Description

The Pilot Joystick All block provides a pilot joystick interface for a Windows platform. Analog is mapped to the joystick X, Y, Z, R, U, and V channels. Buttons and POV are mapped to up to 32 joystick button states and the joystick point-of-view hat.

You can also configure the block to output four axes by setting the **Output configuration** parameter to `FourAxis`.

This block does not produce deployable code.

## Limitations

- If the joystick does not support an *R* (rudder or twist) channel, yaw output is set to zero. Outputs are of type double, except when **Joystick ID** is set to `AllOutputs` mode, which is a `uint32` flagword of bits. On non Microsoft platforms, this block outputs zeros.
- Pitch value has the opposite sense as that delivered by the FlightGear joystick interface.

## Ports

### Output

#### **analog** — Analog output

range [-1, 1] | range [ 0, 1]

Analog output, returned according to:

| Array Number | Channel | Output Range | Joystick                | Description      |
|--------------|---------|--------------|-------------------------|------------------|
| 1            | X       | [-1, 1]      | [left, right]           | Roll command     |
| 2            | Y       | [-1, 1]      | [forward/down, back/up] | Pitch command    |
| 3            | Z       | [ 0, 1]      | [min, max]              | Throttle command |
| 4            | R       | [-1, 1]      | [left, right]           | Yaw command      |
| 5            | U       | [ 0, 1]      | [min, max]              | U channel value  |
| 6            | V       | [ 0, 1]      | [min, max]              | V channel value  |

Data Types: double

#### **buttons** — Button

flagword with 32 button states

Button output, returned as a flagword containing up to 32 button states on the buttons channel. Bit 0 is button 1, Bit 1 is button 2, and so forth.

Data Types: `uint32`

### **POV — Point-of-view**

`hat`

Point-of-view, returned as a hat value in degrees on the POV channel. Zero degrees is straight ahead, 90 degrees is to the left, and so forth.

Data Types: `double`

## **Parameters**

### **Joystick ID — Joystick ID**

`Joystick1 (default) | Joystick2 | None`

Specify the joystick ID as `Joystick1`, `Joystick2`, or `None`.

#### **Programmatic Use**

**Block Parameter:** `JoystickID`

**Type:** character vector

**Values:** `Joystick1 | Joystick2 | None`

**Default:** `'Joystick1'`

### **Output configuration — Joystick output configuration**

`AllOutputs (default) | FourAxis`

Joystick output configuration, specified as `FourAxis` or `AllOutputs`. For more information on the `AllOutputs` configuration, see [Pilot Joystick All](#).

#### **Programmatic Use**

**Block Parameter:** `OutputConfiguration`

**Type:** character vector

**Values:** `FourAxis | AllOutputs`

**Default:** `'FourAxis'`

### **Sample time — Sample time**

`1/30 (default) | -1 | scalar`

Specify the sample time (-1 for inherited), specified as a scalar.

#### **Programmatic Use**

**Block Parameter:** `SampleTime`

**Type:** character vector

**Values:** scalar

**Default:** `'1/30'`

## **See Also**

[Pilot Joystick](#) | [Simulation Pace](#)

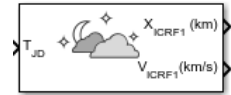
**Introduced in R2007a**



# Planetary Ephemeris

Implement position and velocity of astronomical objects

**Library:** Aerospace Blockset / Environment / Celestial Phenomena



## Description

The Planetary Ephemeris block uses Chebyshev coefficients to implement the position and velocity of the target object relative to the specified center object for a given Julian date. The **Target** parameter specifies an astronomical object. The block implements the ephemerides using the **Center** parameter for an astronomical object as the reference.

The block uses the Chebyshev coefficients that the NASA Jet Propulsion Laboratory provides.

---

**Tip** For  $T_{JD}$ , Julian date input for the block:

- Calculate the date using the Julian Date Conversion block or the Aerospace Toolbox `juliandate` function.
  - Calculate the Julian date using some other means and input it using the Constant block.
- 

This block implements the position and velocity using the International Celestial Reference Frame. If you require the planetary ephemeris position value relative to Earth in Earth-fixed (ECEF) coordinates, use the Direction Cosine Matrix ECI to ECEF block.

## Ports

### Input

#### $T_{JD}$ — Julian date

scalar | positive | between minimum and maximum Julian dates

Julian date, specified as a positive scalar between minimum and maximum Julian dates.

See the **Ephemeris model** parameter for the minimum and maximum Julian dates.

### Dependencies

This port displays if the **Epoch** parameter is set to `Julian date`.

Data Types: `double`

#### $T0_{JD}$ — Fixed Julian date

scalar | positive

Fixed Julian date for a specific epoch that is the most recent midnight at or before the interpolation epoch, specified as a positive scalar. The sum of  $T0_{JD}$  and  $\Delta T_{JD}$  must fall between the minimum and maximum Julian date.

See the **Ephemeris model** parameter for the minimum and maximum Julian dates.

#### Dependencies

This port displays if the **Epoch** parameter is set to  $T_0$  and elapsed Julian time.

Data Types: double

#### $\Delta T_{JD}$ — Elapsed Julian time

scalar | positive

Elapsed Julian time between the fixed Julian date and the ephemeris time, specified as a positive scalar. The sum of  $T_{0JD}$  and  $\Delta T_{JD}$  must fall between the minimum and maximum Julian date.

See the **Ephemeris model** parameter for the minimum and maximum Julian dates.

#### Dependencies

This port displays if the **Epoch** parameter is set to  $T_0$  and elapsed Julian time.

Data Types: double

#### Output

#### $X_{ICRF1}$ — Barycenter position

vector

Barycenter position ( $X_{ICRF1}$ ) of the **Target** object relative to the barycenter of the **Center** object, output as a vector, in km or astronomical units (AU).

---

**Tip** This block outputs the barycenter position in International Celestial Reference Frame (ICRF) coordinates. To convert these coordinates to Earth-centered Earth-fixed (ECEF), use the Direction Cosine Matrix ECI to ECEF block.

---

Data Types: double

#### $V_{ICRF}$ — Velocity

vector

Velocity ( $V_{ICRF}$ ) of the barycenter of the **Target** object relative to the barycenter of the **Center** object, specified as a vector, in km/s or astronomical units (AU)/day.

Data Types: double

#### Parameters

##### Units — Output units

km, km/s (default) | AU, AU/day

Output units, specified as km, km/s or AU, AU/day.

| Units    | Position | Velocity |
|----------|----------|----------|
| km, km/s | km       | km/s     |

| Units      | Position                | Velocity |
|------------|-------------------------|----------|
| Au, AU/day | astronomical units (AU) | AU/day   |

**Programmatic Use****Block Parameter:** kmflag**Type:** character vector**Values:** km, km/s | AU, AU/day**Default:** ' km, km/s '**Epoch — Epoch**Julian date (default) |  $T_0$  and elapsed Julian time

Epoch, specified as:

- Julian date

Julian date to implement the position and velocity of the **Target** object.. When this option is selected, the block has one input port,  $T_{JD}$ .

- $T_0$  and elapsed Julian time

Julian date, specified by two block inputs:

- Fixed Julian date representing a starting epoch.
- Elapsed Julian time between the fixed Julian date ( $T_{0JD}$ ) and the desired model simulation time. The sum of  $T_{0JD}$  and  $\Delta T_{JD}$  must fall between the minimum and maximum Julian date.

**Programmatic Use****Block Parameter:** epochflag**Type:** character vector**Values:** Julian date |  $T_0$  and elapsed Julian time**Default:** 'Julian date'**Ephemeris model — Ephemeris model**

DE405 (default) | DE421 | DE423 | DE430 | DE432t

Select one of the following ephemerides models defined by the Jet Propulsion Laboratory.

| Ephemeris Model | Description   |
|-----------------|---|
| DE405           | Released in 1998. This ephemeris takes into account the Julian date range 2305424.50 (December 9, 1599) to 2525008.50 (February 20, 2201).<br><br>This block implements these ephemerides with respect to the International Celestial Reference Frame version 1.0, adopted in 1998. |
| DE421           | Released in 2008. This ephemeris takes into account the Julian date range 2414992.5 (December 4, 1899) to 2469808.5 (January 2, 2050).<br><br>This block implements these ephemerides with respect to the International Celestial Reference Frame version 1.0, adopted in 1998.     |

| Ephemeris Model | Description  |
|-----------------|--|
| DE423           | Released in 2010. This ephemeris takes into account the Julian date range 2378480.5 (December 16, 1799) to 2524624.5 (February 1, 2200).<br><br>This block implements these ephemerides with respect to the International Celestial Reference Frame version 2.0, adopted in 2010.          |
| DE430           | Released in 2013. This ephemeris takes into account the Julian date range 2287184.5 (December 21, 1549) to 2688976.5 (January 25, 2650).<br><br>This block implements these ephemerides with respect to the International Celestial Reference Frame version 2.0, adopted in 2010.          |
| DE432t          | Released in April 2014. This ephemeris takes into account the Julian date range 2287184.5, (December 21, 1549 ) to 2688976.5, (January 25, 2650).<br><br>This block implements these ephemerides with respect to the International Celestial Reference Frame version 2.0, adopted in 2010. |

---

**Note** This block requires that you download ephemeris data using the Add-On Explorer. To start the Add-On Explorer, in the MATLAB Command Window, type `aeroDataPackage`. on the MATLAB desktop toolstrip, click the **Add-Ons** button.

---

#### Center – Center body

Sun (default) | Mercury | Venus | Earth | Moon | Mars | Jupiter | Saturn | Uranus | Neptune | Pluto | Solar system barycenter | Earth-Moon barycenter

Center body (astronomical object) or reference body, specified as a point of reference for the **Target** barycenter position and velocity measurement.

#### Programmatic Use

**Block Parameter:** nCenter

**Type:** character vector

**Values:** Sun | Mercury | Venus | Earth | Moon | Mars | Jupiter | Saturn | Uranus | Neptune | Pluto | Solar system barycenter | Earth-Moon barycenter

**Default:** 'Sun'

#### Target – Target body

Sun (default) | Mercury | Venus | Earth | Moon | Mars | Jupiter | Saturn | Uranus | Neptune | Pluto | Solar system barycenter | Earth-Moon barycenter

Target body (astronomical object) or reference body, specified as a point of reference for the barycenter position and velocity measurement.

#### Action for out-of-range input – Out-of-range block behavior

None (default) | Warning | Error

Out-of-range block behavior, specified as follows.

| Action | Description |
|--------|-------------|
| None   | No action.  |

| Action          | Description   |
|-----------------|---|
| Warning         | Warning in the MATLAB Command Window, model simulation continues. |
| Error (default) | MATLAB returns an exception, model simulation stops.              |

**Programmatic Use****Block Parameter:** errorflag**Type:** character vector**Values:** 'None' | 'Warning' | 'Error'**Default:** 'Error'**Calculate velocity – Calculate rate of target barycenter**

on (default) | off

Select this check box to calculate the velocity of the **Target** barycenter relative to the **Center** barycenter.

**Programmatic Use****Block Parameter:** velflag**Type:** character vector**Values:** 'off' | 'on' |**Default:** 'on'**References**

- [1] Folkner, W. M., J. G. Williams, D. H. Boggs. "The Planetary and Lunar Ephemeris DE 421." *IPN Progress Report* 42-178, 2009.
- [2] Ma, C. et al. "The International Celestial Reference Frame as Realized by Very Long Baseline Interferometry." *Astronomical Journal*, Vol. 116, 516–546, 1998.
- [3] Vallado, D. A. *Fundamentals of Astrodynamics and Applications*, New York: McGraw-Hill, 1997.

**Extended Capabilities****C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**See Also**

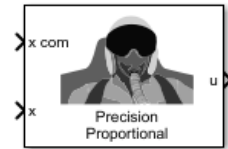
Constant | Direction Cosine Matrix ECI to ECEF | Earth Nutation | Julian Date Conversion | Moon Libration | aeroDataPackage | juliandate

**Introduced in R2013a**

## Precision Pilot Model

Represent precision pilot model

**Library:** Aerospace Blockset / Pilot Models



### Description

The Precision Pilot Model block represents the pilot model described in *Mathematical Models of Human Pilot Behavior* [1]. This pilot model is a single input, single output (SISO) model that represents some aspects of human behavior when controlling aircraft. When modeling human pilot models, use this block for more accuracy than that provided by the Tustin Pilot Model and Crossover Pilot Model blocks.

This block has non-linear behavior. If you want to linearize the block (for example, with one of the `linmod` functions), you might need to change the Pade approximation order. The Precision Pilot Model block implementation incorporates the Transport Delay block with the **Pade order (for linearization)** parameter set to 2 by default. To change this value, use the `set_param` function, for example:

```
set_param(gcb, 'pade', '3')
```

This block is an extension of the Crossover Pilot Model block. It implements the equation described in “Algorithms” on page 4-483.

### Ports

#### Input

##### **x com — Signal command**

scalar

Signal command that the pilot model controls, specified as a scalar.

Data Types: `double`

##### **x — Signal controlled by pilot**

scalar

Signal that the pilot model controls, specified as a scalar.

Data Types: `double`

#### Output

##### **u — Aircraft command**

scalar

Aircraft command, returned as a scalar.

Data Types: double

## Parameters

### Type of control – Aircraft dynamics control

Proportional (default) | Rate or velocity | Acceleration | Second order

Aircraft dynamics control. The equalizer form changes according to these values. For more information, see [2]. To help you decide, this table lists the options and associated dynamics.

| Option (Controlled Element Transfer Function) | Transfer Function of Controlled Element ( $Y_c$ )             | Transfer Function of Pilot ( $Y_p$ )                                       |
|---|---|--|
| Proportional                                  | $K_c$   | Lag-lead, $T_I \gg T_L$  |
| Rate or velocity                              | $\frac{K_c}{s}$   | 1  |
| Acceleration                                  | $\frac{K_c}{s^2}$   | Lead-lag, $T_L \gg T_I$  |
| Second order                                  | $\frac{K_c \omega_n^2}{s^2 + 2\zeta \omega_n s + \omega_n^2}$ | Lead-lag if $\omega_m \ll 2/\tau$ .<br>Lag-lead if $\omega_m \gg 2/\tau$ . |

This table defines the variables used in the list of control options.

| Variable   | Description                        |
|------------|------------------------------------|
| $K_c$      | Aircraft gain.                     |
| $T_I$      | Lag constant.                      |
| $T_L$      | Lead constant.                     |
| $\zeta$    | Damping ratio for the aircraft.    |
| $\omega_n$ | Natural frequency of the aircraft. |

#### Programmatic Use

**Block Parameter:** sw\_popup

**Type:** character vector

**Values:** 'Proportion' | 'Rate or velocity' | 'Acceleration' | 'Second order'

**Default:** 'Proportion'

#### Pilot gain – Pilot gain

1 (default) | scalar

Pilot gain, specified as a double scalar.

#### Programmatic Use

**Block Parameter:** Kp

**Type:** character vector

**Values:** double scalar

**Default:** '1'

#### Pilot time delay(s) – Pilot time delay

0.1 (default) | scalar

Total pilot time delay, specified as a double scalar, in seconds. This value typically ranges from 0.1 s to 0.2 s.

**Programmatic Use**

**Block Parameter:** time\_delay

**Type:** character vector

**Values:** double scalar

**Default:** '0.1'

**Equalizer lead constant — Equalizer lead constant**

1 (default) | scalar

Equalizer lead constant, specified as a double scalar.

**Dependencies**

To enable this parameter, set **Type of control** to Proportional, Acceleration, or Second order.

**Programmatic Use**

**Block Parameter:** TL

**Type:** character vector

**Values:** double scalar

**Default:** '1'

**Equalizer lag constant — Equalizer lag constant**

5 (default) | scalar

Equalizer lag constant, specified as a double scalar.

**Dependencies**

To enable this parameter, set **Type of control** to Proportional, Acceleration, or Second order.

**Programmatic Use**

**Block Parameter:** TI

**Type:** character vector

**Values:** double scalar

**Default:** '5'

**Lag constant for neuromuscular system — Lag constant**

0.1 (default) | scalar

Neuromuscular system lag constant, specified as a double scalar.

**Programmatic Use**

**Block Parameter:** TN1

**Type:** character vector

**Values:** double scalar

**Default:** 0.1

**Undamped natural frequency neuromuscular system (rad/s) — Undamped natural frequency**

20 (default) | scalar

Undamped natural frequency of the neuromuscular system, specified as a double scalar, in rad/s.



**Programmatic Use****Block Parameter:** nat\_freq**Type:** character vector**Values:** double scalar**Default:** 20**Damping neuromuscular system – Damping neuromuscular system**

0.7 (default) | scalar

Damping neuromuscular system, specified as a double scalar.

**Programmatic Use****Block Parameter:** damp**Type:** character vector**Values:** double scalar**Default:** 0.7**Controlled element undamped natural frequency (rad/s) – Controlled element undamped natural frequency**

15 (default) | scalar

Controlled element undamped natural frequency, specified as a double scalar, in rad/s.

**Dependencies**To enable this parameter, set **Type of control** to Second order.**Programmatic Use****Block Parameter:** omega\_m**Type:** character vector**Values:** double scalar**Default:** 15**Algorithms**

When calculating the model, this block also takes into account the neuromuscular dynamics of the pilot. This block implements the following equation:

$$Y_p = K_p e^{-\tau s} \left( \frac{T_L s + 1}{T_I s + 1} \right) \left[ \frac{1}{(T_{N1} s + 1) \left( \frac{s^2}{\omega_N^2} + \frac{2\zeta_N}{\omega_N} s + 1 \right)} \right],$$

where:

| Variable | Description                                 |
|----------|---|
| $K_p$    | Pilot gain.                                 |
| $\tau$   | Pilot delay time.                           |
| $T_L$    | Time lead constant for the equalizer term.  |
| $T_I$    | Time lag constant.                          |
| $T_{N1}$ | Time constant for the neuromuscular system. |

| Variable   | Description                                      |
|------------|--|
| $\omega_N$ | Undamped frequency for the neuromuscular system. |
| $\zeta_N$  | Damping ratio for the neuromuscular system.      |

A sample value for the natural frequency and the damping ratio of a human is 20 rad/s and 0.7, respectively. The term containing the lead-lag term is the equalizer form. This form changes depending on the characteristics of the controlled system. A consistent behavior of the model can occur at different frequency ranges other than the crossover frequency.

## References

- [1] McRuer, D. T., Krendel, E., *Mathematical Models of Human Pilot Behavior*. Advisory Group on Aerospace Research and Development AGARDograph 188, Jan. 1974.
- [2] McRuer, D. T., Graham, D., Krendel, E., and Reisener, W., *Human Pilot Dynamics in Compensatory Systems*. Air Force Flight Dynamics Lab. AFFDL-65-15. 1965.

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

## See Also

Crossover Pilot Model | Transport Delay | Tustin Pilot Model | `linmod`

**Introduced in R2012b**

# Pressure Altitude

Calculate pressure altitude based on ambient pressure

**Library:** Aerospace Blockset / Environment / Atmosphere



## Description

The Pressure Altitude block computes the pressure altitude based on ambient pressure. Pressure altitude is the altitude in the 1976 Committee on the Extension of the Standard Atmosphere (COESA) United States with specified ambient pressure.

Pressure altitude is also known as the mean sea level (MSL) altitude.

The Pressure Altitude block icon port label change based on the input and output units selected from the **Units** list.

## Limitations

- Below the pressure of 0.3961 Pa (approximately 0.00006 psi) and above the pressure of 101325 Pa (approximately 14.7 psi), altitude values are extrapolated logarithmically.
- Air is assumed to be dry and an ideal gas.

## Ports

### Input

#### Port\_1 — Static pressure

scalar | array

Static pressure, specified as a scalar or array.

Data Types: double

### Output

#### Port\_1 — Pressure altitude

scalar | array

Pressure altitude, returned as a scalar or vector.

Data Types: double

## Parameters

### Units — Input units

Metric (MKS) (default) | English

Input units, specified as:

| Units        | Pstatic                     | Alt_p  |
|--------------|-----------------------------|--------|
| Metric (MKS) | Pascal                      | Meters |
| English      | Pound force per square inch | Feet   |

**Programmatic Use****Block Parameter:** units**Type:** character vector**Values:** 'Metric (MKS)' | 'English'**Default:** 'Metric (MKS)'**Action for out-of-range input – Out-of-range block behavior**

Warning (default) | None | Error

Out-of-range block behavior, specified as follows.

| Action          | Description   |
|-----------------|---|
| None            | No action.  |
| Warning         | Warning in the MATLAB Command Window, model simulation continues. |
| Error (default) | MATLAB returns an exception, model simulation stops.              |

**Programmatic Use****Block Parameter:** action**Type:** character vector**Values:** 'None' | 'Warning' | 'Error'**Default:** 'Warning'**References**

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

**Extended Capabilities****C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**See Also**

COESA Atmosphere Model

**Topics**

Ideal Airspeed Correction

**Introduced before R2006a**

# Pressure Conversion

Convert from pressure units to desired pressure units

**Library:** Aerospace Blockset / Utilities / Unit Conversions



## 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 port labels change based on the input and output units selected from the **Initial unit** and the **Final unit** lists.

## Ports

### Input

#### Port\_1 – Pressure

scalar | array

Pressure, specified as a scalar or array, in initial pressure units.

### Dependencies

The input port label depends on the **Initial unit** setting.

Data Types: double

### Output

#### Port\_1 – Pressure

scalar | array

Pressure, returned as a scalar or array, in final pressure units.

### Dependencies

The output port label depends on the **Final unit** setting.

Data Types: double

## Parameters

### Initial unit – Input units

psi (default) | Pa | psf | atm

Input units, specified as:

|     |                            |
|-----|----------------------------|
| psi | Pound mass per square inch |
| Pa  | Pascals                    |

|     |                            |
|-----|----------------------------|
| psf | Pound mass per square foot |
| atm | Atmospheres                |

**Dependencies**

The input port label depends on the **Initial unit** setting.

**Programmatic Use**

**Block Parameter:** IU

**Type:** character vector

**Values:** 'psi' | 'Pa' | 'psf' | 'atm'

**Default:** 'psi'

**Final unit – Output units**

Pa (default) | psi | psf | atm

Output units, specified as:

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

**Dependencies**

The output port label depends on the **Final unit** setting.

**Programmatic Use**

**Block Parameter:** OU

**Type:** character vector

**Values:** 'psi' | 'Pa' | 'psf' | 'atm'

**Default:** 'Pa'

**Extended Capabilities****C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**See Also**

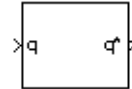
Acceleration Conversion | Angle Conversion | Angular Acceleration Conversion | Angular Velocity Conversion | Density Conversion | Force Conversion | Length Conversion | Mass Conversion | Temperature Conversion | Velocity Conversion

**Introduced before R2006a**

# Quaternion Conjugate

Calculate conjugate of quaternion

**Library:** Aerospace Blockset / Utilities / Math Operations



## Description

The Quaternion Conjugate block calculates the conjugate for a given quaternion. Aerospace Blockset uses quaternions that are defined using the scalar-first convention. For more information on quaternion forms, see “Algorithms” on page 4-489

## Ports

### Input

#### **q** — Input quaternion

quaternion | vector of quaternions

Quaternions in the form of  $[q_0, r_0, \dots, q_1, r_1, \dots, q_2, r_2, \dots, q_3, r_3, \dots]$ , specified as a quaternion or vector.

Data Types: double | bus

### Output

#### **q'** — Quaternion conjugate

quaternion conjugate | vector of quaternion conjugates

Quaternion conjugates in the form of  $[q_0', r_0', \dots, q_1', r_1', \dots, q_2', r_2', \dots, q_3', r_3', \dots]$ , returned as a quaternion or vector.

Data Types: double | bus

## Algorithms

The quaternion has the form of

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

The quaternion conjugate has the form of

$$q' = q_0 - \mathbf{i}q_1 - \mathbf{j}q_2 - \mathbf{k}q_3.$$

## References

- [1] Stevens, Brian L., Frank L. Lewis. *Aircraft Control and Simulation*, Second Edition. Hoboken, NJ: Wiley-Interscience.

## **Extended Capabilities**

### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

### **See Also**

[Quaternion Division](#) | [Quaternion Inverse](#) | [Quaternion Modulus](#) | [Quaternion Multiplication](#) | [Quaternion Norm](#) | [Quaternion Normalize](#) | [Quaternion Rotation](#)

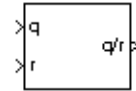
**Introduced before R2006a**



# Quaternion Division

Divide quaternion by another quaternion

**Library:** Aerospace Blockset / Utilities / Math Operations



## Description

The Quaternion Division block divides a given quaternion by another. Aerospace Blockset uses quaternions that are defined using the scalar-first convention. The output is the resulting quaternion from the division or vector of resulting quaternions from division. For the quaternion forms used, see “Algorithms” on page 4-491.

## Ports

### Input

#### **q — Dividend quaternion**

quaternion | vector of quaternions

Dividend quaternions in the form of  $[q_0, p_0, \dots, q_1, p_1, \dots, q_2, p_2, \dots, q_3, p_3, \dots]$ , specified as a quaternion or vector of quaternions.

Data Types: double

#### **r — Divisor quaternion**

quaternion | vector of quaternions

Divisor quaternions in the form of  $[s_0, r_0, \dots, s_1, r_1, \dots, s_2, r_2, \dots, s_3, r_3, \dots]$ , specified as a quaternion or vector of quaternions.

Data Types: double

### Output

#### **q/r — Output quaternion**

quaternion | vector

Output quaternion or vector of resulting quaternions from division.

Data Types: double

## Algorithms

The quaternions have the form of

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

and

$$r = r_0 + \mathbf{i}r_1 + \mathbf{j}r_2 + \mathbf{k}r_3.$$

The resulting quaternion from the division has the form of

$$t = \frac{q}{r} = t_0 + \mathbf{i}t_1 + \mathbf{j}t_2 + \mathbf{k}t_3,$$

where

$$t_0 = \frac{(r_0q_0 + r_1q_1 + r_2q_2 + r_3q_3)}{r_0^2 + r_1^2 + r_2^2 + r_3^2}$$

$$t_1 = \frac{(r_0q_1 - r_1q_0 - r_2q_3 + r_3q_2)}{r_0^2 + r_1^2 + r_2^2 + r_3^2}$$

$$t_2 = \frac{(r_0q_2 + r_1q_3 - r_2q_0 - r_3q_1)}{r_0^2 + r_1^2 + r_2^2 + r_3^2}$$

$$t_3 = \frac{(r_0q_3 - r_1q_2 + r_2q_1 - r_3q_0)}{r_0^2 + r_1^2 + r_2^2 + r_3^2}$$

## References

- [1] Stevens, Brian L., Frank L. Lewis. *Aircraft Control and Simulation*, Second Edition. Hoboken, NJ: Wiley-Interscience.

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

## See Also

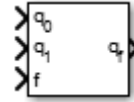
[Quaternion Conjugate](#) | [Quaternion Inverse](#) | [Quaternion Modulus](#) | [Quaternion Multiplication](#) | [Quaternion Norm](#) | [Quaternion Normalize](#) | [Quaternion Rotation](#)

**Introduced before R2006a**

# Quaternion Interpolation

Quaternion interpolation between two quaternions

**Library:** Aerospace Blockset / Utilities / Math Operations



## Description

The Quaternion Interpolation block calculates the quaternion interpolation between two normalized quaternions by an interval fraction. Aerospace Blockset uses quaternions that are defined using the scalar-first convention. Select the interpolation method from SLERP, LERP, or NLERP. For equations used for the interpolation methods, see “Algorithms” on page 4-495.

The two normalized quaternions are the two extremes between which the block calculates the quaternion.

## Ports

### Input

#### $q_0$ — First normalized quaternion

4-by-1 vector | 1-by-4 vector

First normalized quaternion for which to calculate the interpolation. This quaternion must be a normalized quaternion

Data Types: double

#### $q_1$ — Second normalized quaternion

4-by-1 vector | 1-by-4 vector

Second normalized quaternion for which to calculate the interpolation, specified as a 4-by-1 vector or 1-by-4 vector. This quaternion must be a normalized quaternion.

Data Types: double

#### $f$ — Interval fraction

scalar

Interval fraction by which to calculate the quaternion interpolation. This value varies between 0 and 1. It represents the intermediate rotation of the quaternion to be calculated. This fraction affects the interpolation method rotational velocities.

### Dependencies

The interval fraction affects the rotational velocities of the interpolation methods for the **Methods** parameter. For more information on interval fractions, see [1].

Data Types: double

**Output****q<sub>f</sub> — Natural logarithm**

vector

Natural logarithm of quaternion, returned as a vector.

Data Types: double

**Parameters****Methods — Quaternion interpolation method**

SLERP (default) | LERP | NLERP

Quaternion interpolation method to calculate the quaternion interpolation, specified as:

- SLERP  
Quaternion slerp. Spherical linear quaternion interpolation method.
- LERP  
Quaternion lerp. Linear quaternion interpolation method.
- NLERP  
Normalized quaternion linear interpolation method.

**Dependencies**

These methods have different rotational velocities, depending on the interval fraction from input port *f*. For more information on interval fractions, see [1].

**Programmatic Use****Block Parameter:** method**Type:** character vector**Values:** 'SLERP' | 'LERP' | 'NLERP'**Default:** 'SLERP'**Action for out-of-range input — Out-of-range block behavior**

Error (default) | None | Warning

Out-of-range block behavior, specified as follows.

| Action          | Description   |
|-----------------|---|
| None            | No action.  |
| Warning         | Warning in the MATLAB Command Window, model simulation continues. |
| Error (default) | MATLAB returns an exception, model simulation stops.              |

**Programmatic Use****Block Parameter:** action**Type:** character vector**Values:** 'None' | 'Warning' | 'Error'**Default:** 'Error'

## Algorithms

$Slerp(p, q, h) = p(p^*q)^h$  with  $h \in [0, 1]$ .

$LERP(p, q, h) = p(1 - h) + qh$  with  $h \in [0, 1]$ .

With  $r = LERP(p, q, h)$ ,  $NLERP(p, q, h) = \frac{r}{|r|}$ .

## References

- [1] Dam, Erik B., Martin Koch, Martin Lillholm. "Quaternions, Interpolation, and Animation." University of Copenhagen, København, Denmark, 1998.

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

### See Also

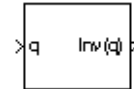
Quaternion Normalize

**Introduced in R2016a**

# Quaternion Inverse

Calculate inverse of quaternion

**Library:** Aerospace Blockset / Utilities / Math Operations



## Description

The Quaternion Inverse block calculates the inverse for a given quaternion. Aerospace Blockset uses quaternions that are defined using the scalar-first convention. For the equations used for the quaternion and quaternion inverse, “Algorithms” on page 4-496.

## Ports

### Input

#### **q – Quaternion**

quaternion | vector of quaternions

Quaternions in the form of  $[q_0, r_0, \dots, q_1, r_1, \dots, q_2, r_2, \dots, q_3, r_3, \dots]$ , specified as a quaternion or vector of quaternions.

Data Types: double

### Output

#### **Inv(q) – Quaternion inverse**

quaternion inverse | vector of quaternion inverses

Quaternion inverse or vector of quaternion inverses.

Data Types: double

## Algorithms

The quaternion has the form of

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

The quaternion inverse has the form of

$$q^{-1} = \frac{q_0 - \mathbf{i}q_1 - \mathbf{j}q_2 - \mathbf{k}q_3}{q_0^2 + q_1^2 + q_2^2 + q_3^2}.$$

## References

- [1] Stevens, Brian L., Frank L. Lewis. *Aircraft Control and Simulation*, Second Edition. Hoboken, NJ: Wiley-Interscience.

## **Extended Capabilities**

### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

### **See Also**

[Quaternion Conjugate](#) | [Quaternion Division](#) | [Quaternion Modulus](#) | [Quaternion Multiplication](#) | [Quaternion Norm](#) | [Quaternion Normalize](#) | [Quaternion Rotation](#)

**Introduced before R2006a**

# Quaternion Modulus

Calculate modulus of quaternion

**Library:** Aerospace Blockset / Utilities / Math Operations



## Description

The Quaternion Modulus block calculates the magnitude for a given quaternion. Aerospace Blockset uses quaternions that are defined using the scalar-first convention. For the equations used for the quaternion and quaternion modulus, see “Algorithms” on page 4-498.

## Ports

### Input

#### **q — Quaternion**

quaternion | vector of quaternions

Quaternions in the form of  $[q_0, r_0, \dots, q_1, r_1, \dots, q_2, r_2, \dots, q_3, r_3, \dots]$ , specified as a quaternion or vector of quaternions.

Data Types: double

### Output

#### **|q| — Quaternion modulus**

quaternion modulus | vector of quaternion moduli

Quaternion modulus or vector of quaternion moduli in the form of  $[|q|, |r|, \dots]$ .

Data Types: double

## Algorithms

The quaternion has the form of

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

The quaternion modulus has the form of

$$|q| = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2}$$

## References

- [1] Stevens, Brian L., Frank L. Lewis. *Aircraft Control and Simulation*, Second Edition. Hoboken, NJ: Wiley-Interscience.



## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

### See Also

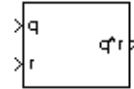
[Quaternion Conjugate](#) | [Quaternion Division](#) | [Quaternion Inverse](#) | [Quaternion Multiplication](#) | [Quaternion Norm](#) | [Quaternion Normalize](#) | [Quaternion Rotation](#)

**Introduced before R2006a**

# Quaternion Multiplication

Calculate product of two quaternions

**Library:** Aerospace Blockset / Utilities / Math Operations



## Description

The Quaternion Multiplication block calculates the product for two given quaternions. Aerospace Blockset uses quaternions that are defined using the scalar-first convention. For more information on the quaternion forms, see “Algorithms” on page 4-500.

## Ports

### Input

#### **q – First quaternion**

quaternion | vector of quaternions

First quaternion, specified as a vector or vector of quaternions. A vector of quaternions has this form, where  $q$  and  $p$  are quaternions:

$$[ q_0, p_0, \dots, q_1, p_1, \dots, q_2, p_2, \dots, q_3, p_3, \dots ]$$

Data Types: double

#### **r – Second quaternion**

quaternion | vector of quaternions

Second quaternion, specified as a vector or vector of quaternions. A vector of quaternions has this form, where  $s$  and  $r$  are quaternions:

$$[ s_0, r_0, \dots, s_1, r_1, \dots, s_2, r_2, \dots, s_3, r_3, \dots ]$$

Data Types: double

### Output

#### **q\*r – Product**

vector | vector of quaternion products

Product of two quaternions, output as a vector or vector of quaternion products.

Data Types: double

## Algorithms

This block uses quaternions of the form of

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

and

$$r = r_0 + \mathbf{i}r_1 + \mathbf{j}r_2 + \mathbf{k}r_3.$$

The quaternion product has the form of

$$t = q \times r = t_0 + \mathbf{i}t_1 + \mathbf{j}t_2 + \mathbf{k}t_3,$$

where

$$t_0 = (r_0q_0 - r_1q_1 - r_2q_2 - r_3q_3)$$

$$t_1 = (r_0q_1 + r_1q_0 - r_2q_3 + r_3q_2)$$

$$t_2 = (r_0q_2 + r_1q_3 + r_2q_0 - r_3q_1)$$

$$t_3 = (r_0q_3 - r_1q_2 + r_2q_1 + r_3q_0)$$

## References

- [1] Stevens, Brian L., Frank L. Lewis. *Aircraft Control and Simulation*, Second Edition. Hoboken, NJ: Wiley-Interscience.

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

## See Also

Quaternion Conjugate | Quaternion Division | Quaternion Inverse | Quaternion Modulus | Quaternion Norm | Quaternion Normalize | Quaternion Rotation

### Topics

“Explore the NASA HL-20 Model” on page 1-5

**Introduced before R2006a**

# Quaternion Norm

Calculate norm of quaternion

**Library:** Aerospace Blockset / Utilities / Math Operations



## Description

The Quaternion Norm block calculates the norm for a given quaternion. Aerospace Blockset uses quaternions that are defined using the scalar-first convention. For the equations used for the quaternion and quaternion norm, see “Algorithms” on page 4-502.

## Ports

### Input

#### **q — Quaternion norm**

quaternion norm | vector of quaternion norms

Quaternions in the form of  $[q_0, r_0, \dots, q_1, r_1, \dots, q_2, r_2, \dots, q_3, r_3, \dots]$ , specified as a quaternion norm or vector of quaternion norms.

Data Types: double

### Output

#### **norm(q) — Quaternion norm**

quaternion norm | vector of quaternion norms

Quaternion norm or vector of quaternion norms in the form of  $[norm(q), norm(r), \dots]$ .

Data Types: double

## Algorithms

The quaternion has the form of

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

The quaternion norm has the form of

$$norm(q) = q_0^2 + q_1^2 + q_2^2 + q_3^2$$

## References

- [1] Stevens, Brian L., Frank L. Lewis. *Aircraft Control and Simulation*, Second Edition. Hoboken, NJ: Wiley-Interscience.

## **Extended Capabilities**

### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

### **See Also**

[Quaternion Conjugate](#) | [Quaternion Division](#) | [Quaternion Inverse](#) | [Quaternion Modulus](#) | [Quaternion Multiplication](#) | [Quaternion Normalize](#) | [Quaternion Rotation](#)

**Introduced before R2006a**

# Quaternion Normalize

Normalize quaternion

**Library:** Aerospace Blockset / Utilities / Math Operations



## Description

The Quaternion Normalize block calculates a normalized quaternion for a given quaternion. Aerospace Blockset uses quaternions that are defined using the scalar-first convention. For the equations used for the quaternion and normalized quaternion, see “Algorithms” on page 4-504.

## Ports

### Input

#### **q** — Quaternion

quaternion | vector of quaternions

Quaternions in the form of  $[q_0, r_0, \dots, q_1, r_1, \dots, q_2, r_2, \dots, q_3, r_3, \dots]$ , specified as a quaternion or vector of quaternions.

Data Types: double

### Output

#### **normal(q)** — Normalized quaternion

normalized quaternion | vector of normalized quaternions

Normalized quaternion or vector of normalized quaternions.

Data Types: double

## Algorithms

The quaternion has the form of

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

The normalized quaternion has the form of

$$\text{normal}(q) = \frac{q_0 + \mathbf{i}q_1 + \mathbf{j}q_2 + \mathbf{k}q_3}{\sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2}}.$$

## References

- [1] Stevens, Brian L., Frank L. Lewis. *Aircraft Control and Simulation*, Second Edition. Hoboken, NJ: Wiley-Interscience.

## **Extended Capabilities**

### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

### **See Also**

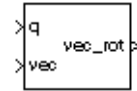
[Quaternion Conjugate](#) | [Quaternion Division](#) | [Quaternion Inverse](#) | [Quaternion Modulus](#) | [Quaternion Multiplication](#) | [Quaternion Norm](#) | [Quaternion Rotation](#)

**Introduced before R2006a**

# Quaternion Rotation

Rotate vector by quaternion

**Library:** Aerospace Blockset / Utilities / Math Operations



## Description

The Quaternion Rotation block rotates a vector by a quaternion. Aerospace Blockset uses quaternions that are defined using the scalar-first convention. For the equations used for the quaternion, vector, and rotated vector, see “Algorithms” on page 4-506.

## Ports

### Input

#### **q — Quaternion**

quaternion | vector

Quaternions in the form of  $[q_0, r_0, \dots, q_1, r_1, \dots, q_2, r_2, \dots, q_3, r_3, \dots]$ , specified as a quaternion or vector of quaternions.

Data Types: double

#### **vec — Vector**

vector | vector of vectors

Vector or vector of vectors in the form of  $[v_1, u_1, \dots, v_2, u_2, \dots, v_3, u_3, \dots]$ .

Data Types: double

### Output

#### **vec\_rot — Rotated quaternion**

rotated quaternion | vector of rotated quaternions

Rotated vector or vector of rotated vectors.

Data Types: double

## Algorithms

The quaternion has the form of

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

The vector has the form of

$$v = \mathbf{i}v_1 + \mathbf{j}v_2 + \mathbf{k}v_3.$$

The rotated vector has the form of



$$v' = \begin{bmatrix} v_1' \\ v_2' \\ v_3' \end{bmatrix} = \begin{bmatrix} (1 - 2q_2^2 - 2q_3^2) & 2(q_1q_2 + q_0q_3) & 2(q_1q_3 - q_0q_2) \\ 2(q_1q_2 - q_0q_3) & (1 - 2q_1^2 - 2q_3^2) & 2(q_2q_3 + q_0q_1) \\ 2(q_1q_3 + q_0q_2) & 2(q_2q_3 - q_0q_1) & (1 - 2q_1^2 - 2q_2^2) \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}$$

## References

- [1] Stevens, Brian L., Frank L. Lewis. *Aircraft Control and Simulation*, Second Edition. Hoboken, NJ: Wiley-Interscience.
- [2] Diebel, James. "Representing Attitude: Euler Angles, Unit Quaternions, and Rotation Vectors." Stanford University, Stanford, California, 2006.

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

## See Also

Quaternion Conjugate | Quaternion Division | Quaternion Inverse | Quaternion Multiplication | Quaternion Norm | Quaternion Normalize

**Introduced before R2006a**

## Quaternions to Direction Cosine Matrix

Convert quaternion vector to direction cosine matrix

**Library:** Aerospace Blockset / Utilities / Axes Transformations



### Description

The Quaternions to Direction Cosine Matrix block transforms a 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. Aerospace Blockset uses quaternions that are defined using the scalar-first convention. For more information, see “Algorithms” on page 4-508.

### Ports

#### Input

##### **q** — Quaternion

4-by-1 vector

Quaternion, specified as a 4-by-1 vector.

Data Types: double

#### Output

##### **DCM<sub>be</sub>** — Direction cosine matrix

3-by-3 matrix

Direction cosine matrix, returned as a 3-by-3 matrix.

Data Types: double

### Algorithms

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:

$$P' = qPq^c$$

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

$$q^c = q_0 - \mathbf{i}q_1 - \mathbf{j}q_2 - \mathbf{k}q_3$$

$$P = 0 + \mathbf{i}x + \mathbf{j}y + \mathbf{k}z$$

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.

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

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

### See Also

Direction Cosine Matrix to Quaternions | Rotation Angles to Quaternions | Direction Cosine Matrix to Rotation Angles | Rotation Angles to Direction Cosine Matrix

**Introduced before R2006a**

## Quaternions to Rodrigues

Convert quaternion to Euler-Rodrigues vector

**Library:** Aerospace Blockset / Utilities / Axes Transformations



### Description

The Quaternions to Rodrigues block converts the 4-by-1 quaternion to the three-element Euler-Rodrigues vector, where quaternion is defined using the scalar-first convention. Aerospace Blockset uses quaternions that are defined using the scalar-first convention. For more information on Euler-Rodrigues vectors, see “Algorithms” on page 4-510.

### Ports

#### Input

##### Quaternion — Quaternion

4-by-1 matrix

Quaternion from which to determine Euler-Rodrigues vector. Quaternion scalar is the first element.

Data Types: double

#### Output

##### rod — Euler-Rodrigues vector

three-element vector

Euler-Rodrigues vector determined from the quaternion.

Data Types: double

### Algorithms

- An Euler-Rodrigues vector  $\vec{b}$  represents a rotation by integrating a direction cosine of a rotation axis with the tangent of half the rotation angle as follows:

$$\vec{b} = [b_x \ b_y \ b_z]$$

where:

$$b_x = \tan\left(\frac{1}{2}\theta\right)s_x,$$

$$b_y = \tan\left(\frac{1}{2}\theta\right)s_y,$$

$$b_z = \tan\left(\frac{1}{2}\theta\right)s_z$$

are the Rodrigues parameters. Vector  $\vec{s}$  represents a unit vector around which the rotation is performed. Due to the tangent, the rotation vector is indeterminate when the rotation angle equals  $\pm\pi$  radians or  $\pm 180$  deg. Values can be negative or positive.

## References

- [1] Dai, J.S. "Euler-Rodrigues formula variations, quaternion conjugation and intrinsic connections." *Mechanism and Machine Theory*, 92, 144-152. Elsevier, 2015.

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

## See Also

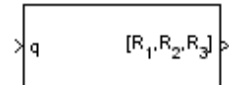
Direction Cosine Matrix to Rodrigues | Rodrigues to Direction Cosine Matrix | Rodrigues to Quaternions | Rodrigues to Rotation Angles | Rotation Angles to Rodrigues

**Introduced in R2017a**

# Quaternions to Rotation Angles

Determine rotation vector from quaternion

**Library:** Aerospace Blockset / Utilities / Axes Transformations



## Description

The Quaternions to Rotation Angles block converts the four-element quaternion vector ( $q_0, q_1, q_2, q_3$ ), into the rotation described by the three rotation angles (R1, R2, R3). The block generates the conversion by comparing elements in the direction cosine matrix (DCM) as a function of the rotation angles. The elements in the DCM are functions of a unit quaternion vector. Aerospace Blockset uses quaternions that are defined using the scalar-first convention. For more information on the direction cosine matrix, see “Algorithms” on page 4-513.

## Limitations

- For the ZYX, ZXY, YXZ, YZX, XYZ, and XZY rotations, the block generates an R2 angle that lies between  $\pm\pi/2$  radians, and R1 and R3 angles that lie between  $\pm\pi$  radians.
- For the 'ZYZ', 'ZXZ', 'YXY', 'YZY', 'YXY', and 'XZX' rotations, the block generates an R2 angle that lies between 0 and  $\pi$  radians, and R1 and R3 angles that lie between  $\pm\pi$  radians. However, in the latter case, when R2 is 0, R3 is set to 0 radians.

## Ports

### Input

#### **q — Quaternion**

4-by-1 vector

Quaternion, specified as a 4-by-1 vector.

Data Types: double

### Output

#### **[R<sub>1</sub>, R<sub>2</sub>, R<sub>3</sub>] — Rotation angles**

3-by-1 vector

Rotation angles, returned 3-by-1 vector, in radians.

Data Types: double

## Parameters

### **Rotation order — Rotation order**

ZYX (default) | ZYZ | ZXY | ZXZ | YXZ | YXY | YZX | YZY | XYZ | YXY | XZY | XZX

Output rotation order for the three rotation angles.

### Programmatic Use

**Block Parameter:** rotationOrder

**Type:** character vector

**Values:** ZYX | ZYZ | ZXY | ZXZ | YXZ | YXY | YZX | YZY | XYZ | XYX | XZY | XZX

**Default:** 'ZYX'

## Algorithms

The elements in the DCM are functions of a unit quaternion vector. For example, for the rotation order z-y-x, the DCM is defined as:

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

The DCM defined by a unit quaternion vector is:

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

From the preceding equation, you can derive the following relationships between DCM elements and individual rotation angles for a ZYX rotation order:

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

where  $\Psi$  is R1,  $\Theta$  is R2, and  $\Phi$  is R3.

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

## See Also

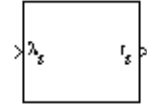
Direction Cosine Matrix to Quaternions | Direction Cosine Matrix to Rotation Angles | Quaternions to Direction Cosine Matrix | Rotation Angles to Direction Cosine Matrix | Rotation Angles to Quaternions

**Introduced in R2007b**

## Radius at Geocentric Latitude

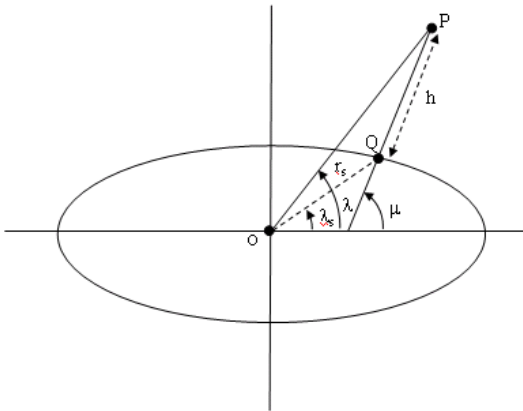
Estimate radius of ellipsoid planet at geocentric latitude

**Library:** Aerospace Blockset / Flight Parameters



### Description

The Radius at Geocentric Latitude block estimates the radius ( $r_s$ ) of an ellipsoid planet at a particular geocentric latitude ( $\lambda_s$ ).



The following equation estimates the ellipsoid radius ( $r_s$ ) using flattening ( $\bar{f}$ ), geocentric latitude ( $\bar{\lambda}_s$ ), and equatorial radius ( $\bar{R}$ ):

$$r_s = \sqrt{\frac{\bar{R}^2}{1 + \left[ \frac{1}{(1 - \bar{f})^2} - 1 \right] \sin^2 \bar{\lambda}_s}}$$

### Ports

#### Input

**$\lambda_s$  — Geocentric latitude**

scalar | vector

Geocentric latitude, specified as a scalar or vector, in degrees.

Data Types: double

#### Output

**$r_s$  — Radius**

scalar | vector

Radius of planet at geocentric latitude, returned as a scalar or vector, in the same units as flattening.



Data Types: double

## Parameters

### Units — Output units

Metric (MKS) (default) | English

Output units, specified as:

| Units        | Equatorial Radius | Radius at Geocentric Latitude |
|--------------|-------------------|-------------------------------|
| Metric (MKS) | Meters            | Meters                        |
| English      | Feet              | Feet                          |

### Dependencies

To enable this parameter, set **Planet model** to Earth (WGS84).

#### Programmatic Use

**Block Parameter:** units

**Type:** character vector

**Values:** 'Metric (MKS)' | 'English'

**Default:** 'Metric (MKS)'

### Planet model — Planet model

Earth (WGS84) (default) | Custom

Planet model to use, Custom or Earth (WGS84).

#### Programmatic Use

**Block Parameter:** ptype

**Type:** character vector

**Values:** 'Earth (WGS84)' | 'Custom'

**Default:** 'Earth (WGS84)'

### Flattening — Flattening of planet

1/298.257223563 (default) | scalar

Flattening of the planet, specified as a double scalar.

### Dependencies

To enable this parameter, set **Planet model** to Custom.

#### Programmatic Use

**Block Parameter:** F

**Type:** character vector

**Values:** double scalar

**Default:** 1/298.257223563

### Equatorial radius of planet — Radius of planet at equator

6378137 (default) | scalar

Radius of the planet at its equator, in the same units as the desired units for ECEF position.

### **Dependencies**

To enable this parameter, set **Planet model** to Custom.

### **Programmatic Use**

**Block Parameter:** R

**Type:** character vector

**Values:** double scalar

**Default:** 6378137

### **References**

- [1] Stevens, Brian L., Frank L. Lewis. *Aircraft Control and Simulation*, New York, John Wiley & Sons, 1992.
- [2] Zipfel, Peter H., *Modeling and Simulation of Aerospace Vehicle Dynamics*. Second Edition. Reston, VA: AIAA Education Series, 2000.

### **Extended Capabilities**

#### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

### **See Also**

Direction Cosine Matrix ECEF to NED | Direction Cosine Matrix ECEF to NED to Latitude and Longitude | ECEF Position to LLA | Geocentric to Geodetic Latitude | Geodetic to Geocentric Latitude | LLA to ECEF Position

**Introduced before R2006a**

# Receive net\_ctrl Packet from FlightGear

Receive net\_ctrl packet from FlightGear

**Library:** Aerospace Blockset / Animation / Flight Simulator Interfaces



## Description

The Receive net\_ctrl Packet from FlightGear block receives a network control and environment data packet, net\_ctrl, from the simulation of a Simulink model in the FlightGear simulator, or from a FlightGear session. This data packet is compatible with a particular version of FlightGear flight simulator. This block supports all signals supported by the FlightGear net\_ctrl data packet. The block arranges the signals into multiple groups. To enable or disable the signal groups, select the FlightGear version. The block inserts zeros for packet values that are part of inactive signal groups.

If you run a model that contains this block in Rapid Accelerator mode, the block produces zeros (0s) and it does not produce deployable code. In Accelerator mode, the block works as expected.

For details on signals and signal groups, see “Output” on page 4-517.

## Ports

### Output

#### net\_ctrl — Controls information from FlightGear

744-by-1 vector | 732-by-1 vector

Controls information from FlightGear, specified as a vector. The block returns packets according to platform and FlightGear version.

| Platform          | FlightGear Version  | Dimension Type  |
|-------------------|---|-----------------|
| Windows and Linux | v2019.1, v2018.3, v2018.2, v2018.1, v2017.3, v2017.1, v2016.3, v2016.1, v3.4, v3.2, v3.0, v2.12, v2.10, v2.8, v2.6, v2.4, v2.0  | 744-by-1 vector |
| macOS             | v2019.1*, v2018.3*, v2018.2*, v2018.1*, v2017.3*, v2017.1*, 2016.3, v2016.1*, v3.4*, v3.2*, v3.0*, v2.12*, v2.10*, v2.8*, v2.6* | 744-by-1 vector |
|                   | v2.4, v2.0  | 732-by-1 vector |

\* On a Mac OS system with FlightGear v2019.1, v2018.3, v2018.2, v2018.1, v2017.3, v2017.1, 2016.3, 2016.1, 3.4, 3.2, 3.0, 2.12, 2.10, 2.8, 2.6, you might see unexpected results (for example, very large or very small data values). For more information, see “Macintosh Platform and FlightGear Version 2.6 or Later” on page 2-33.

Data Types: uint8

### Rx bytes — Received FlightGear packet size

0 | 744 | 732

Received FlightGear packet size, specified as a scalar.

- 0, if no data is received
- Size of the packet in bytes (depending on the FlightGear version and architecture)

| Platform          | FlightGear Version  | Dimension Type |
|-------------------|---|----------------|
| Windows and Linux | —   | 0              |
|                   | v2019.1, v2018.3, v2018.2, v2018.1, v2017.3, v2017.1, v2016.3, v2016.1, v3.4, v3.2, v3.0, v2.12, v2.10, v2.8, v2.6, v2.4, v2.0  | 744            |
| macOS             | —   | 0              |
|                   | v2019.1*, v2018.3*, v2018.2*, v2018.1*, v2017.3*, v2017.1*, 2016.3, v2016.1*, v3.4*, v3.2*, v3.0*, v2.12*, v2.10*, v2.8*, v2.6* | 744            |
|                   | v2.0, v2.4  | 732            |

\* On a macOS system with FlightGear v2019.1, v2018.3, v2018.2, v2018.1, v2017.3, v2017.1, 2016.3, 2016.1, 3.4, 3.2, 3.0, 2.12, 2.10, 2.8, 2.6, you might see unexpected results (for example, very large or very small data values). For more information, see “Macintosh Platform and FlightGear Version 2.6 or Later” on page 2-33.

### Dependencies

This port is enabled by the **Enable received flag port** check box.

Data Types: double

## Parameters

### FlightGear version — FlightGear software version

v2019.1 (default) | v2018.3 | v2018.2 | v2018.1 | v2017.3 | v2017.1 | v2016.3 | v2016.1 | v3.4 | v3.2 | v3.0 | v2.12 | v2.10 | v2.8 | v2.6 | v2.4 | v2.0

Select your FlightGear software version.

**Note** If you are using a FlightGear version older than 2.0, the model displays a notification from the Simulink Upgrade Advisor. Consider upgrading your FlightGear version using the Upgrade Advisor. For more information, see “Supported FlightGear Versions” on page 2-16.

### Programmatic Use

**Block Parameter:** FlightGearVersion

**Type:** character vector

**Values:** v2019.1 | v2018.3 | v2018.2 | v2018.1 | v2017.3 | v2017.1 | v2016.3 | v2016.1 | v3.4 | v3.2 | v3.0 | v2.12 | v2.10 | v2.8 | v2.6 | v2.4 | v2.0

**Default:** 'v2019.1'

### Origin IP address — Origin IP address

127.0.0.1 (default) | scalar

Enter a valid IP address as a dot-decimal string. This IP address must be the address of the computer from which FlightGear is run, for example, 10.10.10.3.

You can also use a MATLAB expression that returns a valid IP address as a character vector. If FlightGear is run on the local computer, leave the default value of 127.0.0.1 (localhost).

To determine the source IP address, you can use one of several techniques, such as:

- Use 127.0.0.1 for the local computer (localhost).
- Ping another computer from a Windowscmd.exe (or Linux shell) prompt:

```
C:\> ping andyspc
```

```
Pinging andyspc [144.213.175.92] with 32 bytes of data:
```

```
Reply from 144.213.175.92: bytes=32 time=30ms TTL=253
```

```
Reply from 144.213.175.92: bytes=32 time=20ms TTL=253
```

```
Reply from 144.213.175.92: bytes=32 time=20ms TTL=253
```

```
Reply from 144.213.175.92: bytes=32 time=20ms TTL=253
```

```
Ping statistics for 144.213.175.92:
```

```
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
```

```
Approximate round trip times in milli-seconds:
```

```
    Minimum = 20ms, Maximum = 30ms, Average = 22ms
```

- On a Windows machine, type ipconfig and use the returned IP address:

```
H:\>ipconfig
```

```
Windows IP Configuration
```

```
Ethernet adapter Local Area Connection:
```

```
    Connection-specific DNS Suffix  . : 
    IP Address. . . . . : 192.168.42.178
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : 192.168.42.254
```

### Programmatic Use

**Block Parameter:** ReceiveAddress

**Type:** character vector

**Values:** scalar

**Default:** '127.0.0.1'

### Origin port — Origin port

5505 (default)

UDP port that the block accepts data from. The sender sends data to the port specified in this parameter. This value must match the **Origin port** parameter of the Generate Run Script block. It must be a unique port number that no other application on the computer uses. The site, <https://>

[en.wikipedia.org/wiki/List\\_of\\_TCP\\_and\\_UDP\\_port\\_numbers](http://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers), lists commonly known UDP port numbers. To identify UDP port numbers already in use on your computer, type:

```
netstat -a -p UDP
```

**Programmatic Use**

**Block Parameter:** ReceivePort

**Type:** character vector

**Values:** scalar

**Default:** '5505'

**Sample time — Sample time**

1/30 (default) | scalar

Specify the sample time (-1 for inherited).

**Programmatic Use**

**Block Parameter:** SampleTime

**Type:** character vector

**Values:** scalar

**Default:** '1/30'

**Enable received flag port — Enable received flag output port**

off (default) | on

Enable a received flag output port. Use this check box to determine if a FlightGear network packet has been received.

**Dependencies**

Selecting this check box enables the **Rx bytes** port.

**Programmatic Use**

**Block Parameter:** packetFlag

**Type:** character vector

**Values:** 'off' | 'on'

**Default:** 'off'

**See Also**

[FlightGear Preconfigured 6DoF Animation](#) | [Generate Run Script](#) | [Pack net\\_fdm Packet for FlightGear](#) | [Send net\\_fdm Packet to FlightGear](#) | [Unpack net\\_ctrl Packet from FlightGear](#)

**Topics**

“Flight Simulator Interface” on page 2-16

“Work with the Flight Simulator Interface” on page 2-20

**External Websites**

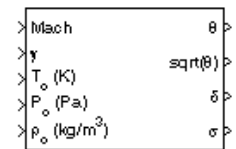
[https://en.wikipedia.org/wiki/List\\_of\\_TCP\\_and\\_UDP\\_port\\_numbers](https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers)

**Introduced in R2012a**

# Relative Ratio

Calculate relative atmospheric ratios

**Library:** Aerospace Blockset / Flight Parameters



## Description

The Relative Ratio block computes the relative atmospheric ratios, including the relative temperature ratio ( $\theta$ ),  $\sqrt{\theta}$ , relative pressure ratio ( $\delta$ ), and relative density ratio ( $\sigma$ ).

$\theta$  represents the ratio of the air stream temperature at a chosen reference station relative to sea level standard atmospheric conditions:

$$\theta = \frac{T}{T_0}$$

$\delta$  represents the ratio of the air stream pressure at a chosen reference station relative to sea level standard atmospheric conditions:

$$\delta = \frac{P}{P_0}$$

$\sigma$  represents the ratio of the air stream density at a chosen reference station relative to sea level standard atmospheric conditions:

$$\sigma = \frac{\rho}{\rho_0}$$

The Relative Ratio block icon displays the input units selected from the **Units** parameter.

## Limitations

For cases in which total temperature, total pressure, or total density ratio is desired (Mach number is nonzero), the total temperature, total pressure, and total densities are calculated assuming perfect gas (with constant molecular weight, constant pressure specific heat, and constant specific heat ratio) and dry air.

## Ports

### Input

#### Mach — Mach number

scalar

Mach number, specified as a scalar.

Data Types: double

**$\gamma$  — Ratio**

scalar

Ratio between the specific heat at constant pressure ( $C_p$ ) and the specific heat at constant volume ( $C_v$ ), specified as a scalar. For example, ( $\gamma = C_p/C_v$ ).

Data Types: double

 **$T_o$  — Static temperature**

scalar

Static temperature, specified as a scalar.

Data Types: double

 **$P_o$  — Static pressure**

scalar

Static pressure, specified as a scalar.

Data Types: double

 **$\rho_o$  — Static density**

scalar

Static density, specified as a scalar.

Data Types: double

**Output** **$\theta$  — Relative temperature ratio**

scalar

Relative temperature ratio ( $\theta$ ), returned as a scalar.

**Dependencies**

To enable this port, select **Theta**.

Data Types: double

**sqrt( $\theta$ ) — Square root of relative temperature ratio**

scalar

Square root of the relative temperature ratio ( $\sqrt{\theta}$ ), returned as a scalar.

**Dependencies**

To enable this port, select **Square root of theta**.

Data Types: double

 **$\delta$  — Relative pressure ratio**

scalar

Relative pressure ratio, ( $\delta$ ), returned as a scalar.



**Dependencies**

To enable this port, select **Delta**.

Data Types: double

 **$\sigma$  – Relative density ratio**

scalar

Relative density ratio, ( $\sigma$ ), returned as a scalar.

**Dependencies**

To enable this port, select **Sigma**.

Data Types: double

**Parameters****Units – Units**

Metric (MKS) (default) | English

Input units, specified as:

| Units        | Tstatic         | Pstatic                     | rho_static                |
|--------------|-----------------|-----------------------------|---------------------------|
| Metric (MKS) | Kelvin          | Pascal                      | Kilograms per cubic meter |
| English      | Degrees Rankine | Pound force per square inch | Slug per cubic foot       |

**Programmatic Use**

**Block Parameter:** units

**Type:** character vector

**Values:** 'Metric (MKS)' | 'English'

**Default:** 'Metric (MKS)'

**Theta – Relative temperature ratio**

on (default) | off

When selected, the block calculates the relative temperature ratio ( $\theta$ ) and static temperature is a required input.

**Programmatic Use**

**Block Parameter:** theta

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'on'

**Square root of theta – Square root of relative temperature ratio**

on (default) | off

When selected, the block calculates the square root of relative temperature ratio ( $\sqrt{\theta}$ ) and static temperature is a required input.

**Dependencies**

Selecting this check box enables the  $\sqrt{\theta}$  output port.

**Programmatic Use**

**Block Parameter:** sq\_theta

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'on'

**Delta — Relative pressure ratio**

on (default) | off

When selected, the block calculates the relative pressure ratio ( $\delta$ ) and static pressure is a required input.

**Programmatic Use**

**Block Parameter:** delta

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'on'

**Sigma — Relative density ratio**

on (default) | off

When selected, the block the relative density ratio ( $\sigma$ ) and static density is a required input.

**Programmatic Use**

**Block Parameter:** sigma

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'on'

**References**

[1] *Aeronautical Vestpocket Handbook*, United Technologies Pratt & Whitney, August, 1986.

**Extended Capabilities****C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

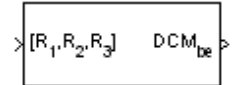
**See Also**

**Introduced before R2006a**

# Rotation Angles to Direction Cosine Matrix

Convert rotation angles to direction cosine matrix

**Library:** Aerospace Blockset / Utilities / Axes Transformations



## Description

The Rotation Angles to Direction Cosine Matrix block determines the direction cosine matrix (DCM) from a given set of rotation angles, R1, R2, and R3, of the first, second, and third rotation angles, respectively. For example, the default rotation angle order ZYX represents a sequence where R1 is z-axis rotation (yaw), R2 is y-axis rotation (pitch), and R3 is x-axis rotation (roll). Use the **Rotation Order** parameter to change the sequence.

## Ports

### Input

**$[R_1, R_2, R_3]$  — Rotation angles**

3-by-1 vector

Rotation angles, specified as a 3-by-1 vector, in radians.

Data Types: double

### Output

**$DCM_{be}$  — Direction cosine matrix**

3-by-3 matrix

Direction cosine matrix that performs coordinate transformations based on rotation angles, returned as a 3-by-3 matrix.

Data Types: double

## Parameters

**Rotation Order — Rotation order**

ZYX (default) | ZYZ | ZXY | ZXZ | YXZ | YXY | YZX | YZY | XYZ | YXX | XZY | XZX

Input rotation order for the three rotation angles.

### Programmatic Use

**Block Parameter:** rotationOrder

**Type:** character vector

**Values:** 'ZYX' | 'ZYZ' | 'ZXY' | 'ZXZ' | 'YXZ' | 'YXY' | 'YZX' | 'YZY' | 'XYZ' | 'YXX' | 'XZY' | 'XZX'

**Default:** 'ZYX'

## **Extended Capabilities**

### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

### **See Also**

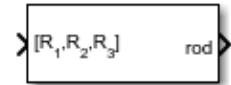
Direction Cosine Matrix to Quaternions | Direction Cosine Matrix to Rotation Angles | Quaternions to Direction Cosine Matrix

### **Introduced in R2007b**

# Rotation Angles to Rodrigues

Convert rotation angles to Euler-Rodrigues vector

**Library:** Aerospace Blockset / Utilities / Axes Transformations



## Description

The Rotation Angles to Rodrigues block converts the rotation described by the three rotation angles  $R_1, R_2, R_3$  into the three-element Euler-Rodrigues vector. For more information on Euler-Rodrigues vectors, see “Algorithms” on page 4-528.

## Ports

### Input

#### **$R_1, R_2, R_3$ — Rotation angles**

three-element vector

Rotation angles, in radians, from which to determine the Euler-Rodrigues vector. Values must be double.

### Output

#### **rod — Euler-Rodrigues vector**

three-element vector

Euler-Rodrigues vector determined from rotation angles.

Data Types: double

## Parameters

#### **Rotation order — Rotation order**

ZYX (default) | ZYX | ZYZ | ZXY | ZXZ | YXZ | YXY | YZX | YZY | XYZ | XYX | XZY | XZX

Rotation order for three wind rotation angles.

The default limitations for the 'ZYX', 'ZXY', 'YXZ', 'YZX', 'XYZ', and 'XZY' sequences generate an  $R_2$  angle that lies between  $\pm\pi/2$  radians ( $\pm 90$  degrees), and  $R_1$  and  $R_3$  angles that lie between  $\pm\pi$  radians ( $\pm 180$  degrees).

The default limitations for the 'ZYZ', 'ZXZ', 'YXY', 'YZY', 'YXX', and 'XZX' sequences generate an  $R_2$  angle that lies between 0 and  $\pi$  radians (180 degrees), and  $R_1$  and  $R_3$  angles that lie between  $\pm\pi$  ( $\pm 180$  degrees).

Rodrigues transformation is not defined for rotation angles equal to  $\pm\pi$  radians ( $\pm 180$  deg).

## Algorithms

An Euler-Rodrigues vector  $\vec{b}$  represents a rotation by integrating a direction cosine of a rotation axis with the tangent of half the rotation angle as follows:

$$\vec{b} = [b_x \ b_y \ b_z]$$

where:

$$b_x = \tan\left(\frac{1}{2}\theta\right)s_x,$$

$$b_y = \tan\left(\frac{1}{2}\theta\right)s_y,$$

$$b_z = \tan\left(\frac{1}{2}\theta\right)s_z$$

are the Rodrigues parameters. Vector  $\vec{s}$  represents a unit vector around which the rotation is performed. Due to the tangent, the rotation vector is indeterminate when the rotation angle equals  $\pm\pi$  radians or  $\pm 180$  deg. Values can be negative or positive.

## References

- [1] Dai, J.S. "Euler-Rodrigues formula variations, quaternion conjugation and intrinsic connections." *Mechanism and Machine Theory*, 92, 144-152. Elsevier, 2015.

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

## See Also

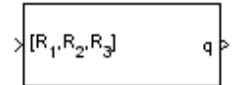
[Direction Cosine Matrix to Rodrigues](#) | [Quaternions to Rodrigues](#) | [Rodrigues to Direction Cosine Matrix](#) | [Rodrigues to Quaternions](#) | [Rodrigues to Rotation Angles](#)

**Introduced in R2017a**

# Rotation Angles to Quaternions

Calculate quaternion from rotation angles

**Library:** Aerospace Blockset / Utilities / Axes Transformations



## Description

The Rotation Angles to Quaternions block converts the rotation described by the three rotation angles (R1, R2, R3) into the four-element quaternion vector  $(q_0, q_1, q_2, q_3)$ , where quaternion is defined using the scalar-first convention. Aerospace Blockset uses quaternions that are defined using the scalar-first convention. For more information on quaternions, see “Algorithms” on page 4-530.

## Limitations

- The limitations for the ZYX, ZXY, YXZ, YZX, XYZ, and XZY implementations generate an R2 angle that is between  $\pm 90$  degrees, and R1 and R3 angles that are between  $\pm 180$  degrees.
- The limitations for the ZYZ, ZXZ, YXY, YZY, XYX, and XZX implementations generate an R2 angle that is between 0 and 180 degrees, and R1 and R3 angles that are between  $\pm 180$  degrees.

## Ports

### Input

**[R<sub>1</sub>, R<sub>2</sub>, R<sub>3</sub>] — Rotation angles**

3-by-1 vector

Rotation angles, specified as a 3-by-1 vector, in radians.

Data Types: double

### Output

**q — Quaternion**

4-by-1 vector

Quaternion, specified as a 4-by-1 vector.

Data Types: double

## Parameters

**Rotation order — Rotation order**

ZYX (default) | ZYZ | ZXY | ZXZ | YXZ | YXY | YZX | YZY | XYZ | YXX | XZY | XZX

Specifies the output rotation order for three wind rotation angles.

**Programmatic Use**

**Block Parameter:** rotationOrder

**Type:** character vector

**Values:** 'ZYX' | 'YZY' | 'ZXY' | 'ZXZ' | 'YXZ' | 'YXY' | 'YZX' | 'YZY' | 'XYZ' | 'XYX' | 'XZY' | 'XZX'

**Default:** 'ZYX'

## Algorithms

A quaternion vector represents a rotation about a unit vector  $(\mu_x, \mu_y, \mu_z)$  through the angle  $\theta$ . 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:

$$i^2 = j^2 = k^2 = -1$$

$$ij = -ji = k$$

$$jk = -kj = i$$

$$ki = -ik = j$$

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.

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

## See Also

Direction Cosine Matrix to Quaternions | Quaternions to Direction Cosine Matrix | Quaternions to Rotation Angles | Rotation Angles to Direction Cosine Matrix

**Introduced in R2007b**



# Revolutions Per Minute (RPM) Indicator

Display measurements for engine revolutions per minute (RPM) in percentage of RPM

**Library:** Aerospace Blockset / Flight Instruments



## Description

The RPM Indicator block displays measurements for engine revolutions per minute in percentage of RPM.

The range of values for RPM goes from 0 to 110 %. Minor ticks represent increments of 5 % RPM and major ticks represent increments of 10 % RPM.

---

**Tip** To facilitate understanding and debugging your model, you can modify instrument block connections in your model during normal and accelerator mode simulations.

---

## Parameters

### Connection — Connect to signal

signal name

Connect to signal for display, selected from list of signal names.

To view the data from a signal, select a signal in the model. The signal appears in the **Connection** table. Select the option button next to the signal you want to display. Click **Apply** to connect the signal.

The table has a row for the signal connected to the block. If there are no signals selected in the model, or the block is not connected to any signals, the table is empty.

### Scale Colors — Ranges of color bands

0 (default) | double | scalar

Ranges of color bands on the outside of the scale, specified as a finite double, or scalar value. Specify the minimum and maximum color range to display on the gauge.

To add a new color, click +. To remove a color, click -.

### Programmatic Use

**Block Parameter:** ScaleColors

**Type:**  $n$ -by-1 struct array

**Values:** struct array with elements Min, Max, and Color

### Label — Block label location

Top (default) | Bottom | Hide

Block label, displayed at the top or bottom of the block, or hidden.

- Top

Show label at the top of the block.

- Bottom

Show label at the bottom of the block.

- Hide

Do not show the label or instructional text when the block is not connected.

**Programmatic Use**

**Block Parameter:** LabelPosition

**Type:** character vector

**Values:** 'Top' | 'Bottom' | 'Hide'

**Default:** 'Top'

**Extended Capabilities****C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

This block is ignored for code generation.

**See Also**

Indicator | Airspeed Indicator | Altimeter | Artificial Horizon | Climb Rate Indicator | Exhaust Gas Temperature (EGT) Indicator | Heading Indicator | Turn Coordinator

**Topics**

“Display Measurements with Cockpit Instruments” on page 2-39

“Programmatically Interact with Gauge Band Colors” on page 2-41

“Flight Instrument Gauges” on page 2-38

**Introduced in R2016a**

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

Implement state-space controller in self-conditioned form

**Library:** Aerospace Blockset / GNC / Control



### Description

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

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

in the self-conditioned form

$$\begin{aligned} \dot{z} &= (A - HC)z + (B - HD)e + Hu_{meas} \\ u_{dem} &= Cz + De \end{aligned}$$

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 System Toolbox command `place` to place the poles at defined locations.

### Limitations

This block requires the Control System Toolbox license.

### Ports

#### Input

##### **e** – Control error

vector

Control error, specified as a vector.

Data Types: `double`

##### **u\_meas** – Achieved actuator positions

vector

Achieved actuator positions, specified as a vector.

Data Types: `double`

**Output****u\_dem — Actuator demands**

vector

Actuator demands, specified as a vector.

Data Types: double

**Parameters****A-matrix — A-matrix of the state-space implementation**

[-1 -0.2;0 -3] (default) | array

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

**Programmatic Use****Block Parameter:** Ak**Type:** character vector**Values:** vector**Default:** '[-1 -0.2;0 -3]'**B-matrix — B-matrix of the state-space implementation**

[1;1] (default) | array

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

**Programmatic Use****Block Parameter:** Bk**Type:** character vector**Values:** vector**Default:** '[1;1]'**C-matrix — C-matrix of the state-space implementation**

[1 0] (default) | array

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

**Programmatic Use****Block Parameter:** Ck**Type:** character vector**Values:** vector**Default:** '[1 0]'**D-matrix — D-matrix of the state-space implementation**

0.02 (default) | array | scalar

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

**Programmatic Use****Block Parameter:** Dk**Type:** character vector**Values:** vector**Default:** '0.02'**Initial state, x\_initial – Initial states**

0 (default) | vector

Initial states for the controller, that is, initial values for the state vector,  $z$ . It should have length equal to the size of the first dimension of  $A$ .

**Programmatic Use****Block Parameter:** x\_initial**Type:** character vector**Values:** vector**Default:** '0'**Poles of A-H\*C – Desired poles**

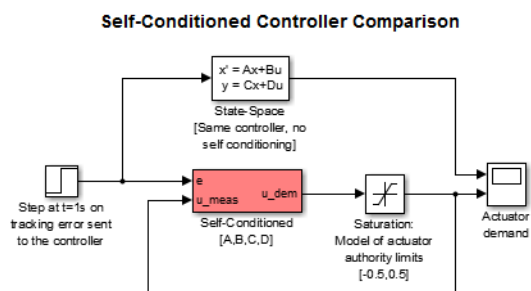
[-5 -2] (default) | vector

Desired poles of  $A-H*C$ , specified as a vector. Hence the number of pole locations defined should be equal to the dimension of the  $A$ -matrix.

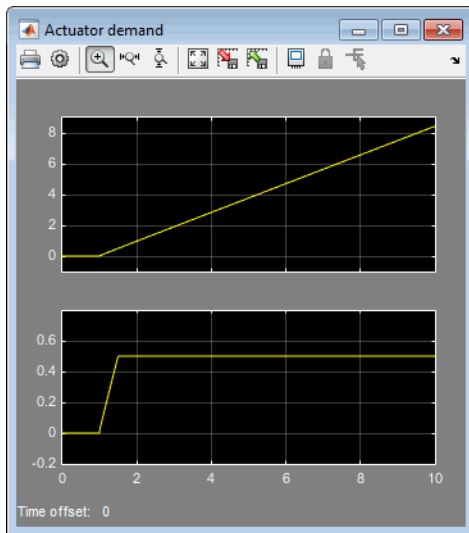
**Programmatic Use****Block Parameter:** vec\_w**Type:** character vector**Values:** vector**Default:** '[-5 -2]'**More About****State-Space Controller**

State-space controller implemented in both self-conditioned and standard state-space forms.

This Simulink model shows a state-space controller implemented in both self-conditioned and standard state-space forms. The actuator authority limits of  $\pm 0.5$  units are modeled by the Saturation block.



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

- [1] Kautsky, Nichols, and Van Dooren, "Robust Pole Assignment in Linear State Feedback," *International Journal of Control*, Vol. 41, Number 5, 1985, pp. 1129-1155.

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

### 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  $[A(v), B(v), C(v), D(v)]$  | Linear Second-Order Actuator | Nonlinear Second-Order Actuator | Saturation

Introduced before R2006a

## Send net\_fdm Packet to FlightGear

Transmit net\_fdm packet to destination IP address and port for FlightGear session

**Library:** Aerospace Blockset / Animation / Flight Simulator Interfaces



### Description

The Send net\_fdm Packet to FlightGear block transmits the net\_fdm packet to FlightGear on the current computer, or a remote computer on the network. The packet is constructed using the Pack net\_fdm Packet for FlightGear block. The destination port should be an unused port that you can use when you launch FlightGear with the FlightGear command line flag:

```
--fdm=network,localhost,5501,5502,5503
```

This block does not produce deployable code.

### Ports

#### Input

**net\_fdm — FlightGear net\_fdm data packet**

scalar

FlightGear net\_fdm data packet, specified as a scalar.

Data Types: uint8

### Parameters

**Destination IP address — Destination IP address for remote computer**

127.0.0.1 (default) | scalar

Destination IP address, specified as a scalar.

You can use one of several techniques to determine the destination IP address, such as:

- Use 127.0.0.1 for the local computer
- Ping another computer from a Windows cmd.exe (or UNIX shell) prompt:

```
C:\> ping andyspc
```

```
Pinging andyspc [144.213.175.92] with 32 bytes of data:
```

```
Reply from 144.213.175.92: bytes=32 time=30ms TTL=253
```

```
Reply from 144.213.175.92: bytes=32 time=20ms TTL=253
```

```
Reply from 144.213.175.92: bytes=32 time=20ms TTL=253
```

```
Reply from 144.213.175.92: bytes=32 time=20ms TTL=253
```

```
Ping statistics for 144.213.175.92:
```

```

Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
  Minimum = 20ms, Maximum = 30ms, Average = 22ms

```

- On a Windows machine, type `ipconfig` and use the returned *IP Address*:

```

H:\>ipconfig

Windows IP Configuration

Ethernet adapter Local Area Connection:

    Connection-specific DNS Suffix  . :
    IP Address. . . . . : 192.168.42.178
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : 192.168.42.254

```

#### **Programmatic Use**

**Block Parameter:** DestinationIpAddress

**Type:** character vector

**Values:** scalar

**Default:** 127.0.0.1

#### **Destination port — Destination port for remote computer**

5502 (default) | scalar

Destination port, specified as a scalar

#### **Programmatic Use**

**Block Parameter:** DestinationPort

**Type:** character vector

**Values:** scalar

**Default:** 5502

#### **Sample time — Sample time**

1/30 (default) | scalar

Sample time (-1 for inherited), specified as a scalar.

#### **Programmatic Use**

**Block Parameter:** SampleTime

**Type:** character vector

**Values:** scalar

**Default:** 1/30

## **See Also**

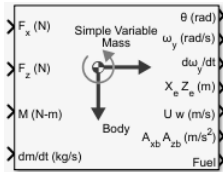
FlightGear Preconfigured 6DoF Animation | Generate Run Script | Pack net\_fdm Packet for FlightGear | Receive net\_ctrl Packet from FlightGear | Unpack net\_ctrl Packet from FlightGear

**Introduced before R2006a**



## Simple Variable Mass 3DOF (Body Axes)

Implement three-degrees-of-freedom equations of motion of simple variable mass with respect to body axes

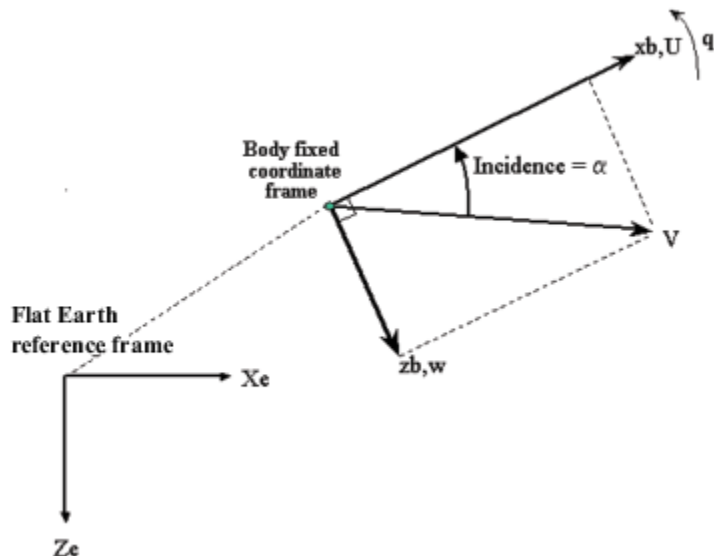


## Library

Equations of Motion/3DOF

## Description

The Simple Variable Mass 3DOF (Body Axes) block considers the rotation in the vertical plane of a body-fixed coordinate frame about a flat Earth reference frame.



The equations of motion are

$$A_{xb} = \dot{u} = \frac{F_x}{m} - \frac{\dot{m}U_{re_b}}{m} - qw - g\sin\theta$$

$$A_{be} = \begin{bmatrix} A_{xe} \\ A_{ze} \end{bmatrix} = \frac{F_b - \dot{m}V_{re}}{m} - \bar{g}$$

$$V_{re_b} = [U_{re} \ W_{re}]_b$$

$$A_{zb} = \dot{w} = \frac{F_z}{m} - \frac{\dot{m}W_{re_b}}{m} + qu + g\cos\theta$$

$$\dot{q} = \frac{M - \dot{I}_{yy}q}{I_{yy}}$$

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

$$\dot{I}_{yy} = \frac{I_{yyfull} - I_{yyempty}}{m_{full} - m_{empty}}\dot{m}$$

where the applied forces are assumed to act at the center of gravity of the body.  $U_{re_b}$  and  $W_{re_b}$  are the relative velocities of the mass flow ( $\dot{m}$ ) being added to or ejected from the body in body-fixed axes.

## Parameters

### Main

### Units

Specifies the input and output units:

| Units                      | Forces | Moment       | Acceleration              | Velocity          | Position | Mass     | Inertia                |
|----------------------------|--------|--------------|---------------------------|-------------------|----------|----------|------------------------|
| Metric (MKS)               | Newton | Newton meter | Meters per second squared | Meters per second | Meters   | Kilogram | Kilogram meter squared |
| English (Velocity in ft/s) | Pound  | Foot pound   | Feet per second squared   | Feet per second   | Feet     | Slug     | Slug foot squared      |
| English (Velocity in kts)  | Pound  | Foot pound   | Feet per second squared   | Knots             | Feet     | Slug     | Slug foot squared      |

### Mass Type

Select the type of mass to use:

|                 |  |
|-----------------|--|
| Fixed           | Mass is constant throughout the simulation.                |
| Simple Variable | Mass and inertia vary linearly as a function of mass rate. |
| Custom Variable | Mass and inertia variations are customizable.              |

The Simple Variable selection conforms to the previously described equations of motion.

### Initial velocity

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

**Initial body attitude**

A scalar value for the initial pitch attitude of the body, ( $\theta_0$ ).

**Initial incidence**

A scalar value for the initial angle between the velocity vector and the body, ( $\alpha_0$ ).

**Initial body rotation rate**

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

**Initial position (x,z)**

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

**Initial mass**

A scalar value for the initial mass of the body.

**Inertia**

A scalar value for the inertia of the body.

**Empty mass**

A scalar value for the empty mass of the body.

**Full mass**

A scalar value for the full mass of the body.

**Empty inertia**

A scalar value for the empty inertia of the body.

**Full inertia**

A scalar value for the full inertia of the body.

**Gravity source**

Specify source of gravity:

|          |  |
|----------|--|
| External | Variable gravity input to block                                  |
| Internal | Constant gravity specified in <b>Acceleration due to gravity</b> |

**Acceleration due to gravity**

A scalar value for the acceleration due to gravity used if internal gravity source is selected. If gravity is to be neglected in the simulation, this value can be set to 0. This parameter appears if you set **Gravity source** to Internal.

**Include mass flow relative velocity**

Select this check box to add a mass flow relative velocity port. This is the relative velocity at which the mass is accreted or ablated.

**Include inertial acceleration**

Select this check box to enable an additional output port for the accelerations in body-fixed axes with respect to the inertial frame. You typically connect this signal to the accelerometer.

**State Attributes**

Assign unique name to each state. You can use state names instead of block paths during linearization.

- To assign a name to a single state, enter a unique name between quotes, for example, 'velocity'.
- To assign names to multiple states, enter a comma-delimited list surrounded by braces, for example, {'a', 'b', 'c'}. Each name must be unique.
- If a parameter is empty (' '), no name assignment occurs.
- The state names apply only to the selected block with the name parameter.
- The number of states must divide evenly among the number of state names.
- You can specify fewer names than states, but you cannot specify more names than states.

For example, you can specify two names in a system with four states. The first name applies to the first two states and the second name to the last two states.

- To assign state names with a variable in the MATLAB workspace, enter the variable without quotes. A variable can be a character vector, cell array, or structure.

**Velocity: e.g., {'u', 'w'}**

Specify velocity state names.

Default value is ''.

**Pitch attitude: e.g., 'theta'**

Specify pitch attitude state name.

Default value is ''.

**Position: e.g., {'Xe', 'Ze'}**

Specify position state names.

Default value is ''.

**Pitch angular rate: e.g., 'q'**

Specify pitch angular rate state name.

Default value is ''.

**Mass: e.g., 'mass'**

Specify mass state name.

Default value is ''.

## Inputs and Outputs

| Input            | Dimension Type | Description  |
|------------------|----------------|--|
| First            |                | Contains the force acting along the body x-axis, ( $F_x$ ).  |
| Second           |                | Contains the force acting along the body z-axis, ( $F_z$ ).  |
| Third            |                | Contains the applied pitch moment, ( $M$ ).  |
| Fourth           |                | Contains one or more rates of change of mass, ( $\dot{m}$ ) (positive if accreted, negative if ablated). |
| Fifth (Optional) |                | Contains the gravity in the selected units.  |

| Input                | Dimension Type     | Description   |
|----------------------|--------------------|---|
| Sixth<br>(Optional)  | Two-element vector | Contains one or more relative velocities at which the mass is accreted to or ablated from the body in body-fixed axes.  |
| Output               | Dimension Type     | Description   |
| First                |                    | Contains the pitch attitude, within $\pm\pi$ , in radians ( $\theta$ ).   |
| Second               |                    | Contains the pitch angular rate, in radians per second ( $q$ ).   |
| Third                |                    | Contains the pitch angular acceleration, in radians per second squared ( $\dot{q}$ ).   |
| Fourth               | Two-element vector | Contains the location of the body, in the flat Earth reference frame, ( $X_e, Z_e$ ).   |
| Fifth                | Two-element vector | Contains the velocity of the body resolved into the body-fixed coordinate frame, ( $u, w$ ).  |
| Sixth                | Two-element vector | Contains the acceleration of the body resolved into the body-fixed coordinate frame, ( $A_x, A_z$ ).  |
| Seventh              | Scalar element     | Contains a flag for fuel tank status, ( <i>Fuel</i> ): <ul style="list-style-type: none"> <li>• 1 indicates that the tank is full.</li> <li>• 0 indicates that the integral is neither full nor empty.</li> <li>• -1 indicates that the tank is empty.</li> </ul> |
| Eighth<br>(Optional) | Two-element vector | Contains the accelerations in body-fixed axes with respect to inertial frame (flat Earth). You typically connect this signal to the accelerometer.  |

## See Also

3DOF (Body Axes)

3DOF (Wind Axes)

Custom Variable Mass 3DOF (Body Axes)

Custom Variable Mass 3DOF (Wind Axes)

Simple Variable Mass 3DOF (Wind Axes)

## Extended Capabilities

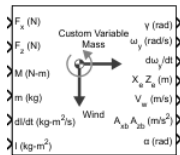
### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

Introduced in R2006a

## Simple Variable Mass 3DOF (Wind Axes)

Implement three-degrees-of-freedom equations of motion of simple variable mass with respect to wind axes

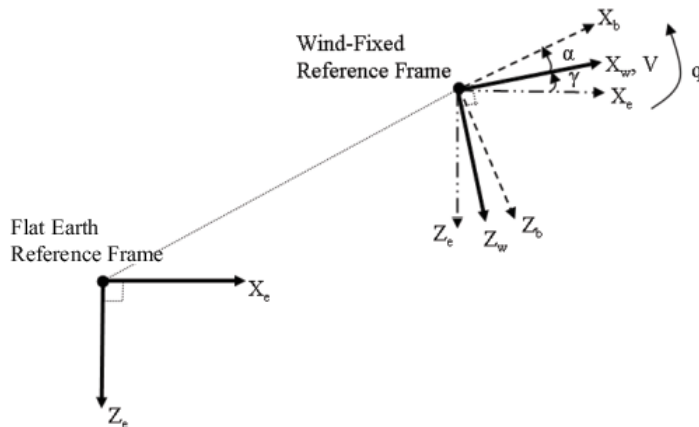


### Library

Equations of Motion/3DOF

### Description

The Simple Variable Mass 3DOF (Wind Axes) block considers the rotation in the vertical plane of a wind-fixed coordinate frame about a flat Earth reference frame.



The equations of motion are

$$\dot{V} = \frac{F_{x_{wind}}}{m} - \frac{\dot{m}Vre_{x_{wind}}}{m} - g\sin\gamma$$

$$A_{be} = \begin{bmatrix} A_{xe} \\ A_{ze} \end{bmatrix} = DCM_{wb} \left[ \frac{F_w - \dot{m}V_{re}}{m} - g \right]$$

$$A_{bb} = \begin{bmatrix} A_{xb} \\ A_{zb} \end{bmatrix} = DCM_{wb} \left[ \frac{F_w - \dot{m}V_{re}}{m} - g - \omega_w \times \bar{V}_w \right]$$

$$\dot{\alpha} = \frac{F_{z_{wind}}}{mV} + q + \frac{g}{V}\cos\gamma - \frac{\dot{m}Vre_{z_{wind}}}{mV}$$

$$\dot{q} = \dot{\theta} = \frac{M_{y_{body}} - \dot{I}_{yy}q}{I_{yy}}$$

$$\dot{y} = q - \dot{\alpha}$$

$$\dot{I}_{yy} = \frac{I_{yyfull} - I_{yyempty}}{m_{full} - m_{empty}}\dot{m}$$

where the applied forces are assumed to act at the center of gravity of the body.  $Vre_w$  is the relative velocity in the wind axes at which the mass flow ( $\dot{m}$ ) is ejected or added to the wind axes.

## Parameters

### Main

### Units

Specifies the input and output units:

| Units                      | Forces | Moment       | Acceleration              | Velocity          | Position | Mass     | Inertia                |
|----------------------------|--------|--------------|---------------------------|-------------------|----------|----------|------------------------|
| Metric (MKS)               | Newton | Newton meter | Meters per second squared | Meters per second | Meters   | Kilogram | Kilogram meter squared |
| English (Velocity in ft/s) | Pound  | Foot pound   | Feet per second squared   | Feet per second   | Feet     | Slug     | Slug foot squared      |
| English (Velocity in kts)  | Pound  | Foot pound   | Feet per second squared   | Knots             | Feet     | Slug     | Slug foot squared      |

### Mass Type

Select the type of mass to use:

|                 |  |
|-----------------|--|
| Fixed           | Mass is constant throughout the simulation.                |
| Simple Variable | Mass and inertia vary linearly as a function of mass rate. |
| Custom Variable | Mass and inertia variations are customizable.              |

The Simple Variable selection conforms to the previously described equations of motion.

### Initial airspeed

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

**Initial flight path angle**

A scalar value for the initial flight path angle of the body, ( $\gamma_0$ ).

**Initial incidence**

A scalar value for the initial angle between the velocity vector and the body, ( $\alpha_0$ ).

**Initial body rotation rate**

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

**Initial position (x,z)**

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

**Initial mass**

A scalar value for the initial mass of the body.

**Inertia body axes**

A scalar value for the inertia of the body.

**Empty mass**

A scalar value for the empty mass of the body.

**Full mass**

A scalar value for the full mass of the body.

**Empty inertia**

A scalar value for the empty inertia of the body.

**Full inertia**

A scalar value for the full inertia of the body.

**Gravity source**

Specify source of gravity:

|          |  |
|----------|--|
| External | Variable gravity input to block                                  |
| Internal | Constant gravity specified in <b>Acceleration due to gravity</b> |

**Acceleration due to gravity**

A scalar value for the acceleration due to gravity used if internal gravity source is selected. If gravity is to be neglected in the simulation, this value can be set to 0. This parameter appears if you set **Gravity source** to Internal.

**Include mass flow relative velocity**

Select this check box to add a mass flow relative velocity port. This is the relative velocity at which the mass is accreted or ablated.

**Include inertial acceleration**

Select this check box to enable an additional output port for the accelerations in body-fixed axes with respect to the inertial frame. You typically connect this signal to the accelerometer.

**State Attributes**

Assign unique name to each state. You can use state names instead of block paths during linearization.



- To assign a name to a single state, enter a unique name between quotes, for example, 'velocity'.
- To assign names to multiple states, enter a comma-delimited list surrounded by braces, for example, {'a', 'b', 'c'}. Each name must be unique.
- If a parameter is empty (' '), no name assignment occurs.
- The state names apply only to the selected block with the name parameter.
- The number of states must divide evenly among the number of state names.
- You can specify fewer names than states, but you cannot specify more names than states.

For example, you can specify two names in a system with four states. The first name applies to the first two states and the second name to the last two states.

- To assign state names with a variable in the MATLAB workspace, enter the variable without quotes. A variable can be a character vector, cell array, or structure.

**Velocity: e.g., 'V'**

Specify velocity state name.

Default value is ''.

**Incidence angle: e.g., 'alpha'**

Specify incidence angle state name.

Default value is ''.

**Flight path angle: e.g., 'gamma'**

Specify flight path angle state name.

Default value is ''.

**Body rotation rate: e.g., 'q'**

Specify body rotation rates state name.

Default value is ''.

**Position: e.g., {'Xe', 'Ze'}**

Specify position state names.

Default value is ''.

**Mass: e.g., 'mass'**

Specify mass state name.

Default value is ''.

## Inputs and Outputs

| Input  | Dimension Type | Description   |
|--------|----------------|---|
| First  |                | Contains the force acting along the wind x-axis, ( $F_x$ ). |
| Second |                | Contains the force acting along the wind z-axis, ( $F_z$ ). |
| Third  |                | Contains the applied pitch moment in body axes, ( $M$ ).    |

| Input            | Dimension Type     | Description  |
|------------------|--------------------|--|
| Fourth           |                    | Contains one or more rates of change of mass, ( $\dot{m}$ ) (positive if accreted, negative if ablated).         |
| Fifth (Optional) |                    | Contains the gravity in the selected units.  |
| Sixth (Optional) | Two-element vector | Contains one or more relative velocities at which the mass is accreted to or ablated from the body in wind axes. |

| Output           | Dimension Type     | Description   |
|------------------|--------------------|---|
| First            |                    | Contains the flight path angle, within $\pm\pi$ , in radians ( $\gamma$ ).  |
| Second           |                    | Contains the pitch angular rate, in radians per second ( $\omega_y$ ).  |
| Third            |                    | Contains the pitch angular acceleration, in radians per second squared ( $d\omega_y/dt$ ).  |
| Fourth           | Two-element vector | Contains the location of the body, in the flat Earth reference frame, ( $X_e, Z_e$ ).   |
| Fifth            | Two-element vector | Contains the velocity of the body resolved into the wind-fixed coordinate frame, ( $V, \theta$ ).   |
| Sixth            | Two-element vector | Contains the acceleration of the body resolved into the body-fixed coordinate frame, ( $A_x, A_z$ ).  |
| Seventh          | Scalar             | Contain the angle of attack, ( $\alpha_0$ ).  |
| Eight            | Scalar element     | Contains a flag for fuel tank status, ( <i>Fuel</i> ): <ul style="list-style-type: none"> <li>• 1 indicates that the tank is full.</li> <li>• 0 indicates that the integral is neither full nor empty.</li> <li>• -1 indicates that the tank is empty.</li> </ul> |
| Ninth (Optional) | Two-element vector | Contains the accelerations in body-fixed axes with respect to inertial frame (flat Earth). You typically connect this signal to the accelerometer.  |

## Reference

Stevens, B. L., and F. L. Lewis, *Aircraft Control and Simulation*, John Wiley & Sons, New York, 1992.

## See Also

3DOF (Body Axes)

3DOF (Wind Axes)

Custom Variable Mass 3DOF (Body Axes)

Custom Variable Mass 3DOF (Wind Axes)

Simple Variable Mass 3DOF (Body Axes)

## **Extended Capabilities**

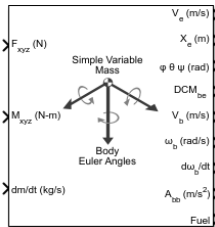
### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**Introduced in R2006a**

## Simple Variable Mass 6DOF (Euler Angles)

Implement Euler angle representation of six-degrees-of-freedom equations of motion of simple variable mass

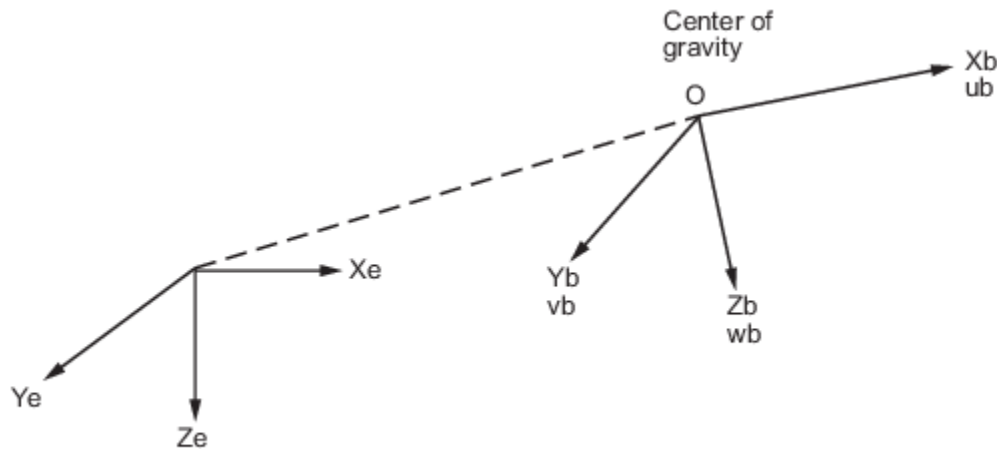


### Library

Equations of Motion/6DOF

### Description

The Simple Variable Mass 6DOF (Euler Angles) block considers the rotation of a body-fixed coordinate frame  $(X_b, Y_b, Z_b)$  about a flat Earth 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 flat Earth reference frame is considered inertial, an excellent approximation that allows the forces due to the Earth's motion relative to the fixed stars to be neglected.



Flat Earth 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.  $Vre_b$  is the relative velocity in the body axes at which the mass flow ( $\dot{m}$ ) is ejected or added to the body in body axes.

$$\begin{aligned}\bar{F}_b &= \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} = m(\dot{\bar{V}}_b + \bar{\omega} \times \bar{V}_b) + \dot{m}\bar{V}_b \\ A_{be} &= \frac{\bar{F}_b - \dot{m}\bar{V}_b}{m} \\ A_{bb} &= \begin{bmatrix} \dot{u}_b \\ \dot{v}_b \\ \dot{w}_b \end{bmatrix} = \frac{\bar{F}_b - \dot{m}\bar{V}_b}{m} - \bar{\omega} \times \bar{V}_b \\ \bar{V}_b &= \begin{bmatrix} u_b \\ v_b \\ w_b \end{bmatrix}, \bar{\omega} = \begin{bmatrix} p \\ q \\ r \end{bmatrix}\end{aligned}$$

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  $O$ .

$$\begin{aligned}\bar{M}_B &= \begin{bmatrix} L \\ M \\ N \end{bmatrix} = I\dot{\bar{\omega}} + \bar{\omega} \times (I\bar{\omega}) + \dot{I}\bar{\omega} \\ I &= \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{yx} & I_{yy} & -I_{yz} \\ -I_{zx} & -I_{zy} & I_{zz} \end{bmatrix}\end{aligned}$$

The inertia tensor is determined using a table lookup which linearly interpolates between  $I_{full}$  and  $I_{empty}$  based on mass ( $m$ ). While the rate of change of the inertia tensor is estimated by the following equation.

$$\dot{I} = \frac{I_{full} - I_{empty}}{m_{full} - m_{empty}} \dot{m}$$

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.

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

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

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

## Parameters

### Main

#### Units

Specifies the input and output units.

| Units                      | Forces | Moment       | Acceleration              | Velocity          | Position | Mass     | Inertia                |
|----------------------------|--------|--------------|---------------------------|-------------------|----------|----------|------------------------|
| Metric (MKS)               | Newton | Newton meter | Meters per second squared | Meters per second | Meters   | Kilogram | Kilogram meter squared |
| English (Velocity in ft/s) | Pound  | Foot pound   | Feet per second squared   | Feet per second   | Feet     | Slug     | Slug foot squared      |
| English (Velocity in kts)  | Pound  | Foot pound   | Feet per second squared   | Knots             | Feet     | Slug     | Slug foot squared      |

#### Mass Type

Select the type of mass to use.

|                 |  |
|-----------------|--|
| Fixed           | Mass is constant throughout the simulation.                |
| Simple Variable | Mass and inertia vary linearly as a function of mass rate. |
| Custom Variable | Mass and inertia variations are customizable.              |

The Simple Variable selection conforms to the previously described equations of motion.

#### Representation

Select the representation to use.

|              |  |
|--------------|--|
| Euler Angles | Use Euler angles within equations of motion. |
| Quaternion   | Use quaternions within equations of motion.  |

The Euler Angles selection conforms to the previously described equations of motion.

#### Initial position in inertial axes

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

#### Initial velocity in body axes

The three-element vector for the initial velocity in the body-fixed coordinate frame.

#### 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.

#### Initial mass

The initial mass of the rigid body.

**Empty mass**

A scalar value for the empty mass of the body.

**Full mass**

A scalar value for the full mass of the body.

**Empty inertia matrix**

A 3-by-3 inertia tensor matrix for the empty inertia of the body.

**Full inertia matrix**

A 3-by-3 inertia tensor matrix for the full inertia of the body.

**Include mass flow relative velocity**

Select this check box to add a mass flow relative velocity port. This is the relative velocity at which the mass is accreted or ablated.

**Include inertial acceleration**

Select this check box to enable an additional output port for the accelerations in body-fixed axes with respect to the inertial frame. You typically connect this signal to the accelerometer.

**State Attributes**

Assign unique name to each state. You can use state names instead of block paths during linearization.

- To assign a name to a single state, enter a unique name between quotes, for example, 'velocity'.
- To assign names to multiple states, enter a comma-delimited list surrounded by braces, for example, {'a', 'b', 'c'}. Each name must be unique.
- If a parameter is empty (' '), no name assignment occurs.
- The state names apply only to the selected block with the name parameter.
- The number of states must divide evenly among the number of state names.
- You can specify fewer names than states, but you cannot specify more names than states.

For example, you can specify two names in a system with four states. The first name applies to the first two states and the second name to the last two states.

- To assign state names with a variable in the MATLAB workspace, enter the variable without quotes. A variable can be a character vector, cell array, or structure.

**Position: e.g., {'Xe', 'Ye', 'Ze'}**

Specify position state names.

Default value is ''.

**Velocity: e.g., {'U', 'v', 'w'}**

Specify velocity state names.

Default value is ''.

**Euler rotation angles: e.g., {'phi', 'theta', 'psi'}**

Specify Euler rotation angle state names. This parameter appears if the **Representation** parameter is set to Euler Angles.

Default value is ''.

**Body rotation rates: e.g., {'p', 'q', 'r'}**

Specify body rotation rate state names.

Default value is ''.

**Mass: e.g., 'mass'**

Specify mass state name.

Default value is ''.

**Inputs and Outputs**

| Input                | Dimension Type          | Description  |
|----------------------|-------------------------|--|
| First                | Vector                  | Contains the three applied forces.   |
| Second               | Vector                  | Contains the three applied moments.  |
| Third                | Scalar                  | Contains one or more rates of change of mass.  |
| Fourth<br>(Optional) | Three-element<br>vector | Contains one or more relative velocities at which the mass is accreted to or ablated from the body in body-fixed axes. |

| Output              | Dimension Type       | Description  |
|---------------------|----------------------|--|
| First               | Three-element vector | Contains the velocity in the flat Earth reference frame.   |
| Second              | Three-element vector | Contains the position in the flat Earth reference frame.   |
| Third               | Three-element vector | Contains the Euler rotation angles [roll, pitch, yaw], within $\pm\pi$ , in radians.   |
| Fourth              | 3-by-3 matrix        | Applies to the coordinate transformation from flat Earth axes to body-fixed axes.  |
| Fifth               | Three-element vector | Contains the velocity in the body-fixed frame.   |
| Sixth               | Three-element vector | Contains the angular rates in body-fixed axes, in radians per second.  |
| Seventh             | Three-element vector | Contains the angular accelerations in body-fixed axes, in radians per second squared.  |
| Eight               | Three-element vector | Contains the accelerations in body-fixed axes with respect to body frame.  |
| Ninth               | Scalar element       | Contains a flag for fuel tank status: <ul style="list-style-type: none"> <li>• 1 indicates that the tank is full.</li> <li>• 0 indicates that the integral is neither full nor empty.</li> <li>• -1 indicates that the tank is empty.</li> </ul> |
| Tenth<br>(Optional) | Three-element vector | Contains the accelerations in body-fixed axes with respect to inertial frame (flat Earth). You typically connect this signal to the accelerometer.   |

**Assumptions and Limitations**

The block assumes that the applied forces are acting at the center of gravity of the body.



## Reference

Stevens, Brian, and Frank Lewis, *Aircraft Control and Simulation*. Second Edition. Hoboken, NJ: John Wiley & Sons, 2003.

Zipfel, Peter H., *Modeling and Simulation of Aerospace Vehicle Dynamics*. Second Edition. Reston, VA: AIAA Education Series, 2007.

## See Also

6DOF (Euler Angles)

6DOF (Quaternion)

6DOF ECEF (Quaternion)

6DOF Wind (Quaternion)

6DOF Wind (Wind Angles)

6th Order Point Mass (Coordinated Flight)

Custom Variable Mass 6DOF (Euler Angles)

Custom Variable Mass 6DOF (Quaternion)

Custom Variable Mass 6DOF ECEF (Quaternion)

Custom Variable Mass 6DOF Wind (Quaternion)

Custom Variable Mass 6DOF Wind (Wind Angles)

Simple Variable Mass 6DOF (Quaternion)

Simple Variable Mass 6DOF ECEF (Quaternion)

Simple Variable Mass 6DOF Wind (Quaternion)

Simple Variable Mass 6DOF Wind (Wind Angles)

“About Aerospace Coordinate Systems” on page 2-8

## Extended Capabilities

### C/C++ Code Generation

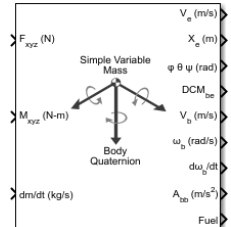
Generate C and C++ code using Simulink® Coder™.

### Introduced in R2006a

## Simple Variable Mass 6DOF (Quaternion)

Implement quaternion representation of six-degrees-of-freedom equations of motion of simple variable mass with respect to body axes

**Library:** Aerospace Blockset / Equations of Motion / 6DOF



### Description

The Simple Variable Mass 6DOF (Quaternion) implements a quaternion representation of six-degrees-of-freedom equations of motion of simple variable mass with respect to body axes.

For a description of the coordinate system and the translational dynamics, see the description for the Simple Variable Mass 6DOF (Euler Angles) block. Aerospace Blockset uses quaternions that are defined using the scalar-first convention. For more information on the integration of the rate of change of the quaternion vector, see “Algorithms” on page 4-564.

### Limitations

The block assumes that the applied forces are acting at the center of gravity of the body.

### Ports

#### Input

##### $F_{xyz}$ (N) — Applied forces

three-element vector

Applied forces, specified as three-element vector.

Data Types: double

##### $M_{xyz}$ (N-m) — Applied moments

three-element vector

Applied moments, specified as three-element vector.

Data Types: double

##### $dm/dt$ (kg/s) — Rate of change of mass

scalar

One or more rates of change of mass (positive if accreted, negative if ablated), specified as a scalar.

Data Types: double

**$V_{re}$  (m/s) — Relative velocity**

three-element vector

One or more relative velocities, specified as a three-element vector, at which the mass is accreted to or ablated from the body in body-fixed axes.

Data Types: double

**Output** **$V_e$  (m/s) — Velocity in flat Earth reference frame**

three-element vector

Velocity in the flat Earth reference frame, returned as a three-element vector.

Data Types: double

 **$X_e$  (m) — Position in flat Earth reference frame**

three-element vector

Position in the flat Earth reference frame, returned as a three-element vector.

Data Types: double

 **$\varphi$   $\theta$   $\psi$  (rad) — Euler rotation angles**

three-element vector

Euler rotation angles [roll, pitch, yaw], returned as three-element vector, in radians.

Data Types: double

 **$DCM_{be}$  — Coordinate transformation**

3-by-3 matrix

Coordinate transformation from flat Earth axes to body-fixed axes, returned as a 3-by-3 matrix.

Data Types: double

 **$V_b$  (m/s) — Velocity in body-fixed frame**

three-element vector

Velocity in body-fixed frame, returned as a three-element vector.

Data Types: double

 **$\omega_b$  (rad/s) — Angular rates in body-fixed axes**

three-element vector

Angular rates in body-fixed axes, returned as a three-element vector, in radians per second.

Data Types: double

 **$d\omega_b/dt$  — Angular accelerations**

three-element vector

Angular accelerations in body-fixed axes, returned as a three-element vector, in radians per second squared.

Data Types: double

**A<sub>bb</sub> m/s<sup>2</sup> — Accelerations in body-fixed axes**

three-element vector

Accelerations in body-fixed axes with respect to body frame, returned as a three-element vector.

Data Types: double

**Fuel — Fuel tank status**

scalar

Fuel tank status, returned as

- 1 — Tank is full.
- 0 — Tank is neither full nor empty.
- -1 — Tank is empty.

Data Types: double

**A<sub>be</sub> m/s<sup>2</sup> — Accelerations with respect to inertial frame**

three-element vector

Accelerations in body-fixed axes with respect to inertial frame (flat Earth), returned as a three-element vector. You typically connect this signal to the accelerometer.

**Dependencies**

This port appears only when the **Include inertial acceleration** check box is selected.

Data Types: double

**Parameters****Main****Units — Input and output units**

Metric (MKS) (default) | English (Velocity in ft/s) | English (Velocity in kts)

Input and output units, specified as Metric (MKS), English (Velocity in ft/s), or English (Velocity in kts).

| Units                      | Forces | Moment       | Acceleration              | Velocity          | Position | Mass     | Inertia                |
|----------------------------|--------|--------------|---------------------------|-------------------|----------|----------|------------------------|
| Metric (MKS)               | Newton | Newton meter | Meters per second squared | Meters per second | Meters   | Kilogram | Kilogram meter squared |
| English (Velocity in ft/s) | Pound  | Foot pound   | Feet per second squared   | Feet per second   | Feet     | Slug     | Slug foot squared      |
| English (Velocity in kts)  | Pound  | Foot pound   | Feet per second squared   | Knots             | Feet     | Slug     | Slug foot squared      |

**Programmatic Use****Block Parameter:** Units

**Type:** character vector

**Values:** Metric (MKS) | English (Velocity in ft/s) | English (Velocity in kts)

**Default:** Metric (MKS)

### Mass Type – Mass type

Fixed (default) | Simple Variable | Custom Variable

Mass type, specified according to the following table.

| Mass Type       | Description  | Default for  |
|-----------------|--|--|
| Fixed           | Mass is constant throughout the simulation.                | <ul style="list-style-type: none"> <li>6DOF (Quaternion)</li> <li>6DOF (Euler Angles)</li> </ul>   |
| Simple Variable | Mass and inertia vary linearly as a function of mass rate. | <ul style="list-style-type: none"> <li>Simple Variable Mass 6DOF (Euler Angles)</li> <li>Simple Variable Mass 6DOF (Quaternion)</li> </ul> |
| Custom Variable | Mass and inertia variations are customizable.              | <ul style="list-style-type: none"> <li>Custom Variable Mass 6DOF (Quaternion)</li> </ul>   |

The Simple Variable selection conforms to the previously described equations of motion.

### Programmatic Use

**Block Parameter:** mtype

**Type:** character vector

**Values:** Fixed | Simple Variable | Custom Variable

**Default:** Simple Variable

### Representation – Equations of motion representation

Quaternion | Euler Angles

Equations of motion representation, specified according to the following table.

| Representation | Description                                  | Default For   |
|----------------|--|---|
| Euler Angles   | Use Euler angles within equations of motion. | <ul style="list-style-type: none"> <li>6DOF (Euler Angles)</li> <li>Simple Variable Mass 6DOF (Euler Angles)</li> </ul>   |
| Quaternion     | Use quaternions within equations of motion.  | <ul style="list-style-type: none"> <li>6DOF (Quaternion)</li> <li>Custom Variable Mass 6DOF (Quaternion)</li> <li>Simple Variable Mass 6DOF (Quaternion)</li> </ul> |

The Quaternion selection conforms to the previously described equations of motion.

### Programmatic Use

**Block Parameter:** rep

**Type:** character vector

**Values:** Euler Angles | Quaternion

**Default:** Quaternion

**Initial position in inertial axes [Xe,Ye,Ze] — Position in inertial axes**`[0 0 0]` (default) | three-element vector

Initial location of the body in the flat Earth reference frame, specified as a three-element vector.

**Programmatic Use****Block Parameter:** `xme_0`**Type:** character vector**Values:** `[0 0 0]` | three-element vector**Default:** `[0 0 0]`**Initial velocity in body axes [U,v,w] — Velocity in body axes**`[0 0 0]` (default) | three-element vector

The three-element vector for the initial velocity in the body-fixed coordinate frame.

**Programmatic Use****Block Parameter:** `Vm_0`**Type:** character vector**Values:** `[0 0 0]` | three-element vector**Default:** `[0 0 0]`**Initial Euler orientation [roll, pitch, yaw] — Initial Euler orientation**`[0 0 0]` (default) | three-element vector

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

**Programmatic Use****Block Parameter:** `eul_0`**Type:** character vector**Values:** `[0 0 0]` | three-element vector**Default:** `[0 0 0]`**Initial body rotation rates [p,q,r] — Initial body rotation**`[0 0 0]` (default) | three-element vector

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

**Programmatic Use****Block Parameter:** `pm_0`**Type:** character vector**Values:** `[0 0 0]` | three-element vector**Default:** `[0 0 0]`**Initial mass — Initial mass**`1.0` (default) | scalar

The initial mass of the rigid body.

**Programmatic Use****Block Parameter:** `mass_0`**Type:** character vector**Values:** `1.0` | scalar**Default:** `1.0`**Inertia — Inertia**`eye(3)` (default) | scalar

A scalar value for the inertia of the body.

**Programmatic Use**

**Block Parameter:** `inertia`

**Type:** character vector

**Values:** `eye(3)` | scalar

**Default:** `eye(3)`

**Empty mass — Empty mass**

`0.5` (default) | scalar

A scalar value for the empty mass of the body.

**Programmatic Use**

**Block Parameter:** `mass_e`

**Type:** character vector

**Values:** `0.5` | scalar

**Default:** `0.5`

**Full mass — Full mass of body**

`2.0` (default) | scalar

A scalar value for the full mass of the body.

**Programmatic Use**

**Block Parameter:** `mass_f`

**Type:** character vector

**Values:** `2.0` | scalar

**Default:** `2.0`

**Empty inertia matrix — Empty inertia matrix**

`eye(3)` (default) | 3-by-3 matrix

A 3-by-3 inertia tensor matrix for the empty inertia of the body.

**Programmatic Use**

**Block Parameter:** `inertia_e`

**Type:** character vector

**Values:** `eye(3)` | scalar

**Default:** `eye(3)`

**Full inertia matrix — Full inertia of body**

`2*eye(3)` (default) | 3-by-3 matrix

A 3-by-3 inertia tensor matrix for the full inertia of the body.

**Programmatic Use**

**Block Parameter:** `inertia_f`

**Type:** character vector

**Values:** `2*eye(3)` | scalar

**Default:** `2*eye(3)`

**Gain for quaternion normalization — Gain**

`1.0` (default) | scalar

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

**Programmatic Use****Block Parameter:** `k_quat`**Type:** character vector**Values:** `1.0` | scalar**Default:** `1.0`**Include mass flow relative velocity — Mass flow relative velocity port**`off` (default) | `on`

Select this check box to add a mass flow relative velocity port. This is the relative velocity at which the mass is accreted or ablated.

**Programmatic Use****Block Parameter:** `vre_flag`**Type:** character vector**Values:** `off` | `on`**Default:** `off`**Include inertial acceleration — Include inertial acceleration port**`off` (default) | `on`

Select this check box to add an inertial acceleration port.

**Programmatic Use****Block Parameter:** `abi_flag`**Type:** character vector**Values:** `off` | `on`**Default:** `off`**State Attributes**

Assign unique name to each state. You can use state names instead of block paths during linearization.

- To assign a name to a single state, enter a unique name between quotes, for example, `'velocity'`.
- To assign names to multiple states, enter a comma-delimited list surrounded by braces, for example, `{'a', 'b', 'c'}`. Each name must be unique.
- If a parameter is empty (`' '`), no name assignment occurs.
- The state names apply only to the selected block with the name parameter.
- The number of states must divide evenly among the number of state names.
- You can specify fewer names than states, but you cannot specify more names than states.

For example, you can specify two names in a system with four states. The first name applies to the first two states and the second name to the last two states.

- To assign state names with a variable in the MATLAB workspace, enter the variable without quotes. A variable can be a character vector, cell array, or structure.

**Position: e.g., {'Xe', 'Ye', 'Ze'} — Position state name**`' '` (default) | comma-delimited list surrounded by braces

Specify position state names.



**Programmatic Use****Block Parameter:** xme\_statename**Type:** character vector**Values:** '' | comma-delimited list surrounded by braces**Default:** ''**Velocity: e.g., {'U', 'v', 'w'} – Velocity state name**

'' (default) | comma-delimited list surrounded by braces

Specify velocity state names.

**Programmatic Use****Block Parameter:** Vm\_statename**Type:** character vector**Values:** '' | comma-delimited list surrounded by braces**Default:** ''**Quaternion vector: e.g., {'qr', 'qi', 'qj', 'qk'} – Quaternion vector state name**

'' (default) | comma-delimited list surrounded by braces

Specify quaternion vector state names. This parameter appears if the **Representation** parameter is set to Quaternion.**Programmatic Use****Block Parameter:** quat\_statename**Type:** character vector**Values:** '' | comma-delimited list surrounded by braces**Default:** ''**Body rotation rates: e.g., {'p', 'q', 'r'} – Body rotation state name**

'' (default) | comma-delimited list surrounded by braces

Specify body rotation rate state names.

**Programmatic Use****Block Parameter:** pm\_statename**Type:** character vector**Values:** '' | comma-delimited list surrounded by braces**Default:** ''**Euler rotation angles: e.g., {'phi', 'theta', 'psi'} – Euler rotation state name**

'' (default) | comma-delimited list surrounded by braces

Specify Euler rotation angle state names.

**Programmatic Use****Block Parameter:** eul\_statename**Type:** character vector**Values:** '' | comma-delimited list surrounded by braces**Default:** ''**Mass: e.g., 'mass' – Mass state name**

'' (default) | comma-delimited list surrounded by braces

Specify mass state name.

**Programmatic Use****Block Parameter:** mass\_statename**Type:** character vector**Values:** ' ' | comma-delimited list surrounded by braces**Default:** ' '**Algorithms**

The equation of the integration of the rate of change of the quaternion vector follows. The gain  $K$  drives the norm of the quaternion state vector to 1.0 should  $\varepsilon$  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.

$$\begin{bmatrix} \dot{q}_0 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 0 & -p & -q & -r \\ p & 0 & r & -q \\ q & -r & 0 & p \\ r & q & -p & 0 \end{bmatrix} \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} + K\varepsilon \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix}$$

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

**References**

- [1] Stevens, Brian, and Frank Lewis, *Aircraft Control and Simulation*. Second Edition. Hoboken, NJ: John Wiley & Sons, 2003.
- [2] Zipfel, Peter H., *Modeling and Simulation of Aerospace Vehicle Dynamics*. Second Edition. Reston, VA: AIAA Education Series, 2007.

**Extended Capabilities****C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**See Also**

6DOF (Euler Angles) | 6DOF (Quaternion) | 6DOF ECEF (Quaternion) | 6DOF Wind (Quaternion) | 6DOF Wind (Wind Angles) | Custom Variable Mass 6DOF (Euler Angles) | Custom Variable Mass 6DOF (Quaternion) | Custom Variable Mass 6DOF ECEF (Quaternion) | Custom Variable Mass 6DOF Wind (Quaternion) | Custom Variable Mass 6DOF Wind (Wind Angles) | Simple Variable Mass 6DOF (Euler Angles) | Simple Variable Mass 6DOF ECEF (Quaternion) | Simple Variable Mass 6DOF Wind (Quaternion) | Simple Variable Mass 6DOF Wind (Wind Angles)

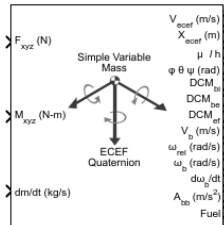
**Topics**

"About Aerospace Coordinate Systems" on page 2-8

**Introduced in R2006a**

# Simple Variable Mass 6DOF ECEF (Quaternion)

Implement quaternion representation of six-degrees-of-freedom equations of motion of simple variable mass in Earth-centered Earth-fixed (ECEF) coordinates

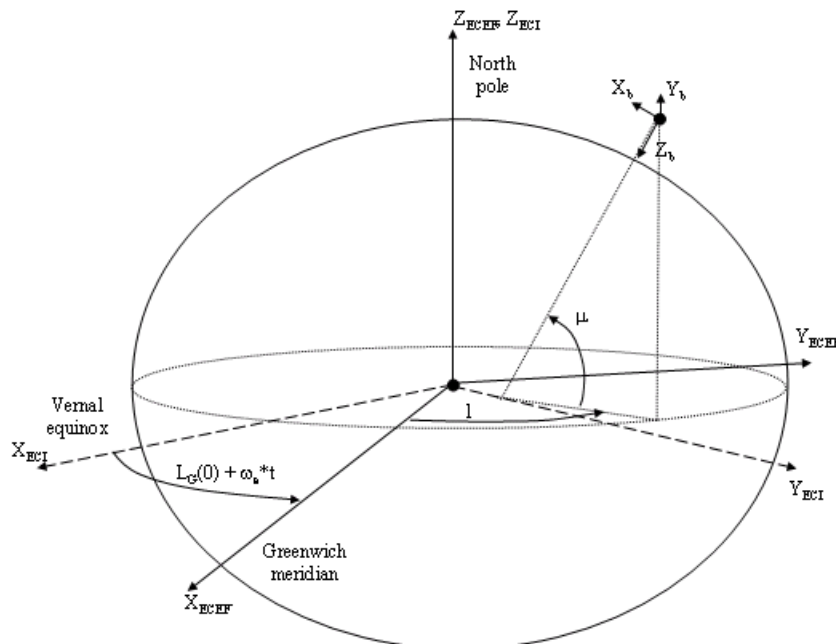


## Library

Equations of Motion/6DOF

## Description

The Simple Variable Mass 6DOF ECEF (Quaternion) block considers the rotation of a Earth-centered Earth-fixed (ECEF) coordinate frame ( $X_{ECEF}$ ,  $Y_{ECEF}$ ,  $Z_{ECEF}$ ) about an Earth-centered inertial (ECI) reference frame ( $X_{ECI}$ ,  $Y_{ECI}$ ,  $Z_{ECI}$ ). The origin of the ECEF coordinate frame is the center of the Earth, additionally the body of interest is assumed to be rigid, an assumption that eliminates the need to consider the forces acting between individual elements of mass. The representation of the rotation of ECEF frame from ECI frame is simplified to consider only the constant rotation of the ellipsoid Earth ( $\omega_e$ ) including an initial celestial longitude ( $L_G(0)$ ). This excellent approximation allows the forces due to the Earth's complex motion relative to the “fixed stars” to be neglected.



The translational motion of the ECEF coordinate frame is given below, where the applied forces  $[F_x F_y F_z]^T$  are in the body frame.  $V_{re_b}$  is the relative velocity in the wind axes at which the mass flow ( $\dot{m}$ ) is ejected or added to the body axes.

$$\begin{aligned} \bar{F}_b &= \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} = m \left( \dot{\bar{V}}_b + \bar{\omega}_b \times \bar{V}_b + DCM_{bf} \bar{\omega}_e \times \bar{V}_b + DCM_{bf} (\bar{\omega}_e \times (\bar{\omega}_e \times \bar{X}_f)) \right) \\ &\quad + \dot{m} (\bar{V}_{re_b} + DCM_{bf} (\bar{\omega}_e \times \bar{X}_f)) \\ A_{bb} &= \begin{bmatrix} \dot{u}_b \\ \dot{v}_b \\ \dot{w}_b \end{bmatrix} = \frac{\bar{F}_b - \dot{m} (\bar{V}_{re_b} + DCM_{bf} (\bar{\omega}_e \times \bar{X}_f))}{m} \\ &\quad - [\bar{\omega}_b \times \bar{V}_b + DCM_{bf} \bar{\omega}_e \times \bar{V}_b + DCM_{bf} (\bar{\omega}_e (\bar{\omega}_e \times \bar{X}_f))] \\ A_{becef} &= \frac{\bar{F}_b - \dot{m} (\bar{V}_{re_b} + DCM_{bf} (\bar{\omega}_e \times \bar{X}_f))}{m} \end{aligned}$$

where the change of position in ECEF  $\dot{\bar{x}}_f(\dot{\bar{x}}_i)$  is calculated by

$$\dot{\bar{x}}_f = DCM_{fb} \bar{V}_b$$

and the velocity of the body with respect to ECEF frame, expressed in body frame ( $\bar{V}_b$ ), angular rates of the body with respect to ECI frame, expressed in body frame ( $\bar{\omega}_b$ ). Earth rotation rate ( $\bar{\omega}_e$ ), and relative angular rates of the body with respect to north-east-down (NED) frame, expressed in body frame ( $\bar{\omega}_{rel}$ ) are defined as

$$\begin{aligned} \bar{V}_b &= \begin{bmatrix} u \\ v \\ w \end{bmatrix} \quad \bar{\omega}_{rel} = \begin{bmatrix} p \\ q \\ r \end{bmatrix} \quad \bar{\omega}_e = \begin{bmatrix} 0 \\ 0 \\ \omega_e \end{bmatrix} \\ \bar{\omega}_b &= \bar{\omega}_{rel} + DCM_{bf} \bar{\omega}_e + DCM_{be} \bar{\omega}_{ned} \\ \bar{\omega}_{ned} &= \begin{bmatrix} \dot{\mu} \cos \mu \\ -\dot{\mu} \\ -\dot{\mu} \sin \mu \end{bmatrix} = \begin{bmatrix} V_E / (N + h) \\ -V_N / (M + h) \\ V_E \tan \mu / (N + h) \end{bmatrix} \end{aligned}$$

The rotational dynamics of the body defined in 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.

$$\begin{aligned} \bar{M}_b &= \begin{bmatrix} L \\ M \\ N \end{bmatrix} = \bar{I} \dot{\bar{\omega}}_b + \bar{\omega}_b \times (\bar{I} \bar{\omega}_b) + \dot{\bar{I}} \bar{\omega}_b \\ I &= \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{yx} & I_{yy} & -I_{yz} \\ -I_{zx} & -I_{zy} & I_{zz} \end{bmatrix} \end{aligned}$$

The inertia tensor is determined using a table lookup which linearly interpolates between  $I_{full}$  and  $I_{empty}$  based on mass ( $m$ ). The rate of change of the inertia tensor is estimated by the following equation.

$$\dot{I} = \frac{I_{full} - I_{empty}}{m_{full} - m_{empty}} \dot{m}$$

The integration of the rate of change of the quaternion vector is given below.

$$\begin{bmatrix} \dot{q}_0 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} = -1/2 \begin{bmatrix} 0 & \omega_b(1) & \omega_b(2) & \omega_b(3) \\ -\omega_b(1) & 0 & -\omega_b(3) & \omega_b(2) \\ -\omega_b(2) & \omega_b(3) & 0 & -\omega_b(1) \\ -\omega_b(3) & -\omega_b(2) & \omega_b(1) & 0 \end{bmatrix} \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix}$$

Aerospace Blockset uses quaternions that are defined using the scalar-first convention.

## Parameters

### Main

### Units

Specifies the input and output units:

| Units                      | Forces | Moment       | Acceleration              | Velocity          | Position | Mass     | Inertia                |
|----------------------------|--------|--------------|---------------------------|-------------------|----------|----------|------------------------|
| Metric (MKS)               | Newton | Newton meter | Meters per second squared | Meters per second | Meters   | Kilogram | Kilogram meter squared |
| English (Velocity in ft/s) | Pound  | Foot pound   | Feet per second squared   | Feet per second   | Feet     | Slug     | Slug foot squared      |
| English (Velocity in kts)  | Pound  | Foot pound   | Feet per second squared   | Knots             | Feet     | Slug     | Slug foot squared      |

### Mass type

Select the type of mass to use:

|                 |   |
|-----------------|---|
| Fixed           | Mass is constant throughout the simulation (see 6DOF ECEF (Quaternion)).                        |
| Simple Variable | Mass and inertia vary linearly as a function of mass rate.                                      |
| Custom Variable | Mass and inertia variations are customizable (see Custom Variable Mass 6DOF ECEF (Quaternion)). |

The Simple Variable selection conforms to the previously described equations of motion.

**Initial position in geodetic latitude, longitude and altitude**

The three-element vector for the initial location of the body in the geodetic reference frame. Latitude and longitude values can be any value. However, latitude values of +90 and -90 may return unexpected values because of singularity at the poles.

**Initial velocity in body axes**

The three-element vector containing the initial velocity of the body with respect to the ECEF frame, expressed in the body frame.

**Initial Euler orientation**

The three-element vector containing the initial Euler rotation angles [roll, pitch, yaw], in radians. Euler rotation angles are those between the body and NED coordinate systems.

**Initial body rotation rates**

The three-element vector for the initial angular rates of the body with respect to the NED frame, expressed the body frame, in radians per second.

**Initial mass**

The mass of the rigid body.

**Inertia**

A scalar value for the inertia of the body.

**Empty mass**

A scalar value for the empty mass of the body.

**Full mass**

A scalar value for the full mass of the body.

**Empty inertia matrix**

A 3-by-3 inertia tensor matrix for the empty inertia of the body.

**Full inertia matrix**

A 3-by-3 inertia tensor matrix for the full inertia of the body.

**Include mass flow relative velocity**

Select this check box to add a mass flow relative velocity port. This is the relative velocity at which the mass is accreted or ablated.

**Include inertial acceleration**

Select this check box to enable an additional output port for the accelerations in body-fixed axes with respect to the inertial frame. You typically connect this signal to the accelerometer.

**Planet model**

Specifies the planet model to use: Custom or Earth (WGS84).

**Equatorial radius of planet**

Specifies the radius of the planet at its equator. The units of the equatorial radius parameter should be the same as the units for ECEF position. This option is only available when **Planet model** is set to Custom.

**Flattening**

Specifies the flattening of the planet. This option is only available when **Planet model** is set to Custom.

**Rotational rate**

Specifies the scalar rotational rate of the planet in rad/s. This option is only available when **Planet model** is set to Custom.

**Celestial longitude of Greenwich source**

Specifies the source of Greenwich meridian's initial celestial longitude:

|          |  |
|----------|--|
| Internal | Use celestial longitude value from <b>Celestial longitude of Greenwich</b> . |
| External | Use external input for celestial longitude value.                            |

**Celestial longitude of Greenwich**

The initial angle between Greenwich meridian and the x-axis of the ECI frame. This parameter appears if you set **Celestial longitude of Greenwich source** to Internal.

**Include inertial acceleration**

Select this check box to enable an additional output port for the accelerations in body-fixed axes with respect to the inertial frame. You typically connect this signal to the accelerometer.

**State Attributes**

Assign unique name to each state. You can use state names instead of block paths during linearization.

- To assign a name to a single state, enter a unique name between quotes, for example, 'velocity'.
- To assign names to multiple states, enter a comma-delimited list surrounded by braces, for example, {'a', 'b', 'c'}. Each name must be unique.
- If a parameter is empty (' '), no name assignment occurs.
- The state names apply only to the selected block with the name parameter.
- The number of states must divide evenly among the number of state names.
- You can specify fewer names than states, but you cannot specify more names than states.

For example, you can specify two names in a system with four states. The first name applies to the first two states and the second name to the last two states.

- To assign state names with a variable in the MATLAB workspace, enter the variable without quotes. A variable can be a character vector, cell array, or structure.

**Quaternion vector: e.g., {'qr', 'qi', 'qj', 'qk'}**

Specify quaternion vector state names.

Default value is ''.

**Body rotation rates: e.g., {'p', 'q', 'r'}**

Specify body rotation rate state names.

Default value is ''.

**Velocity: e.g., {'U', 'v', 'w'}**

Specify velocity state names.

Default value is ''.

**ECEF position: e.g., {'Xecef', 'Yecef', 'Zecef'}**

Specify the ECEF position state names.

Default value is ''.

**Inertial position: e.g., {'Xeci', 'Yeci', 'Zeci'}**

Specify the inertial position state names.

Default value is ''.

**Celestial longitude of Greenwich: e.g., 'LG'**

Specify the Celestial longitude of Greenwich state name.

Default value is ''.

**Mass: e.g., 'mass'**

Specify mass state name.

Default value is ''.

## Inputs and Outputs

| Input             | Dimension Type       | Description  |
|-------------------|----------------------|--|
| First             | Vector               | Contains the three applied forces in body-fixed axes.  |
| Second            | Vector               | Contains the three applied moments in body-fixed axes.   |
| Third             | Scalar               | Contains one or more rates of change of mass (positive if accreted, negative if ablated).                      |
| Fourth (Optional) | Three-element vector | Contains one or more relative velocities at which the mass is accreted to or ablated from the body-fixed axes. |

| Output | Dimension Type       | Description  |
|--------|----------------------|--|
| First  | Three-element vector | Contains the velocity of body respect to ECEF frame, expressed in ECEF frame.  |
| Second | Three-element vector | Contains the position in the ECEF reference frame.   |
| Third  | Three-element vector | Contains the position in geodetic latitude, longitude and altitude, in degrees, degrees and selected units of length respectively.         |
| Fourth | Three-element vector | Contains the body rotation angles [roll, pitch, yaw], in radians. Euler rotation angles are those between body and NED coordinate systems. |
| Fifth  | 3-by-3 matrix        | Applies to the coordinate transformation from ECI axes to body-fixed axes.   |
| Sixth  | 3-by-3 matrix        | Applies to the coordinate transformation from NED axes to body-fixed axes.   |



| Output                | Dimension Type       | Description  |
|-----------------------|----------------------|--|
| Seventh               | 3-by-3 matrix        | Applies to the coordinate transformation from ECEF axes to NED axes.   |
| Eighth                | Three-element vector | Contains the velocity of body with respect to ECEF frame, expressed in body frame.   |
| Ninth                 | Three-element vector | Contains the relative angular rates of body with respect to NED frame, expressed in body frame, in radians per second.   |
| Tenth                 | Three-element vector | Contains the angular rates of the body with respect to ECI frame, expressed in body frame, in radians per second.  |
| Eleventh              | Three-element vector | Contains the angular accelerations of the body with respect to ECI frame, expressed in body frame, in radians per second squared.  |
| Twelfth               | Three-element vector | Contains the accelerations in body-fixed axes with respect to body frame.  |
| Thirteenth            | Scalar               | Is an element containing a flag for fuel tank status: <ul style="list-style-type: none"> <li>• 1 indicates that the tank is full.</li> <li>• 0 indicates that the integral is neither full nor empty.</li> <li>• -1 indicates that the tank is empty.</li> </ul> |
| Fourteenth (Optional) | Three-element vector | Contains the accelerations in body-fixed axes with respect to ECEF frame.  |

## Assumptions and Limitations

This implementation assumes that the applied forces are acting at the center of gravity of the body.

This implementation generates a geodetic latitude that lies between  $\pm 90$  degrees, and longitude that lies between  $\pm 180$  degrees. Additionally, the MSL altitude is approximate.

The Earth is assumed to be ellipsoidal. By setting flattening to 0.0, a spherical planet can be achieved. The Earth's precession, nutation, and polar motion are neglected. The celestial longitude of Greenwich is Greenwich Mean Sidereal Time (GMST) and provides a rough approximation to the sidereal time.

The implementation of the ECEF coordinate system assumes that the origin is at the center of the planet, the x-axis intersects the Greenwich meridian and the equator, the z-axis is the mean spin axis of the planet, positive to the north, and the y-axis completes the right-hand system.

The implementation of the ECI coordinate system assumes that the origin is at the center of the planet, the x-axis is the continuation of the line from the center of the Earth toward the vernal equinox, the z-axis points in the direction of the mean equatorial plane's north pole, positive to the north, and the y-axis completes the right-hand system.

## References

Stevens, B. L., and F. L. Lewis, *Aircraft Control and Simulation, Second Edition*, John Wiley & Sons, New York, 2003.

McFarland, Richard E., *A Standard Kinematic Model for Flight simulation at NASA-Ames*, NASA CR-2497.

“Supplement to Department of Defense World Geodetic System 1984 Technical Report: Part I - Methods, Techniques and Data Used in WGS84 Development,” DMA TR8350.2-A.

## See Also

6DOF (Euler Angles)

6DOF (Quaternion)

6DOF ECEF (Quaternion)

6DOF Wind (Quaternion)

6DOF Wind (Wind Angles)

6th Order Point Mass (Coordinated Flight)

Custom Variable Mass 6DOF (Euler Angles)

Custom Variable Mass 6DOF (Quaternion)

Custom Variable Mass 6DOF ECEF (Quaternion)

Custom Variable Mass 6DOF Wind (Quaternion)

Custom Variable Mass 6DOF Wind (Wind Angles)

Simple Variable Mass 6DOF (Euler Angles)

Simple Variable Mass 6DOF (Quaternion)

Simple Variable Mass 6DOF Wind (Quaternion)

Simple Variable Mass 6DOF Wind (Wind Angles)

“About Aerospace Coordinate Systems” on page 2-8

## Extended Capabilities

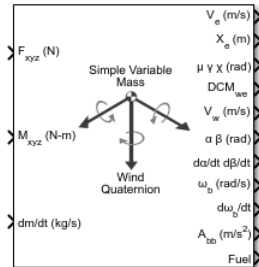
### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

### **Introduced in R2006a**

## Simple Variable Mass 6DOF Wind (Quaternion)

Implement quaternion representation of six-degrees-of-freedom equations of motion of simple variable mass with respect to wind axes

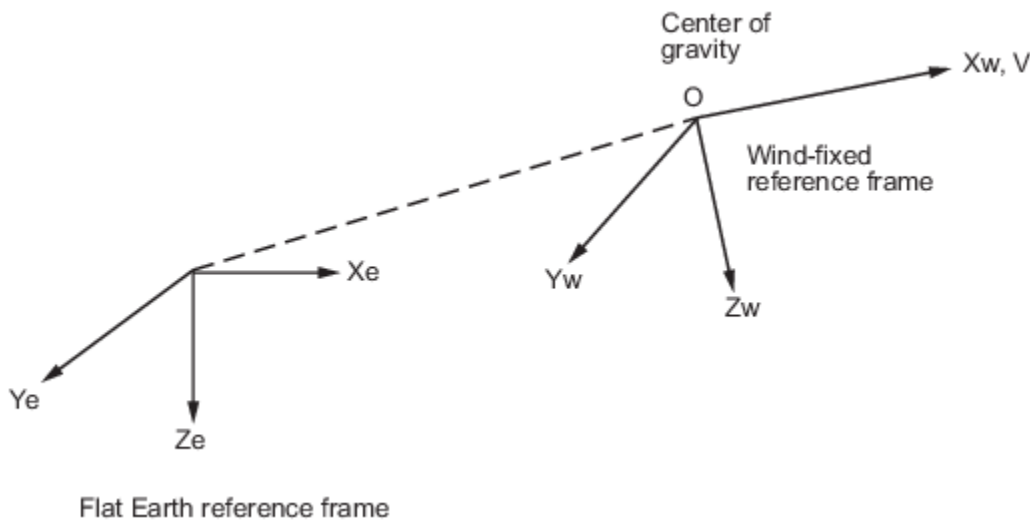


### Library

Equations of Motion/6DOF

### Description

The Simple Variable Mass 6DOF Wind (Quaternion) block considers the rotation of a wind-fixed coordinate frame ( $X_w, Y_w, Z_w$ ) about an flat Earth reference frame ( $X_e, Y_e, Z_e$ ). The origin of the wind-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 flat Earth reference frame is considered inertial, an excellent approximation that allows the forces due to the Earth's motion relative to the "fixed stars" to be neglected.



The translational motion of the wind-fixed coordinate frame is given below, where the applied forces  $[F_x F_y F_z]^T$  are in the wind-fixed frame.  $V_{re_w}$  is the relative velocity in the wind axes at which the mass flow ( $\dot{m}$ ) is ejected or added to the body.

$$\vec{F}_w = \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} = m(\dot{\vec{V}}_w + \bar{\omega}_w \times \vec{V}_w) + \dot{m}\vec{V}re_w$$

$$\vec{V}_w = \begin{bmatrix} V \\ 0 \\ 0 \end{bmatrix}, \bar{\omega}_w = \begin{bmatrix} p_w \\ q_w \\ r_w \end{bmatrix} = DMC_{wb} \begin{bmatrix} p_b - \dot{\beta} \sin \alpha \\ q_b - \dot{\alpha} \\ r_b + \dot{\beta} \cos \alpha \end{bmatrix}, \bar{w}_b = \begin{bmatrix} p_b \\ q_b \\ r_b \end{bmatrix}$$

The rotational dynamics of the body-fixed frame are given below, where the applied moments are [L MN]<sup>T</sup>, and the inertia tensor *I* is with respect to the origin O. Inertia tensor *I* is much easier to define in body-fixed frame.

$$\vec{M}_b = \begin{bmatrix} L \\ M \\ N \end{bmatrix} = I\dot{\bar{\omega}}_b + \bar{\omega}_b \times (I\bar{\omega}_b) + \dot{I}\bar{\omega}_b$$

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

The inertia tensor is determined using a table lookup which linearly interpolates between *I<sub>full</sub>* and *I<sub>empty</sub>* based on mass (*m*). While the rate of change of the inertia tensor is estimated by the following equation.

$$\dot{I} = \frac{I_{full} - I_{empty}}{m_{full} - m_{empty}} \dot{m}$$

The integration of the rate of change of the quaternion vector is given below.

$$\begin{bmatrix} \dot{q}_0 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} = -1/2 \begin{bmatrix} 0 & p & q & r \\ -p & 0 & -r & q \\ -q & r & 0 & -p \\ -r & -q & p & 0 \end{bmatrix} \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix}$$

Aerospace Blockset uses quaternions that are defined using the scalar-first convention.

## Parameters

### Main

### Units

Specifies the input and output units:

| Units        | Forces | Moment       | Acceleration              | Velocity          | Position | Mass     | Inertia                |
|--------------|--------|--------------|---------------------------|-------------------|----------|----------|------------------------|
| Metric (MKS) | Newton | Newton meter | Meters per second squared | Meters per second | Meters   | Kilogram | Kilogram meter squared |

| Units                            | Forces | Moment     | Acceleration               | Velocity           | Position | Mass | Inertia              |
|----------------------------------|--------|------------|----------------------------|--------------------|----------|------|----------------------|
| English<br>(Velocity<br>in ft/s) | Pound  | Foot pound | Feet per second<br>squared | Feet per<br>second | Feet     | Slug | Slug foot<br>squared |
| English<br>(Velocity<br>in kts)  | Pound  | Foot pound | Feet per second<br>squared | Knots              | Feet     | Slug | Slug foot<br>squared |

**Mass Type**

Select the type of mass to use:

|                 |  |
|-----------------|--|
| Fixed           | Mass is constant throughout the simulation.                |
| Simple Variable | Mass and inertia vary linearly as a function of mass rate. |
| Custom Variable | Mass and inertia variations are customizable.              |

The Simple Variable selection conforms to the previously described equations of motion.

**Representation**

Select the representation to use:

|             |   |
|-------------|---|
| Wind Angles | Use wind angles within equations of motion. |
| Quaternion  | Use quaternions within equations of motion. |

The Quaternion selection conforms to the previously described equations of motion.

**Initial position in inertial axes**

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

**Initial airspeed, sideslip angle, and angle of attack**

The three-element vector containing the initial airspeed, initial sideslip angle and initial angle of attack.

**Initial wind orientation**

The three-element vector containing the initial wind angles [bank, flight path, and heading], in radians.

**Initial body rotation rates**

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

**Initial mass**

The initial mass of the rigid body.

**Inertia in body axis**

A scalar value for the inertia of the body.

**Empty mass**

A scalar value for the empty mass of the body.

**Full mass**

A scalar value for the full mass of the body.

**Empty inertia matrix**

A 3-by-3 inertia tensor matrix for the empty inertia of the body, in body-fixed axes.

**Full inertia matrix**

A 3-by-3 inertia tensor matrix for the full inertia of the body, in body-fixed axes.

**Include mass flow relative velocity**

Select this check box to add a mass flow relative velocity port. This is the relative velocity at which the mass is accreted or ablated.

**Include inertial acceleration**

Select this check box to enable an additional output port for the accelerations in body-fixed axes with respect to the inertial frame. You typically connect this signal to the accelerometer.

**State Attributes**

Assign unique name to each state. You can use state names instead of block paths during linearization.

- To assign a name to a single state, enter a unique name between quotes, for example, 'velocity'.
- To assign names to multiple states, enter a comma-delimited list surrounded by braces, for example, {'a', 'b', 'c'}. Each name must be unique.
- If a parameter is empty (' '), no name assignment occurs.
- The state names apply only to the selected block with the name parameter.
- The number of states must divide evenly among the number of state names.
- You can specify fewer names than states, but you cannot specify more names than states.

For example, you can specify two names in a system with four states. The first name applies to the first two states and the second name to the last two states.

- To assign state names with a variable in the MATLAB workspace, enter the variable without quotes. A variable can be a character vector, cell array, or structure.

**Position: e.g., {'Xe', 'Ye', 'Ze'}**

Specify position state names.

Default value is ''.

**Velocity: e.g., 'V'**

Specify velocity state name.

Default value is ''.

**Incidence angle: e.g., 'alpha'**

Specify incidence angle state name.

Default value is ''.

**Sideslip angle: e.g., 'beta'**

Specify sideslip angle state name.

Default value is ''.

**Wind orientation: e.g., {'mu', 'gamma', 'chi'}**

Specify wind orientation state names. This parameter appears if the **Representation** parameter is set to Wind Angles.

Default value is ''.

**Quaternion vector: e.g., {'qr', 'qi', 'qj', 'qk'}**

Specify quaternion vector state names. This parameter appears if the **Representation** parameter is set to **Quaternion**.

Default value is ''.

**Body rotation rates: e.g., {'p', 'q', 'r'}**

Specify body rotation rate state names.

Default value is ''.

**Mass: e.g., 'mass'**

Specify mass state name.

Default value is ''.

## Inputs and Outputs

| Input             | Dimension Type   | Description  |
|-------------------|------------------|--|
| First             | Vector           | Contains the three applied forces in wind-fixed axes.  |
| Second            | Vector           | Contains the three applied moments in body-fixed axes.   |
| Third             | Scalar or vector | Contains one or more rates of change of mass.  |
| Fourth (Optional) | Vector           | Contains one or more relative velocities at which the mass is accreted to or ablated from the body in wind axes. |

| Output  | Dimension Type       | Description   |
|---------|----------------------|---|
| First   | Three-element vector | Contains the velocity in the flat Earth reference frame.  |
| Second  | Three-element vector | Contains the position in the flat Earth reference frame.  |
| Third   | Three-element vector | Contains the wind rotation angles [bank, flight path, heading], in radians.                                 |
| Fourth  | 3-by-3 matrix        | Applies to the coordinate transformation from flat Earth axes to wind-fixed axes.                           |
| Fifth   | Three-element vector | Contains the velocity in the wind-fixed frame.  |
| Sixth   | Two-element vector   | Contains the angle of attack and sideslip angle, in radians.  |
| Seventh | Two-element vector   | Contains the rate of change of angle of attack and rate of change of sideslip angle, in radians per second. |
| Eighth  | Three-element vector | Contains the angular rates in body-fixed axes, in radians per second.                                       |
| Ninth   | Three-element vector | Contains the angular accelerations in body-fixed axes, in radians per second squared.                       |
| Tenth   | Three-element vector | Contains the accelerations in body-fixed axes with respect to body frame.                                   |

| Output   | Dimension Type       | Description  |
|----------|----------------------|--|
| Eleventh | Scalar element       | Contains a flag for fuel tank status: <ul style="list-style-type: none"> <li>• 1 indicates that the tank is full.</li> <li>• 0 indicates that the integral is neither full nor empty.</li> <li>• -1 indicates that the tank is empty.</li> </ul> |
| Twelfth  | Three-element vector | Contains the accelerations in body-fixed axes with respect to inertial frame (flat Earth). You typically connect this signal to the accelerometer.   |

## Assumptions and Limitations

The block assumes that the applied forces are acting at the center of gravity of the body.

## References

Stevens, Brian, and Frank Lewis, *Aircraft Control and Simulation*, Second Edition, John Wiley & Sons, 2003.

Zipfel, Peter H., *Modeling and Simulation of Aerospace Vehicle Dynamics*. Second Edition, AIAA Education Series, 2007.

## See Also

6DOF (Euler Angles)

6DOF (Quaternion)

6DOF ECEF (Quaternion)

6DOF Wind (Quaternion)

6DOF Wind (Wind Angles)

6th Order Point Mass (Coordinated Flight)

Custom Variable Mass 6DOF (Euler Angles)

Custom Variable Mass 6DOF (Quaternion)

Custom Variable Mass 6DOF ECEF (Quaternion)

Custom Variable Mass 6DOF Wind (Quaternion)

Custom Variable Mass 6DOF Wind (Wind Angles)

Simple Variable Mass 6DOF (Euler Angles)

Simple Variable Mass 6DOF (Quaternion)

Simple Variable Mass 6DOF ECEF (Quaternion)



Simple Variable Mass 6DOF Wind (Wind Angles)

“About Aerospace Coordinate Systems” on page 2-8

## **Extended Capabilities**

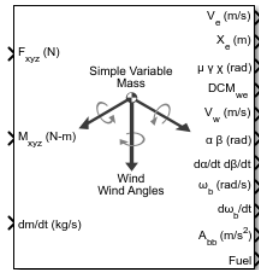
### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**Introduced in R2006a**

## Simple Variable Mass 6DOF Wind (Wind Angles)

Implement wind angle representation of six-degrees-of-freedom equations of motion of simple variable mass



### Library

Equations of Motion/6DOF

### Description

For a description of the coordinate system employed and the translational dynamics, see the block description for the Simple Variable Mass 6DOF (Quaternion) block.

The relationship between the wind angles,  $[\mu\gamma\chi]^T$ , can be determined by resolving the wind rates into the wind-fixed coordinate frame.

$$\begin{bmatrix} p_w \\ q_w \\ r_w \end{bmatrix} = \begin{bmatrix} \dot{\mu} \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\mu & \sin\mu \\ 0 & -\sin\mu & \cos\mu \end{bmatrix} \begin{bmatrix} 0 \\ \dot{\gamma} \\ 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\mu & \sin\mu \\ 0 & -\sin\mu & \cos\mu \end{bmatrix} \begin{bmatrix} \cos\gamma & 0 & -\sin\gamma \\ 0 & 1 & 0 \\ \sin\gamma & 0 & \cos\gamma \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ \dot{\chi} \end{bmatrix} \equiv J^{-1} \begin{bmatrix} \dot{\mu} \\ \dot{\gamma} \\ \dot{\chi} \end{bmatrix}$$

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

$$\begin{bmatrix} \dot{\mu} \\ \dot{\gamma} \\ \dot{\chi} \end{bmatrix} = J \begin{bmatrix} p_w \\ q_w \\ r_w \end{bmatrix} = \begin{bmatrix} 1 & (\sin\mu\tan\gamma) & (\cos\mu\tan\gamma) \\ 0 & \cos\mu & -\sin\mu \\ 0 & \frac{\sin\mu}{\cos\gamma} & \frac{\cos\mu}{\cos\gamma} \end{bmatrix} \begin{bmatrix} p_w \\ q_w \\ r_w \end{bmatrix}$$

The body-fixed angular rates are related to the wind-fixed angular rate by the following equation.

$$\begin{bmatrix} p_w \\ q_w \\ r_w \end{bmatrix} = DMC_{wb} \begin{bmatrix} p_b - \dot{\beta}\sin\alpha \\ q_b - \dot{\alpha} \\ r_b + \dot{\beta}\cos\alpha \end{bmatrix}$$

Using this relationship in the wind rate vector equations, gives the relationship between the wind rate vector and the body-fixed angular rates.

$$\begin{bmatrix} \dot{\mu} \\ \dot{\gamma} \\ \dot{\chi} \end{bmatrix} = J \begin{bmatrix} p_w \\ q_w \\ r_w \end{bmatrix} = \begin{bmatrix} 1 (\sin\mu\tan\gamma) (\cos\mu\tan\gamma) \\ 0 \cos\mu & -\sin\mu \\ 0 \frac{\sin\mu}{\cos\gamma} & \frac{\cos\mu}{\cos\gamma} \end{bmatrix} DMC_{wb} \begin{bmatrix} p_b - \dot{\beta}\sin\alpha \\ q_b - \dot{\alpha} \\ r_b + \dot{\beta}\cos\alpha \end{bmatrix}$$

## Parameters

### Main

#### Units

Specifies the input and output units:

| Units                      | Forces | Moment       | Acceleration              | Velocity          | Position | Mass     | Inertia                |
|----------------------------|--------|--------------|---------------------------|-------------------|----------|----------|------------------------|
| Metric (MKS)               | Newton | Newton meter | Meters per second squared | Meters per second | Meters   | Kilogram | Kilogram meter squared |
| English (Velocity in ft/s) | Pound  | Foot pound   | Feet per second squared   | Feet per second   | Feet     | Slug     | Slug foot squared      |
| English (Velocity in kts)  | Pound  | Foot pound   | Feet per second squared   | Knots             | Feet     | Slug     | Slug foot squared      |

#### Mass Type

Select the type of mass to use:

|                 |  |
|-----------------|--|
| Fixed           | Mass is constant throughout the simulation.                |
| Simple Variable | Mass and inertia vary linearly as a function of mass rate. |
| Custom Variable | Mass and inertia variations are customizable.              |

The Simple Variable selection conforms to the previously described equations of motion.

#### Representation

Select the representation to use:

|             |   |
|-------------|---|
| Wind Angles | Use wind angles within equations of motion. |
| Quaternion  | Use quaternions within equations of motion. |

The Wind Angles selection conforms to the previously described equations of motion.

#### Initial position in inertial axes

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

#### Initial airspeed, sideslip angle, and angle of attack

The three-element vector containing the initial airspeed, initial sideslip angle and initial angle of attack.

#### Initial wind orientation

The three-element vector containing the initial wind angles [bank, flight path, and heading], in radians.

**Initial body rotation rates**

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

**Initial mass**

The initial mass of the rigid body.

**Empty mass**

A scalar value for the empty mass of the body.

**Full mass**

A scalar value for the full mass of the body.

**Empty inertia matrix**

A 3-by-3 inertia tensor matrix for the empty inertia of the body, in body-fixed axes.

**Full inertia matrix**

A 3-by-3 inertia tensor matrix for the full inertia of the body, in body-fixed axes.

**Include mass flow relative velocity**

Select this check box to add a mass flow relative velocity port. This is the relative velocity at which the mass is accreted or ablated.

**Include inertial acceleration**

Select this check box to enable an additional output port for the accelerations in body-fixed axes with respect to the inertial frame. You typically connect this signal to the accelerometer.

**State Attributes**

Assign unique name to each state. You can use state names instead of block paths during linearization.

- To assign a name to a single state, enter a unique name between quotes, for example, 'velocity'.
- To assign names to multiple states, enter a comma-delimited list surrounded by braces, for example, {'a', 'b', 'c'}. Each name must be unique.
- If a parameter is empty (' '), no name assignment occurs.
- The state names apply only to the selected block with the name parameter.
- The number of states must divide evenly among the number of state names.
- You can specify fewer names than states, but you cannot specify more names than states.

For example, you can specify two names in a system with four states. The first name applies to the first two states and the second name to the last two states.

- To assign state names with a variable in the MATLAB workspace, enter the variable without quotes. A variable can be a character vector, cell array, or structure.

**Position: e.g., {'Xe', 'Ye', 'Ze'}**

Specify position state names.

Default value is ''.

**Velocity: e.g., 'V'**

Specify velocity state name.

Default value is ''.

**Incidence angle: e.g., 'alpha'**

Specify incidence angle state name.

Default value is ''.

**Sideslip angle: e.g., 'beta'**

Specify sideslip angle state name.

Default value is ''.

**Wind orientation: e.g., {'mu', 'gamma', 'chi'}**

Specify wind orientation state names. This parameter appears if the **Representation** parameter is set to `Wind Angles`.

Default value is ''.

**Body rotation rates: e.g., {'p', 'q', 'r'}**

Specify body rotation rates state names.

Default value is ''.

**Mass: e.g., 'mass'**

Specify mass state name.

Default value is ''.

**Inputs and Outputs**

| Input             | Dimension Type       | Description  |
|-------------------|----------------------|--|
| First             | Vector               | Contains the three applied forces in wind-fixed axes.  |
| Second            | Vector               | Contains the three applied moments in body-fixed axes.   |
| Third             | Scalar or vector     | Contains one or more rates of change of mass. This value is positive if the mass is added (accreted) to the body in wind axes. It is negative if the mass is ejected (ablated) from the body in wind axes. |
| Fourth (Optional) | Three-element vector | Contains one or more relative velocities at which the mass is accreted to or ablated from the body in wind axes.   |
| Output            | Dimension Type       | Description  |
| First             | Three-element vector | Contains the velocity in the fixed Earth reference frame.  |
| Second            | Three-element vector | Contains the position in the flat Earth reference frame.   |
| Third             | Three-element vector | Contains the wind rotation angles [bank, flight path, heading], within $\pm\pi$ , in radians.  |
| Fourth            | 3-by-3 matrix        | Applies to the coordinate transformation from flat Earth axes to wind-fixed axes.  |
| Fifth             | Three-element vector | Contains the velocity in the wind-fixed frame.   |

| Output           | Dimension Type       | Description  |
|------------------|----------------------|--|
| Sixth            | Two-element vector   | Contains the angle of attack and sideslip angle, in radians.   |
| Seventh          | Two-element vector   | Contains the rate of change of angle of attack and rate of change of sideslip angle, in radians per second.  |
| Eighth           | Three-element vector | Contains the angular rates in body-fixed axes, in radians per second.  |
| Ninth            | Three-element vector | Contain the angular accelerations in body-fixed axes, in radians per second squared.   |
| Tenth            | Three-element vector | Contains the accelerations in body-fixed axes with respect to body frame.  |
| Eleventh         | Scalar element       | Contains a flag for fuel tank status: <ul style="list-style-type: none"> <li>• 1 indicates that the tank is full.</li> <li>• 0 indicates that the integral is neither full nor empty.</li> <li>• -1 indicates that the tank is empty.</li> </ul> |
| Twelfth (Output) | Three-element vector | Contains the accelerations in body-fixed axes with respect to inertial frame (flat Earth). You typically connect this signal to the accelerometer.   |

## Assumptions and Limitations

The block assumes that the applied forces are acting at the center of gravity of the body.

## References

Stevens, Brian, and Frank Lewis, *Aircraft Control and Simulation*, Second Edition, John Wiley & Sons, 2003.

Zipfel, Peter H., *Modeling and Simulation of Aerospace Vehicle Dynamics*. Second Edition, AIAA Education Series, 2007.

## See Also

6DOF (Euler Angles)

6DOF (Quaternion)

6DOF ECEF (Quaternion)

6DOF Wind (Quaternion)

6DOF Wind (Wind Angles)

6th Order Point Mass (Coordinated Flight)

Custom Variable Mass 6DOF (Euler Angles)

Custom Variable Mass 6DOF (Quaternion)

Custom Variable Mass 6DOF ECEF (Quaternion)

Custom Variable Mass 6DOF Wind (Quaternion)

Custom Variable Mass 6DOF Wind (Wind Angles)

Simple Variable Mass 6DOF (Euler Angles)

Simple Variable Mass 6DOF (Quaternion)

Simple Variable Mass 6DOF ECEF (Quaternion)

Simple Variable Mass 6DOF Wind (Quaternion)

“About Aerospace Coordinate Systems” on page 2-8

## **Extended Capabilities**

### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**Introduced in R2006a**

## Simulation Pace

Set simulation rate for animation viewing

**Library:** Aerospace Blockset / Animation / Animation Support Utilities



### Description

The Simulation Pace block lets you run model simulation at a slower pace so that you can comfortably view connected animations and understand and observe the system behavior. Visualizing simulations at a slower rate makes it easier to understand underlying system design, identify design issues and demonstrate near real-time behavior. You can view the results and inspect your system while the simulation is in progress.

Use this block in scenarios where one simulation-second is completed in a few wall clock time milliseconds.

When configuring this block, also consider the block sample time, which affects the simulation pace. The default is 1/30th of a second, which corresponds to a 30 frames-per-second visualization rate (typical for desktop computers). For more information, see “Sample time” on page 4-0 .

To use this block:

- Set the model solver to **Fixed-step**.
- Use a discrete sample time.

This block does not produce deployable code.

### Ports

#### Output

##### Port\_1 — Pace error

scalar

Pace error, specified as a scalar.

The block optionally outputs the pace error value (*simulationTime* minus *ClockTime*), in seconds. The pace error is positive if the simulation is running faster than the specified pace and negative if slower than the specified pace.

Outputting the pace error from the block lets you record the overall pace achieved during the simulation or routing the signal to other blocks to determine if the simulation is too slow to keep up with the specified pace.

#### Dependencies

To enable this port, select the **Output pace error (sec)** check box.

Data Types: double



## Parameters

### Simulation pace — Ratio of simulation time to clock time

1 (default) | scalar

Ratio of simulation time to clock time, specified as a scalar, in seconds of simulation time per second of clock time.

#### Programmatic Use

**Block Parameter:** OutputPaceError

**Type:** character vector

**Values:** '1' | scalar

**Default:** '1'

### Sleep mode — Control simulation pace

Auto (default) | MATLAB Thread | Off | Busy-Wait

Control simulation pace of model using one of these methods. MATLAB Thread, Busy-Wait, and Auto slow down the simulation pace at simulation-second 0.1 to wait for the wall clock to get to time 1. Use this parameter when one simulation-second is completed in a few wall clock time milliseconds.

- Auto — Use the model configuration parameter setting **Enable pacing to slow down simulation** to control the simulation pace. If the model configuration parameter setting **Enable pacing to slow down simulation** is not selected, the block behaves as though the MATLAB Thread option is selected.
- MATLAB Thread — Use the operating system `sleep` function during simulation to wait for the wall clock to get to time 1.
- Off — Disable the pace functionality and let the simulation run as fast as possible.
- Busy-Wait — Use a while loop in conjunction with the Simstruct to wait for the simulation to wait for the wall clock to get to time 1.

#### Programmatic Use

**Block Parameter:** SleepMode

**Type:** character vector

**Values:** 'MATLAB Thread' | 'Off' | 'Busy-Wait' | 'Auto'

**Default:** 'Auto'

### Output pace error — Display pace error

off (default) | on

Select this check box to output the pace error value (*simulationTime* minus *ClockTime*), in seconds. The pace error is positive if the simulation is running faster than the specified pace and negative if slower than the specified pace. To disable the display, clear this check box .

#### Programmatic Use

**Block Parameter:** OutputPaceError

**Type:** character vector

**Values:** 'off' | 'on'

**Default:** 'off'

### Sample time — Sample time

1/30 (default) | -1 | scalar | vector

Specify the sample time as a scalar. The default 1/30th of a second corresponds to a 30 frames-per-second visualization rate (typical for desktop computers). To set how often the Simulink interface synchronizes with the wall clock, use this parameter.

The block sample time must be:

- Discrete
- Greater than 0.0 or an inherited sample time (-1)

The block sample time and its optional offset time ( $[T_s, T_o]$ ) must be finite and discrete.

---

**Caution** Choose as slow a sample time as needed for smooth animation, since oversampling has little benefit and undersampling can cause animation jumpiness. Undersampling can also potentially block the MATLAB main thread on your computer.

---

#### **Programmatic Use**

**Block Parameter:** SampleTime

**Type:** character vector

**Values:** scalar | vector

**Default:** '1/30'

## **Algorithms**

The simulation pace is implemented by putting the entire MATLAB thread to sleep until it must run again to keep up the pace. The Simulink software is single threaded and runs on the one MATLAB thread, so only one Simulation Pace block can be active at a time.

## **Extended Capabilities**

### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

## **See Also**

Pilot Joystick

### **Topics**

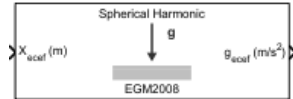
“Specify Sample Time” (Simulink)

“Simulation Pacing” (Simulink)

**Introduced before R2006a**

# Spherical Harmonic Gravity Model

Implement spherical harmonic representation of planetary gravity



## Library

Environment/Gravity

## Description

The Spherical Harmonic Gravity Model block implements the mathematical representation of spherical harmonic planetary gravity based on planetary gravitational potential. It provides a convenient way to describe a planet gravitational field outside of its surface in spherical harmonic expansion.

You can use spherical harmonics to modify the magnitude and direction of spherical gravity ( $-GM/r^2$ ). The most significant or largest spherical harmonic term is the second degree zonal harmonic,  $J_2$ , which accounts for oblateness of a planet.

Use this block if you want more accurate gravity values than spherical gravity models. For example, nonatmospheric flight applications might require higher accuracy.

## Parameters

### Units

Specifies the parameter and output units:

| Units        | Height |
|--------------|--------|
| Metric (MKS) | Meters |
| English      | Feet   |

### Degree

Specify the degree of harmonic model. Recommended degrees are:

| Planet Model | Degree |
|--------------|--------|
| EGM2008      | 120    |
| EGM96        | 70     |
| LP100K       | 60     |
| LP165P       | 60     |
| GMM2B        | 60     |
| EIGENGL04C   | 70     |

**Action for out-of-range input**

Specify if out-of-range input invokes a warning, error, or no action.

**Planet model**

Specify the planetary model. From the list, select:

| Planet Model | Notes   |
|--------------|---|
| EGM2008      | Earth — Is the latest Earth spherical harmonic gravitational model from National Geospatial-Intelligence Agency (NGA). This block provides the WGS-84 version of this gravitational model. You can use the EGM96 planetary model if you need to use the older standard for Earth. |
| EGM96        | Earth   |
| LP100K       | Moon — Is best for lunar orbit determination based upon computational time required to compute orbits. This planet model was created in approximately the same year as LP165P with similar data.  |
| LP165P       | Moon — Is best for extended lunar mission orbit accuracy. This planet model was created in approximately the same year as LP165P with similar data.   |
| GMM2B        | Mars  |
| Custom       | Enables you to specify your own planetary model. This option enables the <b>Planet mat-file</b> parameter.  |
| EIGENGL04C   | Earth — Supports the gravity field model, EIGEN-GL04C ( <a href="http://icgem.gfz-potsdam.de/tom_longtime">http://icgem.gfz-potsdam.de/tom_longtime</a> ). This model is an upgrade to EIGEN-CG03C.   |

When defining your own planetary model, the **Degree** parameter is limited to the maximum value for int16. When inputting a large degree, you might receive an out-of-memory error. For more information about avoiding out-of-memory errors in the MATLAB environment, see “Resolve “Out of Memory” Errors” (MATLAB).

**Planet mat-file**

Specify a MAT-file that contains definitions for a custom planetary model. The `aerogmm2b.mat` file in the Aerospace Toolbox is the default MAT-file for a custom planetary model.

This file must contain:

| Variable      | Description  |
|---------------|--|
| <i>Re</i>     | Scalar of planet equatorial radius in meters (m).  |
| <i>GM</i>     | Scalar of planetary gravitational parameter in meters cubed per second squared ( $\text{m}^3/\text{s}^2$ )                   |
| <i>degree</i> | Scalar of maximum degree.  |
| <i>C</i>      | $(\text{degree}+1)$ -by- $(\text{degree}+1)$ matrix containing normalized spherical harmonic coefficients matrix, <i>C</i> . |
| <i>S</i>      | $(\text{degree}+1)$ -by- $(\text{degree}+1)$ matrix containing normalized spherical harmonic coefficients matrix, <i>S</i> . |

When using a large value for **Degree**, you might receive an out-of-memory error. For more information about avoiding out-of-memory errors in the MATLAB environment, see “Resolve “Out of Memory” Errors” (MATLAB).

## References

- [1] Gottlieb, R. G., “Fast Gravity, Gravity Partial, Normalized Gravity, Gravity Gradient Torque and Magnetic Field: Derivation, Code and Data,” *Technical Report NASA Contractor Report 188243*, NASA Lyndon B. Johnson Space Center, Houston, Texas, February 1993.
- [2] Vallado, D. A., *Fundamentals of Astrodynamics and Applications*, McGraw-Hill, New York, 1997.
- [3] “NIMA TR8350.2: Department of Defense World Geodetic System 1984, Its Definition and Relationship with Local Geodetic Systems”.
- [4] Konopliv, A. S., S. W. Asmar, E. Carranza, W. L. Sjogren, D. N. Yuan., “Recent Gravity Models as a Result of the Lunar Prospector Mission, Icarus”, Vol. 150, no. 1, pp 1-18, 2001.
- [5] Lemoine, F. G., D. E. Smith, D.D. Rowlands, M.T. Zuber, G. A. Neumann, and D. S. Chinn, “An improved solution of the gravity field of Mars (GMM-2B) from Mars Global Surveyor”, *Journal Of Geophysical Research*, Vol. 106, No. E10, pp 23359-23376, October 25, 2001.
- [6] Kenyon S., J. Factor, N. Pavlis, and S. Holmes, “Towards the Next Earth Gravitational Model”, Society of Exploration Geophysicists 77th Annual Meeting, San Antonio, Texas, September 23-28, 2007.
- [7] Pavlis, N.K., S.A. Holmes, S.C. Kenyon, and J.K. Factor, “An Earth Gravitational Model to Degree 2160: EGM2008”, presented at the 2008 General Assembly of the European Geosciences Union, Vienna, Austria, April 13-18, 2008.
- [8] Grueber, T., and A. Köhl, “Validation of the EGM2008 Gravity Field with GPS-Leveling and Oceanographic Analyses”, presented at the IAG International Symposium on Gravity, Geoid & Earth Observation 2008, Chania, Greece, June 23-27, 2008.
- [9] Förste, C., Flechtner, F., Schmidt, R., König, R., Meyer, U., Stubenvoll, R., Rothacher, M., Barthelmes, F., Neumayer, H., Biancale, R., Bruinsma, S., Lemoine, J.M., Loyer, S., “A Mean Global Gravity Field Model From the Combination of Satellite Mission and Altimetry/ Gravimetry Surface Data - EIGEN-GL04C”, *Geophysical Research Abstracts*, Vol. 8, 03462, 2006.
- [10] Hill, K. A. (2007). Autonomous Navigation in Libration Point Orbits. Doctoral dissertation, University of Colorado, Boulder.
- [11] Colombo, Oscar L., “Numerical Methods for Harmonic Analysis on the Sphere”, Reports of the department of Geodetic Science, Report No. 310, The Ohio State University, Columbus, OH., March 1981.
- [12] Colombo, Oscar L., “The Global Mapping of Gravity with Two Satellites”, Nederlands Geodetic Commission, vol. 7 No. 3, Delft, The Netherlands, 1984., Reports of the department of Geodetic Science, Report No. 310, The Ohio State University, Columbus, OH., March 1981.
- [13] Jones, Brandon A. (2010). Efficient Models for the Evaluation and Estimation of the Gravity Field. Doctoral dissertation, University of Colorado, Boulder.

## **Extended Capabilities**

### **C/C++ Code Generation**

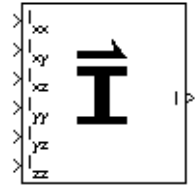
Generate C and C++ code using Simulink® Coder™.

**Introduced in R2010a**

# Symmetric Inertia Tensor

Create inertia tensor from moments and products of inertia

**Library:** Aerospace Blockset / Mass Properties



## Description

The Symmetric Inertia Tensor block creates an inertia tensor from moments and products of inertia. Each input corresponds to an element of the tensor.

The inertia tensor has the form of:

$$Inertia = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{xy} & I_{yy} & -I_{yz} \\ -I_{xz} & -I_{yz} & I_{zz} \end{bmatrix}$$

## Ports

### Input

#### **$I_{xx}$ — Moment of inertia**

scalar

Moment of inertia about the x-axis, specified as a scalar.

Data Types: double

#### **$I_{xy}$ — Product of inertia in xy plane**

scalar

Product of inertia in the xy plane, specified as a scalar.

Data Types: double

#### **$I_{xz}$ — Product of inertia in xz plane**

scalar

Product of inertia in the xz plane, specified as a scalar.

Data Types: double

#### **$I_{yy}$ — Moment of inertia about y-axis**

scalar

Moment of inertia about the y-axis, specified as a scalar.

Data Types: double

**$I_{yz}$  — Product of inertia in yz plane**

scalar

Product of inertia in the yz plane, specified as a scalar.

Data Types: double

 **$I_{zz}$  — Moment of inertia about z-axis**

scalar

Moment of inertia about the z-axis, specified as a scalar.

Data Types: double

**Output** **$I$  — Inertia tensor**

3-by-3 matrix

Symmetric inertia tensor, returned as a 3-by-3 matrix.

Data Types: double

**Extended Capabilities****C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**See Also**[Create 3x3 Matrix](#)**Introduced before R2006a**



# Temperature Conversion

Convert from temperature units to desired temperature units

**Library:** Aerospace Blockset / Utilities / Unit Conversions



## 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 port labels change based on the input and output units selected from the **Initial unit** and the **Final unit** lists.

## Ports

### Input

#### Port\_1 — Temperature

scalar | array

Temperature, specified as a scalar or array, in initial temperature units.

#### Dependencies

The input port label depends on the **Initial unit** setting.

Data Types: double

### Output

#### Port\_1 — Temperature

scalar | array

Temperature, returned as a scalar or array, in final temperature units.

#### Dependencies

The output port label depends on the **Final unit** setting.

Data Types: double

## Parameters

### Initial unit — Input units

R (default) | F | C | K

Input units, specified as:

|   |        |
|---|--------|
| K | Kelvin |
|---|--------|

|   |                    |
|---|--------------------|
| F | Degrees Fahrenheit |
| C | Degrees Celsius    |
| R | Degrees Rankine    |

### Dependencies

The input port label depends on the **Initial unit** setting.

### Programmatic Use

**Block Parameter:** IU

**Type:** character vector

**Values:** 'K' | 'F' | 'C' | 'R'

**Default:** 'R'

### Final unit – Output units

K (default) | F | C | R

Output units, specified as:

|   |                    |
|---|--------------------|
| K | Kelvin             |
| F | Degrees Fahrenheit |
| C | Degrees Celsius    |
| R | Degrees Rankine    |

### Dependencies

The output port label depends on the **Final unit** setting.

### Programmatic Use

**Block Parameter:** OU

**Type:** character vector

**Values:** 'K' | 'F' | 'C' | 'R'

**Default:** 'K'

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

## See Also

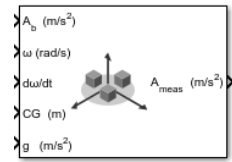
Acceleration Conversion | Angle Conversion | Angular Acceleration Conversion | Angular Velocity Conversion | Density Conversion | Force Conversion | Length Conversion | Mass Conversion | Pressure Conversion | Velocity Conversion

**Introduced before R2006a**

# Three-axis Accelerometer

Implement three-axis accelerometer

**Library:** Aerospace Blockset / GNC / Navigation



## Description

The Three-Axis Accelerometer block implements an accelerometer on each of the three axes. For more information on the ideal measured accelerations, see “Algorithms” on page 4-601.

Optionally, to apply discretizations to the Three-Axis Accelerometer block inputs and dynamics along with nonlinearizations of the measured accelerations, use the Saturation block.

The Three-axis Accelerometer block icon displays the input and output units selected from the **Units** parameter.

## Limitations

- Vibropendulous error and hysteresis effects are not accounted for in this block.
- This block is not intended to model the internal dynamics of different forms of the instrument.

## Ports

### Input

#### **$A_b$ — Actual accelerations**

three-element vector

Actual accelerations in body-fixed axes, specified as a three-element vector, in the units specified in the **Units** parameter.

Data Types: double

#### **$\omega$ — Angular rates**

three-element vector

Angular rates in body-fixed axes, specified as a three-element vector, in radians per second.

Data Types: double

#### **$d\omega/dt$ — Angular accelerations**

three-element vector

Angular accelerations in body-fixed axes, specified as a three-element vector, in radians per second squared.

Data Types: double

**CG — Location of center of gravity**

three-element vector

Location of the center of gravity, specified as a three-element vector, in the units specified in the **Units** parameter.

Data Types: double

**g — Gravity**

three-element vector

Gravity in body axis, specified as a three-element vector, in the units specified in the **Units** parameter.

Data Types: double

**Output****A<sub>meas</sub> — Measured accelerations**

three-element vector

Measured accelerations from the accelerometer, returned as a three-element vector, in the units specified in the **Units** parameter.

Data Types: double

**Parameters****Units — Units**

Metric (MKS) (default) | English

Input and output units, specified as:

| Units        | Acceleration              | Length |
|--------------|---------------------------|--------|
| Metric (MKS) | Meters per second squared | Meters |
| English      | Feet per second squared   | Feet   |

**Programmatic Use****Block Parameter:** units**Type:** character vector**Values:** 'Metric (MKS)' | 'English'**Default:** 'Metric (MKS)'**Accelerometer location — Accelerometer location**

[0 0 0] (default) | three-element vector

Location of the accelerometer group, specified as a three-element vector, measured from the zero datum (typically the nose) to aft, to the right of the vertical centerline, and above the horizontal centerline. This measurement reference is the same for the center of gravity input. The units are the units specified in the **Units** parameter.

**Programmatic Use****Block Parameter:** acc**Type:** character vector

**Values:** three-element vector

**Default:** '[0 0 0]'

### **Subtract gravity — Subtract gravity**

on (default) | off

To subtract gravity from acceleration readings, select this check box.

#### **Programmatic Use**

**Block Parameter:** gtype

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'on'

### **Second order dynamics — Second-order dynamics**

on (default) | off

To apply second-order dynamics to acceleration readings, select this check box.

#### **Programmatic Use**

**Block Parameter:** dtype\_a

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'on'

### **Natural frequency (rad/sec) — Natural frequency**

190 (default) | scalar

Natural frequency of the accelerometer, specified as a double scalar, in radians per second.

#### **Programmatic Use**

**Block Parameter:** w\_a

**Type:** character vector

**Values:** double scalar

**Default:** '190'

### **Damping ratio — Damping ratio**

0.707 (default) | scalar

Damping ratio of the accelerometer, specified as a double scalar, with no dimensions.

#### **Programmatic Use**

**Block Parameter:** z\_a

**Type:** character vector

**Values:** double scalar

**Default:** '0.707'

### **Scale factors and cross-coupling — Scale factors and cross coupling**

[1 0 0; 0 1 0; 0 0 1] (default) | 3-by-3 matrix

Scale factors and cross-coupling, specified as a 3-by-3 matrix, to skew the accelerometer from body axes and to scale accelerations along body axes.

#### **Programmatic Use**

**Block Parameter:** a\_sf\_cc

**Type:** character vector

**Values:** 3-by-3 matrix

**Default:** '[1 0 0; 0 1 0; 0 0 1]'

#### **Measurement bias — Measurement bias**

[0 0 0] (default) | three-element vector

Long-term biases along the accelerometer axes, specified as a three-element vector, in the units specified in the **Units** parameter.

#### **Programmatic Use**

**Block Parameter:** a\_bias

**Type:** character vector

**Values:** 3-by-3 matrix

**Default:** '[0 0 0]'

#### **Update rate (sec) — Update rate**

0 (default) | scalar

Update rate of the accelerometer, specified as a double scalar, in seconds. An update rate of 0 creates a continuous accelerometer. If the **Noise on** check box is selected and the update rate is 0, the block updates the noise at a rate of 0.1.

---

**Tip** If you:

- Update this parameter value to 0 (continuous)
- Configure a fixed-step solver for the model

you must also select the **Automatically handle rate transition for data transfer** check box in the **Solver** pane. This check box enables the software to handle rate transitions correctly.

---

#### **Programmatic Use**

**Block Parameter:** a\_Ts

**Type:** character vector

**Values:** double scalar

**Default:** '0'

#### **Noise on — White noise**

on (default) | off

To apply white noise to acceleration readings, select this check box.

#### **Programmatic Use**

**Block Parameter:** a\_rand

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'on'

#### **Noise seeds — Noise seeds**

[23093 23094 23095] (default) | three-element vector

Scalar seeds for the Gaussian noise generator for each axis of the accelerometer, specified as a three-element vector.

**Dependencies**

To enable this parameter, select **Noise on**.

**Programmatic Use**

**Block Parameter:** a\_seeds

**Type:** character vector

**Values:** three-element vector

**Default:** '[23093 23094 23095]'

**Noise power — Noise power**

[0.001 0.001 0.001] (default) | three-element vector

Height of the power spectral density (PSD) of the white noise for each axis of the accelerometer, specified as a three-element vector, in:

- (m/s<sup>2</sup>)/Hz when **Units** is set to Metric (MKS)
- (ft/s<sup>2</sup>)/Hz when **Units** is set to English

**Dependencies**

To enable this parameter, select **Noise on**.

**Programmatic Use**

**Block Parameter:** a\_pow

**Type:** character vector

**Values:** three-element vector

**Default:** '[0.001 0.001 0.001]'

**Lower and upper output limits — Minimum and maximum values of acceleration**

[-inf -inf -inf inf inf inf] (default) | six-element vector

Three minimum values and three maximum values of acceleration in each of the accelerometer axes, specified as a six-element vector, in the units specified in the **Units** parameter.

**Programmatic Use**

**Block Parameter:** a\_sat

**Type:** character vector

**Values:** six-element vector

**Default:** '[-inf -inf -inf inf inf inf]'

**Algorithms**

The ideal measured accelerations ( $\bar{A}_{imeas}$ ) include the acceleration in body axes at the center of gravity ( $\bar{A}_b$ ) and lever arm effects due to the accelerometer not being at the center of gravity. Optionally, gravity in body axes can be removed. This is represented by the equation:

$$\bar{A}_{imeas} = \bar{A}_b + \bar{\omega}_b \times (\bar{\omega}_b \times \bar{d}) + \dot{\bar{\omega}}_b \times \bar{d} - \bar{g}$$

where  $\bar{\omega}_b$  are body-fixed angular rates,  $\dot{\bar{\omega}}_b$  are body-fixed angular accelerations, and  $\bar{d}$  is the lever arm. The lever arm ( $\bar{d}$ ) is defined as the distances that the accelerometer group is forward, right, and below the center of gravity:

$$\bar{d} = \begin{bmatrix} d_x \\ d_y \\ d_z \end{bmatrix} = \begin{bmatrix} -(x_{acc} - x_{CG}) \\ y_{acc} - y_{CG} \\ -(z_{acc} - z_{CG}) \end{bmatrix}$$

The orientation of the axes used to determine the location of the accelerometer group ( $x_{acc}$ ,  $y_{acc}$ ,  $z_{acc}$ ) and center of gravity ( $x_{CG}$ ,  $y_{CG}$ ,  $z_{CG}$ ) is from the zero datum (typically the nose) to aft, to the right of the vertical centerline and above the horizontal centerline. The  $x$ -axis and  $z$ -axis of these measurement axes are opposite the body-fixed axes that produce the negative signs in the lever arms for the  $x$ -axis and  $z$ -axis.

Measured accelerations ( $\bar{A}_{meas}$ ) output by this block contain error sources and are defined as

$$\bar{A}_{meas} = \bar{A}_{imeas} \times \bar{A}_{SFCC} + \bar{A}_{bias} + noise,$$

where  $\bar{A}_{SFCC}$  is a 3-by-3 matrix of scaling factors on the diagonal and misalignment terms in the nondiagonal, and  $\bar{A}_{bias}$  are the biases.

## References

- [1] Rogers, R. M., *Applied Mathematics in Integrated Navigation Systems*, AIAA Education Series, 2000.

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

## See Also

Saturation | Three-axis Gyroscope | Three-axis Inertial Measurement Unit

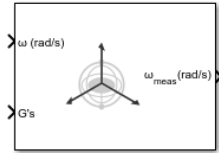
**Introduced before R2006a**



# Three-axis Gyroscope

Implement three-axis gyroscope

**Library:** Aerospace Blockset / GNC / Navigation



## Description

The Three-Axis Gyroscope block implements a gyroscope on each of the three axes. For more information on the measured body angular rates, see “Algorithms” on page 4-606.

Optionally, to apply discretizations to the block inputs and dynamics along with nonlinearizations of the measured body angular rates, use the Saturation block.

## Limitations

- Anisoelastic bias and aniso inertial bias effects are not accounted for in this block.
- This block is not intended to model the internal dynamics of different forms of the instrument.

## Ports

### Input

#### $\omega$ — Angular rates

three-element vector

Angular rates in the body-fixed axes, specified as a three-element vector, in radians per second.

Data Types: double

#### $G$ 's — Accelerations

three-element vector

Accelerations in the body-fixed axes, specified as a three-element vector, in Gs.

Data Types: double

### Output

#### $\omega_{meas}$ — Measured angular rates

three-element vector

Measured angular rates from the gyroscope, returned as a three-element vector, in radians per second.

Data Types: double

## Parameters

### Second order dynamics — Second-order dynamics

on (default) | off

To apply second-order dynamics to gyroscope readings, select this check box.

#### Programmatic Use

**Block Parameter:** dtype\_g

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'on'

### Natural frequency (rad/sec) — Natural frequency

190 (default) | scalar

Natural frequency of the gyroscope, specified as a double scalar, in radians per second.

#### Programmatic Use

**Block Parameter:** w\_g

**Type:** character vector

**Values:** double scalar

**Default:** '190'

### Damping ratio — Damping ratio

0.707 (default) | scalar

Damping ratio of the gyroscope, specified as a double scalar.

#### Programmatic Use

**Block Parameter:** z\_g

**Type:** character vector

**Values:** double scalar

**Default:** '0.707'

### Scale factors and cross-coupling — Scale factors and cross coupling

[1 0 0; 0 1 0; 0 0 1] (default) | 3-by-3 matrix

Scale factors and cross-coupling, specified as a 3-by-3 matrix, to skew the gyroscope from body axes and to scale accelerations along body axes.

#### Programmatic Use

**Block Parameter:** g\_sf\_cc

**Type:** character vector

**Values:** 3-by-3 matrix

**Default:** '[1 0 0; 0 1 0; 0 0 1]'

### Measurement bias — Measurement bias

[0 0 0] (default) | three-element vector

Long-term biases along the gyroscope axes, specified as a three-element vector, in radians per second.

#### Programmatic Use

**Block Parameter:** g\_bias

**Type:** character vector

**Values:** 3-by-3 matrix

**Default:** '[0 0 0]'

### G-sensitive bias — Maximum change in rates

[0 0 0] (default) | three-element vector

Maximum change in rates due to linear acceleration, specified as a three-element vector, in radians per second per g-unit.

#### Programmatic Use

**Block Parameter:** g\_sen

**Type:** character vector

**Values:** three-element vector

**Default:** '[0 0 0]'

### Update rate (sec) — Update rate

0 (default) | scalar

Update rate of the gyroscope, specified as a double scalar, in seconds. An update rate of 0 creates a continuous gyroscope. If the **Noise on** check box is selected and the update rate is 0, the block updates the noise at a rate of 0.1.

---

**Tip** If you:

- Update this parameter value to 0 (continuous)
- Configure a fixed-step solver for the model

you must also select the **Automatically handle rate transition for data transfer** check box in the **Solver** pane. This check box enables the software to handle rate transitions correctly.

---

#### Programmatic Use

**Block Parameter:** g\_Ts

**Type:** character vector

**Values:** double scalar

**Default:** '0'

### Noise on — White noise

on (default) | off

To apply white noise to the gyroscope readings, select this check box.

#### Programmatic Use

**Block Parameter:** g\_rand

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'on'

### Noise seeds — Noise seeds

[23093 23094 23095] (default) | three-element vector

Scalar seeds for the Gaussian noise generator for each axis of the gyroscope, specified as a three-element vector.

**Dependencies**

To enable this parameter, select **Noise on**.

**Programmatic Use**

**Block Parameter:** `g_seeds`

**Type:** character vector

**Values:** three-element vector

**Default:** `'[23093 23094 23095]'`

**Noise power — Noise power**

`[0.0001 0.0001 0.0001]` (default) | three-element vector

Height of the power spectral density (PSD) of the white noise for each axis of the gyroscope, specified as a three-element vector, in  $(\text{rad/s})^2/\text{Hz}$ .

**Dependencies**

To enable this parameter, select **Noise on**.

**Programmatic Use**

**Block Parameter:** `g_pow`

**Type:** character vector

**Values:** three-element vector

**Default:** `'[0.0001 0.0001 0.0001]'`

**Lower and upper output limits — Minimum and maximum values of angular rates**

`[-inf -inf -inf inf inf inf]` (default) | six-element vector

Three minimum values and three maximum values of angular rates in each of gyroscope axes, specified as a six-element vector, in radians per second.

**Programmatic Use**

**Block Parameter:** `g_sat`

**Type:** character vector

**Values:** six-element vector

**Default:** `'[-inf -inf -inf inf inf inf]'`

**Algorithms**

The measured body angular rates ( $\bar{\omega}_{meas}$ ) include the body angular rates ( $\bar{\omega}_b$ ), errors, and, optionally, the discretizations and nonlinearizations of the signals:

$$\bar{\omega}_{meas} = \bar{\omega}_b \times \bar{\omega}_{SFCC} + \bar{\omega}_{bias} + Gs \times \bar{\omega}_{gsens} + noise$$

where  $\bar{\omega}_{SFCC}$  is a 3-by-3 matrix of scaling factors on the diagonal and misalignment terms in the nondiagonal,  $\bar{\omega}_{bias}$  are the biases, ( $Gs$ ) are the  $Gs$  on the gyroscope, and  $\bar{\omega}_{gsens}$  are the  $G$ -sensitive biases.

**References**

- [1] Rogers, R. M., *Applied Mathematics in Integrated Navigation Systems*, AIAA Education Series, 2000.

## **Extended Capabilities**

### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

### **See Also**

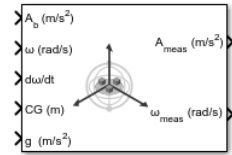
Saturation | Three-axis Accelerometer | Three-axis Inertial Measurement Unit

**Introduced before R2006a**

# Three-axis Inertial Measurement Unit

Implement three-axis inertial measurement unit (IMU)

**Library:** Aerospace Blockset / GNC / Navigation



## Description

The Three-Axis Inertial Measurement Unit block implements an inertial measurement unit (IMU) containing a three-axis accelerometer and a three-axis gyroscope.

For a description of the equations and application of errors, see Three-axis Accelerometer and Three-axis Gyroscope.

The Three-axis Inertial Measurement Unit block icon displays the input and output units selected from the **Units** parameter.

## Limitations

- Vibropendulous error, hysteresis affects, anisoelastic bias, and aniso inertial bias are not accounted for in this block.
- This block is not intended to model the internal dynamics of different forms of the instrument.

## Ports

### Input

#### $A_b$ — Actual accelerations

three-element vector

Actual accelerations in body-fixed axes, specified as a three-element vector, in the units specified in the **Units** parameter.

Data Types: double

#### $\omega$ — Angular rates

three-element vector

Angular rates in body-fixed axes, specified as a three-element vector, in radians per second.

Data Types: double

#### $d\omega/dt$ — Angular accelerations

three-element vector

Angular accelerations in body-fixed axes, specified as a three-element vector, in radians per second squared.

Data Types: double

**CG — Location of center of gravity**

three-element vector

Location of the center of gravity, specified as a three-element vector, in the units specified in the **Units** parameter.

Data Types: double

**g — Gravity**

three-element vector

Gravity in body axis, specified as a three-element vector, in the units specified in the **Units** parameter.

Data Types: double

**Output****A<sub>meas</sub> — Measured accelerations**

three-element vector

Measured accelerations from the accelerometer, returned as a three-element vector, in the units specified in the **Units** parameter.

Data Types: double

**w<sub>meas</sub> — Measured angular rates**

three-element vector

Measured angular rates from the gyroscope, returned as a three-element vector, in radians per second.

Data Types: double

**Parameters****Units — Units**

Metric (MKS) (default) | English

Input and output units, specified as:

| Units        | Acceleration              | Length |
|--------------|---------------------------|--------|
| Metric (MKS) | Meters per second squared | Meters |
| English      | Feet per second squared   | Feet   |

**Programmatic Use****Block Parameter:** units**Type:** character vector**Values:** 'Metric (MKS)' | 'English'**Default:** 'Metric (MKS)'**IMU location — IMU location**

[0 0 0] (default) | three-element vector

The location of the IMU, which is also the accelerometer group location, is measured from the zero datum (typically the nose) to aft, to the right of the vertical centerline, and above the horizontal

centerline. This measurement reference is the same for the center of gravity input. The units are in the units specified in the **Units** parameter.

**Programmatic Use**

**Block Parameter:** imu

**Type:** character vector

**Values:** three-element vector

**Default:** '[0 0 0]'

**Update rate (sec) — Update rate**

0 (default) | scalar

Update rate of the accelerometer and gyroscope, specified as a double scalar, in seconds. An update rate of 0 creates a continuous accelerometer and continuous gyroscope. If the **Noise on** check box is selected and the update rate is 0, the block updates the noise at a rate of 0.1.

---

**Tip** If you:

- Update this parameter value to 0 (continuous)
- Configure a fixed-step solver for the model

you must also select the **Automatically handle rate transition for data transfer** check box in the **Solver** pane. This check box enables the software to handle rate transitions correctly.

---

**Programmatic Use**

**Block Parameter:** i\_Ts

**Type:** character vector

**Values:** double scalar

**Default:** '0'

**Second order dynamics for accelerometer — Second-order dynamics**

on (default) | off

To apply second-order dynamics to acceleration readings, select this check box.

**Programmatic Use**

**Block Parameter:** dtype\_a

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'on'

**Accelerometer natural frequency (rad/sec) — Accelerometer natural frequency**

190 (default) | scalar

Natural frequency of the accelerometer, specified as a double scalar, in radians per second.

**Dependencies**

To enable this parameter, select **Second order dynamics for accelerometer**.

**Programmatic Use**

**Block Parameter:** w\_a

**Type:** character vector

**Values:** double scalar



**Default:** '190'

### Accelerometer damping ratio — Accelerometer damping ratio

0.707 (default) | scalar

Damping ratio of the accelerometer, specified as a double scalar, with no dimensions.

#### Dependencies

To enable this parameter, select **Second order dynamics for accelerometer**.

#### Programmatic Use

**Block Parameter:** z\_a

**Type:** character vector

**Values:** double scalar

**Default:** '0.707'

### Accelerometer scale factor and cross-coupling — Scale factors and cross coupling

[1 0 0; 0 1 0; 0 0 1] (default) | 3-by-3 matrix

Scale factors and cross-coupling, specified as a 3-by-3 matrix, to skew the accelerometer from body axes and to scale accelerations along body axes.

#### Programmatic Use

**Block Parameter:** a\_sf\_cc

**Type:** character vector

**Values:** 3-by-3 matrix

**Default:** '[1 0 0; 0 1 0; 0 0 1]'

### Accelerometer measurement bias — Accelerometer measurement bias

[0 0 0] (default) | three-element vector

Long-term biases along the accelerometer axes, specified as a three-element vector, in the units specified in the **Units** parameter.

#### Programmatic Use

**Block Parameter:** a\_bias

**Type:** character vector

**Values:** three-element vector

**Default:** '[0 0 0]'

### Accelerometer upper and lower limits — Minimum and maximum values of acceleration

[-inf -inf -inf inf inf inf] (default) | six-element vector

Three minimum values and three maximum values of acceleration in each of accelerometer axes, specified as a six-element vector, in units specified in the **Units** parameter.

#### Programmatic Use

**Block Parameter:** a\_sat

**Type:** character vector

**Values:** six-element vector

**Default:** '[-inf -inf -inf inf inf inf]'

### Second-order dynamics for gyro — Gyroscope second-order dynamics

on (default) | off

To apply second-order dynamics to gyroscope readings, select this check box.

**Programmatic Use****Block Parameter:** dtype\_g**Type:** character vector**Values:** 'on' | 'off'**Default:** 'on'**Gyro natural frequency (rad/sec) — Gyroscope natural frequency**

190 (default) | scalar

Natural frequency of the gyroscope, specified as a double scalar, in radians per second.

**Dependencies**

To enable this parameter, select **Second-order dynamics for gyro**.

**Programmatic Use****Block Parameter:** w\_g**Type:** character vector**Values:** double scalar**Default:** '190'**Gyro damping ratio — Gyroscope damping ratio**

0.707 (default) | scalar

Damping ratio of the gyroscope, specified as a double scalar.

**Dependencies**

To enable this parameter, select **Second-order dynamics for gyro**.

**Programmatic Use****Block Parameter:** z\_g**Type:** character vector**Values:** double scalar**Default:** '0.707'**Gyro scale factors and cross-coupling — Gyroscope scale factors and cross-coupling**

[1 0 0; 0 1 0; 0 0 1] (default) | 3-by-3 matrix

Gyroscope scale factors and cross-coupling, specified as a 3-by-3 matrix, to skew the gyroscope from body axes and to scale angular rates along body axes.

**Programmatic Use****Block Parameter:** g\_sf\_cc**Type:** character vector**Values:** 3-by-3 matrix**Default:** '[1 0 0; 0 1 0; 0 0 1]'**Gyro measurement bias — Gyroscope measurement bias**

[0 0 0] (default) | three-element vector

Long-term biases along the gyroscope axes, specified a three-element vector, in radians per second.

**Programmatic Use****Block Parameter:** g\_bias

**Type:** character vector  
**Values:** three-element vector  
**Default:** '[0 0 0]'

#### **G-sensitive bias — Maximum change in rates**

[0 0 0] (default) | three-element vector

Maximum change in rates due to linear acceleration, specified as a three-element vector, in radians per second per g-unit.

##### **Programmatic Use**

**Block Parameter:** g\_sens  
**Type:** character vector  
**Values:** three-element vector  
**Default:** '[0 0 0]'

#### **Gyro upper and lower limits — Minimum and maximum values of angular rates**

[-inf -inf -inf inf inf inf] (default) | six-element vector

Three minimum values and three maximum values of angular rates in each of the gyroscope axes, specified as a six-element vector, in radians per second.

##### **Programmatic Use**

**Block Parameter:** g\_sat  
**Type:** character vector  
**Values:** six-element vector  
**Default:** '[-inf -inf -inf inf inf inf]'

#### **Noise on — White noise**

on (default) | off

To apply white noise to acceleration and gyroscope readings, select this check box.

##### **Programmatic Use**

**Block Parameter:** i\_rand  
**Type:** character vector  
**Values:** 'on' | 'off'  
**Default:** 'on'

#### **Noise seeds — Noise seeds**

[23093 23094 23095 23096 23097 23098] (default) | six-element vector

Scalar seeds for the Gaussian noise generator for each axis of the accelerometer and gyroscope, specified as a six-element vector.

##### **Dependencies**

To enable this parameter, select **Noise on**.

##### **Programmatic Use**

**Block Parameter:** i\_seeds  
**Type:** character vector  
**Values:** six-element vector  
**Default:** '[23093 23094 23095 23096 23097 23098]'

#### **Noise power — Noise power**

[0.001 0.001 0.001 0.0001 0.0001 0.0001] (default) | six-element vector

Height of the power spectral density (PSD) of the white noise for each axis of the accelerometer and gyroscope, specified as a six-element vector, in:

- (m/s<sup>2</sup>)/Hz when **Units** is set to `Metric` (MKS)
- (ft/s<sup>2</sup>)/Hz when **Units** is set to `English`

### **Dependencies**

To enable this parameter, select **Noise on**.

### **Programmatic Use**

**Block Parameter:** `i_pow`

**Type:** character vector

**Values:** six-element vector

**Default:** `'[0.001 0.001 0.001 0.0001 0.0001 0.0001]'`

### **References**

[1] Rogers, R. M., *Applied Mathematics in Integrated Navigation Systems*, AIAA Education Series, 2000.

### **Extended Capabilities**

#### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

### **See Also**

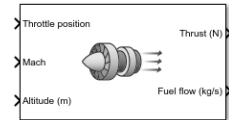
[Calculate Range | Three-axis Accelerometer | Three-axis Gyroscope](#)

**Introduced before R2006a**

# Turbofan Engine System

Implement first-order representation of turbofan engine with controller

**Library:** Aerospace Blockset / 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. For more information on this system, see “Algorithms” on page 4-618.

The Turbofan Engine System block icon displays the input and output units selected from the **Units** parameter.

## Limitations

- The atmosphere is at standard day conditions and an ideal gas.
- The 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.
- This engine system is assumed to have a high bypass ratio.

## Ports

### Input

#### Throttle position — Throttle position

scalar | vector

Throttle position, specified as a scalar or vector. This value can vary from zero to one, corresponding to no and full throttle.

Data Types: double

#### Mach — Mach number

scalar

Mach number, specified as a scalar.

Data Types: double

#### Altitude — Altitude

scalar | vector

Altitude, specified as scalar or vector, in the units specified in the **Units** parameter.

Data Types: double

**Initial thrust – Initial thrust**

scalar | vector

Initial thrust, specified as a scalar or vector, in the units specified in the **Units** parameter.

**Dependencies**

To enable this port, set **Initial thrust source** to External.

Data Types: double

**Output****Thrust – Thrust**

scalar | vector

Thrust, returned as a scalar or vector, in the units specified in the **Units** parameter.

Data Types: double

**Fuel flow – Fuel flow**

scalar | vector

Fuel flow, returned as scalar or vector, in the units specified in the **Units** parameter units per second.

Data Types: double

**Parameters****Units – Input and output units**

Metric (MKS) (default) | English

Input and output units, specified as:

| Units        | Altitude | Thrust      | Fuel Flow             |
|--------------|----------|-------------|-----------------------|
| Metric (MKS) | Meters   | Newtons     | Kilograms per second  |
| English      | Feet     | Pound force | Pound mass per second |

**Programmatic Use****Block Parameter:** units**Type:** character vector**Values:** 'Metric (MKS)' | 'English'**Default:** 'Metric (MKS)'**Initial thrust source – Initial thrust source**

Internal (default) | External

Initial thrust, specified as:

|          |   |
|----------|---|
| Internal | Use the value of the <b>Initial thrust</b> parameter. |
| External | Use external input for initial thrust value.          |

**Programmatic Use****Block Parameter:** ic\_source

**Type:** character vector  
**Values:** 'Internal' | 'External'  
**Default:** 'Internal'

**Initial thrust – Initial**

0 (default) | scalar

Initial thrust value, specified as a double scalar.

**Programmatic Use**

**Block Parameter:** IC

**Type:** character vector

**Values:** double scalar

**Default:** '0'

**Maximum sea-level static thrust – Maximum thrust at sea-level**

45000 (default) | scalar

Maximum thrust at sea-level, specified as a double scalar, at a Mach value of 0.

**Programmatic Use**

**Block Parameter:** Fmax

**Type:** character vector

**Values:** double scalar

**Default:** '45000'

**Fastest engine time constant at sea-level static (sec) – Fastest engine time at sea level**

1 (default) | scalar

Fastest engine time at sea level, specified as a double scalar.

**Programmatic Use**

**Block Parameter:** tau

**Type:** character vector

**Values:** double scalar

**Default:** '1'

**Sea-level static thrust specific fuel consumption – Thrust-specific fuel consumption at sea level**

0.35 (default) | scalar

Thrust-specific fuel consumption at sea level, specified as a double scalar, in specified mass units per hour per specified thrust units, at:

- Mach value of 0
- Maximum thrust

**Programmatic Use**

**Block Parameter:** SFC

**Type:** character vector

**Values:** double scalar

**Default:** '0.35'

**Ratio of installed thrust to uninstalled thrust – Coefficient representing loss**

0.9 (default) | scalar

Coefficient representing the loss in thrust due to engine installation, specified as a double value.

**Programmatic Use**

**Block Parameter:** Nt

**Type:** character vector

**Values:** double scalar

**Default:** '0.9'

## Algorithms

This system is represented by a first-order system with unitless heuristic lookup tables for thrust, thrust specific fuel consumption (TSFC), and the engine time constant. For the lookup table data, thrust is a function of throttle position and the Mach number, TSFC is a function of thrust and the 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  $\delta$  and relative temperature ratio  $\theta$ , and scaled by maximum sea level static thrust, the fastest engine time constant at the sea level static, sea level static thrust specific fuel consumption, and the ratio of installed thrust to uninstalled thrust.

## References

- [1] *Aeronautical Vestpocket Handbook*, United Technologies Pratt & Whitney, August, 1986.
- [2] Raymer, D. P., *Aircraft Design: A Conceptual Approach*, AIAA Education Series, Washington, DC, 1989.
- [3] Hill, P. G., and C. R. Peterson, *Mechanics and Thermodynamics of Propulsion*, Addison-Wesley Publishing Company, Reading, Massachusetts, 1970.

## Extended Capabilities

**C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

## See Also

**Introduced before R2006a**



# Turn Coordinator

Display measurements on turn coordinator and inclinometer

**Library:** Aerospace Blockset / Flight Instruments



## Description

The Turn Coordinator block displays measurements on a gyroscopic turn rate instrument and on an inclinometer.

- The gyroscopic turn rate instrument shows the rate of heading change of the aircraft as a tilting of the aircraft symbol in the gauge.
- The inclinometer shows whether the turn is coordinated, slipping, or skidding by the position of the ball.

When the ball is centered, the turn is coordinated. When the ball is off center, the turn is slipping or skidding. The turn rate instrument has marks for wings level and for a standard rate turn. A standard rate turn is a heading change of 3 degrees per second, also known as a two minute turn.

The input for gyroscopic turn rate instruments and inclinometers is in degrees. The turn rate value is input as the degrees of tilt of the aircraft symbol in the gauge. The standard rate turn marks are at angles of  $\pm 15$  degrees. Tilt angle values are limited to  $\pm 20$  degrees, whereas inclinometer angles are limited to  $\pm 15$  degrees.

Combine the turn indicator and inclinometer signals in a Mux block in order:

- 1 Turn indicator
- 2 Inclinometer

For example, turn indicator and inclinometer values of `[15 0]` indicate a coordinated, standard rate turn.

---

**Tip** To facilitate understanding and debugging your model, you can modify instrument block connections in your model during normal and accelerator mode simulations.

---

## Parameters

### Connection — Connect to signal

signal name | 2-element signal

Connect to 2-element signal for display, selected from a list of signal names. The 2-element signal consists of turn indicator and inclinometer signals combined in a Mux block, in degrees. You connect and display this combined signal. This input cannot be a bus signal.

To view the data from a signal, select a signal in the model. The signal appears in the **Connection** table. Select the option button next to the signal you want to display. Click **Apply** to connect the signal.

The table has a row for the signal connected to the block. If there are no signals selected in the model, or the block is not connected to any signals, the table is empty.

**Label — Block label location**

Top (default) | Bottom | Hide

Block label, displayed at the top or bottom of the block, or hidden.

- Top

Show label at the top of the block.

- Bottom

Show label at the bottom of the block.

- Hide

Do not show the label or instructional text when the block is not connected.

**Programmatic Use**

**Block Parameter:** LabelPosition

**Type:** character vector

**Values:** 'Top' | 'Bottom' | 'Hide'

**Default:** 'Top'

**Extended Capabilities****C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**See Also**

Airspeed Indicator | Altimeter | Artificial Horizon | Climb Rate Indicator | Exhaust Gas Temperature (EGT) Indicator | Heading Indicator | Revolutions Per Minute (RPM) Indicator

**Topics**

“Display Measurements with Cockpit Instruments” on page 2-39

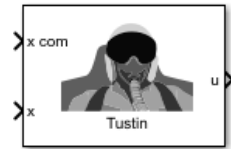
“Flight Instrument Gauges” on page 2-38

**Introduced in R2016a**

# Tustin Pilot Model

Represent Tustin pilot model

**Library:** Aerospace Blockset / Pilot Models



## Description

The Tustin Pilot Model block represents the pilot model that A. Tustin describes in *The Nature of the Operator's Response in Manual Control, and its Implications for Controller Design* [1]. When modeling human pilot models, use this block for the least accuracy, compared to that provided by the Crossover Pilot Model and Precision Pilot Model blocks. This block requires less input than those blocks, and provides better performance. However, the results might be less accurate.

This pilot model is a single input, single output (SISO) model that represents human behavior, and is based on the transfer function described in “Algorithms” on page 4-622.

This block has nonlinear behavior. If you want to linearize the block (for example, with one of the `linmod` functions), you might need to change the Pade approximation order. The Tustin Pilot Model block implementation incorporates the Transport Delay block with the **Pade order (for linearization)** parameter set to 2 by default. To change this value, use the `set_param` function, for example:

```
set_param(gcb, 'pade', '3')
```

## Ports

### Input

#### **x com — Signal command**

scalar

Signal command that the pilot model controls, specified as a scalar.

Data Types: `double`

#### **x — Signal controlled by pilot**

scalar

Signal controlled by pilot, specified as a scalar.

Data Types: `double`

### Output

#### **u — Aircraft command**

scalar

Aircraft command, returned as a scalar.

Data Types: double

## Parameters

### Pilot gain — Pilot gain

1 (default) | scalar

Pilot gain, specified as a double scalar.

#### Programmatic Use

**Block Parameter:** Kp

**Type:** character vector

**Values:** double scalar

**Default:** '1'

### Pilot time delay(s) — Pilot time delay

0.1 (default) | scalar

Total pilot time delay, specified as a double scalar, in seconds. This value typically ranges from 0.1 s to 0.2 s.

#### Programmatic Use

**Block Parameter:** time\_delay

**Type:** character vector

**Values:** double scalar

**Default:** '0.1'

### Pilot lead constant — Pilot lead constant

5 (default) | scalar

Pilot lead constant, specified as a double scalar.

#### Programmatic Use

**Block Parameter:** T

**Type:** character vector

**Values:** double scalar

**Default:** '5'

## Algorithms

This pilot model is a single input, single output (SISO) model that represents human behavior, based on the transfer function:

$$\frac{u(s)}{e(s)} = \frac{K_p(1 + Ts)}{s} e^{-\tau s}.$$

In this equation:

| Variable | Description    |
|----------|----------------|
| $K_p$    | Pilot gain.    |
| $T$      | Lead constant. |

| Variable | Description  |
|----------|--|
| $\tau$   | Transport delay time caused by the pilot neuromuscular system. |
| $u(s)$   | Input to the aircraft model and output to the pilot model.     |
| $e(s)$   | Error between the desired pilot value and the actual value.    |

## References

- [1] Tustin, A., *The Nature of the Operator's Response in Manual Control, and its Implications for Controller Design*. Convention on Automatic Regulators and Servo Mechanisms. May, 1947.

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

## See Also

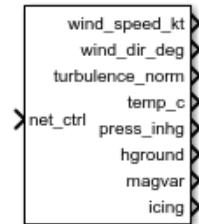
Crossover Pilot Model | Precision Pilot Model | Transport Delay | `linmod`

**Introduced in R2012b**

## Unpack net\_ctrl Packet from FlightGear

Unpack net\_ctrl variable packet received from FlightGear

**Library:** Aerospace Blockset / Animation / Flight Simulator Interfaces



### Description

The Unpack net\_ctrl Packet from FlightGear block unpacks net\_ctrl variable packets received from FlightGear via the Receive net\_ctrl Packet from FlightGear block, and makes them available for the Simulink environment.

### Ports

#### Input

**net\_ctrl — FlightGear packet to be unpacked**

array

FlightGear packet to be unpacked, specified as an array.

Data Types: uint8

#### Output

##### Environment Outputs

**wind\_speed\_kt — Wind speed**

scalar

Wind speed, specified as a scalar, in knots.

##### Dependencies

To enable this port, select the **Show environment outputs** check box.

Data Types: double

**wind\_dir\_deg — Wind direction**

scalar

Wind direction, specified as a scalar, in deg.

##### Dependencies

To enable this port, select the **Show environment outputs** check box.

Data Types: double

**turbulence\_norm — Turbulence norm**

scalar

Turbulence norm, specified as a scalar.

**Dependencies**To enable this port, select the **Show environment outputs** check box.

Data Types: double

**temp\_c — Ambient temperature**

scalar

Ambient temperature, specified as a scalar, in deg C.

**Dependencies**To enable this port, select the **Show environment outputs** check box.

Data Types: double

**press\_inhg — Ambient pressure**

scalar

Ambient pressure, specified as a scalar, in inHg.

**Dependencies**To enable this port, select the **Show environment outputs** check box.

Data Types: double

**hground — Ground elevation**

scalar

Ground elevation, specified as a scalar, in m.

**Dependencies**To enable this port, select the **Show environment outputs** check box.

Data Types: double

**magvar — Local magnetic variation**

scalar

Local magnetic variation, specified as a scalar.

**Dependencies**To enable this port, select the **Show environment outputs** check box.

Data Types: double

**icing — Icing status**

scalar

Icing status, specified as a scalar, in deg.

**Dependencies**

To enable this port, select the **Show environment outputs** check box.

Data Types: `uint32`

**Control Surface Position Inputs****aileron — Normalized aileron position**

1 | scalar

Normalized aileron position [-1,1], specified as a scalar.

**Dependencies**

To enable this port, select the **Show control surface position outputs** check box.

Data Types: `double`

**elevator — Normalized elevator position**

1 | scalar

Normalized elevator position [-1,1], specified as a scalar.

**Dependencies**

To enable this port, select the **Show control surface position outputs** check box.

Data Types: `double`

**rudder — Normalized rudder position**

1 | scalar

Normalized rudder position [-1,1], specified as a scalar.

**Dependencies**

To enable this port, select the **Show control surface position outputs** check box.

Data Types: `double`

**aileron\_trim — Normalized aileron trim position**

scalar

Normalized aileron trim position [-1,1], specified as a scalar.

**Dependencies**

To enable this port, select the **Show control surface position outputs** check box.

Data Types: `double`

**elevator\_trim — Normalized elevator trim position**

1 | scalar

Normalized elevator trim position [-1,1], specified as a scalar.

**Dependencies**

To enable this port, select the **Show control surface position outputs** check box.



Data Types: double

**rudder\_trim — Normalized rudder trim position**

1 | scalar

Normalized rudder trim position [-1,1], specified as a scalar.

**Dependencies**

To enable this port, select the **Show control surface position outputs** check box.

Data Types: double

**flaps — Normalized flaps position**

1 | scalar

Normalized flaps position [-0,1], specified as a scalar.

**Dependencies**

To enable this port, select the **Show control surface position outputs** check box.

Data Types: double

**spoilers — Normalized spoilers position**

1 | scalar

Normalized spoilers position [0,1], specified as a scalar.

**Dependencies**

To enable this port, select the **Show control surface position outputs** check box.

Data Types: single

**speedbrake — Normalized speedbrake position**

1 | scalar

Normalized speedbrake position [0,1], specified as a scalar.

**Dependencies**

To enable this port, select the **Show control surface position outputs** check box.

Data Types: single

**flaps\_power — Power for flaps**

1 | scalar

Power for flaps, specified as a scalar. A value of 1 indicates that power is available.

**Dependencies**

To enable this port, select the **Show control surface position outputs** check box.

Data Types: uint32

**flap\_motor\_ok — Flap motor powered**

scalar

Flap motor powered, specified as a scalar.

**Dependencies**

To enable this port, select the **Show control surface position outputs** check box.

Data Types: uint32

**Engine/Fuel Outputs****num\_engines — Number of valid engines**

scalar

Number of valid engines, specified as a scalar.

**Dependencies**

To enable this port, select the **Show engine/fuel outputs** check box.

Data Types: uint32

**master\_bat — Master battery switch**

vector

Master battery switch, specified as a vector.

**Dependencies**

To enable this port, select the **Show engine/fuel outputs** check box.

Data Types: uint32

**master\_alt — Master alternator switch**

vector

Master alternator switch, specified as a vector.

**Dependencies**

To enable this port, select the **Show engine/fuel outputs** check box.

Data Types: uint32

**magnetos — Magnetos switch**

scalar

Magnetos switch, specified as a scalar.

**Dependencies**

To enable this port, select the **Show engine/fuel outputs** check box.

Data Types: uint32

**starter\_power — Power to start motor**

1 | vector

Power to starter motor, specified as a vector. A value of 1 indicates that power is available.

**Dependencies**

To enable this port, select the **Show engine/fuel outputs** check box.

Data Types: uint32

**throttle — Normalized throttle position**

1 | vector

Normalized throttle position [0,1], specified as a vector.

**Dependencies**

To enable this port, select the **Show engine/fuel outputs** check box.

Data Types: double

**mixture — Normalized mixture lever position**

1 | vector

Normalized mixture lever position [0,1], specified as a vector.

**Dependencies**

To enable this port, select the **Show engine/fuel outputs** check box.

Data Types: double

**condition — Normalized condition**

1 | vector

Normalized condition [0,1], specified as a vector.

**Dependencies**

To enable this port, select the **Show engine/fuel outputs** check box.

Data Types: uint32

**fuel\_pump\_power — Normalized speedbrake position**

1 | scalar

Power to fuel pump, specified as a vector. A value of 1 indicates that pump is on.

**Dependencies**

To enable this port, select the **Show engine/fuel outputs** check box.

Data Types: uint32

**prop\_advance — Propeller advance**

1 | vector

Propeller advance [0,1], specified as a vector.

**Dependencies**

To enable this port, select the **Show engine/fuel outputs** check box.

Data Types: double

**feed\_tank\_to — Feed tank to switch**

vector

Feed tank to switch, specified as a vector.

**Dependencies**To enable this port, select the **Show engine/fuel outputs** check box.

Data Types: uint32

**reverse — Reverse switch**

vector

Reverse switch, specified as a vector.

**Dependencies**To enable this port, select the **Show engine/fuel outputs** check box.

Data Types: uint32

**engine\_ok — Engine status indicator**

vector

Engine status indicator, specified as a vector.

**Dependencies**To enable this port, select the **Show engine/fuel outputs** check box.

Data Types: uint32

**mag\_left\_ok — Left magneto status indicator**

vector

Left magneto status indicator, specified as a vector.

**Dependencies**To enable this port, select the **Show engine/fuel outputs** check box.

Data Types: uint32

**mag\_right\_ok — Right magneto status indicator**

vector

Right magneto status indicator, specified as a vector.

**Dependencies**To enable this port, select the **Show engine/fuel outputs** check box.

Data Types: uint32

**spark\_plugs\_ok — Normalized speedbrake position**

vector

Spark plugs status indicator, specified as a vector. A value of 0 indicates that the plugs have failed.

**Dependencies**

To enable this port, select the **Show engine/fuel outputs** check box.

Data Types: uint32

**oil\_press\_status — Oil pressure status indicator**

0 | 1 | 2 | scalar

Oil pressure status indicator, specified as a vector.

- 0 — Normal oil pressure
- 1 — Low oil pressure
- 2 — Failed oil pressure

**Dependencies**

To enable this port, select the **Show engine/fuel outputs** check box.

Data Types: uint32

**fuel\_pump\_ok — Fuel management status indicator**

vector

Fuel management status indicator, specified as a vector.

**Dependencies**

To enable this port, select the **Show engine/fuel outputs** check box.

Data Types: uint32

**num\_tanks — Number of valid tanks**

scalar

Number of valid tanks, specified as a scalar.

**Dependencies**

To enable this port, select the **Show engine/fuel outputs** check box.

Data Types: uint32

**fuel\_selector — Fuel selector**

scalar

Fuel selector, specified as a vector.

- 0 — Off
- 1 — On

**Dependencies**

To enable this port, select the **Show engine/fuel outputs** check box.

Data Types: single

**xfer\_pump — Specify transfer**

vector

Specifies transfer from array value to tank, specified by value as a vector.

**Dependencies**

To enable this port, select the **Show engine/fuel outputs** check box.

Data Types: uint32

**cross\_feed — Cross feed valve**

scalar

Cross feed valve, specified as a scalar.

- 0 — False
- 1 — On

**Dependencies**

To enable this port, select the **Show engine/fuel outputs** check box.

Data Types: single

**Landing Gear Outputs****brake\_left — Left brake pedal position pilot**

scalar

Left brake pedal position pilot, specified as a scalar.

**Dependencies**

To enable this port, select the **Show landing gear outputs** check box.

Data Types: double

**brake\_right — Right brake pedal position pilot**

scalar

Right brake pedal position pilot, specified as a scalar.

**Dependencies**

To enable this port, select the **Show landing gear outputs** check box.

Data Types: double

**copilot\_brake\_left — Left brake pedal position pilot**

scalar

Left brake pedal position pilot, specified as a scalar.

**Dependencies**

To enable this port, select the **Show landing gear outputs** check box.

Data Types: double

**copilot\_brake\_right — Right brake pedal position pilot**

scalar

Right brake pedal position pilot, specified as a scalar.

**Dependencies**

To enable this port, select the **Show landing gear outputs** check box.

Data Types: double

**brake\_parking — Brake parking position**

scalar

Brake parking position, specified as a scalar.

**Dependencies**

To enable this port, select the **Show landing gear outputs** check box.

Data Types: double

**gear\_handle — Gear handle position**

scalar

Gear handle position, specified as a scalar.

- 0 — Gear handle up
- 1 — Gear handle down

**Dependencies**

To enable this port, select the **Show landing gear outputs** check box.

Data Types: uint32

**Avionic Outputs**

**master\_avionics — Master avionics switch**

scalar

Master avionics switch, specified as a scalar.

**Dependencies**

To enable this port, select the **Show avionic outputs** check box.

Data Types: uint32

**comm\_1 — Comm 1 frequency**

scalar

Comm 1 frequency, specified as a scalar, in Hz.

**Dependencies**

To enable this port, select the **Show avionic outputs** check box.

Data Types: double

**comm\_2 — Comm 2 frequency**

scalar

Comm 2 frequency, specified as a scalar, in Hz.

**Dependencies**

To enable this port, select the **Show avionic outputs** check box.

Data Types: double

**nav\_1 — Nav 1 frequency**

scalar

Nav 1 frequency, specified as a scalar, in Hz.

**Dependencies**

To enable this port, select the **Show avionic outputs** check box.

Data Types: double

**nav\_2 — Nav 2 frequency**

scalar

Nav 2 frequency, specified as a scalar, in Hz.

**Dependencies**

To enable this port, select the **Show avionic outputs** check box.

Data Types: double

**Parameters****FlightGear version — FlightGear version**

v2019.1 (default) | v2018.3 | v2018.2 | v2018.1 | v2017.3 | v2017.1 | v2016.3 | v2016.1 | v3.4 | v3.2 | v3.0 | v2.12 | v2.10 | v2.8 | v2.6 | v2.4 | v2.0

FlightGear software version, selected from the list.

---

**Note** If you are using a FlightGear version older than 2.0, the model displays a notification from the Simulink Upgrade Advisor. Consider upgrading your FlightGear version using the Upgrade Advisor. For more information, see “Supported FlightGear Versions” on page 2-16.

---

**Programmatic Use**

**Block Parameter:** FlightGearVersion

**Type:** character vector

**Values:** v2019.1 | v2018.3 | v2018.2 | v2018.1 | v2017.3 | v2017.1 | v2016.3 | v2016.1 | v3.4 | v3.2 | v3.0 | v2.12 | v2.10 | v2.8 | v2.6 | v2.4 | v2.0

**Default:** 'v2019.1'

**Show control surface position outputs — Control surface position outputs**

off (default) | on

Select this check box to include the control surface position outputs from the FlightGear net\_ctrl data packet.



**Dependencies**

Select this check box to enable these input ports.

**Signal Group 1: Control surface position outputs**

| Name                 | Units             | Type   | Width | Description                              |
|----------------------|-------------------|--------|-------|--|
| <i>aileron</i>       | 1 (dimensionless) | double | 1     | Normalized aileron position [-1,1]       |
| <i>elevator</i>      | 1 (dimensionless) | double | 1     | Normalized elevator position [-1,1]      |
| <i>rudder</i>        | 1 (dimensionless) | double | 1     | Normalized rudder position [-1,1]        |
| <i>aileron_trim</i>  | 1 (dimensionless) | double | 1     | Normalized aileron trim position [-1,1]  |
| <i>elevator_trim</i> | 1 (dimensionless) | double | 1     | Normalized elevator trim position [-1,1] |
| <i>rudder_trim</i>   | 1 (dimensionless) | double | 1     | Normalized rudder trim position [-1,1]   |
| <i>flaps</i>         | 1 (dimensionless) | double | 1     | Normalized flaps position [-0,1]         |
| <i>spoilers</i>      | 1 (dimensionless) | double | 1     | Normalized spoilers position [0,1]       |
| <i>speedbrake</i>    | 1 (dimensionless) | double | 1     | Normalized speedbrake position [0,1]     |
| <i>flaps_power</i>   | 1 (dimensionless) | uint32 | 1     | Power for flaps (1 = power available)    |
| <i>flap_motor_ok</i> | —                 | uint32 | 1     | Flap motor powered                       |

**Programmatic Use**

**Block Parameter:** ShowControlSurfacePositionOutputs

**Type:** character vector

**Values:** 'off' | 'on'

**Default:** 'off'

**Show engine/fuel outputs – Engine/fuel outputs**

off (default) | on

Select this check box to include the engine and fuel outputs from the FlightGear net\_ctrl data packet.

**Dependencies**

Select this check box to enable these input ports.

**Signal Group 2: Engine/fuel outputs**

| <b>Name</b>             | <b>Units</b>      | <b>Type</b> | <b>Width</b> | <b>Description</b>  |
|-------------------------|-------------------|-------------|--------------|---|
| <i>num_engines</i>      | —                 | uint32      | 1            | Number of valid engines   |
| <i>master_bat</i>       | —                 | uint32      | 4            | Master battery switch   |
| <i>master_alt</i>       | —                 | uint32      | 4            | Master alternator switch  |
| <i>magnetos</i>         | —                 | uint32      | 4            | Magnetos switch   |
| <i>starter_power</i>    | —                 | uint32      | 4            | Power to starter motor (1 = starter power available)                  |
| <i>throttle</i>         | 1 (dimensionless) | double      | 4            | Normalized throttle position [0,1]                                    |
| <i>mixture</i>          | 1 (dimensionless) | double      | 4            | Normalized mixture lever position [0,1]                               |
| <i>condition</i>        | 1 (dimensionless) | double      | 4            | Normalized condition [0,1]  |
| <i>fuel_pump_power</i>  | —                 | uint32      | 4            | Power to fuel pump 1 = on)  |
| <i>prop_advance</i>     | 1 (dimensionless) | double      | 4            | Propeller advance [0,1]   |
| <i>feed_tank_to</i>     | —                 | uint32      | 4            | Feed tank to switch   |
| <i>reverse</i>          | —                 | uint32      | 4            | Reverse switch  |
| <i>engine_ok</i>        | —                 | uint32      | 4            | Engine status indicator   |
| <i>mag_left_ok</i>      | —                 | uint32      | 4            | Left magneto status indicator   |
| <i>mag_right_ok</i>     | —                 | uint32      | 4            | Right magneto status indicator  |
| <i>spark_plugs_ok</i>   | —                 | uint32      | 4            | Spark plugs status indicator (0 = failed plugs)                       |
| <i>oil_press_status</i> | —                 | uint32      | 4            | Oil pressure status indicator (0 = normal, 1 = low, 2 = full failure) |
| <i>fuel_pump_ok</i>     | —                 | uint32      | 4            | Fuel management status indicator                                      |
| <i>num_tanks</i>        | —                 | uint32      | 1            | Number of valid tanks   |
| <i>fuel_selector</i>    | —                 | uint32      | 8            | Fuel selector. (0 = off, 1 = on)                                      |
| <i>xfer_pump</i>        | —                 | uint32      | 5            | Specifies transfer from array value to tank specified by value        |
| <i>cross_feed</i>       | —                 | uint32      | 1            | Cross feed valve (0 = false, 1 = on)                                  |

**Programmatic Use****Block Parameter:** ShowEngineFuelOutputs**Type:** character vector**Values:** 'off' | 'on'

**Default:** 'off'

### Show landing gear outputs – Landing gear outputs

off (default) | on

Select this check box to include the landing gear outputs from the FlightGear net\_ctrl data packet.

#### Dependencies

Select this check box to enable these input ports.

### Signal Group 3: Landing gear outputs

| Name                       | Units | Type   | Width | Description   |
|----------------------------|-------|--------|-------|---|
| <i>brake_left</i>          | —     | double | 1     | Left brake pedal position pilot                                 |
| <i>brake_right</i>         | —     | double | 1     | Right brake pedal position pilot                                |
| <i>copilot_brake_left</i>  | —     | double | 1     | Left brake pedal position copilot                               |
| <i>copilot_brake_right</i> | —     | double | 1     | Right brake pedal position copilot                              |
| <i>brake_parking</i>       | —     | double | 1     | Brake parking position  |
| <i>gear_handle</i>         | —     | uint32 | 1     | Gear handle position (1 = gear handle down, 0 = gear handle up) |

#### Programmatic Use

**Block Parameter:** ShowLandingGearOutputs

**Type:** character vector

**Values:** 'off' | 'on'

**Default:** 'off'

### Show avionics outputs – Avionic outputs

off (default) | on

Select this check box to include the avionics outputs from the FlightGear net\_ctrl data packet.

#### Dependencies

Select this check box to enable these input ports.

### Signal Group 4: Avionics outputs

| Name                   | Units | Type   | Width | Description            |
|------------------------|-------|--------|-------|------------------------|
| <i>master_avionics</i> | —     | uint32 | 1     | Master avionics switch |
| <i>comm_1</i>          | Hz    | double | 1     | Comm 1 frequency       |
| <i>comm_2</i>          | Hz    | double | 1     | Comm 2 frequency       |
| <i>nav_1</i>           | Hz    | double | 1     | Nav 1 frequency        |
| <i>nav_2</i>           | Hz    | double | 1     | Nav 2 frequency        |

**Programmatic Use****Block Parameter:** ShowAvionicOutputs**Type:** character vector**Values:** 'off' | 'on'**Default:** 'off'**Show environment outputs — Environment outputs**

on (default) | off

Select this check box to include the environment outputs from the FlightGear net\_ctrl data packet.

**Dependencies**

Select this check box to enable these input ports.

**Signal Group 5: Environment outputs**

| Name                   | Units | Type   | Width | Description              |
|------------------------|-------|--------|-------|--------------------------|
| <i>wind_speed_kt</i>   | knot  | double | 1     | Wind speed               |
| <i>wind_dir_deg</i>    | deg   | double | 1     | Wind direction           |
| <i>turbulence_norm</i> | —     | double | 1     | Turbulence norm          |
| <i>temp_c</i>          | deg C | double | 1     | Ambient temperature      |
| <i>press_inhg</i>      | inHg  | double | 1     | Ambient pressure         |
| <i>hground</i>         | m     | double | 1     | Ground elevation         |
| <i>magvar</i>          | deg   | double | 1     | Local magnetic variation |
| <i>icing</i>           | -     | uint32 | 1     | Icing status             |

**Programmatic Use****Block Parameter:** ShowEnvironmentOutputs**Type:** character vector**Values:** 'off' | 'on'**Default:** 'on'**Sample time — Sample time**

1/30 (default) | scalar

Specify the sample time (-1 for inherited), as a scalar.

**Programmatic Use****Block Parameter:** SampleTime**Type:** character vector**Values:** scalar**Default:** '1/30'**See Also**

FlightGear Preconfigured 6DoF Animation | Generate Run Script | Pack net\_fdm Packet for FlightGear | Receive net\_ctrl Packet from FlightGear | Send net\_fdm Packet to FlightGear

**Topics**

“Flight Simulator Interface” on page 2-16

“Work with the Flight Simulator Interface” on page 2-20

**Introduced in R2012a**

# Velocity Conversion

Convert from velocity units to desired velocity units

**Library:** Aerospace Blockset / Utilities / Unit Conversions



## 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 port labels change based on the input and output units selected from the **Initial unit** and the **Final unit** lists.

## Ports

### Input

#### Port\_1 – Velocity

scalar | array

Velocity, specified as a scalar or array, in initial velocity units.

#### Dependencies

The input port label depends on the **Initial unit** setting.

Data Types: double

### Output

#### Port\_1 – Velocity

scalar | array

Velocity, returned as a scalar or array, in final velocity units.

#### Dependencies

The output port label depends on the **Final unit** setting.

Data Types: double

## Parameters

### Initial unit – Input units

ft/s (default) | m/s | km/s | in/s | km/h | mph | kts | ft/min

Input units, specified as:

|      |                   |
|------|-------------------|
| 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 |
| ft/min | Feet per minute         |

### Dependencies

The input port label depends on the **Initial unit** setting.

### Programmatic Use

**Block Parameter:** IU

**Type:** character vector

**Values:** 'm/s' | 'ft/s' | 'km/s' | 'in/s' | 'km/h' | 'mph' | 'kts' | 'ft/min'

**Default:** 'ft/s'

### Final unit – Output units

m/s (default) | ft/s | km/s | in/s | km/h | mph | kts | ft/min

Output units, specified as:

|        |                         |
|--------|-------------------------|
| 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 |
| ft/min | Feet per minute         |

### Dependencies

The output port label depends on the **Final unit** setting.

### Programmatic Use

**Block Parameter:** OU

**Type:** character vector

**Values:** 'm/s' | 'ft/s' | 'km/s' | 'in/s' | 'km/h' | 'mph' | 'kts' | 'ft/min'

**Default:** 'm/s'

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

**See Also**

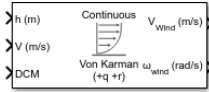
Acceleration Conversion | Angle Conversion | Angular Acceleration Conversion | Angular Velocity Conversion | Density Conversion | Force Conversion | Length Conversion | Mass Conversion | Pressure Conversion | Temperature Conversion

**Introduced before R2006a**



# Von Karman Wind Turbulence Model (Continuous)

Generate continuous wind turbulence with Von Kármán velocity spectra



## Library

Environment/Wind

## Description

The Von Kármán Wind Turbulence Model (Continuous) block uses the Von Kármán 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 and Military Handbook MIL-HDBK-1797.

According to the military references, turbulence is 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  $\Omega$  radians per meter, the circular frequency  $\omega$  is calculated by multiplying  $V$  by  $\Omega$ . The following table displays the component spectra functions:

|                     | MIL-F-8785C  | MIL-HDBK-1797   |
|---------------------|--|---|
| <b>Longitudinal</b> |  |   |
| $\Phi_u(\omega)$    | $\frac{2\sigma_u^2 L_u}{\pi V} \cdot \frac{1}{\left[1 + (1.339 L_u \frac{\omega}{V})^2\right]^{5/6}}$  | $\frac{2\sigma_u^2 L_u}{\pi V} \cdot \frac{1}{\left[1 + (1.339 L_u \frac{\omega}{V})^2\right]^{5/6}}$   |
| $\Phi_p(\omega)$    | $\frac{\sigma_w^2}{V L_w} \cdot \frac{0.8 \left(\frac{\pi L_w}{4b}\right)^{1/3}}{1 + \left(\frac{4b\omega}{\pi V}\right)^2}$                       | $\frac{\sigma_w^2}{2V L_w} \cdot \frac{0.8 \left(\frac{2\pi L_w}{4b}\right)^{1/3}}{1 + \left(\frac{4b\omega}{\pi V}\right)^2}$                      |
| <b>Lateral</b>      |  |   |
| $\Phi_v(\omega)$    | $\frac{\sigma_v^2 L_v}{\pi V} \cdot \frac{1 + \frac{8}{3} (1.339 L_v \frac{\omega}{V})^2}{\left[1 + (1.339 L_v \frac{\omega}{V})^2\right]^{11/6}}$ | $\frac{2\sigma_v^2 L_v}{\pi V} \cdot \frac{1 + \frac{8}{3} (2.678 L_v \frac{\omega}{V})^2}{\left[1 + (2.678 L_v \frac{\omega}{V})^2\right]^{11/6}}$ |
| $\Phi_r(\omega)$    | $\frac{\mp \left(\frac{\omega}{V}\right)^2}{1 + \left(\frac{3b\omega}{\pi V}\right)^2} \cdot \Phi_v(\omega)$                                       | $\frac{\mp \left(\frac{\omega}{V}\right)^2}{1 + \left(\frac{3b\omega}{\pi V}\right)^2} \cdot \Phi_v(\omega)$  |
| <b>Vertical</b>     |  |   |

|                  | MIL-F-8785C   | MIL-HDBK-1797  |
|------------------|---|--|
| $\Phi_w(\omega)$ | $\frac{\sigma_w^2 L_w}{\pi V} \cdot \frac{1 + \frac{8}{3}(1.339L_w \frac{\omega}{V})^2}{\left[1 + (1.339L_w \frac{\omega}{V})^2\right]^{11/6}}$ | $\frac{2\sigma_w^2 L_w}{\pi V} \cdot \frac{1 + \frac{8}{3}(2.678L_w \frac{\omega}{V})^2}{\left[1 + (2.678L_w \frac{\omega}{V})^2\right]^{11/6}}$ |
| $\Phi_q(\omega)$ | $\frac{\pm \left(\frac{\omega}{V}\right)^2}{1 + \left(\frac{4b\omega}{\pi V}\right)^2} \cdot \Phi_w(\omega)$                                    | $\frac{\pm \left(\frac{\omega}{V}\right)^2}{1 + \left(\frac{4b\omega}{\pi V}\right)^2} \cdot \Phi_w(\omega)$                                     |

The variable  $b$  represents the aircraft wingspan. The variables  $L_u, L_v, L_w$  represent the turbulence scale lengths. The variables  $\sigma_u, \sigma_v, \sigma_w$  represent the turbulence intensities:

The spectral density definitions of turbulence angular rates are defined in the references as three variations, which are displayed in the following table:

$$\begin{array}{lll}
 p_g = \frac{\partial w_g}{\partial y} & q_g = \frac{\partial w_g}{\partial x} & r_g = -\frac{\partial v_g}{\partial x} \\
 p_g = \frac{\partial w_g}{\partial y} & q_g = \frac{\partial w_g}{\partial x} & r_g = \frac{\partial v_g}{\partial x} \\
 p_g = -\frac{\partial w_g}{\partial y} & q_g = -\frac{\partial w_g}{\partial x} & r_g = \frac{\partial v_g}{\partial x}
 \end{array}$$

The variations affect only the vertical ( $q_g$ ) and lateral ( $r_g$ ) turbulence angular rates.

Keep in mind that the longitudinal turbulence angular rate spectrum,  $\Phi_p(\omega)$ , is a rational function. The rational function is derived from curve-fitting a complex algebraic function, not the vertical turbulence velocity spectrum,  $\Phi_w(\omega)$ , multiplied by a scale factor. Because the turbulence angular rate spectra contribute less to the aircraft gust response than the turbulence velocity spectra, it may explain the variations in their definitions.

The variations lead to the following combinations of vertical and lateral turbulence angular rate spectra.

| Vertical          | Lateral           |
|-------------------|-------------------|
| $\Phi_q(\omega)$  | $-\Phi_r(\omega)$ |
| $\Phi_q(\omega)$  | $\Phi_r(\omega)$  |
| $-\Phi_q(\omega)$ | $\Phi_r(\omega)$  |

To generate a signal with the correct characteristics, a unit variance, band-limited white noise signal is passed through forming filters. The forming filters are approximations of the Von Kármán velocity spectra which are valid in a range of normalized frequencies of less than 50 radians. These filters can be found in both the Military Handbook MIL-HDBK-1797 and the reference by Ly and Chan.

The following two tables display the transfer functions.

|                     | MIL-F-8785C |
|---------------------|-------------|
| <b>Longitudinal</b> |             |

| <b>MIL-F-8785C</b>   |  |
|----------------------|--|
| $H_u(s)$             | $\frac{\sigma_u \sqrt{\frac{2}{\pi}} \cdot \frac{L_u}{V} \left(1 + 0.25 \frac{L_u}{V} s\right)}{1 + 1.357 \frac{L_u}{V} s + 0.1987 \left(\frac{L_u}{V}\right)^2 s^2}$  |
| $H_p(s)$             | $\sigma_w \sqrt{\frac{0.8}{V}} \cdot \frac{\left(\frac{\pi}{4b}\right)^{1/6}}{L_w^{1/3} \left(1 + \left(\frac{4b}{\pi V}\right) s\right)}$   |
| <b>Lateral</b>       |  |
| $H_v(s)$             | $\frac{\sigma_v \sqrt{\frac{1}{\pi}} \cdot \frac{L_v}{V} \left(1 + 2.7478 \frac{L_v}{V} s + 0.3398 \left(\frac{L_v}{V}\right)^2 s^2\right)}{1 + 2.9958 \frac{L_v}{V} s + 1.9754 \left(\frac{L_v}{V}\right)^2 s^2 + 0.1539 \left(\frac{L_v}{V}\right)^3 s^3}$       |
| $H_r(s)$             | $\frac{\mp \frac{s}{V}}{\left(1 + \left(\frac{3b}{\pi V}\right) s\right)} \cdot H_v(s)$  |
| <b>Vertical</b>      |  |
| $H_w(s)$             | $\frac{\sigma_w \sqrt{\frac{1}{\pi}} \cdot \frac{L_w}{V} \left(1 + 2.7478 \frac{L_w}{V} s + 0.3398 \left(\frac{L_w}{V}\right)^2 s^2\right)}{1 + 2.9958 \frac{L_w}{V} s + 1.9754 \left(\frac{L_w}{V}\right)^2 s^2 + 0.1539 \left(\frac{L_w}{V}\right)^3 s^3}$       |
| $H_q(s)$             | $\frac{\pm \frac{s}{V}}{\left(1 + \left(\frac{4b}{\pi V}\right) s\right)} \cdot H_w(s)$  |
| <b>MIL-HDBK-1797</b> |  |
| <b>Longitudinal</b>  |  |
| $H_u(s)$             | $\frac{\sigma_u \sqrt{\frac{2}{\pi}} \cdot \frac{L_u}{V} \left(1 + 0.25 \frac{L_u}{V} s\right)}{1 + 1.357 \frac{L_u}{V} s + 0.1987 \left(\frac{L_u}{V}\right)^2 s^2}$  |
| $H_p(s)$             | $\sigma_w \sqrt{\frac{0.8}{V}} \cdot \frac{\left(\frac{\pi}{4b}\right)^{1/6}}{(2L_w)^{1/3} \left(1 + \left(\frac{4b}{\pi V}\right) s\right)}$  |
| <b>Lateral</b>       |  |
| $H_v(s)$             | $\frac{\sigma_v \sqrt{\frac{1}{\pi}} \cdot \frac{2L_v}{V} \left(1 + 2.7478 \frac{2L_v}{V} s + 0.3398 \left(\frac{2L_v}{V}\right)^2 s^2\right)}{1 + 2.9958 \frac{2L_v}{V} s + 1.9754 \left(\frac{2L_v}{V}\right)^2 s^2 + 0.1539 \left(\frac{2L_v}{V}\right)^3 s^3}$ |
| $H_r(s)$             | $\frac{\mp \frac{s}{V}}{\left(1 + \left(\frac{3b}{\pi V}\right) s\right)} \cdot H_v(s)$  |

| <b>MIL-HDBK-1797</b> |  |
|----------------------|--|
| <b>Vertical</b>      |  |
| $H_w(s)$             | $\frac{\sigma_w \sqrt{\frac{1}{\pi}} \cdot \frac{2L_w}{V} \left( 1 + 2.7478 \frac{2L_w}{V} s + 0.3398 \left( \frac{2L_w}{V} \right)^2 s^2 \right)}{1 + 2.9958 \frac{2L_w}{V} s + 1.9754 \left( \frac{2L_w}{V} \right)^2 s^2 + 0.1539 \left( \frac{2L_w}{V} \right)^3 s^3}$ |
| $H_q(s)$             | $\frac{\pm \frac{s}{V}}{\left( 1 + \left( \frac{4b}{\pi V} \right) s \right)} \cdot H_w(s)$  |

Divided into two distinct regions, the turbulence scale lengths and intensities are functions of altitude.

**Note** The same transfer functions result after evaluating the turbulence scale lengths. The differences in turbulence scale lengths and turbulence transfer functions balance offset.

**Low-Altitude Model (Altitude < 1000 feet)**

According to the military references, the turbulence scale lengths at low altitudes, where  $h$  is the altitude in feet, are represented in the following table:

| <b>MIL-F-8785C</b>                                | <b>MIL-HDBK-1797</b>                               |
|---|--|
| $L_w = h$   | $2L_w = h$   |
| $L_u = L_v = \frac{h}{(0.177 + 0.000823h)^{1.2}}$ | $L_u = 2L_v = \frac{h}{(0.177 + 0.000823h)^{1.2}}$ |

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.

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

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

The turbulence axes orientation in this region is defined as follows:

- Longitudinal turbulence velocity,  $u_g$ , aligned along the horizontal relative mean wind vector
- Vertical turbulence velocity,  $w_g$ , aligned with vertical.

At this altitude range, the output of the block is transformed into body coordinates.

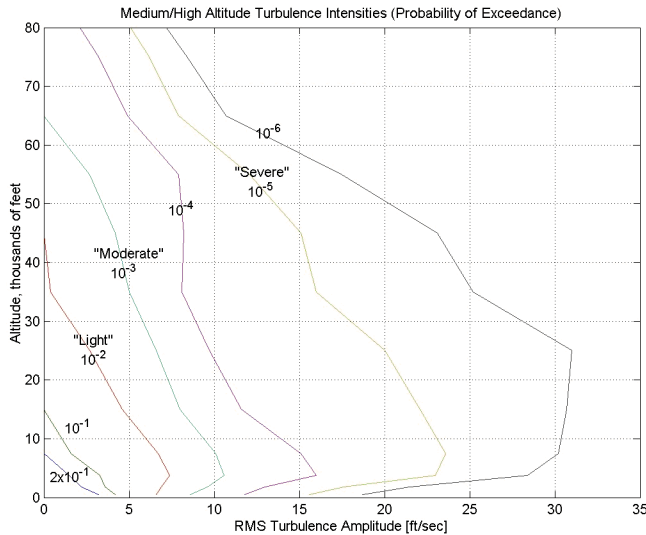
**Medium/High Altitudes (Altitude > 2000 feet)**

For medium to high altitudes the turbulence scale lengths and intensities are based on the assumption that the turbulence is isotropic. In the military references, the scale lengths are represented by the following equations:

|                                     |                                       |
|-------------------------------------|---------------------------------------|
| <b>MIL-F-8785C</b>                  | <b>MIL-HDBK-1797</b>                  |
| $L_u = L_v = L_w = 2500 \text{ ft}$ | $L_u = 2L_v = 2L_w = 2500 \text{ ft}$ |

The turbulence intensities are determined from a lookup table that provides the turbulence intensity as a function of altitude and the probability of the turbulence intensity being exceeded. The relationship of the turbulence intensities is represented in the following equation:  $\sigma_u = \sigma_v = \sigma_w$ .

The turbulence axes orientation in this region is defined as being aligned with the body coordinates:



**Between Low and Medium/High Altitudes (1000 feet < Altitude < 2000 feet)**

At altitudes between 1000 feet and 2000 feet, the turbulence velocities and turbulence angular rates are determined by linearly interpolating between the value from the low altitude model at 1000 feet transformed from mean horizontal wind coordinates to body coordinates and the value from the high altitude model at 2000 feet in body coordinates.

**Parameters**

**Units**

Define the units of wind speed due to the turbulence.

| Units                      | Wind Velocity | Altitude | Air Speed     |
|----------------------------|---------------|----------|---------------|
| Metric (MKS)               | Meters/second | Meters   | Meters/second |
| English (Velocity in ft/s) | Feet/second   | Feet     | Feet/second   |
| English (Velocity in kts)  | Knots         | Feet     | Knots         |

**Specification**

Define which military reference to use. This affects the application of turbulence scale lengths in the lateral and vertical directions

**Model type**

Select the wind turbulence model to use:

|                               |   |
|-------------------------------|---|
| Continuous Von Karman (+q -r) | Use continuous representation of Von Kármán velocity spectra with positive vertical and negative lateral angular rates spectra. |
| Continuous Von Karman (+q +r) | Use continuous representation of Von Kármán velocity spectra with positive vertical and lateral angular rates spectra.          |
| Continuous Von Karman (-q +r) | Use continuous representation of Von Kármán velocity spectra with negative vertical and positive lateral angular rates spectra. |
| Continuous Dryden (+q -r)     | Use continuous representation of Dryden velocity spectra with positive vertical and negative lateral angular rates spectra.     |
| Continuous Dryden (+q +r)     | Use continuous representation of Dryden velocity spectra with positive vertical and lateral angular rates spectra.              |
| Continuous Dryden (-q +r)     | Use continuous representation of Dryden velocity spectra with negative vertical and positive lateral angular rates spectra.     |
| Discrete Dryden (+q -r)       | Use discrete representation of Dryden velocity spectra with positive vertical and negative lateral angular rates spectra.       |
| Discrete Dryden (+q +r)       | Use discrete representation of Dryden velocity spectra with positive vertical and lateral angular rates spectra.                |
| Discrete Dryden (-q +r)       | Use discrete representation of Dryden velocity spectra with negative vertical and positive lateral angular rates spectra.       |

The Continuous Von Kármán selections conform to the transfer function descriptions.

**Wind speed at 6 m defines the low altitude intensity**

The measured wind speed at a height of 20 feet (6 meters) provides the intensity for the low-altitude turbulence model.

**Wind direction at 6 m (degrees clockwise from north)**

The measured wind direction at a height of 20 feet (6 meters) is an angle to aid in transforming the low-altitude turbulence model into a body coordinates.

**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 the military references, a figure of 1750 feet is recommended for the longitudinal turbulence scale length of the Dryden spectra.

---

**Note** An alternate scale length value changes the power spectral density asymptote and gust load.

---

**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**

| Input  | Dimension Type | Description                                    |
|--------|----------------|--|
| First  | scalar         | Contains the altitude in units selected.       |
| Second | scalar         | Contains the aircraft speed in units selected. |
| Third  | 3-by-3 matrix  | Contains a direction cosine matrix.            |

| Output | Dimension Type       | Description   |
|--------|----------------------|---|
| First  | Three-element signal | Contains the turbulence velocities, in the selected units.    |
| Second | Three-element signal | Contains the turbulence angular rates, in radians per second. |

**Assumptions and Limitations**

The frozen turbulence field assumption is valid for the cases of mean-wind velocity and the root-mean-square turbulence velocity, or intensity, are small relative to the aircraft's ground speed.

The turbulence model describes an average of all conditions for clear air turbulence because the following factors are not incorporated into the model:

- Terrain roughness
- Lapse rate
- Wind shears

- Mean wind magnitude
- Other meteorological factions (except altitude)

## References

U.S. Military Handbook MIL-HDBK-1797, 19 December 1997.

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

Chalk, C., Neal, P., Harris, T., Pritchard, F., Woodcock, R., "Background Information and User Guide for MIL-F-8785B(ASG), `Military Specification-Flying Qualities of Piloted Airplanes'," AD869856, Cornell Aeronautical Laboratory, August 1969.

Hoblitt, F., *Gust Loads on Aircraft: Concepts and Applications*, AIAA Education Series, 1988.

Ly, U., Chan, Y., "Time-Domain Computation of Aircraft Gust Covariance Matrices," AIAA Paper 80-1615, Atmospheric Flight Mechanics Conference, Danvers, MA., August 11-13, 1980.

McRuer, D., Ashkenas, I., Graham, D., *Aircraft Dynamics and Automatic Control*, Princeton University Press, July 1990.

Moorhouse, D., Woodcock, R., "Background Information and User Guide for MIL-F-8785C, `Military Specification-Flying Qualities of Piloted Airplanes'," ADA119421, Flight Dynamic Laboratory, July 1982.

McFarland, R., "A Standard Kinematic Model for Flight Simulation at NASA-Ames," NASA CR-2497, Computer Sciences Corporation, January 1975.

Tatom, F., Smith, R., Fichtl, G., "Simulation of Atmospheric Turbulent Gusts and Gust Gradients," AIAA Paper 81-0300, Aerospace Sciences Meeting, St. Louis, MO., January 12-15, 1981.

Yeager, J., "Implementation and Testing of Turbulence Models for the F18-HARV Simulation," NASA CR-1998-206937, Lockheed Martin Engineering & Sciences, March 1998.

## See Also

Dryden Wind Turbulence Model (Continuous)

Dryden Wind Turbulence Model (Discrete)

Discrete Wind Gust Model

Wind Shear Model

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

### Introduced in R2006b



# WGS84 Gravity Model

Implement 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 the **Type of gravity model** parameter.

The WGS84 Gravity Model block icon displays the input and output units selected from the **Units** list.

## Parameters

### 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:

| Units        | Height | Gravity                   |
|--------------|--------|---------------------------|
| Metric (MKS) | Meters | Meters per second squared |
| English      | Feet   | Feet per second squared   |

### Exclude Earth's atmosphere

Select for the value for the Earth's gravitational field to exclude the mass of the atmosphere.

Clear for the value for the Earth's gravitational field to include the mass of the atmosphere.

This option is available only 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. 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 available only with **Type of gravity model WGS84 Close Approximation** or **WGS84 Exact**.

### Input Julian date

When selected, another input port, JD, appears on the block mask. Select this check box if you want to manually specify the Julian date for the block. Otherwise, the block calculates the Julian date given the values of **Month**, **Day**, and **Year**. The year must be after January 1, 2000 (2451545). Selecting this block disables the **Month**, **Day**, and **Year** parameters.

### Month

Specifies the month used to calculate Julian centuries from Epoch J2000.0.

This option is available only with **Type of gravity model WGS84 Close Approximation** or **WGS84 Exact** and only when **Preprocessing reference frame** is selected. It is disabled if you select **Input Julian Date**.

### Day

Specifies the day used to calculate Julian centuries from Epoch J2000.0.

This option is available only with **Type of gravity model WGS84 Close Approximation** or **WGS84 Exact** and only when **Preprocessing reference frame** is selected. It is disabled if you select **Input Julian Date**.

### Year

Specifies the year used to calculate Julian centuries from Epoch J2000.0. The year must be 2000 or greater.

This option is available only with **Type of gravity model WGS84 Close Approximation** or **WGS84 Exact** and only when **Preprocessing reference frame** is selected. It is disabled if you select **Input Julian Date**.

### 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 available only with **Type of gravity model WGS84 Close Approximation** or **WGS84 Exact**.

### Action for out-of-range input

Specify if out-of-range input invokes a warning, error, or no action.

## Inputs and Outputs

| Input | Dimension Type                          | Description   |
|-------|---|---|
| First | Three-element vector or $M$ -by-3 array | Contains the position in geodetic latitude, longitude and altitude, with units in degrees, degrees, and selected units of length respectively. Altitude must be less than 20,000.0 m (approximately 65,620.0 feet). |

| Input                | Dimension Type                             | Description   |
|----------------------|--|---|
| Second<br>(Optional) | Scalar                                     | Contains the Julian centuries that you specified.   |
| Output               | Dimension Type                             | Description   |
| Output               | Three-element vector<br>or $M$ -by-3 array | Gravity in the north-east-down (NED) coordinate system. The Taylor Series and Close Approximation methods output only normal gravity (down in the NED Coordinate system). The Exact method outputs both normal and tangent gravity (down and north in the NED coordinate system). |

## 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 is sufficient near the surface when submicrogal precision is not necessary. At medium and high geodetic heights, it is less accurate.

The WGS84 Close Approximation model gives results with submicrogal precision.

To predict and determine a satellite orbit with high accuracy, use the EGM96 through degree and order 70.

## Examples

See the Environment Models subsystem in the Airframe subsystem of the `aeroblk_HL20` model for an example of this block.

## Reference

[1] NIMA TR8350.2: "Department of Defense World Geodetic System 1984, Its Definition and Relationship with Local Geodetic Systems."

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

**Introduced before R2006a**

## Wind Angles to Direction Cosine Matrix

Convert wind angles to direction cosine matrix

**Library:** Aerospace Blockset / Utilities / Axes Transformations



### Description

The Wind Angles to Direction Cosine Matrix block converts three wind rotation angles into a 3-by-3 direction cosine matrix (DCM). The DCM matrix performs the coordinate transformation of a vector in earth axes ( $ox_0, oy_0, oz_0$ ) into a vector in wind axes ( $ox_3, oy_3, oz_3$ ). For more information on the direction cosine matrix, see “Algorithms” on page 4-654.

This implementation generates a flight path angle that lies between  $\pm 90$  degrees, and bank and heading angles that lie between  $\pm 180$  degrees.

### Ports

#### Input

**$\mu \ \gamma \ \chi$  — Wind rotation angles**

3-by-1 vector

Wind rotation angles, specified as a 3-by-1 vector, in radians.

Data Types: `double`

#### Output

**$DCM_{we}$  — Direction cosine matrix**

3-by-3 matrix

Direction cosine matrix, returned as a 3-by-3 matrix.

Data Types: `double`

### Algorithms

The DCM matrix performs the coordinate transformation of a vector in earth axes ( $ox_0, oy_0, oz_0$ ) into a vector in wind axes ( $ox_3, oy_3, oz_3$ ). The order of the axis rotations required to bring this about is:

- 1 A rotation about  $oz_0$  through the heading angle ( $\chi$ ) to axes ( $ox_1, oy_1, oz_1$ )
- 2 A rotation about  $oy_1$  through the flight path angle ( $\gamma$ ) to axes ( $ox_2, oy_2, oz_2$ )
- 3 A rotation about  $ox_2$  through the bank angle ( $\mu$ ) to axes ( $ox_3, oy_3, oz_3$ )

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

$$\begin{bmatrix} ox_3 \\ oy_3 \\ oz_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\mu & \sin\mu \\ 0 & -\sin\mu & \cos\mu \end{bmatrix} \begin{bmatrix} \cos\gamma & 0 & -\sin\gamma \\ 0 & 1 & 0 \\ \sin\gamma & 0 & \cos\gamma \end{bmatrix} \begin{bmatrix} \cos\chi & \sin\chi & 0 \\ -\sin\chi & \cos\chi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} ox_0 \\ oy_0 \\ oz_0 \end{bmatrix}$$

Combining the three axis transformation matrices defines the following DCM:

$$DCM_{we} = \begin{bmatrix} \cos\gamma\cos\chi & \cos\gamma\sin\chi & -\sin\gamma \\ (\sin\mu\sin\gamma\cos\chi - \cos\mu\sin\chi) & (\sin\mu\sin\gamma\sin\chi + \cos\mu\cos\chi) & \sin\mu\cos\gamma \\ (\cos\mu\sin\gamma\cos\chi + \sin\mu\sin\chi) & (\cos\mu\sin\gamma\sin\chi - \sin\mu\cos\chi) & \cos\mu\cos\gamma \end{bmatrix}$$

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

### See Also

Direction Cosine Matrix Body to Wind | Direction Cosine Matrix to Rotation Angles | Direction Cosine Matrix to Wind Angles | Rotation Angles to Direction Cosine Matrix

**Introduced before R2006a**

## Wind Angular Rates

Calculate wind angular rates from body angular rates, angle of attack, sideslip angle, rate of change of angle of attack, and rate of change of sideslip

**Library:** Aerospace Blockset / Flight Parameters



### Description

The Wind Angular Rates block supports the equations of motion in wind-fixed frame models by calculating the wind-fixed angular rates ( $p_w$ ,  $q_w$ ,  $r_w$ ). For more information on the equation used for the calculation, see “Algorithms” on page 4-657.

### Ports

#### Input

##### $\alpha$ $\beta$ — Angles of attack and sideslip

2-by-1 vector

Angle of attack and sideslip, specified as a 2-by-1 vector, in radians.

Data Types: double

##### $d\alpha/dt$ $d\beta/dt$ — Rates of change

2-by-1 vector

Rate of change of the angle of attack and rate of change of the sideslip, specified as a 2-by-1 vector, in radians per second.

Data Types: double

##### $\omega$ — Body angular rates

three-element vector

Body angular rates, specified as a three-element vector, in radians per second.

Data Types: double

#### Output

##### $\omega_w$ — Wind angular rates

three-element vector

Wind angular rates, returned as a three-element vector, in radians per second.

Data Types: double

## Algorithms

The body-fixed angular rates ( $p_b$ ,  $q_b$ ,  $r_b$ ), angle of attack ( $\alpha$ ), sideslip angle ( $\beta$ ), rate of change of angle of attack ( $\dot{\alpha}$ ), and rate of change of sideslip ( $\dot{\beta}$ ) are related to the wind-fixed angular rates as illustrated in the following equation:

$$\begin{bmatrix} p_w \\ q_w \\ r_w \end{bmatrix} = \begin{bmatrix} \cos\alpha\cos\beta & \sin\beta & \sin\alpha\cos\beta \\ -\cos\alpha\sin\beta & \cos\beta & -\sin\alpha\sin\beta \\ -\sin\alpha & 0 & \cos\alpha \end{bmatrix} \begin{bmatrix} p_b - \dot{\beta}\sin\alpha \\ q_b - \dot{\alpha} \\ r_b + \dot{\beta}\cos\alpha \end{bmatrix}$$

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

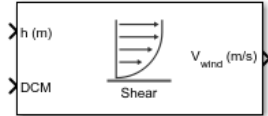
### See Also

3DOF (Body Axes) | 6DOF Wind (Quaternion) | 6DOF Wind (Wind Angles) | Custom Variable Mass 3DOF (Body Axes) | Custom Variable Mass 6DOF Wind (Quaternion) | Custom Variable Mass 6DOF Wind (Wind Angles) | Simple Variable Mass 3DOF (Body Axes) | Simple Variable Mass 6DOF Wind (Quaternion) | Simple Variable Mass 6DOF Wind (Wind Angles)

**Introduced before R2006a**

## Wind Shear Model

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 flat Earth 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.

## Parameters

### Units

Define the units of wind shear.

| Units                      | Wind          | Altitude |
|----------------------------|---------------|----------|
| Metric (MKS)               | Meters/second | Meters   |
| English (Velocity in ft/s) | Feet/second   | Feet     |
| English (Velocity in kts)  | Knots         | Feet     |

### Flight phase

Select flight phase:

- Category C — Terminal Flight Phases



- Other

### Wind speed at 6 m (20 feet) altitude (m/s, f/s, or knots)

The measured wind speed at an altitude of 20 feet (6 m) above the ground.

### Wind direction at 6 m (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.

## Inputs and Outputs

| Input  | Dimension Type                 | Description   |
|--------|--------------------------------|---|
| First  |                                | Contains the altitude in units selected.                                    |
| Second | 3-by-3 direction cosine matrix |   |
| Output | Dimension Type                 | Description   |
| First  | 3-by-1 vector                  | Contains the mean wind speed in the body axes frame, in the selected units. |

## Examples

See Wind Shear Model in aeroblk\_HL20 for an example of this block.

## Reference

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

## See Also

Discrete Wind Gust Model

Dryden Wind Turbulence Model (Continuous)

Dryden Wind Turbulence Model (Discrete)

Von Karman Wind Turbulence Model (Continuous)

## Extended Capabilities

### C/C++ Code Generation

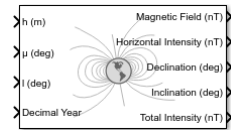
Generate C and C++ code using Simulink® Coder™.

Introduced before R2006a

## World Magnetic Model

Calculate Earth's magnetic field at specific location and time using World Magnetic Model

**Library:** Aerospace Blockset / Environment / Gravity



### Description

The World Magnetic Model block implements the mathematical representation of the National Geospatial Intelligence Agency (NGA) World Magnetic Model. The World Magnetic Model block calculates the Earth magnetic field vector, horizontal intensity, declination, inclination, and total intensity at a specified location and time. The reference frame is north-east-down (NED).

---

**Note** You cannot use this block to model the Earth magnetic field above an altitude of 1,000,000 meters.

---

### Limitations

All specifications have these limitations:

- The internal calculation of decimal year does not take into account local time or leap seconds.
- The specifications describe only the long-wavelength spatial magnetic fluctuations in the Earth's core. Intermediate and short-wavelength fluctuations, contributed from the crustal field (the mantle and crust), are not included. Also, the substantial fluctuations of the geomagnetic field, which occur constantly during magnetic storms and almost constantly in the disturbance field (auroral zones), are not included.

In addition, each specification has these limitations:

- WMM2015v2 supersedes WMM2015(v1). Consider replacing WMM2015(v1) with WMM2015v2 when used for navigation and other systems. WMM2015v2 was released by National Oceanic and Atmospheric Administration (NOAA) in February 2019 to correct performance degradation issues in the Arctic region for January 1, 2015, to December 31, 2019. Therefore, it is still acceptable to use WMM2015(v1) in systems below 55 degrees latitude in the Northern hemisphere.
- The WMM2020 specification produces data that is reliable five years after the epoch of the model, which is January 1, 2020.
- The WMM2015 specification produces data that is reliable five years after the epoch of the model, which is January 1, 2015.
- The WMM2010 specification produces data that is reliable five years after the epoch of the model, which is January 1, 2010.
- The WMM2005 specification produces data that is reliable five years after the epoch of the model, which is January 1, 2005.
- The WMM2000 specification produces data that is reliable five years after the epoch of the model, which is January 1, 2000.

## Ports

### Input

#### **h (m) – Height**

scalar

Height, specified as a scalar, in selected units.

Data Types: double

#### **$\mu$ (deg) – Latitude**

scalar

Latitude, specified as a scalar, in degrees.

Data Types: double

#### **l (deg) – Longitude**

scalar

Longitude, specified as a scalar, in degrees.

Data Types: double

#### **Decimal Year – Desired year**

scalar

Desired year in a decimal format to include any fraction of the year that has already passed. The value is the current year plus the number of days that have passed in this year divided by 365.

For example, to calculate the decimal year, `dyear`, for March 21, 2015:

```
dyear=decyear('21-March-2015','dd-mmm-yyyy')
```

```
dyear =  
    2.0152e+03
```

Data Types: double

### Output

#### **Magnetic Field (nT) – Magnetic field**

vector

Magnetic field, returned as a vector, in selected units.

Data Types: double

#### **Horizontal Intensity (nT) – Horizontal intensity**

scalar

Horizontal intensity, returned as a scalar, in specified units.

Data Types: double

#### **Declination (deg) – Declination**

scalar

Declination, returned as a scalar, in degrees.

Data Types: double

### **Inclination (deg) – Inclination**

scalar

Inclination, returned as a scalar, in degrees.

Data Types: double

### **Total Intensity (nT) – Total intensity**

scalar

Total intensity, returned as a scalar, in selected units.

Data Types: double

## **Parameters**

### **WMM coefficients – World Magnetic Model coefficient file**

WMM2020 (2020-2025) (default) | WMM2015 V2 (2015-2020) | WMM2000 (2000-2005) | WMM2005 (2005-2010) | WMM2010 (2010-2015) | WMM2015 V1 (2015-2020) | Custom

World Magnetic Model coefficient file, selected from the list.

- WMM2000 (2000-2005) – World Magnetic Model 2000 coefficient file
- WMM2005 (2005-2010) – World Magnetic Model 2005 coefficient file
- WMM2010 (2010-2015) – World Magnetic Model 2010 coefficient file
- WMM2015 V1 (2015-2020) – World Magnetic Model 2015(v1) coefficient file
- WMM2015 V2 (2015-2020) – World Magnetic Model 2015v2 coefficient file
- WMM2020 (2020-2025) – World Magnetic Model 2020 coefficient file
- Custom – Specify your own World Magnetic Model coefficient file. You can download a World Magnetic Model coefficient file from The NOAA World Magnetic Model.

### **Dependencies**

Selecting Custom enables the **Custom .COF file** parameter.

#### **Programmatic Use**

**Block Parameter:** model

**Type:** character vector

**Values:** 'WMM2020 (2020-2025)' | 'WMM2015 V2 (2015-2020)' | 'WMM2000 (2000-2005)' | 'WMM2005 (2005-2010)' | 'WMM2010 (2010-2015)' | 'WMM2015 V1 (2015-2020)' | 'Custom'

**Default:** 'WMM2020 (2020-2025)'

### **Custom .COF file – Custom World Magnetic Model coefficient file**

'WMM2020COF' (default) | any coefficient file name

World Magnetic Model coefficient file, downloaded from The NOAA World Magnetic Model. For example, if you want to download a coefficient file not yet listed in the **WMM coefficients** list.

**Dependencies**

To enable this parameter, select Custom for the **WMM coefficients** parameter.

**Programmatic Use**

**Block Parameter:** customFile

**Type:** character vector

**Values:** 'WMM2020.COF' | any coefficient file name

**Default:** 'WMM2020.COF'

**Units – Input and output units**

Metric (MKS) (default) | English

Input and output units:

| Units        | Height | Magnetic Field | Horizontal Intensity | Total Intensity |
|--------------|--------|----------------|----------------------|-----------------|
| Metric (MKS) | Meters | Nanotesla      | Nanotesla            | Nanotesla       |
| English      | Feet   | Nanogauss      | Nanogauss            | Nanogauss       |

**Programmatic Use**

**Block Parameter:** units

**Type:** character vector

**Values:** 'Metric (MKS)' | 'English'

**Default:** 'Metric (MKS)'

**Input decimal year – Input decimal year**

on (default) | off

When selected, the decimal year is an input for the World Magnetic Model block. Otherwise, specify a date using the dialog parameters of **Month**, **Day**, and **Year**.

**Programmatic Use**

**Block Parameter:** time\_in

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'on'

**Month – Input month**

January (default) | February | March | April | May | June | July | August | September | October | November | December

Month to calculate decimal year.

**Programmatic Use**

**Block Parameter:** month

**Type:** character vector

**Values:** 'January' | 'February' | 'March' | 'April' | 'May' | 'June' | 'July' | 'August' | 'September' | 'October' | 'November' | 'December'

**Default:** 'January'

**Day – Input day**

1 (default) | 1 to 31

Day to calculate decimal year.

**Programmatic Use****Block Parameter:** day**Type:** character vector**Values:** '1' to '31'**Default:** '1'**Year — Input year**

2020 (default) | any year

Year to calculate decimal year.

**Programmatic Use****Block Parameter:** year**Type:** character vector**Values:** any year**Default:** '2020'**Action for out-of-range input — Out-of-range action**

Error (default) | Warning | None

Specify if out-of-range input invokes a warning, error, or no action.

**Programmatic Use****Block Parameter:** action**Type:** character vector**Values:** 'Error' | 'Warning' | 'None'**Default:** 'Error'**Output horizontal intensity — Output horizontal intensity**

on (default) | off

When selected, the horizontal intensity is output.

**Programmatic Use****Block Parameter:** h\_out**Type:** character vector**Values:** 'on' | 'off'**Default:** 'on'**Output declination — Output declination**

on (default) | off

When selected, the declination or the angle between true north and the magnetic field vector (positive eastwards) is output.

**Programmatic Use****Block Parameter:** dec\_out**Type:** character vector**Values:** 'on' | 'off'**Default:** 'on'**Output inclination — Output inclination**

on (default) | off

When selected, the inclination or the angle between the horizontal plane and the magnetic field vector (positive downwards) is output.

**Programmatic Use****Block Parameter:** inc\_out**Type:** character vector**Values:** 'on' | 'off'**Default:** 'on'**Output total intensity – Output total intensity**

on (default) | off

When selected, the total intensity is output.

**Programmatic Use****Block Parameter:** ti\_out**Type:** character vector**Values:** 'on' | 'off'**Default:** 'on'**Extended Capabilities****C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**See Also**

International Geomagnetic Reference Field 12 | decyear

**External Websites**

<https://www.ngdc.noaa.gov/geomag/WMM/DoDWMM.shtml>

**Introduced in R2019b**

# World Magnetic Model 2000

Calculate Earth's magnetic field at specific location and time using World Magnetic Model 2000 (WMM2000)

## Library

Environment/Gravity

## Description

The WMM2000 block implements the mathematical representation of the National Geospatial Intelligence Agency (NGA) World Magnetic Model 2000. The WMM2000 block calculates the Earth's magnetic field vector, horizontal intensity, declination, inclination, and total intensity at a specified location and time. The reference frame is north-east-down (NED).

## Parameters

### Units

Specifies the input and output units:

| Units        | Height | Magnetic Field | Horizontal Intensity | Total Intensity |
|--------------|--------|----------------|----------------------|-----------------|
| Metric (MKS) | Meters | Nanotesla      | Nanotesla            | Nanotesla       |
| English      | Feet   | Nanogauss      | Nanogauss            | Nanogauss       |

### Input decimal year

When selected, the decimal year is an input for the World Magnetic Model 2000 block. Otherwise, a date must be specified using the dialog parameters of **Month**, **Day**, and **Year**.

### Month

Specifies the month used to calculate decimal year.

### Day

Specifies the day used to calculate decimal year.

### Year

Specifies the year used to calculate decimal year.

### Action for out of range input

Specify if out-of-range input invokes a warning, error or no action.

### Output horizontal intensity

When selected, the horizontal intensity is output.

### Output declination

When selected, the declination, the angle between true north and the magnetic field vector (positive eastwards), is output.

### Output inclination

When selected, the inclination, the angle between the horizontal plane and the magnetic field vector (positive downwards), is output.



**Output total intensity**

When selected, the total intensity is output.

**Inputs and Outputs**

| Input                | Dimension Type | Description   |
|----------------------|----------------|---|
| First                |                | Contains the height, in selected units.   |
| Second               |                | Contains the latitude in degrees.   |
| Third                |                | Contains the longitude in degrees.  |
| Fourth<br>(Optional) |                | Contains the desired year in a decimal format to include any fraction of the year that has already passed. The value is the current year plus the number of days that have passed in this year divided by 365.<br><br>The following code illustrates how to calculate the decimal year, <code>dyear</code> , for March 21, 2005:<br><br><pre>%%BEGIN CODE%% dyear=decyear('21-March-2005','dd-mmm-yyyy') %%END CODE%%</pre> |

| Output               | Dimension Type | Description   |
|----------------------|----------------|---|
| First                |                | Contains the magnetic field vector in selected units. |
| Second<br>(Optional) |                | Contains the horizontal intensity in selected units.  |
| Third<br>(Optional)  |                | Contains the declination in degrees.                  |
| Fourth<br>(Optional) |                | Contains the inclination in degrees.                  |
| Fifth (Optional)     |                | Contains the total intensity in selected units.       |

**Limitations**

The WMM2000 specification produces data that is reliable five years after the epoch of the model, which is January 1, 2000.

The internal calculation of decimal year does not take into account local time or leap seconds.

The WMM2000 specification describes only the long-wavelength spatial magnetic fluctuations due to the Earth's core. Intermediate and short-wavelength fluctuations, contributed from the crustal field (the mantle and crust), are not included. Also, the substantial fluctuations of the geomagnetic field, which occur constantly during magnetic storms and almost constantly in the disturbance field (auroral zones), are not included.

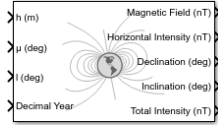
**Reference**

Macmillan, S. and J. M. Quinn, 2000. "The Derivation of the World Magnetic Model 2000," *British Geological Survey Technical Report WM/00/17R*.

<https://www.ngdc.noaa.gov/geomag/WMM/DoDWMM.shtml>

## See Also

International Geomagnetic Reference Field 12, World Magnetic Model 2010



## Compatibility

---

**Note** World Magnetic Model 2000 will be removed in a future version. For model years between 2000 and the start of 2015, use International Geomagnetic Reference Field 12. For model years between 2015 and later, use World Magnetic Model.

---

**Introduced before R2006a**

# World Magnetic Model 2005

Calculate Earth's magnetic field at specific location and time using World Magnetic Model 2005 (WMM2005)

## Library

Environment/Gravity

## Description

The WMM2005 block implements the mathematical representation of the National Geospatial Intelligence Agency (NGA) World Magnetic Model 2005. The WMM2005 block calculates the Earth's magnetic field vector, horizontal intensity, declination, inclination, and total intensity at a specified location and time. The reference frame is north-east-down (NED).

---

**Note** You cannot use this block to model the Earth magnetic field above an altitude of 1,000,000 meters.

---

## Parameters

### Units

Specifies the input and output units:

| Units        | Height | Magnetic Field | Horizontal Intensity | Total Intensity |
|--------------|--------|----------------|----------------------|-----------------|
| Metric (MKS) | Meters | Nanotesla      | Nanotesla            | Nanotesla       |
| English      | Feet   | Nanogauss      | Nanogauss            | Nanogauss       |

### Input decimal year

When selected, the decimal year is an input for the World Magnetic Model 2005 block. Otherwise, a date must be specified using the dialog parameters of **Month**, **Day**, and **Year**.

### Month

Specifies the month used to calculate decimal year.

### Day

Specifies the day used to calculate decimal year.

### Year

Specifies the year used to calculate decimal year.

### Action for out of range input

Specify if out-of-range input invokes a warning, error or no action.

### Output horizontal intensity

When selected, the horizontal intensity is output.

**Output declination**

When selected, the declination, the angle between true north and the magnetic field vector (positive eastwards), is output.

**Output inclination**

When selected, the inclination, the angle between the horizontal plane and the magnetic field vector (positive downwards), is output.

**Output total intensity**

When selected, the total intensity is output.

**Inputs and Outputs**

| Input                | Dimension Type | Description  |
|----------------------|----------------|--|
| First                |                | Contains the height, in selected units.  |
| Second               |                | Contains the latitude in degrees.  |
| Third                |                | Contains the longitude in degrees.   |
| Fourth<br>(Optional) |                | Contains the desired year in a decimal format to include any fraction of the year that has already passed. The value is the current year plus the number of days that have passed in this year divided by 365. |
|                      |                | The following code illustrates how to calculate the decimal year, <code>dyear</code> , for March 21, 2005:   |
|                      |                | <pre>%%%BEGIN CODE%%% dyear=decyear('21-March-2005','dd-mmm-yyyy') %%%END CODE%%%</pre>  |

| Output               | Dimension Type | Description  |
|----------------------|----------------|--|
| First                | Vector         | Contains the magnetic field in selected units.       |
|                      |                | The reference frame is north-east-down (NED).        |
| Second<br>(Optional) |                | Contains the horizontal intensity in selected units. |
| Third<br>(Optional)  |                | Contains the declination in degrees.                 |
| Fourth<br>(Optional) |                | Contains the inclination in degrees.                 |
| Fifth (Optional)     |                | Contains the total intensity in selected units.      |

**Limitations**

The WMM2005 specification produces data that is reliable five years after the epoch of the model, which is January 1, 2005.

The internal calculation of decimal year does not take into account local time or leap seconds.

The WMM2005 specification describes only the long-wavelength spatial magnetic fluctuations due to the Earth's core. Intermediate and short-wavelength fluctuations, contributed from the crustal field (the mantle and crust), are not included. Also, the substantial fluctuations of the geomagnetic field,

which occur constantly during magnetic storms and almost constantly in the disturbance field (auroral zones), are not included.

## Reference

<https://www.ngdc.noaa.gov/geomag/WMM/DoDWMM.shtml>

## See Also

International Geomagnetic Reference Field 12, World Magnetic Model 2010

|              |                           |
|--------------|---------------------------|
| h (m)        | Magnetic Field (nT)       |
| $\mu$ (deg)  | Horizontal Intensity (nT) |
| I (deg)      | Declination (deg)         |
|              | Inclination (deg)         |
| Decimal Year | Total Intensity (nT)      |

## Compatibility

---

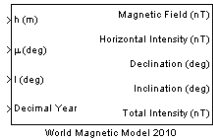
**Note** World Magnetic Model 2005 will be removed in a future version. For model years between 2000 and the start of 2015, use International Geomagnetic Reference Field 12. For model years between 2015 and later, use World Magnetic Model.

---

**Introduced before R2006a**

# World Magnetic Model 2010

Calculate Earth's magnetic field at specific location and time using World Magnetic Model 2010 (WMM2010)



## Library

Environment/Gravity

## Description

The WMM2010 block implements the mathematical representation of the National Geospatial Intelligence Agency (NGA) World Magnetic Model 2010. The WMM2010 block calculates the Earth's magnetic field vector, horizontal intensity, declination, inclination, and total intensity at a specified location and time. The reference frame is north-east-down (NED).

---

**Note** You cannot use this block to model the Earth magnetic field above an altitude of 1,000,000 meters.

---

## Parameters

### Units

Specifies the input and output units:

| Units        | Height | Magnetic Field | Horizontal Intensity | Total Intensity |
|--------------|--------|----------------|----------------------|-----------------|
| Metric (MKS) | Meters | Nanotesla      | Nanotesla            | Nanotesla       |
| English      | Feet   | Nanogauss      | Nanogauss            | Nanogauss       |

### Input decimal year

When selected, the decimal year is an input for the World Magnetic Model 2010 block. Otherwise, a date must be specified using the dialog parameters of **Month**, **Day**, and **Year**.

### Month

Specifies the month used to calculate decimal year.

### Day

Specifies the day used to calculate decimal year.

### Year

Specifies the year used to calculate decimal year.

**Action for out-of-range input**

Specify if out-of-range input invokes a warning, error or no action.

**Output horizontal intensity**

When selected, the horizontal intensity is output.

**Output declination**

When selected, the declination, the angle between true north and the magnetic field vector (positive eastwards), is output.

**Output inclination**

When selected, the inclination, the angle between the horizontal plane and the magnetic field vector (positive downwards), is output.

**Output total intensity**

When selected, the total intensity is output.

**Inputs and Outputs**

| Input                | Dimension Type | Description  |
|----------------------|----------------|--|
| First                |                | Contains the height, in selected units.  |
| Second               |                | Contains the latitude in degrees.  |
| Third                |                | Contains the longitude in degrees.   |
| Fourth<br>(Optional) |                | Contains the desired year in a decimal format to include any fraction of the year that has already passed. The value is the current year plus the number of days that have passed in this year divided by 365. |
|                      |                | The following code illustrates how to calculate the decimal year, dyear, for March 21, 2010:   |
|                      |                | <pre> %%%BEGIN CODE%%% dyear=decyear('21-March-2010','dd-mmm-yyyy') %%%END CODE%%% </pre>  |

| Output               | Dimension Type | Description  |
|----------------------|----------------|--|
| First                | Vector         | Contains the magnetic field in selected units.       |
| Second<br>(Optional) |                | Contains the horizontal intensity in selected units. |
| Third<br>(Optional)  |                | Contains the declination in degrees.                 |
| Fourth<br>(Optional) |                | Contains the inclination in degrees.                 |
| Fifth (Optional)     |                | Contains the total intensity in selected units.      |

**Limitations**

The WMM2010 specification produces data that is reliable five years after the epoch of the model, which is January 1, 2015.

The internal calculation of decimal year does not take into account local time or leap seconds.

The WMM2010 specification describes only the long-wavelength spatial magnetic fluctuations due to the Earth's core. Intermediate and short-wavelength fluctuations, contributed from the crustal field (the mantle and crust), are not included. Also, the substantial fluctuations of the geomagnetic field, which occur constantly during magnetic storms and almost constantly in the disturbance field (auroral zones), are not included.

## Reference

<https://www.ngdc.noaa.gov/geomag/WMM/DoDWMM.shtml>

## See Also

World Magnetic Model 2000, World Magnetic Model 2005

## Compatibility

---

**Note** World Magnetic Model 2010 will be removed in a future version. For model years between 2000 and the start of 2015, use International Geomagnetic Reference Field 12. For model years between 2015 and later, use World Magnetic Model.

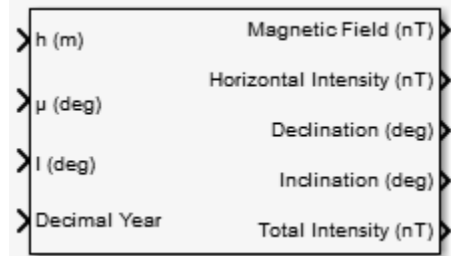
---

**Introduced in R2010a**



# World Magnetic Model 2015

Calculate Earth's magnetic field at specific location and time using World Magnetic Model 2015 (WMM2015)



## Library

Environment/Gravity

## Description

The WMM2015 block implements the mathematical representation of the National Geospatial Intelligence Agency (NGA) World Magnetic Model 2015. The WMM2015 block calculates the Earth's magnetic field vector, horizontal intensity, declination, inclination, and total intensity at a specified location and time. The reference frame is north-east-down (NED).

**Note** You cannot use this block to model the Earth magnetic field above an altitude of 1,000,000 meters.

## Parameters

### Units

Specifies the input and output units:

| Units        | Height | Magnetic Field | Horizontal Intensity | Total Intensity |
|--------------|--------|----------------|----------------------|-----------------|
| Metric (MKS) | Meters | Nanotesla      | Nanotesla            | Nanotesla       |
| English      | Feet   | Nanogauss      | Nanogauss            | Nanogauss       |

### Input decimal year

When selected, the decimal year is an input for the World Magnetic Model 2015 block. Otherwise, a date must be specified using the dialog parameters of **Month**, **Day**, and **Year**.

### Month

Specifies the month used to calculate decimal year.

### Day

Specifies the day used to calculate decimal year.

**Year**

Specifies the year used to calculate decimal year.

**Action for out-of-range input**

Specify if out-of-range input invokes a warning, error or no action.

**Output horizontal intensity**

When selected, the horizontal intensity is output.

**Output declination**

When selected, the declination, the angle between true north and the magnetic field vector (positive eastwards), is output.

**Output inclination**

When selected, the inclination, the angle between the horizontal plane and the magnetic field vector (positive downwards), is output.

**Output total intensity**

When selected, the total intensity is output.

**Inputs and Outputs**

| Input                | Dimension Type | Description  |
|----------------------|----------------|--|
| First                |                | Contains the height, in selected units.  |
| Second               |                | Contains the latitude in degrees.  |
| Third                |                | Contains the longitude in degrees.   |
| Fourth<br>(Optional) |                | Contains the desired year in a decimal format to include any fraction of the year that has already passed. The value is the current year plus the number of days that have passed in this year divided by 365. |
|                      |                | The following code illustrates how to calculate the decimal year, dyear, for March 21, 2015:   |
|                      |                | <pre> %%%BEGIN CODE%%% dyear=decyear('21-March-2015','dd-mmm-yyyy') %%%END CODE%%% </pre>  |
| Output               | Dimension Type | Description  |
| First                | Vector         | Contains the magnetic field in selected units.   |
| Second<br>(Optional) |                | Contains the horizontal intensity in selected units.   |
| Third<br>(Optional)  |                | Contains the declination in degrees.   |
| Fourth<br>(Optional) |                | Contains the inclination in degrees.   |
| Fifth (Optional)     |                | Contains the total intensity in selected units.  |

**Limitations**

The WMM2015 specification produces data that is reliable five years after the epoch of the model, which is January 1, 2015.

The internal calculation of decimal year does not take into account local time or leap seconds.

The WMM2015 specification describes only the long-wavelength spatial magnetic fluctuations due to the Earth's core. Intermediate and short-wavelength fluctuations, contributed from the crustal field (the mantle and crust), are not included. Also, the substantial fluctuations of the geomagnetic field, which occur constantly during magnetic storms and almost constantly in the disturbance field (auroral zones), are not included.

## Reference

<https://www.ngdc.noaa.gov/geomag/WMM/DoDWMM.shtml>

## See Also

World Magnetic Model 2000, World Magnetic Model 2005

## Compatibility

---

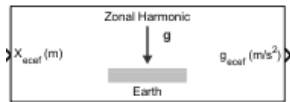
**Note** World Magnetic Model 2015 will be removed in a future version. For model years between 2000 and the start of 2015, use International Geomagnetic Reference Field 12. For model years between 2015 and later, use World Magnetic Model.

---

## Introduced in R2015a

## Zonal Harmonic Gravity Model

Calculate zonal harmonic representation of planetary gravity



### Library

Environment/Gravity

### Description

The Zonal Harmonic Gravity Model block calculates the zonal harmonic representation of planetary gravity at a specific location based on planetary gravitational potential. This block provides a convenient way to describe the gravitational field of a planet outside its surface.

By default, the block uses the fourth order zonal coefficient for Earth to calculate the zonal harmonic gravity. It also allows you to specify the second or third zonal coefficient.

`gravityzonal` is implemented using the following planetary parameter values for each planet:

| Planet  | Equatorial Radius (Re) in Meters | Gravitational Parameter (GM) in m <sup>3</sup> /s <sup>2</sup> | Zonal Harmonic Coefficients (J Values)       |
|---------|----------------------------------|--|--|
| Earth   | 6378.1363e3                      | 3.986004415e14   | [ 0.0010826269 -0.0000025323 -0.0000016204 ] |
| Jupiter | 71492.e3                         | 1.268e17   | [0.01475 0 -0.00058]                         |
| Mars    | 3397.2e3                         | 4.305e13   | [ 0.001964 0.000036 ]                        |
| Mercury | 2439.0e3                         | 2.2032e13  | 0.00006                                      |
| Moon    | 1738.0e3                         | 4902.799e9   | 0.0002027                                    |
| Neptune | 24764e3                          | 6.809e15   | 0.004  |
| Saturn  | 60268.e3                         | 3.794e16   | [0.01645 0 -0.001]                           |
| Uranus  | 25559.e3                         | 5.794e15   | 0.012  |
| Venus   | 6052.0e3                         | 3.257e14   | 0.000027                                     |

### Parameters

#### Units

Specify the input units:

| Units        | Position | Equatorial Radius | Gravitational Parameter         |
|--------------|----------|-------------------|---------------------------------|
| Metric (MKS) | Meters   | Meters            | Meters cubed per second squared |
| English      | Feet     | Feet              | Feet cubed per second squared   |

### Degree

Specify the degree of harmonic model.

- 2 — Second degree, J2. Most significant or largest spherical harmonic term, which accounts for the oblateness of a planet.
- 3 — Third degree, J3.
- 4 — Fourth degree, J4 (default).

### Action for out-of-range input

Specify if out-of-range input invokes a warning, error, or no action.

### Planet model

Specify the planetary model. From the list, select Mercury, Venus, Earth, Moon, Mars, Jupiter, Saturn, Uranus, Neptune, or Custom.

Selecting Custom enables you to specify your own planetary model. This option enables the **Equatorial radius**, **Gravitational parameter**, and **J values** parameters.

Selecting Mercury, Venus, Moon, Uranus, or Neptune limits the degree to 2.

Selecting Mars limits the degree to 3.

### Equatorial radius

Specify the planetary equatorial radius in the length units that the **Units** parameter defines.

### Gravitational parameter

Specify the planetary gravitational parameter in the length units cubed per second squared that the **Units** parameter defines.

### J values

Specify a 3-element array that defines the zonal harmonic coefficients.

## Inputs and Outputs

This block accepts only scalar inputs (m=1).

| Input | Dimension Type | Description  |
|-------|----------------|--|
| First | m-by-3 matrix  | Contains planet-centered planet-fixed coordinates from the center of the planet in the selected length units. If <b>Planet model</b> has a value of Earth, this matrix contains Earth-centered Earth-fixed (ECEF) coordinates. |

| <b>Output</b> | <b>Dimension Type</b> | <b>Description</b>  |
|---------------|-----------------------|---|
| First         | m-by-3 array          | Contains gravity values in the x-axis, y-axis and z-axis of the planet-centered planet-fixed coordinates in the selected length units per second squared. |

## References

Vallado, D. A., *Fundamentals of Astrodynamics and Applications*, McGraw-Hill, New York, 1997.

Fortescue, P., J. Stark, G. Swinerd, (Eds.). *Spacecraft Systems Engineering*, Third Edition, Wiley & Sons, West Sussex, 2003.

Tewari, A., *Atmospheric and Space Flight Dynamics Modeling and Simulation with MATLAB and Simulink*, Birkhäuser, Boston, 2007.

## Extended Capabilities

### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

### **Introduced in R2009b**

# Functions

---

## **asbAddOns**

Start Add-On Explorer for aerospace Add-Ons

### **Syntax**

```
asbAddOns
```

### **Description**

asbAddOns starts the Add-On Explorer, from which you can download, install, or uninstall Add-Ons for Aerospace Blockset and Aerospace Toolbox.

### **Examples**

#### **Start Add-On Explorer for Aerospace Add-Ons**

Start Add-On Explorer for Aerospace Blockset and Aerospace Toolbox Add-Ons.

```
asbAddOns
```

The Add-On Explorer starts populated with the aerospace product Add-Ons.

### **See Also**

aeroDataPackage

**Introduced in R2019a**



# asbCubeSatMissionAnalysis

Create and launch new instance of CubeSat Mission Analysis live script

## Syntax

```
[SavedFile] = asbCubeSatMissionAnalysis
[SavedFile] = asbCubeSatMissionAnalysis(FileName)
```

## Description

[SavedFile] = asbCubeSatMissionAnalysis creates and starts a new instance of the CubeSat Mission Analysis live script with the default name `CubeSatMissionAnalysis_currentdate.mlx` and assigns the name to SavedFile.

[SavedFile] = asbCubeSatMissionAnalysis(FileName) creates and starts a new instance of the CubeSat mission analysis live script with the name `FileName.mlx`.

## Examples

### Create and Start CubeSat Mission Analysis Live Script with Default Name

Create and start a CubeSat mission analysis live script with a default name.

```
[SavedFile] = asbCubeSatMissionAnalysis
SavedFile =
    'CubeSatMissionAnalysis_20190607T104952.mlx'
```

### Create and Start CubeSat Mission Analysis Live Script with Custom Name

Create and start a CubeSat mission analysis live script with a custom name, `MyCubeSatMissionAnalysis`.

```
[SavedFile] = asbCubeSatMissionAnalysis('MyCubeSatMissionAnalysis')
SavedFile =
    'MyCubeSatMissionAnalysis.mlx'
```

## Input Arguments

### FileName — File name or path for CubeSat mission analysis live script

`CubeSatMissionAnalysis_currentdate.mlx` (default) | file name

File name or path at which to save the CubeSat mission analysis live script.

Example: `MyCubeSatMissionAnalysis`

Data Types: char | string

## **Output Arguments**

### **SavedFile — CubeSat mission analysis live script file name**

file name

Name of CubeSat mission analysis live script, returned as a file name.

### **See Also**

CubeSat Vehicle | ecef2eci | eci2ecef | generateAccessTable | greenwichsrt |  
ijk2keplerian | juliandate | keplerian2ijk

**Introduced in R2019a**

# asbFlightControlAnalysis

Start flight control analysis template

## Syntax

```
asbFlightControlAnalysis()
asbFlightControlAnalysis(configuration)
asbFlightControlAnalysis(configuration,modelToAnalyze)
asbFlightControlAnalysis(configuration,modelToAnalyze,airframe)
```

## Description

`asbFlightControlAnalysis()` creates a flight control analysis template for a 3DOF configuration.

`asbFlightControlAnalysis(configuration)` creates a flight control analysis template for a specified configuration.

`asbFlightControlAnalysis(configuration,modelToAnalyze)` creates a flight control analysis model with the specified model name.

`asbFlightControlAnalysis(configuration,modelToAnalyze,airframe)` creates a flight control analysis template for a specified airframe model.

## Examples

### Start Flight Control Analysis Template for 3DOF Configuration

Start default flight control analysis template for 3DOF configuration.

```
asbFlightControlAnalysis
```

### Start Flight Control Analysis Template for 6DOF Configuration

Start default flight control analysis template for 6DOF configuration.

```
asbFlightControlAnalysis('6DOF')
```

### Start Flight Control Analysis Template Using a Different Airframe Model

Start the 3DOF flight control analysis template `SkyHoggAnalysisModel` and trim the model around the `opSpecDefault` operating point specification object. The example then linearizes the airframe model around the `opTrim` operating point and calculates the short- and long-period (phugoid) mode characteristics of `linSys`.

```
asbFlightControlAnalysis('3DOF', 'SkyHoggAnalysisModel');
opSpecDefault = SkyHogg3DOF0pSpec('SkyHoggAnalysisModel');
opTrim = trimAirframe('SkyHoggAnalysisModel', opSpecDefault);
```

```
linSys = linearizeAirframe('SkyHoggAnalysisModel', opTrim)  
flyingQual = computeLongitudinalFlyingQualities('SkyHoggAnalysisModel', linSys)
```

## Input Arguments

### **configuration** — Configuration for flight control analysis

'3DOF' (default) | '6DOF'

Configuration for flight control analysis

Data Types: char | string

### **modelToAnalyze** — Name for flight control analysis model being created

model name

Name for flight control analysis model being created.

Data Types: char | string

### **airframe** — Airframe to analyze

airframe subsystem specified as a block path (default) | airframe model specified as a model name

Airframe to analyze, specified as an airframe model name (inserted as a referenced model).  
Otherwise, the subsystem must be loaded.

Data Types: char | string

## See Also

[computeLateralDirectionalFlyingQualities](#) | [computeLongitudinalFlyingQualities](#) | [linearizeAirframe](#) | [trimAirframe](#)

## Topics

“Analyze Dynamic Response and Flying Qualities of Aerospace Vehicles” on page 2-45

## Introduced in R2018b

# computeLateralDirectionalFlyingQualities

Calculate dutch roll mode, roll mode, and spiral mode characteristics of state-space model

## Syntax

```
computeLateralDirectionalFlyingQualities(modelToAnalyze)
computeLateralDirectionalFlyingQualities(modelToAnalyze, linSys)
computeLateralDirectionalFlyingQualities(modelToAnalyze, linSys, generatePlots)
```

## Description

`computeLateralDirectionalFlyingQualities(modelToAnalyze)` calculates the lateral-directional flying qualities (dutch roll mode, roll mode, and spiral mode) characteristics using the linear system state-space model selected in the input dialog window and compares the results against MIL-F-8785C requirements.

`computeLateralDirectionalFlyingQualities(modelToAnalyze, linSys)` calculates lateral-directional flying qualities using the linear system state-space model selected in the input dialog window. The state-space model must have at least these four states:

- v
- p
- r
- phi

or all eight of these states:

- v
- p
- r
- phi
- U
- w
- q
- theta

To create a compatible state-space model, use the `linearizeAirframe` function.

`computeLateralDirectionalFlyingQualities(modelToAnalyze, linSys, generatePlots)` calculates lateral-directional flying qualities using linear system state-space model `linSys`.

## Examples

### Calculate Lateral-Directional Flying Qualities While Specifying a State-Space Model

Calculate lateral-directional flying qualities of `linSys`, a state space model of airframe `DehavillandBeaverAnalysisModel`. This example starts the flight control analysis template using

`asbFlightControlAnalysis` and trims the model around the `opSpecDefault` operating point specification object. It then linearizes the airframe model around the `opTrim` operating point and calculates the dutch roll mode, roll mode, and spiral mode characteristics of `linSys`. The example also optionally calculates the longitudinal characteristics for the model.

```
asbFlightControlAnalysis('6DOF', 'DehavillandBeaverAnalysisModel');
opSpecDefault = DehavillandBeaver6DOF0pSpec('DehavillandBeaverAnalysisModel');
opTrim = trimAirframe('DehavillandBeaverAnalysisModel', opSpecDefault);
linSys = linearizeAirframe('DehavillandBeaverAnalysisModel', opTrim);
lonFlyingQual = computeLongitudinalFlyingQualities('DehavillandBeaverAnalysisModel', linSys)
latFlyingQual = computeLateralDirectionalFlyingQualities('DehavillandBeaverAnalysisModel', linSys)
```

### Compare Lateral-Directional Flying Qualities Results to MIL-F-8785C Requirements

Compare against MIL-F\_8785C requirements.

Calculate lateral-directional flying qualities for the model `DehavillandBeaverAnalysisModel` and specify a state-space, `linSys`.

```
asbFlightControlAnalysis('6DOF', 'DehavillandBeaverAnalysisModel');
opSpecDefault = DehavillandBeaver6DOF0pSpec('DehavillandBeaverAnalysisModel');
opTrim = trimAirframe('DehavillandBeaverAnalysisModel', opSpecDefault);
linSys = linearizeAirframe('DehavillandBeaverAnalysisModel', opTrim);
lonFlyingQual = computeLongitudinalFlyingQualities('DehavillandBeaverAnalysisModel', linSys);
latFlyingQual = computeLateralDirectionalFlyingQualities('DehavillandBeaverAnalysisModel', linSys);
```

Get the list of lateral-directional flying qualities for the model.

```
latFlyingQual
```

```
latFlyingQual =
```

```
struct with fields:
    DutchRollMode: [1x1 struct]
    RollMode: [1x1 struct]
    SpiralMode: [1x1 struct]
```

Get the dutch roll flying quality.

```
latFlyingQual.DutchRollMode
```

```
ans =
```

```
struct with fields:
    root: [2x1 double]
    oscillatoryMode: 'Dutch Roll Mode'
    MILF8785CRequirement: 'Satisfies MIL-F-8785C Level 1 Criteria (zeta_d >= 0.08, omega_n >= 0.4)'
    DampingRatio: 0.4337
    NaturalFreq: 1.4872
    TimeToDoubleAmp: -1.0747
    TimeConstant: []
    response: 'converging oscillatory motion'
    description: 'complex conjugate pair with negative real components'
```

Get the results of the comparison with the MIL-F-8785C requirements.

```
latFlyingQual.DutchRollMode.MILF8785CRequirement
```

```
ans =  
'Satisfies MIL-F-8785C Level 1 Criteria (zeta_d >= 0.08, omega_n >= 0.4, zeta_d*omega_n_d >= 0.15)'
```

## Input Arguments

### **modelToAnalyze** — Model on which to perform flight control analysis

model name

Model on which to perform flight control analysis using the linear state-space model `linSys`.

Data Types: `char` | `string`

### **linSys** — Linear state-space model object

linear state-space model object name

Linear state-space model object used to perform flight control analysis on `modelToAnalyze`.

Data Types: `char` | `string`

### **generatePlots** — Display pole-zero map

model name

Display pole-zero map for the linear system state-space model.

Data Types: `char` | `string`

## Limitations

This function requires the Simulink Control Design license

## See Also

[asbFlightControlAnalysis](#) | [computeLongitudinalFlyingQualities](#) | [linearizeAirframe](#) | [trimAirframe](#)

## Topics

“Analyze Dynamic Response and Flying Qualities of Aerospace Vehicles” on page 2-45

## Introduced in R2019a

## computeLongitudinalFlyingQualities

Calculate short-period and long-period (phugoid) mode characteristics of specified state-space model

### Syntax

```
computeLongitudinalFlyingQualities(modelToAnalyze)
computeLongitudinalFlyingQualities(modelToAnalyze,linSys)
computeLongitudinalFlyingQualities(modelToAnalyze,linSys,generatePlots)
```

### Description

`computeLongitudinalFlyingQualities(modelToAnalyze)` calculates longitudinal flying qualities (short-period and phugoid mode) using the linear system state-space model selected in the input dialog window and compares the results against MIL-F-8785C requirements.

`computeLongitudinalFlyingQualities(modelToAnalyze,linSys)` calculates longitudinal flying qualities (short-period and phugoid mode) using the linear system state-space model selected in the input dialog window. The state-space model must have at least these four states:

- U
- w
- q
- theta

or all eight of these states:

- U
- w
- q
- theta
- v
- p
- r
- phi

To create a usable state-space model, use the `linearizeAirframe` function.

`computeLongitudinalFlyingQualities(modelToAnalyze,linSys,generatePlots)` calculates longitudinal flying qualities (short-period and phugoid mode) using linear system state-space model `linSys`.

### Examples

#### Calculate Longitudinal Flying Qualities While Specifying a State-Space Model

Calculate longitudinal flying qualities of `linSys`, a state space model `DehavillandBeaverAnalysisModel` This example starts the flight control analysis template using



asbFlightControlAnalysis and trims the model around the opSpecDefault operating point specification object. It then linearizes the airframe model around the opTrim operating point and calculates the short- and long-period (phugoid) mode characteristics of linSys.

```
asbFlightControlAnalysis('3D0F', 'SkyHoggAnalysisModel');
opSpecDefault = SkyHogg3D0F0pSpec('SkyHoggAnalysisModel');
opTrim = trimAirframe('SkyHoggAnalysisModel', opSpecDefault);
linSys = linearizeAirframe('SkyHoggAnalysisModel', opTrim)
flyingQual = computeLongitudinalFlyingQualities('SkyHoggAnalysisModel', linSys)
```

## Compare Longitudinal Flying Qualities Results to MIL-F-8785C Requirements

Compare against MIL-F\_8785C requirements.

Calculate longitudinal flying qualities for the model DehavillandBeaverAnalysisModel and specify a state-space, linSys.

```
asbFlightControlAnalysis('6D0F', 'DehavillandBeaverAnalysisModel');
opSpecDefault = DehavillandBeaver6D0F0pSpec('DehavillandBeaverAnalysisModel');
opTrim = trimAirframe('DehavillandBeaverAnalysisModel', opSpecDefault);
linSys = linearizeAirframe('DehavillandBeaverAnalysisModel', opTrim);
lonFlyingQual = computeLongitudinalFlyingQualities('DehavillandBeaverAnalysisModel', linSys);
latFlyingQual = computeLateralDirectionalFlyingQualities('DehavillandBeaverAnalysisModel', linSys);
```

Get the list of longitudinal flying qualities for the model.

```
lonFlyingQual
```

```
lonFlyingQual =
```

```
struct with fields:
    PhugoidMode: [1x1 struct]
    ShortPeriodMode: [1x1 struct]
```

Get the phugoid flying quality.

```
lonFlyingQual.PhugoidMode
```

```
ans =
```

```
struct with fields:
    root: [2x1 double]
    oscillatoryMode: 'Phugoid (Long-Period Mode)'
    MILF8785CRequirement: 'Satisfies MIL-F-8785C Level 1 Criteria (zeta_ph >= 0.04)'
    DampingRatio: 0.0926
    NaturalFreq: 0.1540
    TimeToDoubleAmp: -48.5747
    TimeConstant: []
    response: 'converging oscillatory motion'
    description: 'complex conjugate pair with negative real components'
```

Get the results of the comparison with the MIL-F-8785C requirements.

```
lonFlyingQual.PhugoidMode.MILF8785CRequirement
```

```
ans =  
    'Satisfies MIL-F-8785C Level 1 Criteria (zeta_ph >= 0.04)'
```

## Input Arguments

### **modelToAnalyze** — Model on which to perform flight control analysis

model name

Model on which to perform flight control analysis using the linear state-space model `linSys`.

Data Types: `char` | `string`

### **linSys** — Linear state-space model object

linear state-space model object name

Linear state-space model object used to perform flight control analysis on `modelToAnalyze`.

Data Types: `char` | `string`

### **generatePlots** — Display pole-zero map

model name

Display pole-zero map for the linear system state-space model.

Data Types: `char` | `string`

## See Also

`asbFlightControlAnalysis` | `computeLateralDirectionalFlyingQualities` |  
`linearizeAirframe` | `trimAirframe`

## Topics

“Analyze Dynamic Response and Flying Qualities of Aerospace Vehicles” on page 2-45

## Introduced in R2018b

## ecef2eci

Position and velocity vectors in Earth-centered inertial mean-equator mean-equinox

### Syntax

```
[r_eci] = ecef2eci(utc,r_ecef)
[r_eci,v_eci] = ecef2eci( ___,v_ecef)
[r_eci,v_eci,a_eci] = ecef2eci( ___,a_ecef)
[r_eci,v_eci,a_eci] = ecef2eci( ___,Name,Value)
```

### Description

`[r_eci] = ecef2eci(utc,r_ecef)` calculates the position vector in the Earth-centered inertial mean-equator mean-equinox (J2000) coordinate system for a given position vector in the Earth-centered Earth-fixed (ECEF) coordinate system at a specific Universal Coordinated Time (UTC).

`[r_eci,v_eci] = ecef2eci( ___,v_ecef)` calculates the position and velocity vectors for given position and velocity vectors.

`[r_eci,v_eci,a_eci] = ecef2eci( ___,a_ecef)` calculates the position, velocity, acceleration vectors for given position, velocity, and acceleration vectors.

`[r_eci,v_eci,a_eci] = ecef2eci( ___,Name,Value)` calculates the position, velocity, and acceleration vectors at a higher precision using Earth orientation parameters.

### Examples

#### Convert ECEF Position and Velocity to ECI

Convert the Earth-centered Earth-fixed (ECEF) position and velocity to Earth-centered inertial (ECI) at 12:00 on January 4, 2019.

```
r_ecef = [-5762640 -1682738 3156028];
v_ecef = [3832 -4024 4837];
utc = [2019 1 4 12 0 0];
[r_eci, v_eci] = ecef2eci(utc, r_ecef, v_ecef);
```

```
r_eci =
    1.0e+06 *
    -2.9818
     5.2070
     3.1616
```

```
v_eci =
    1.0e+03 *
    -3.3837
    -4.8870
     4.8430
```

## Convert ECEF Position to ECI Including Polar Motion Effects

Convert the ECEF position to ECI at 12:00 on January 4, 2019, including the effects of polar motion.

```
r_ecef = [-5762640 -1682738 3156028];
utc = [2019 1 4 12 0 0];
mjd = mjuliandate(utc);
pm = polarMotion(mjd, 'action', 'none')*180/pi;
r_eci = ecef2eci(utc, r_ecef, 'pm', pm);
```

```
r_eci =
    1.0e+06 *
    -2.9818
     5.2070
     3.1616
```

## Input Arguments

### utc — Universal Coordinated Time

1-by-6 array | 1-by-6 matrix

Universal Coordinated Time (UTC) in the order year, month, day, hour, minutes, and seconds, specified as 1-by-6 array of UTC values:

| Time Value        | Enter   |
|-------------------|---|
| Year              | Double value that is a whole number greater than 1, such as 2013.             |
| Month             | Double value that is a whole number greater than 0, within the range 1 to 12. |
| Day               | Double value that is a whole number greater than 0, within the range 1 to 31. |
| Hour              | Double value that is a whole number greater than 0, within the range 1 to 24. |
| Minute and second | Double value that is a whole number greater than 0, within the range 1 to 60. |

Example: [2000 1 12 4 52 12.4]

Data Types: double

### r\_ecef — Position components

3-by-1 array

Array of Earth-centered Earth-fixed position components, specified as a 3-by-1 array.

Data Types: double

### v\_ecef — Velocity components

3-by-1 array

Earth-centered Earth-fixed velocity components, specified as a 3-by-1 array.

Data Types: double

**a\_ecef — Acceleration components**

3-by-1 array

Earth-centered Earth-fixed acceleration components, specified as a 3-by-1 array.

Data Types: `double`

**Name-Value Pair Arguments**

Specify optional comma-separated pairs of `Name`, `Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1, Value1, . . . , NameN, ValueN`.

Example: `[-0.2530e-6 -0.0188e-6]`

**dAT — Difference between TAI and UTC**

0 (default) | scalar

Difference between International Atomic Time (TAI) and Universal Coordinated Time (UTC), specified as a scalar, in seconds.

Example: 32

Data Types: `double`

**dUT1 — Difference between TAI and UTC**

0 (default) | scalar

Difference between International Atomic Time (TAI) and Universal Coordinated Time (UTC), specified as a scalar, in seconds.

Example: 0.234

Data Types: `double`

**pm — Polar displacement**

array of zeroes (default) | 1-by-2 array

Polar displacements due to the motion of Earth crust along the x- and y-axis, in degrees.

---

**Tip** To calculate the displacement, use the `polarMotion` function.

---

Example: `pm = polarMotion(mjd, 'action', 'none')*180/pi;`

Data Types: `double`

**dCIP — Adjustment to the CIP location**

1-by-2 array

Adjustment to the location of the Celestial Intermediate Pole (CIP), in degrees, specified as a comma-separated pair consisting of `dCIP` and an `M`-by-2 array. This location (`dDeltaX`, `dDeltaY`) is along the x- and y- axes. By default, this function assumes a 1-by-2 array of zeroes.

For historical values, see the International Earth Rotation and Reference Systems Service Web site (<https://www.iers.org>) and navigate to the Earth orientation data Data/Products page.

- $M$ -by-2 array

Specify an  $M$ -by-2 array of location adjustment values, where  $M$  is the number of direction cosine or transformation matrices to be converted. Each row corresponds to one set of  $dDeltaX$  and  $dDeltaY$  values.

Example: [-0.2530e-6 -0.0188e-6]

Data Types: double

### **lod — Excess length of day**

0 (default) | scalar

Excess length of day (difference between astronomically determined duration of day and 86400 SI seconds), specified as a scalar, in seconds.

Example: 32

Data Types: double

## **Output Arguments**

### **r\_eci — Position components**

3-by-1 array

Earth-centered inertial mean-equator mean-equinox (J2000) position components, specified as a 3-by-1 array.

### **v\_eci — Velocity components**

3-by-1 array

Earth-centered inertial mean-equator mean-equinox (J2000) velocity components, specified as a 3-by-1 array.

### **a\_eci — Acceleration components**

3-by-1 array

Earth-centered inertial mean-equator mean-equinox (J2000) acceleration components, specified as a 3-by-1 array.

## **References**

- [1] Vallado, D. A. *Fundamentals of Astrodynamics and Applications*. alg. 4. New York: McGraw-Hill, 1997.

## **See Also**

CubeSat Vehicle | aeroReadIERSData | asbCubeSatMissionAnalysis | dcmeCi2ecef | deltaCIP | deltaUT1 | eci2ecef | generateAccessTable | greenwicksrt | polarMotion

**Introduced in R2019a**

## eci2ecef

Position, velocity, and acceleration vectors in Earth-centered Earth-fixed (ECEF) coordinate system

### Syntax

```
[r_ecef,v_ecef,a_ecef] = eci2ecef(utc,r_eci,v_eci,a_eci)
[r_ecef,v_ecef,a_ecef] = eci2ecef(utc,r_eci,v_eci,a_eci,Name,Value)
```

### Description

`[r_ecef,v_ecef,a_ecef] = eci2ecef(utc,r_eci,v_eci,a_eci)` calculates position, velocity, and acceleration vectors in Earth-centered Earth-fixed (ITRF) coordinate system for given position, velocity, and acceleration vectors in the Earth-centered inertial mean-equator mean-equinox (J2000) coordinate system at a specific Universal Coordinated Time (UTC).

`[r_ecef,v_ecef,a_ecef] = eci2ecef(utc,r_eci,v_eci,a_eci,Name,Value)` calculates the position, velocity, and acceleration vectors at a higher precision using Earth orientation parameters.

### Examples

#### Convert ECI Position and Velocity to ECEF

Convert ECI position and velocity to ECEF at 12:00 on January 4, 2019.

```
r_eci = [-2981784 5207055 3161595];
v_eci = [-3384 -4887 4843];
utc = [2019 1 4 12 0 0];
[r_ecef, v_ecef] = eci2ecef(utc, r_eci, v_eci)
```

```
r_ecef =
    1.0e+06 *
    -5.7627
    -1.6827
     3.1560
```

```
v_ecef =
    1.0e+03 *
     3.8319
    -4.0243
     4.8370
```

#### Convert ECI Position to ECEF Including Polar Motion Effects

Convert ECI position to ECEF at 12:00 on January 4, 2019 including effects of polar motion.

```
r_eci = [-2981784 5207055 3161595];
utc = [2019 1 4 12 0 0];
mjd = mjdliandate(utc);
```

```

pm = polarMotion(mjd, 'action', 'none')*180/pi;
r_ecef = eci2ecef(utc, r_eci, 'pm', pm)

r_ecef =
    1.0e+06 *
    -5.7627
    -1.6827
     3.1560

```

## Input Arguments

### utc — Universal Coordinated Time

1-by-6 array | 1-by-6 matrix

Universal Coordinated Time (UTC) in the order year, month, day, hour, minutes, and seconds, specified as 1-by-6 array of UTC values:

| Time Value        | Enter   |
|-------------------|---|
| Year              | Double value that is a whole number greater than 1, such as 2013.             |
| Month             | Double value that is a whole number greater than 0, within the range 1 to 12. |
| Day               | Double value that is a whole number greater than 0, within the range 1 to 31. |
| Hour              | Double value that is a whole number greater than 0, within the range 1 to 24. |
| Minute and second | Double value that is a whole number greater than 0, within the range 1 to 60. |

Example: [2000 1 12 4 52 12.4]

Data Types: double

### r\_eci — Position components

3-by-1 array

Earth-centered inertial mean-equator mean-equinox (J2000) position components, specified as a 3-by-1 array.

Data Types: double

### v\_eci — Velocity components

3-by-1 array

Earth-centered inertial mean-equator mean-equinox (J2000) velocity components, specified as a 3-by-1 array.

Data Types: double

### a\_eci — Acceleration components

3-by-1 array

Earth-centered inertial mean-equator mean-equinox (J2000) acceleration components, specified as a 3-by-1 array.



Data Types: double

### Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name`, `Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1, Value1, . . . , NameN, ValueN`.

Example: [-0.2530e-6 -0.0188e-6]

### dAT — Difference between TAI and UTC

0 (default) | scalar

Difference between International Atomic Time (TAI) and Universal Coordinated Time (UTC), specified as a scalar, in seconds.

Example: 32

Data Types: double

### dUT1 — Difference between TAI and UTC

0 (default) | scalar

Difference between International Atomic Time (TAI) and Universal Coordinated Time (UTC), specified as a scalar, in seconds.

Example: 0.234

Data Types: double

### pm — Polar displacement

array of zeroes (default) | 1-by-2 array

Polar displacements due to the motion of Earth crust along the  $x$ - and  $y$ -axis, in degrees.

---

**Tip** To calculate the displacement, use the `polarMotion` function.

---

Example: `pm = polarMotion(mjd, 'action', 'none')*180/pi;`

Data Types: double

### dCIP — Adjustment to the CIP

1-by-2 array

Adjustment to the location of the Celestial Intermediate Pole (CIP), in degrees, specified as a comma-separated pair consisting of `dCIP` and an  $M$ -by-2 array. This location ( $dDeltaX$ ,  $dDeltaY$ ) is along the  $x$ - and  $y$ - axes. By default, this function assumes a 1-by-2 array of zeroes.

For historical values, see the International Earth Rotation and Reference Systems Service Web site (<https://www.iers.org>) and navigate to the Earth Orientation Data Data/Products page.

- $M$ -by-2 array

Specify a  $M$ -by-2 array of location adjustment values, where  $M$  is the number of direction cosine or transformation matrices to be converted. Each row corresponds to one set of  $dDeltaX$  and  $dDeltaY$  values.

Example: [-0.2530e-6 -0.0188e-6]

Data Types: double

### **lod — Excess length of day**

0 (default) | scalar

Excess length of day (difference between astronomically determined duration of day and 86400 SI seconds), specified as a scalar, in seconds.

Example: 32

Data Types: double

## **Output Arguments**

### **r\_ecef — Position components**

3-by-1 array

Earth-centered Earth-fixed position components, specified as a 3-by-1 array.

### **v\_ecef — Velocity components**

3-by-1 array

Earth-centered Earth-fixed velocity components, specified as a 3-by-1 array.

### **a\_ecef — Acceleration components**

3-by-1 array

Earth-centered Earth-fixed acceleration components, specified as a 3-by-1 array.

## **References**

[1] Vallado, D. A. *Fundamentals of Astrodynamics and Applications*. alg. 4. New York: McGraw-Hill, 1997.

## **See Also**

CubeSat Vehicle | aeroReadIERSData | asbCubeSatMissionAnalysis | dcmeci2ecef | deltaCIP | deltaUT1 | ecef2eci | generateAccessTable | greenwichsrt | polarMotion

**Introduced in R2019a**

# generateAccessTable

Calculate line-of-sight (LOS) mutual visibility between CubeSat vehicle and ground station

## Syntax

```
[accesstable,inview] = generateAccessTable(time,lla,gs_latlon)
[accesstable,inview] = generateAccessTable(time,lla,gs_latlon,method)
```

## Description

[accesstable,inview] = generateAccessTable(time,lla,gs\_latlon) uses spherical Earth geometry to calculate line-of-sight visibility at times, time, between a CubeSat vehicle of position lla and a ground station located at gs\_latlon.

[accesstable,inview] = generateAccessTable(time,lla,gs\_latlon,method) uses spherical Earth geometry or topographical data to calculate line-of-sight visibility.

## Examples

### Approximate LOS Access Between a CubeSat Vehicle and Ground Station

Approximate line-of-site (LOS) access between a CubeSat vehicle in a 500 km circular 45 deg inclined orbit with 155 deg right ascension of the ascending node (RAAN) and a ground station located at MathWorks headquarters over one hour beginning at 12:00 on January 1, 2020:

```
load('exampleValues_generateAccessTable.mat','lat','lon','alt');
time = datetime(2020,1,1,12,0,0) + seconds(1:60*60);
latlon_MathWorks = [42, -71];
[accessTable, inView] = generateAccessTable(time, [lat lon alt], latlon_MathWorks);
```

## Input Arguments

### time — Array of latitudes, longitudes, and altitudes

*M* column vector

Array of latitudes, longitudes, and altitudes, specified as an *M* column vector, where *M* is the length of the array. The array is a MATLAB datetime data type.

Data Types: double

### lla — Geodetic latitudes, longitudes, and altitudes

*M*-by-3 vector

CubeSat geodetic latitudes (deg), longitudes (deg), and altitudes (m), specified as an *M*-by-3 vector.

Data Types: double

### gs\_latlon — Ground station geodetic latitude and longitude

2-by-1 array

Ground station geodetic latitude (deg) and longitude (deg), specified as a 2-by-1 array.

Data Types: `double`

**method — Computation method**

`spherical` | `topographical`

Computation method, specified as `spherical` or `topographical`.

Data Types: `double`

## Output Arguments

**accesstable — Summary of access intervals**

`[]` (default) | `table`

Table summarizing access intervals during which line-of-sight (LOS) visibility is established between the CubeSat vehicle and the ground station. `accesstable` is empty if no access windows are identified. Access interval results are output as a table.

Data Types: `table`

**inview — Established mutual visibility**

logical vector of  $M$

Established mutual visibility at `time`, returned as a logical vector of  $M$ , the length of the array.

Data Types: `logical`

## See Also

`CubeSat Vehicle` | `asbCubeSatMissionAnalysis` | `datetime` | `ecef2eci` | `eci2ecef` | `greenwichsrt` | `ijk2keplerian` | `juliandate` | `keplerian2ijk`

**Introduced in R2019a**

# greenwichsrt

Greenwich mean and apparent sidereal times

## Syntax

```
[thGMST, thGAST] = greenwichSRT(utcJD)
[thGMST, thGAST] = greenwichSRT(utcJD, dUT1, dAT)
```

## Description

`[thGMST, thGAST] = greenwichSRT(utcJD)` calculates Greenwich mean and apparent sidereal times at a specific Universal Coordinated Time (UTC). Mean sidereal time accounts only for secular motion (precession). Apparent sidereal time includes secular and periodic contributions.

`[thGMST, thGAST] = greenwichSRT(utcJD, dUT1, dAT)` calculates Greenwich mean and apparent sidereal times at a higher precision using Earth orientation parameters.

## Examples

### Calculate Greenwich Sidereal Times

Calculate Greenwich sidereal times at 12:00 on January 4, 2019.

```
jd = juliandate([2019 1 4 12 0 0]);
[thGMST, thGAST] = greenwichSRT(jd)
```

```
thGMST =
    283.8103
```

```
thGAST =
    283.8065
```

## Input Arguments

### utcJD — UTC as Julian date

scalar

Universal Coordinated Time (UTC) as a Julian date, specified as a scalar.

---

**Tip** To calculate the Julian date for a particular date, use the `juliandate` function.

---

Data Types: double

### dUT1 — Difference between CUT and UT1

0 (default) | scalar

Difference between the Coordinated Universal Time (UTC) and Universal Time (UT1), specified as a scalar, in seconds.

**dAT – Difference between TAI and UTC**

0 (default) | scalar

Difference between International Atomic Time (TAI) and Coordinated Universal Time (UTC), specified as a scalar, in seconds.

**Output Arguments****thGMST – Greenwich mean sidereal time**

scalar

Greenwich mean sidereal time, specified as a scalar, in seconds.

**thGAST – Greenwich apparent sidereal time**

scalar

Greenwich apparent sidereal time, specified as a scalar, in seconds.

**Limitations**

This function requires the Mapping Toolbox license.

**References**

[1] Vallado, D. A. *Fundamentals of Astrodynamics and Applications*. alg. 1 and eqs. 1-63. New York: McGraw-Hill, 1997.

**See Also**

CubeSat Vehicle | asbCubeSatMissionAnalysis | dcmeci2ecef | ecef2eci | eci2ecef | generateAccessTable | greenwichsrt

**Introduced in R2019a**

# ijk2keplerian

Keplerian orbit elements using position and velocity vectors

## Syntax

```
[a,ecc,incl,RAAN,argp,nu,truelon,arglat,lonper] = ijk2keplerian(r_ijk, v_ijk)
```

## Description

[a,ecc,incl,RAAN,argp,nu,truelon,arglat,lonper] = ijk2keplerian(r\_ijk, v\_ijk) calculates Keplerian orbit elements for given position and velocity vectors in the geocentric equatorial coordinate system.

## Examples

### Convert IJK Position and Velocity

Convert the geocentric equatorial coordinate system (IJK) position and velocity to Keplerian orbital elements.

```
r_ijk = [-2981784 5207055 3161595];
v_ijk = [-3384 -4887 4843];
[a, ecc, incl, RAAN, argp, nu, truelon, arglat, lonper] = ...
    ijk2keplerian(r_ijk, v_ijk)
```

```
a =
    6.7845e+06
```

```
ecc =
    9.1950e-04
```

```
incl =
    51.7528
```

```
RAAN =
    95.2570
```

```
argp =
    106.4005
```

```
nu =
    290.0096
```

```
truelon =
    NaN
```

```
arglat =
    NaN
```

```
lonper =  
    NaN
```

## Input Arguments

### **r\_ijk** — Position component

0 (default) | 3-by-1 array

Geocentric equatorial position components, specified as a 3-by-1 array.

Data Types: `double`

### **v\_ijk** — Velocity component

0 (default) | 3-by-1 array

Geocentric equatorial velocity components, specified as a 3-by-1 array.

Data Types: `double`

## Output Arguments

### **a** — Semi-major axis

scalar

Semimajor axis (half of the longest diameter) of the orbit, returned as a scalar, in meters.

Data Types: `double`

### **ecc** — Orbit eccentricity

scalar value greater than or equal to 0

Orbit eccentricity (deviation of orbital curve from circular), returned as a scalar.

Data Types: `double`

### **incl** — Inclination

scalar value from 0 to 180

Inclination (tilt angle) of the orbit, in degrees.

Data Types: `double`

### **RAAN** — Right ascension of ascending node

scalar value from 0 to 360

Angle in the equatorial plane from the x-axis to the location of the ascending node (point at which the satellite crosses the equator from south to north), in degrees.

Data Types: `double`

### **argp** — Angle between CubeSat ascending node and periapsis

scalar value from 0 to 360

Angle between the CubeSat ascending node and periapsis (closest point of orbit to Earth), in degrees.

Data Types: `double`



**nu — Angle between periapsis and current position of CubeSat**

scalar value from 0 to 360

Angle between periapsis and current position of CubeSat, in degrees.

Data Types: double

**trueLon — Angle between x-axis and CubeSat position vector**

scalar value from 0 to 360

Angle between the x-axis and CubeSat position vector, in degrees.

Data Types: double

**argLat — Angle between ascending node and CubeSat position vector**

scalar value from 0 to 360

Angle between the ascending node and the CubeSat position vector, in degrees.

Data Types: double

**lonper — Angle between x-axis and eccentricity vector**

scalar value from 0 to 360

Angle between the x-axis and the eccentricity vector, in degrees.

Data Types: double

**References**

[1] Vallado, D. A. *Fundamentals of Astrodynamics and Applications*. alg. 5. McGraw-Hill, 1997.

**See Also**

CubeSat Vehicle | aeroReadIERSData | asbCubeSatMissionAnalysis | dcmeci2ecef | deltaCIP | deltaUT1 | ecef2eci | eci2ecef | generateAccessTable | greenwichsrt | keplerian2ijk | polarMotion

**Introduced in R2019a**

## keplerian2ijk

Position and velocity vectors in geocentric equatorial coordinate system using Keplerian orbit elements

### Syntax

```
[r_ijk,v_ijk] = keplerian2ijk(a,ecc,incl,RAAN,argp,nu)
[r_ijk,v_ijk] = keplerian2ijk( ___,Name,Value)
```

### Description

`[r_ijk,v_ijk] = keplerian2ijk(a,ecc,incl,RAAN,argp,nu)` calculates the position and velocity vectors in the geocentric equatorial coordinate system (IJK) for given Keplerian orbit elements of noncircular, inclined orbits.

`[r_ijk,v_ijk] = keplerian2ijk( ___,Name,Value)` specifies orbit element properties using one or more name-value pair arguments. For example, 'truelon', '17' specifies the angle between the x-axis and CubeSat position vector. Specify name-value pair arguments after all other input arguments.

### Examples

#### Convert Keplerian Orbital Elements

Convert Keplerian orbital elements to geocentric equatorial coordinate system (IJK) position and velocity.

```
a = 6786230;
ecc = .01;
incl = 52;
RAAN = 95;
argp = 93;
nu = 300;
[r_ijk, v_ijk] = keplerian2ijk(a, ecc, incl, RAAN, argp, nu)
```

```
r_ijk =
  1.0e+06 *
   -2.7489
    5.4437
    2.8977
```

```
v_ijk =
  1.0e+03 *
   -3.5694
   -4.5794
    5.0621
```

## Convert Keplerian Orbital Elements for Equatorial Orbit

Convert Keplerian orbital elements to geocentric equatorial coordinate system (IJK) position and velocity for equatorial orbit.

```
a = 6786230;
ecc = .1;
incl = 0;
RAAN = 95;
argp = 93;
nu = 300;
lonper = 45;
[r_ijk, v_ijk] = keplerian2ijk(a, ecc, incl, RAAN, argp, nu, 'lonper', lonper)
```

```
r_ijk =
    1.0e+06 *
    6.1804
   -1.6560
         0
```

```
v_ijk =
    1.0e+03 *
    1.4489
    7.9848
         0
```

## Input Arguments

### **a** — Semi-major axis

scalar

Semimajor axis (half of the longest diameter) of the orbit, specified as a scalar, in meters.

Data Types: double

### **ecc** — Orbit eccentricity

0 (default) | scalar value greater than or equal to 0

Orbit eccentricity (deviation of orbital curve from circular), specified as a scalar.

Data Types: double

### **incl** — Inclination

0 (default) | scalar value from 0 to 180

Inclination (tilt angle) of the orbit, in degrees.

Data Types: double

### **RAAN** — Right ascension of ascending node

0 (default) | scalar value from 0 to 360

Angle in the equatorial plane from the x-axis to the location of the ascending node, point at which the satellite crosses the equator from south to north, in degrees. The function does not use this value for equatorial orbits.

Data Types: double

**argp — Angle between CubeSat ascending node and periapsis** $\theta$  (default) | scalar value from 0 to 360

Angle between the CubeSat ascending node and the periapsis (closest point of orbit to Earth), in degrees. The function does not use this value for circular and equatorial orbits.

Data Types: double

**nu — Angle between periapsis and current position of CubeSat** $\theta$  (default) | scalar value from 0 to 360

Angle between the periapsis and the current position of CubeSat, in degrees. The function does not use this value for circular orbits.

Data Types: double

**Name-Value Pair Arguments**

Specify optional comma-separated pairs of `Name`, `Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1, Value1, . . . , NameN, ValueN`.

Example: 45

**trueLon — Angle between x-axis and CubeSat position vector** $\theta$  (default) | scalar value from 0 to 360

Angle between the x-axis and the CubeSat position vector, in degrees. The function uses this value only for circular equatorial orbits (where eccentricity and inclination are zero).

Data Types: double

**argLat — Angle between ascending node and CubeSat position vector** $\theta$  (default) | scalar value from 0 to 360

Angle between the ascending node and the CubeSat position vector, in degrees. The function uses this value only for circular inclined orbits (where eccentricity is zero and inclination is nonzero).

Data Types: double

**lonper — Angle between x-axis and eccentricity vector** $\theta$  | scalar value from 0 to 360

Angle between the x-axis and the eccentricity vector, in degrees. The function uses this value only for noncircular equatorial orbits (where eccentricity is nonzero and inclination is zero).

Data Types: double

**Output Arguments****r\_ijk — Position component**

3-by-1 array

Geocentric equatorial position components, specified as a 3-by-1 array.

**v\_ijk — Velocity component**

3-by-1 array

Geocentric equatorial velocity components, specified as a 3-by-1 array.

## References

[1] Vallado, D. A. *Fundamentals of Astrodynamics and Applications*. alg. 5. McGraw-Hill, 1997.

## See Also

CubeSat Vehicle | aeroReadIERSData | asbCubeSatMissionAnalysis | dcmeci2ecef | deltaCIP | deltaUT1 | ecef2eci | eci2ecef | generateAccessTable | greenwichsrt | ijk2keplerian | keplerian2ijk | polarMotion

**Introduced in R2019a**

## linearizeAirframe

Linearize airframe model around operating points

### Syntax

```
linSys = linearizeAirframe(modelToAnalyze)
linSys = linearizeAirframe(modelToAnalyze)
linSys = linearizeAirframe(modelToAnalyze,opPoint)
linSys = linearizeAirframe(modelToAnalyze,opPoint,generatePlots)
```

### Description

`linSys = linearizeAirframe(modelToAnalyze)` linearizes an airframe model around a specified operating point or operating point specification object and generates an output state-space model that contains only longitudinal states. A **Linearize Airframe** dialog window prompts you to select an operating point or operating point specification object from the base workspace. If an operating point or operating point specification object does not exist in the base workspace, click the **Launch Trim Tool** button in the **Linearize Airframe** dialog window. This button starts the Simulink Control Design Model Linearizer in which you can create the operating point specification object. The `linearizeAirframe` function uses this object as the operating condition around which to linearize the airframe model.

`linSys = linearizeAirframe(modelToAnalyze)` linearizes an airframe model around the specified operating point object or operating point specification object.

`linSys = linearizeAirframe(modelToAnalyze,opPoint)` linearizes an airframe model around the specified operating point object or operating point specification object.

`linSys = linearizeAirframe(modelToAnalyze,opPoint,generatePlots)` displays bode and step plot results of longitudinal linearization.

### Examples

#### Linearize Model Around a Provided Operating Point Specification Object

Linearize the model `SkyHoggAnalysisModel` around the operating point, `opTrim`. This example starts the flight control analysis template using `asbFlightControlAnalysis` and trims the model around the `opSpecDefault` operating point specification object. It then linearizes the airframe model around the `opTrim` operating point and calculates the short- and long-period (phugoid) mode characteristics of `linSys`.

```
asbFlightControlAnalysis('3DOF', 'SkyHoggAnalysisModel');
opSpecDefault = SkyHogg3DOF0pSpec('SkyHoggAnalysisModel');
opTrim = trimAirframe('SkyHoggAnalysisModel', opSpecDefault);
```

```
linSys = linearizeAirframe('SkyHoggAnalysisModel', opTrim)
flyingQual = computeLongitudinalFlyingQualities('SkyHoggAnalysisModel', linSys)
```

## Input Arguments

### **modelToAnalyze** — Model on which to perform flight control analysis

model name

Model on which to perform flight control analysis. This model must be previously created with the `asbFlightControlAnalysis` function.

Data Types: char | string

### **opPoint** — Operating point object

operating point object

Operating point object used to linearize the model `modelToAnalyze`.

Data Types: char | string

### **generatePlots** — Display pole-zero map

model name

Display pole-zero map for the linear system state-space model.

Data Types: char | string

## Output Arguments

### **linSys** — State-space model object

linear state-space model object name

State space model object representing the linearized airframe model at a specified operating point.

Data Types: char | string

## Limitations

This function requires the Simulink Control Design license.

## See Also

**Model Linearizer** | `asbFlightControlAnalysis` | `computeLateralDirectionalFlyingQualities` | `computeLongitudinalFlyingQualities` | `trimAirframe`

## Topics

“Analyze Dynamic Response and Flying Qualities of Aerospace Vehicles” on page 2-45

## Introduced in R2018b

## linearizeLongitudinalAirframe (Obsolete)

Linearize airframe model around operating points

### Syntax

```
linearizeLongitudinalAirframe(modelToAnalyze)
linearizeLongitudinalAirframe(modelToAnalyze,opPoint)
linearizeLongitudinalAirframe(modelToAnalyze,opPoint,generatePlots)
```

### Description

---

**Note** This function is obsolete. Use `linearizeAirframe` instead.

---

`linearizeLongitudinalAirframe(modelToAnalyze)` linearizes an airframe model around a specified operating point or operating point specification object and generates an output state-space model that contains only longitudinal states. A **Linearize Airframe** dialog window prompts you to select an operating point or operating point specification object from the base workspace. If an operating point or operating point specification object does not exist in the base workspace, click the **Launch Trim Tool** button in the **Trim Airframe** dialog window. This button starts the Simulink Control Design Model Linearizer in which you can create the operating point specification object. From this object, the `linearizeLongitudinalAirframe` function creates the operating point.

`linearizeLongitudinalAirframe(modelToAnalyze,opPoint)` linearizes an airframe model around the specified operating point object or operating point specification object.

`linearizeLongitudinalAirframe(modelToAnalyze,opPoint,generatePlots)` displays bode and step plot results of longitudinal linearization.

### Examples

#### Linearize Model While Specifying an Operating Point Specification Object

Linearize the model `SkyHoggAnalysisModel` and specify an operating point, `opTrim`. This example starts the flight control analysis template using `asbFlightControlAnalysis` and trims the model around the `opSpecDefault` operating point specification object. It then linearizes the airframe model around the `opTrim` operating point and calculates the short- and long-period (phugoid) mode characteristics of `linSys`.

```
asbFlightControlAnalysis('3DOF', 'SkyHoggAnalysisModel');
opSpecDefault = SkyHogg3DOF0pSpec('SkyHoggAnalysisModel');
opTrim = trimAirframe('SkyHoggAnalysisModel', opSpecDefault);
linSys = linearizeLongitudinalAirframe('SkyHoggAnalysisModel', opTrim)
flyingQual = computeLongitudinalFlyingQualities('SkyHoggAnalysisModel', linSys)
```

### Input Arguments

**modelToAnalyze** — Model on which to perform flight control analysis  
model name



Model on which to perform flight control analysis using the linear state-space model `linSys`. This model must be previously created with the `asbFlightControlAnalysis` function.

Data Types: `char` | `string`

#### **opPoint — Linear state-space model**

linear state-space model name

Linear state-space model used to perform flight control analysis on `modelToAnalyze`.

Data Types: `char` | `string`

#### **generatePlots — Display pole-zero map**

model name

Display pole-zero map for the linear system state-space model.

Data Types: `char` | `string`

## **Limitations**

This function requires the Simulink Control Design license.

## **See Also**

**Model Linearizer** | `asbFlightControlAnalysis` | `computeLongitudinalFlyingQualities` | `linearizeAirframe` | `trimAirframe`

## **Topics**

“Analyze Dynamic Response and Flying Qualities of Aerospace Vehicles” on page 2-45

## **Introduced in R2018b**

## trimAirframe

Trim airframe around operating point specification object

### Syntax

```
trimAirframe(modelToAnalyze)
trimAirframe(modelToAnalyze,opSpec)
```

### Description

`trimAirframe(modelToAnalyze)` trims the airframe around an operating point specification object. A **Trim Airframe** dialog window prompts you to select an operating point specification object from the base workspace. If an operating point specification object does not exist in the base workspace, click the **Launch Trim Tool** button in the **Trim Airframe** dialog window. This button starts the Simulink Control Design Model Linearizer in which you can create the operating point specification object. From this object, the `trimAirframe` function trims the airframe.

`trimAirframe(modelToAnalyze,opSpec)` trims the airframe model around the specified operating point specification object.

### Examples

#### Trim Model While Specifying an Operating Point Specification Object

Trim the model `SkyHoggAnalysisModel` around an operating point specification object, `opSpecDefault`. This example starts the flight control analysis template using `asbFlightControlAnalysis` and trims the model around the `opSpecDefault` operating point. It then linearizes the airframe model around the `opTrim` operating point specification object and calculates the short- and long-period (phugoid) mode characteristics of `linSys`.

```
asbFlightControlAnalysis('3DOF', 'SkyHoggAnalysisModel');
opSpecDefault = SkyHogg3DOF0pSpec('SkyHoggAnalysisModel');
opTrim = trimAirframe('SkyHoggAnalysisModel', opSpecDefault);
linSys = linearizeAirframe('SkyHoggAnalysisModel', opTrim)
flyingQual = computeLongitudinalFlyingQualities('SkyHoggAnalysisModel', linSys)
```

### Input Arguments

#### modelToAnalyze — Model on which to perform flight control analysis

model name

Model on which to perform flight control analysis using the linear state-space model `linSys`. This model must be previously created with the `asbFlightControlAnalysis` function.

Data Types: `char` | `string`

#### opSpec — Linear state-space model

linear state-space model name

Linear state-space model used to perform flight control analysis on `modelToAnalyze`.

Data Types: char | string

## Limitations

This function requires the Simulink Control Design license.

## See Also

**Model Linearizer** | `asbFlightControlAnalysis` | `computeLateralDirectionalFlyingQualities` | `computeLongitudinalFlyingQualities` | `linearizeAirframe`

## Topics

“Analyze Dynamic Response and Flying Qualities of Aerospace Vehicles” on page 2-45

## Introduced in R2018b



# Examples

---

- “1903 Wright Flyer and Pilot with Scopes for Data Visualization” on page 6-2
- “1903 Wright Flyer and Pilot with Simulink® 3D Animation™” on page 6-4
- “Fly the De Havilland Beaver” on page 6-7
- “Lightweight Airplane Design” on page 6-9
- “Multiple Aircraft with Collaborative Control” on page 6-25
- “HL-20 with Flight Instrumentation Blocks” on page 6-27
- “HL-20 with Simulink® 3D Animation™ and Flight Instrumentation Blocks” on page 6-32
- “HL-20 Project with Optional FlightGear Interface” on page 6-37
- “Quaternion Estimate from Measured Rates” on page 6-39
- “Indicated Airspeed from True Airspeed Calculation” on page 6-40
- “Six Degree of Freedom Motion Platform” on page 6-42
- “Gravity Models with Precessing Reference Frame” on page 6-45
- “True Airspeed from Indicated Airspeed Calculation” on page 6-48
- “Airframe Trim and Linearize with Simulink® Control Design™” on page 6-50
- “Airframe Trim and Linearize with Control System Toolbox™” on page 6-53
- “Self-conditioned Controller Comparison” on page 6-57
- “Quadcopter Project” on page 6-59
- “Electrical Component Analysis for Hybrid and Electric Aircraft” on page 6-66

## 1903 Wright Flyer and Pilot with Scopes for Data Visualization

This model shows how to model the Wright Brother's 1903 Flyer modeled in Simulink®, and Aerospace Blockset™ software. This model simulates the longitudinal motion of the Flyer in response to the pitch commands of a simulated pilot.

December 17, 2003 marked the centennial of the first powered, heavier-than-air controlled flight. This first flight happened at Kitty Hawk, North Carolina, on December 17, 1903 at 10:30 am. With a flight lasting only 12 seconds and traveling a distance of 120 feet, Orville Wright piloted his way into flight history. Three other flights occurred that day with Wilbur and Orville taking turns at the controls. Each of the flights was of increasing distance. The fourth and final flight of the day completed by Wilbur was an impressive 59 seconds traveling 852 feet. The 1903 Flyer would not take to the skies again. After the last flight of the day, the Flyer was damaged beyond repair when it was caught by a gust of wind and rolled over.

Additional information about the 1903 Flyer can be found at NASA Web Site: [Re-Living The Wright Way](#) and on MathWorks® web site: [The Wright Stuff Celebrating The 1903 Flyer](#)

A technical reference is Hooven, Frederick J., 'Longitudinal Dynamics of the Wright Brothers' Early Flyers 'A Study in Computer Simulation of Flight', from *The Wright Flyer An Engineering Perspective* edited by Howard S. Wolko, 1987.

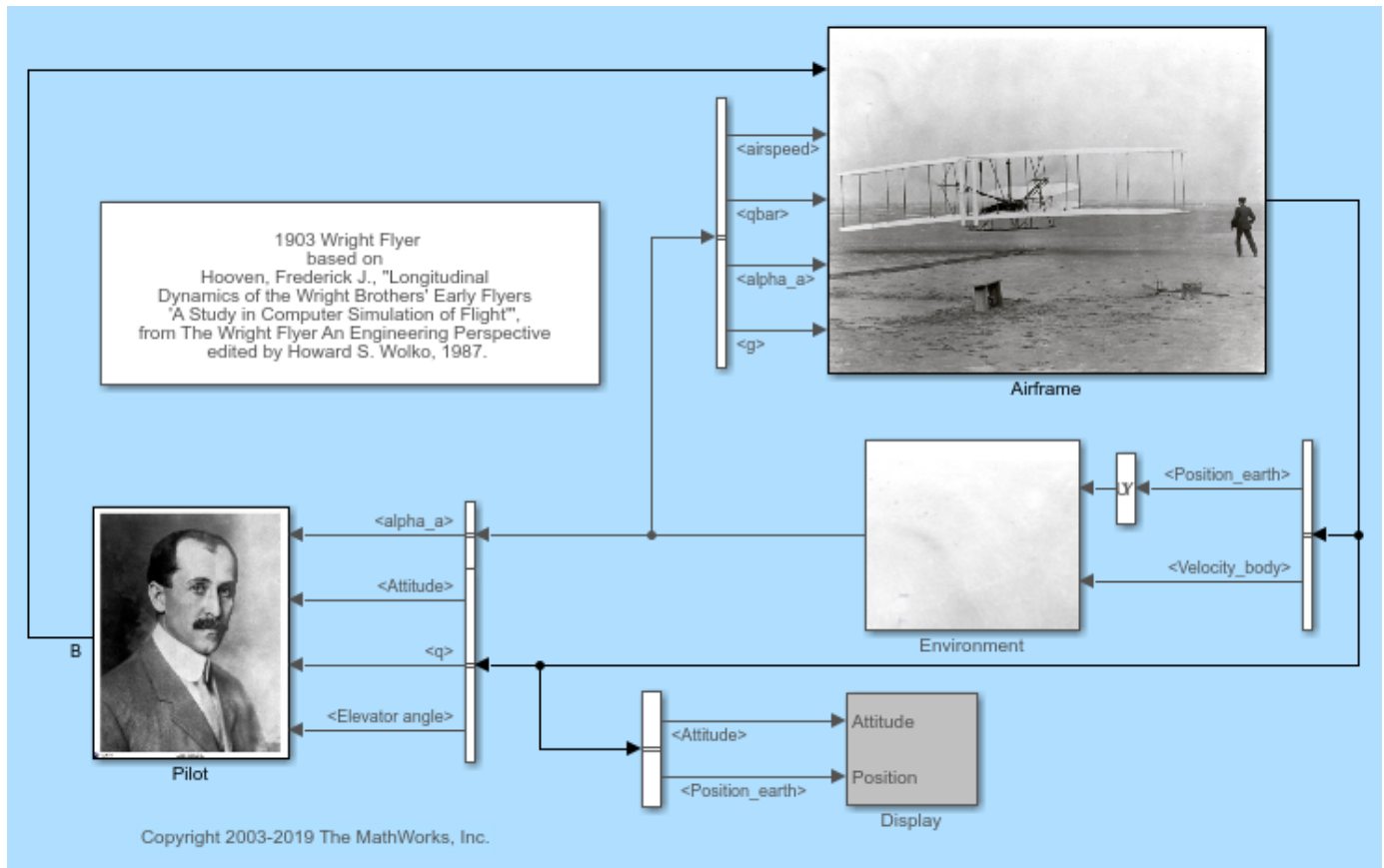
Note that the following warning messages are from a Simulink assertion block, used to determine if the Flyer has landed or stalled.

Landing

Warning: Assertion detected in 'aeroblk\_wf\_3dof\_noVR/Airframe/Touch Down?/Check Touch Down/Land?' at time 2.529176.

Hitting Ground

Warning: Assertion detected in 'aeroblk\_wf\_3dof\_noVR/Airframe/Touch Down?/Altitude?' at time 2.529176.



## 1903 Wright Flyer and Pilot with Simulink® 3D Animation™

This model shows how to model the Wright Brother's 1903 Flyer modeled in Simulink®, Aerospace Blockset™ and Simulink® 3D Animation™ software. This model simulates the longitudinal motion of the Flyer in response to the pitch commands of a simulated pilot.

December 17, 2003 marked the centennial of the first powered, heavier-than-air controlled flight. This first flight happened at Kitty Hawk, North Carolina, on December 17, 1903 at 10:30 am. With a flight lasting only 12 seconds and traveling a distance of 120 feet, Orville Wright piloted his way into flight history. Three other flights occurred that day with Wilbur and Orville taking turns at the controls. Each of the flights was of increasing distance. The fourth and final flight of the day completed by Wilbur was an impressive 59 seconds traveling 852 feet. The 1903 Flyer would not take to the skies again. After the last flight of the day, the Flyer was damaged beyond repair when it was caught by a gust of wind and rolled over.

Additional information about the 1903 Flyer can be found at NASA Web Site: [Re-Living The Wright Way](#) and on MathWorks® web site: [The Wright Stuff Celebrating The 1903 Flyer](#)

A technical reference is Hooven, Frederick J., 'Longitudinal Dynamics of the Wright Brothers' Early Flyers 'A Study in Computer Simulation of Flight', from *The Wright Flyer An Engineering Perspective* edited by Howard S. Wolko, 1987.

Note that the following warning messages are from a Simulink assertion block, used to determine if the Flyer has landed or stalled.

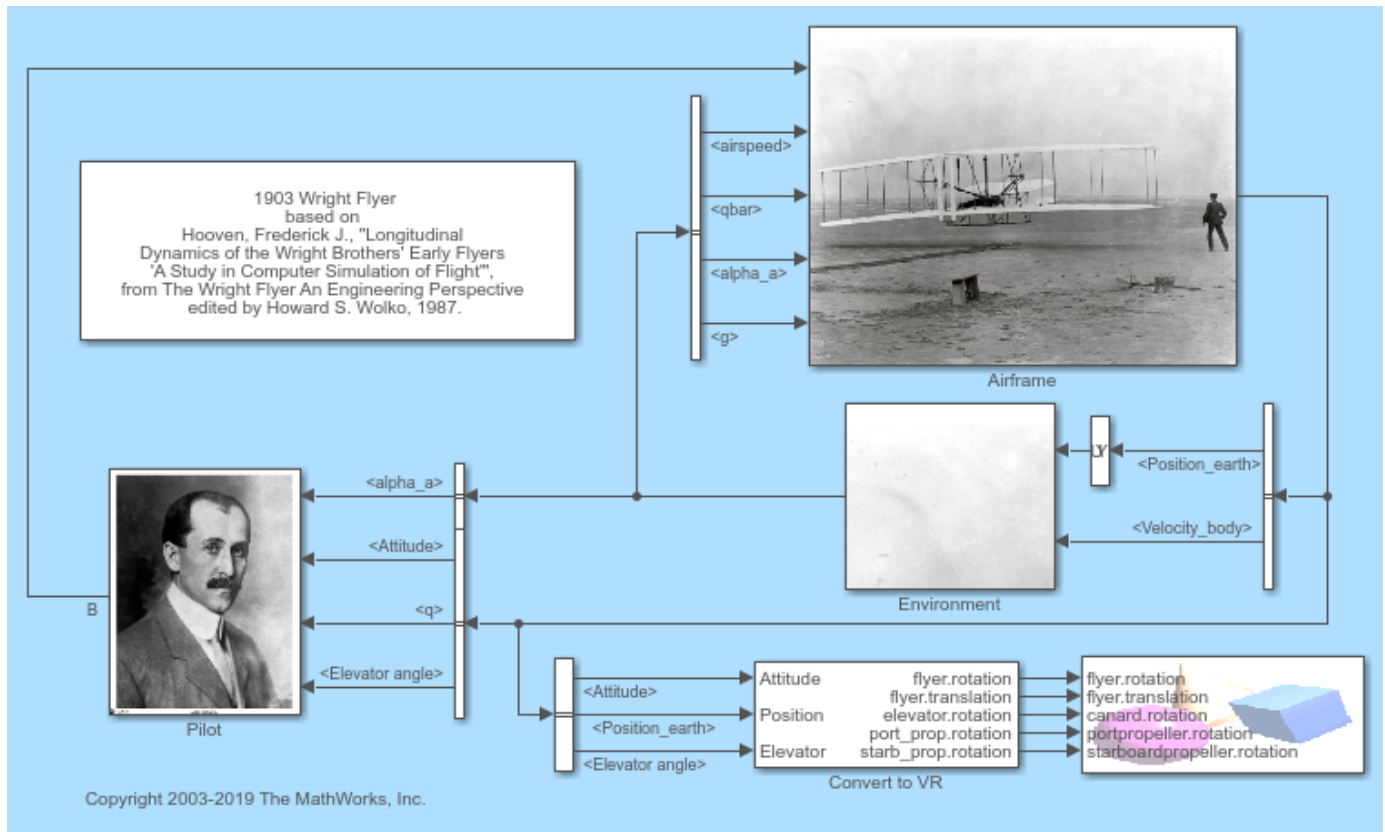
Landing

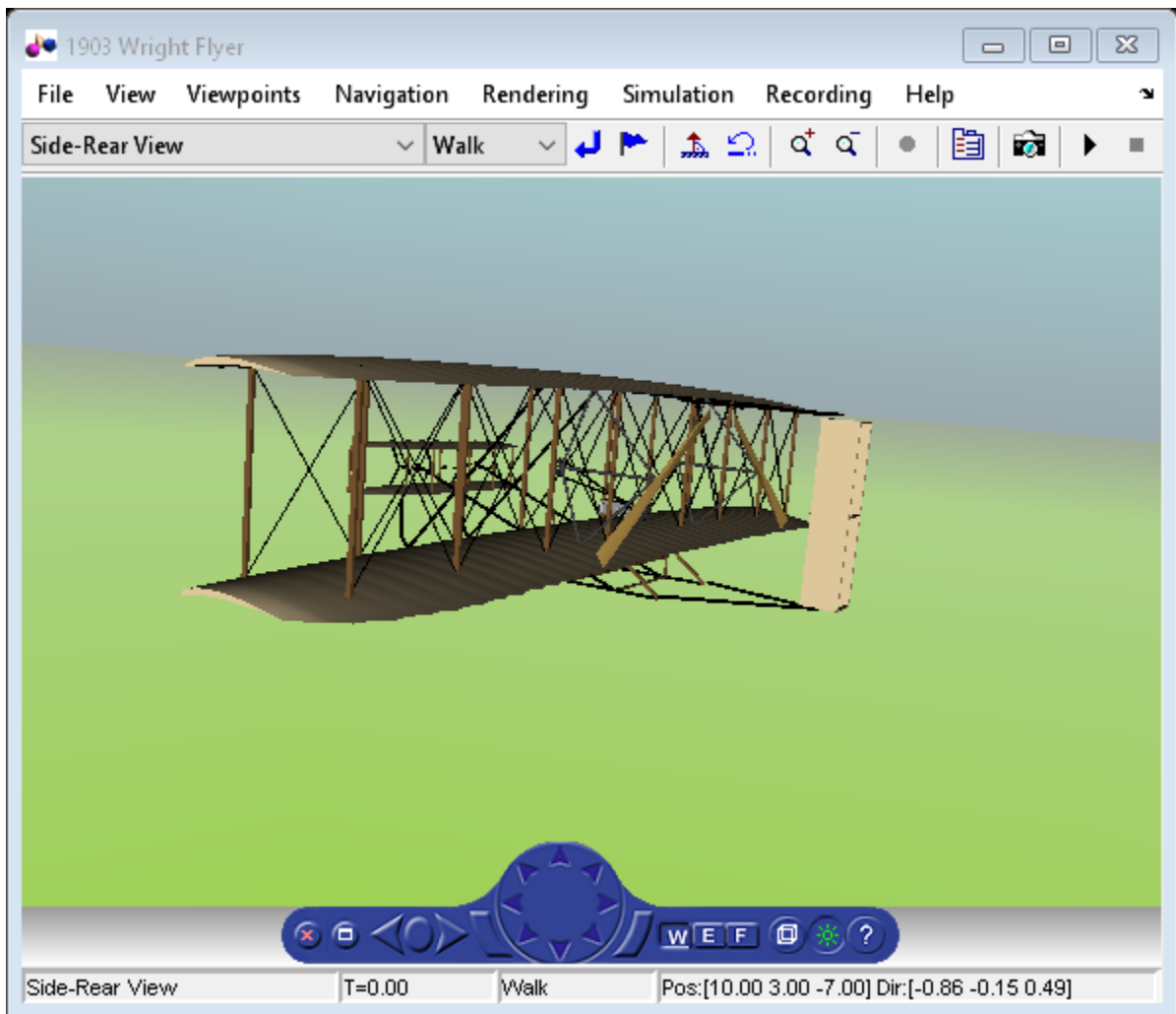
Warning: Assertion detected in 'aeroblk\_wf\_3dof/Airframe/Touch Down?/Check Touch Down/Land?' at time 2.529176.

Hitting Ground

Warning: Assertion detected in 'aeroblk\_wf\_3dof/Airframe/Touch Down?/Altitude?' at time 2.529176.







## Fly the De Havilland Beaver

This model shows how to model the De Havilland Beaver using Simulink® and Aerospace Blockset™ software. It also shows how to use a pilot's joystick to fly the De Havilland Beaver. This model has been color-coded to aid in locating Aerospace Blockset blocks. The red blocks are Aerospace Blockset blocks, the orange blocks are subsystems containing additional Aerospace Blockset blocks, and the white blocks are Simulink blocks.

The De Havilland Beaver model includes the airframe dynamics and aerodynamics. Effects of the environment are also modeled, such as wind profiles for the landing phase. Visualization for this model is done via an interface to FlightGear, an open source flight simulator package.

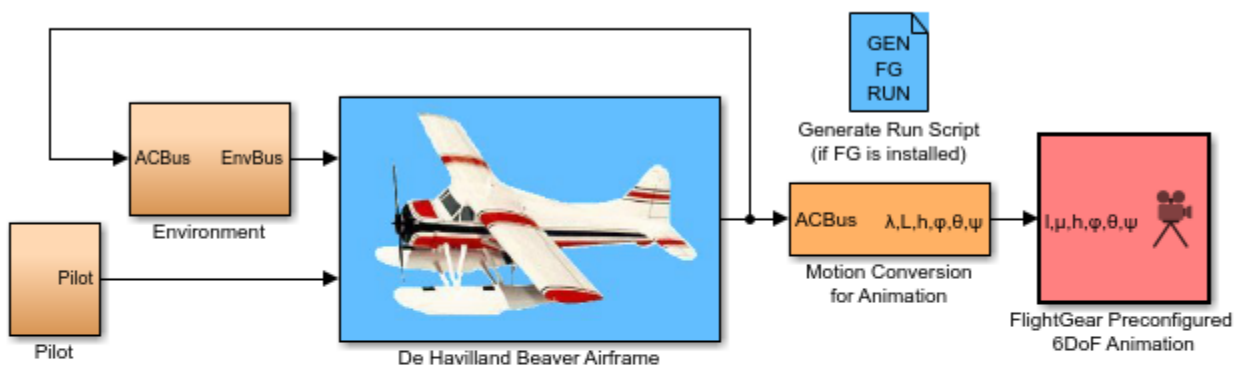
For more information on the FlightGear interface, read these documentation topics:

“Flight Simulator Interface” on page 2-16

“Work with the Flight Simulator Interface” on page 2-20

“Run the HL-20 Example with FlightGear” on page 2-29

### Fly the De Havilland Beaver



De Havilland Beaver model  
1.76  
Based on original work created by  
Marc Rauw for Delft University of Technology  
<http://www.dutchroll.com>

How to run the De Havilland model:

1. See the Aerospace Blockset User's Guide for instructions to set up FlightGear.
2. Install the `dhc2` geometry model to FlightGear's `data/Aircraft` directory. The geometry is downloadable from [www.flightgear.org](http://www.flightgear.org).
3. To start FlightGear, generate run script and run generated batch file by typing `dos('runfg.bat &')` at the MATLAB command line.

Copyright 1990-2019 The MathWorks, Inc.

The De Havilland Beaver was first flown in 1947. Today it is still prized by pilots for its reliability and versatility. The De Havilland Beaver can be operated on wheels, skis or float landing gear.

Speed maximum: 110 kts, Altitude maximum: 10,000 ft, Range maximum: 400 nm, Load: 6 passengers, Crew: 1 member.

# Lightweight Airplane Design

This model shows how to use MathWorks® products to address the technical and process challenges of aircraft design using the design of a lightweight aircraft.

To run this example model, you need Aerospace Blockset™ software and its required products. Additional products you will need to explore this model further are:

- Control System Toolbox™
- Simulink® Control Design™
- Simulink® Design Optimization™

The design process is iterative; you will try many vehicle configurations before selecting the final one. Ideally, you perform iterations before building any hardware. The challenge is to perform the iterations quickly. Typically, different groups work on different steps of the process. Effective collaboration among these groups and the right set of tools are essential to addressing this challenge.

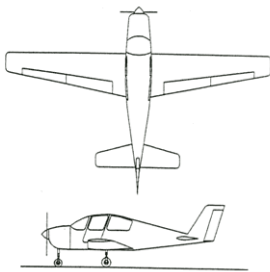
## Defining Vehicle Geometry

The geometry of this lightweight aircraft is from reference 1. The original design objective for this geometry was a four-seat general aviation aircraft that was safe, simple to fly, and easily maintainable with specific mission and performance constraints. For more details on these constraints, see reference 1.

Potential performance requirements for this aircraft include:

- Level cruise speed
- Acceptable rate of climb
- Acceptable stall speed.

For the aircraft flight control, rate of climb is the design requirement and assumed to be greater than 2 meters per second (m/s) at 2,000 meters.



**Figure 1:** Lightweight four-seater monoplane [1].

## Determining Vehicle Aerodynamic Characteristics

The aircraft's geometrical configuration determines its aerodynamic characteristics, and therefore its performance and handling qualities. Once you choose the geometric configuration, you can obtain the aerodynamic characteristics by means of:

- Analytical prediction

- Wind tunnel testing of the scaled model or a full-sized prototype
- Flight tests.

While wind tunnel tests and flight tests provide high-fidelity results, they are expensive and time-consuming, because they must be performed on the actual hardware. It is best to use these methods when the aircraft's geometry is finalized. **Note:** Analytical prediction is a quicker and less expensive way to estimate aerodynamic characteristics in the early stages of design.

In this example, we will use Digital Datcom, a popular software program, for analytical prediction. The U.S. Air Force developed it as a digital version of its Data Compendium (DATCOM). This software is publicly available.

To start, create a Digital Datcom input file that defines the geometric configuration of our aircraft and the flight conditions that we will need to obtain the aerodynamic coefficients.

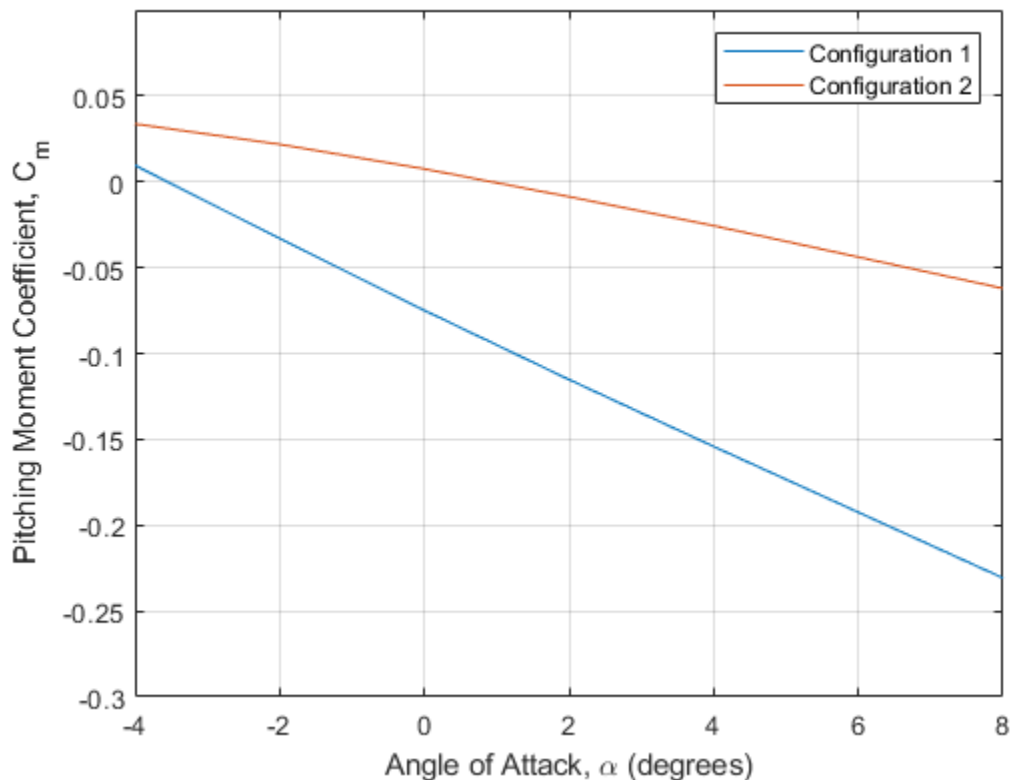
```
$FLTCON NMACH=4.0,MACH(1)=0.1,0.2,0.3,0.35$
$FLTCON NALT=8.0,ALT(1)=1000.0,3000.0,5000.0,7000.0,9000.0,
11000.0,13000.0,15000.0$
$FLTCON NALPHA=10.,ALSCHD(1)=-16.0,-12.0,-8.0,-4.0,-2.0,0.0,2.0,
ALSCHD(8)=4.0,8.0,12.0,LOOP=2.0$
$OPTINS SREF=225.8,CBARR=5.75,BLREF=41.15$
$SYNTHS XCG=7.9,ZCG=-1.4,XW=6.1,ZW=0.0,ALIW=1.1,XH=20.2,
ZH=0.4,ALIH=0.0,XV=21.3,ZV=0.0,VERTUP=.TRUE.$
$BODY NX=10.0,
X(1)=-4.9,0.0,3.0,6.1,9.1,13.3,20.2,23.5,25.9,
R(1)=0.0,1.0,1.75,2.6,2.6,2.6,2.0,1.0,0.0$
$WGPLNF CHRDT=4.0,SSPNE=18.7,SSPN=20.6,CHRDR=7.2,SAVSI=0.0,CHSTAT=0.25,
TWISTA=-1.1,SSPNDD=0.0,DHDADI=3.0,DHDADO=3.0,TYPE=1.0$
$HTPLNF CHRDT=2.3,SSPNE=5.7,SSPN=6.625,CHRDR=0.25,SAVSI=11.0,
CHSTAT=1.0,TWISTA=0.0,TYPE=1.0$
$VTPLNF CHRDT=2.7,SSPNE=5.0,SSPN=5.2,CHRDR=5.3,SAVSI=31.3,
CHSTAT=0.25,TWISTA=0.0,TYPE=1.0$
$SYMFLP NDELTA=5.0,DELTA(1)=-20.,-10.,0.,10.,20.,PHETE=.0522,
CHRFI=1.3,
CHRF0=1.3,SPANFI=.1,SPANF0=6.0,FTYPE=1.0,CB=1.3,TC=.0225,
PHETEP=.0391,NTYPE=1.$
NACA-W-4-0012
NACA-H-4-0012
NACA-V-4-0012
CASEID SKYHOGG BODY-WING-HORIZONTAL TAIL-VERTICAL TAIL CONFIG
DAMP
NEXT CASE
```

Digital Datcom provides the vehicle's aerodynamic stability and control derivatives and coefficients at specified flight conditions. Flight control engineers can gain insight into the vehicle's performance and handling characteristics by examining stability and control derivatives. We must import this data into the MATLAB® technical computing environment for analysis. Normally, this is a manual process.

With the Aerospace Toolbox software, we can bring multiple Digital Datcom output files into the MATLAB technical computing environment with just one command. There is no need for manual input. Each Digital Datcom output is imported into the MATLAB technical computing environment as a cell array of structures, with each structure corresponding to a different Digital Datcom output file. After importing the Digital Datcom output, we can run multiple configurations through Digital Datcom and compare the results in the MATLAB technical computing environment.

In our model, we need to check whether the vehicle is inherently stable. To do this, we can use Figure 2 to check whether the pitching moment described by the corresponding coefficient,  $C_m$ , provides a restoring moment for the aircraft. A restoring moment returns the aircraft angle of attack to zero.

In configuration 1 (Figure 2),  $C_m$  is negative for some angles of attack less than zero. This means that this configuration will not provide a restoring moment for those negative angles of attack and will not provide the flight characteristics that are desirable. Configuration 2 fixes this problem by moving the center of gravity rearward. Shifting the center of gravity produces a  $C_m$  that provides a restoring moment for all negative angles of attack.



**Figure 2:** Visual analysis of Digital Datcom pitching moment coefficients.

### Creating Flight Vehicle Simulation

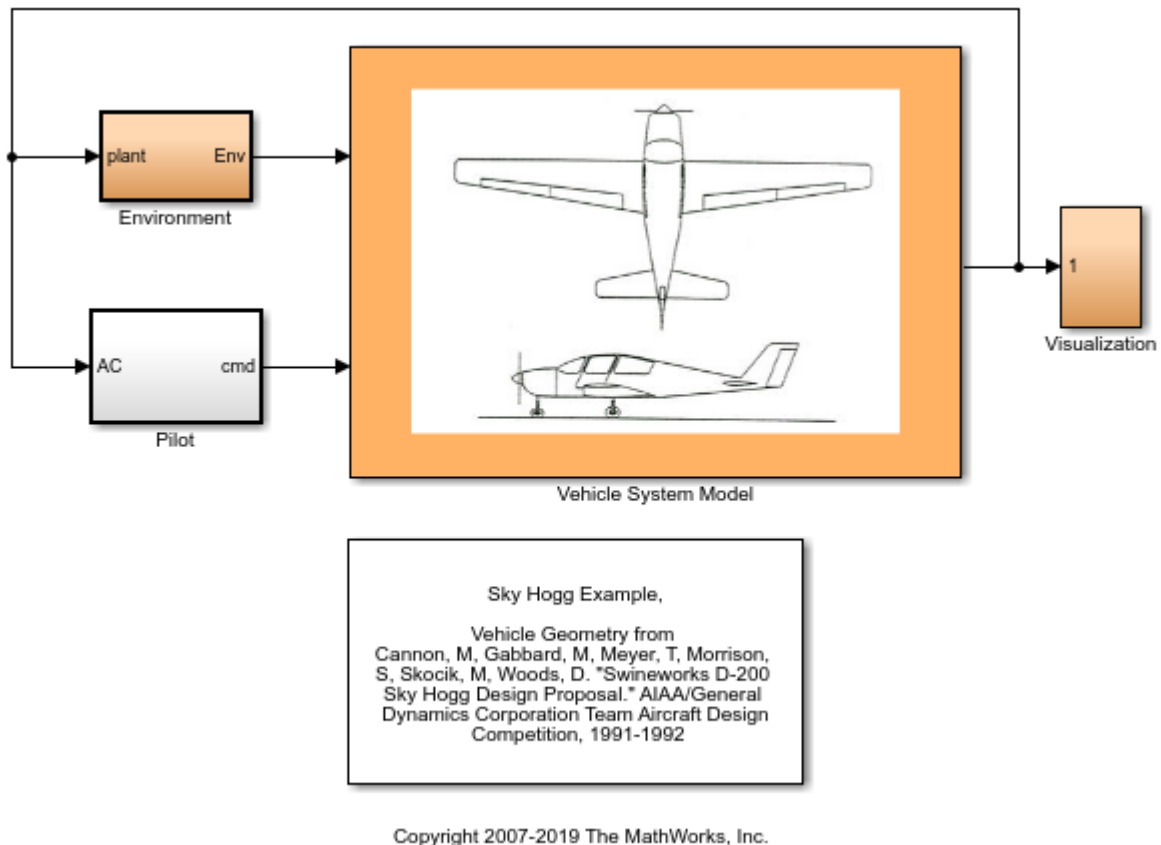
Once we determine aerodynamic stability and control derivatives, we can build an open-loop plant model to evaluate the aircraft longitudinal dynamics. Once the model is complete, we can show it to colleagues, including those who do not have Simulink® software, by using Simulink® Report Generator™ software to export the model to a Web view. A Web view is an interactive HTML replica of the model that lets you navigate model hierarchy and check the properties of subsystems, blocks, and signals.

A typical plant model includes the following components:

- **Equations of motion:** calculate vehicle position and attitude from forces and moments
- **Forces and moments:** calculate aerodynamic, gravity, and thrust forces and moments

- **Actuator positions:** calculate displacements based on actuator commands
- **Environment:** include environmental effects of wind disturbances, gravity, and atmosphere
- **Sensors:** model the behavior of the measurement devices

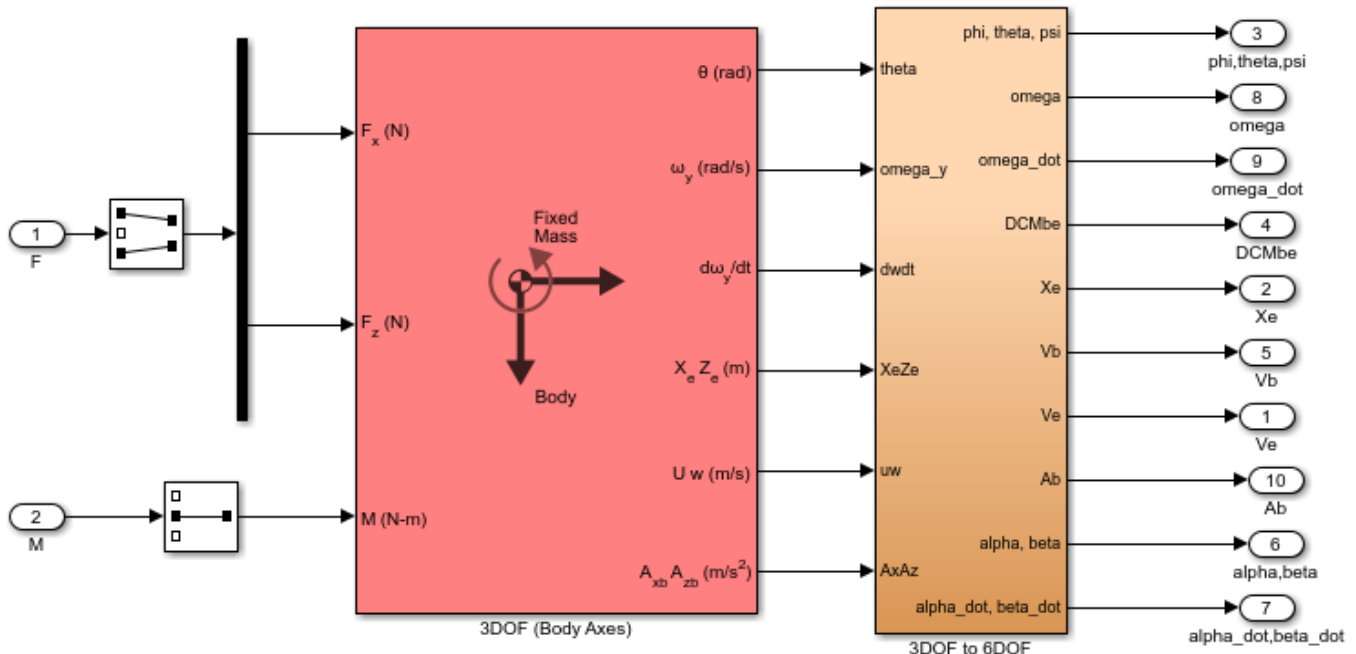
We can implement most of this functionality using Aerospace Blockset™ blocks. This model highlights subsystems containing Aerospace Blockset blocks in orange. It highlights Aerospace Blockset blocks in red.



**Figure 3:** Top Level of Lightweight Aircraft Model

We begin by building a plant model using a 3DOF block from the Equations of Motion library in the Aerospace Blockset library (Figure 4). This model will help us determine whether the flight vehicle is longitudinally stable and controllable. We design our subsystem to have the same interface as a six degrees-of-freedom (DOF) version. When we are satisfied with three DOF performance, stability, and controllability, we can implement the six DOF version, iterating on the other control surface geometries until we achieve the desired behavior from the aircraft.

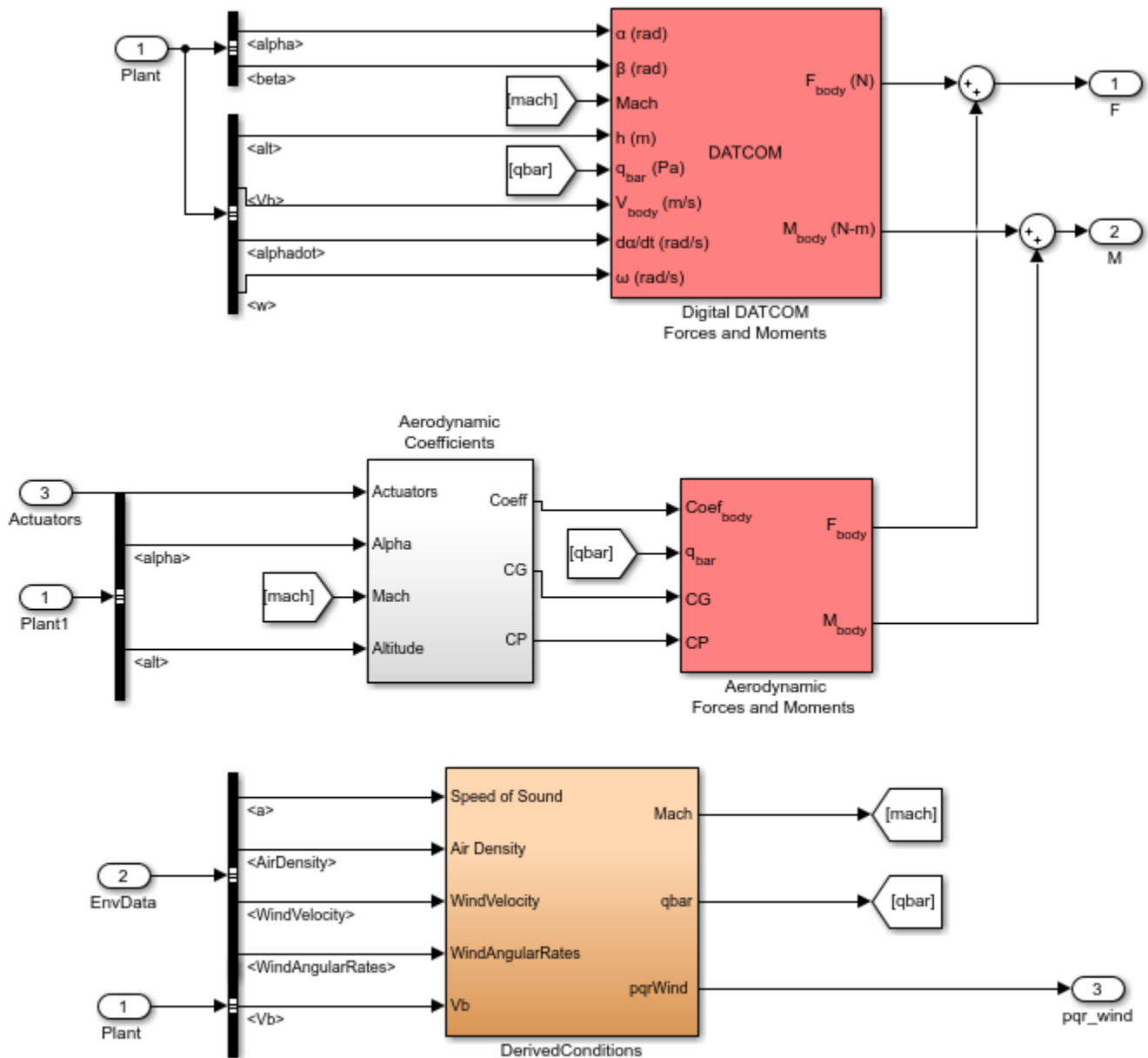




**Figure 4:** Equations of Motion implemented using 3DoF Euler block from the Aerospace Blockset library.

To calculate the aerodynamic forces and moments acting on our vehicle, we use a Digital Datcom Forces and Moments block from the Aerospace Blockset library (Figure 5). This block uses a structure that Aerospace Toolbox creates when it imports aerodynamic coefficients from Digital Datcom.

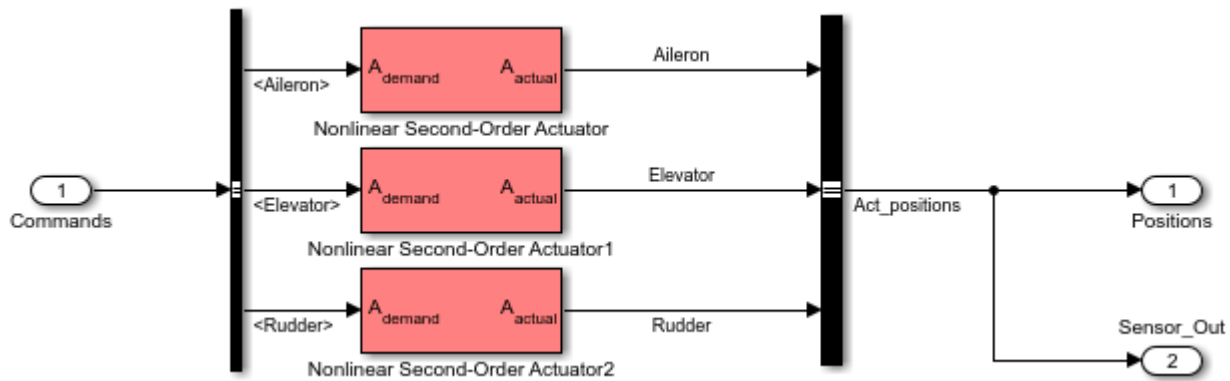
For some Digital Datcom cases, dynamic derivative have values for only the first angle of attack. The missing data points can be filled with the values for the first angle of attack, since these derivatives are independent of angle of attack. To see example code of how to fill in missing data in Digital Datcom data points, you can examine the `asbPrepDatcom` function.



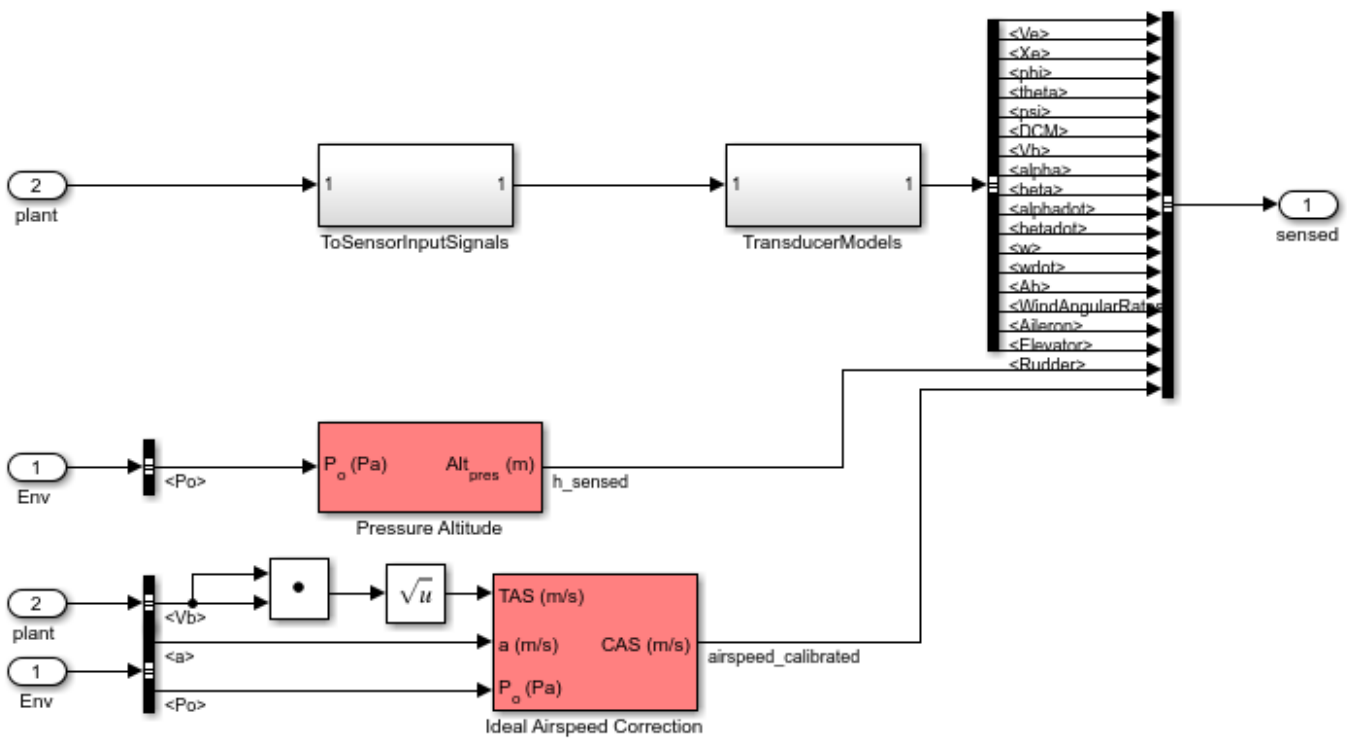
Aerodynamics model may add landing gear and ground effects at a later time.

**Figure 5:** Aerodynamic Forces and Moments implemented in part with the Aerospace Blockset Digital Datcom Forces and Moment block.

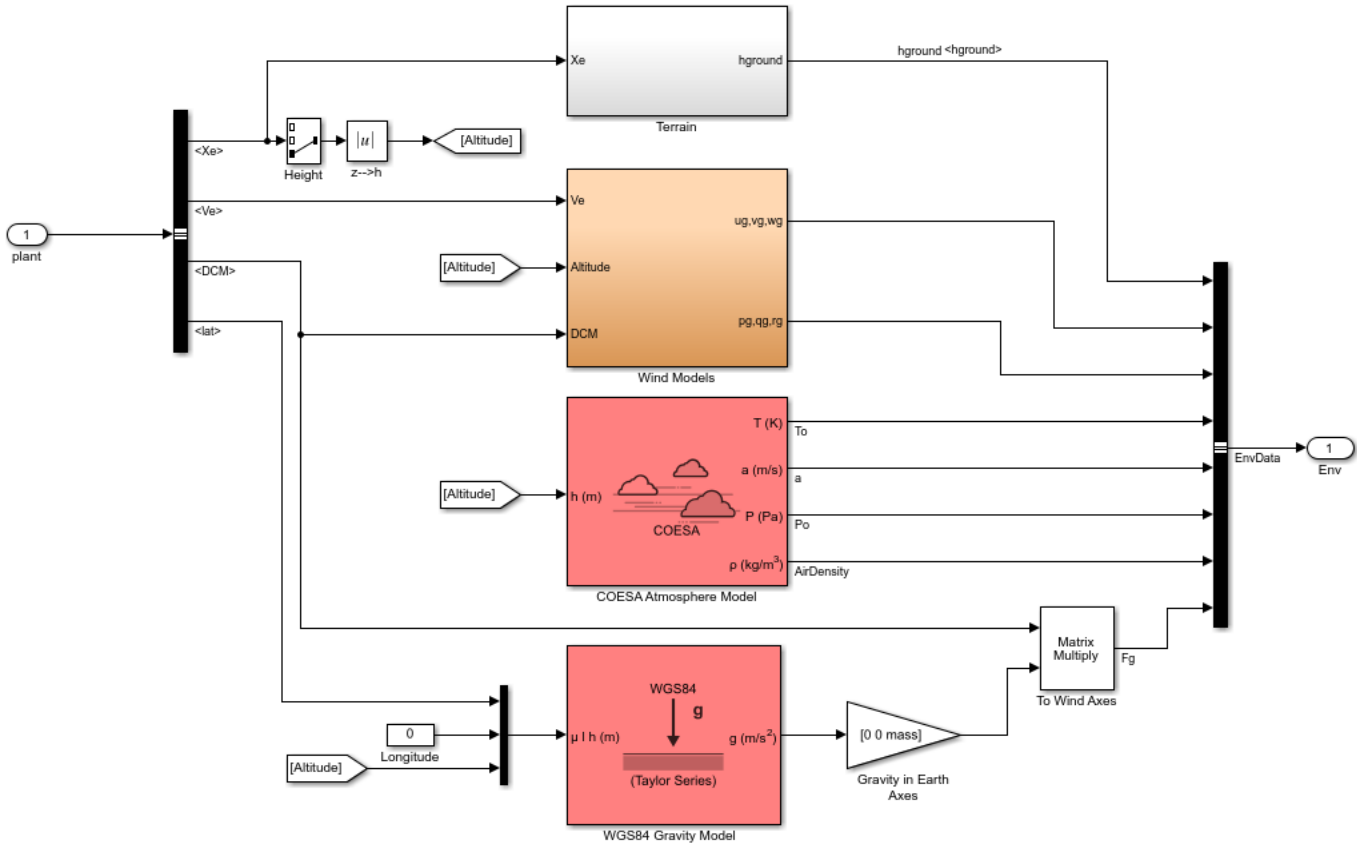
We also use Aerospace Blockset blocks to create actuator, sensor, and environment models (Figures 6, 7, and 8, respectively). **Note:** In addition to creating the following parts of the model, we use standard Aerospace Blockset blocks to ensure that we convert from body axes to wind axes and back correctly.



**Figure 6:** Implementation of actuator models using Aerospace Blockset blocks.



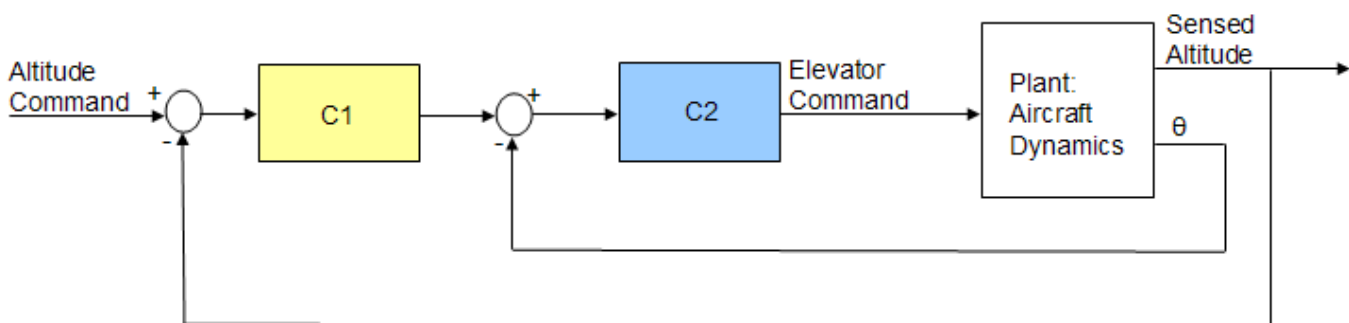
**Figure 7:** Implementation of flight sensor model using Aerospace Blockset blocks.



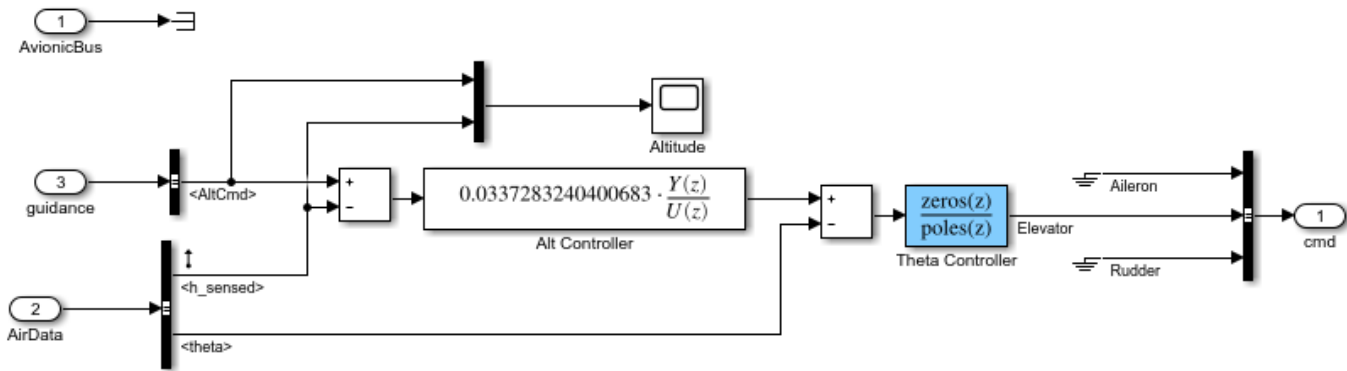
**Figure 8:** Environmental effect of wind, atmosphere, and gravity using Aerospace Blockset blocks.

### Designing Flight Control Laws

Once we have created the Simulink plant model, we design a longitudinal controller that commands elevator position to control altitude. The traditional two-loop feedback control structure chosen for this design (Figure 9) has an outer loop for controlling altitude (compensator C1 in yellow) and an inner loop for controlling pitch angle (compensator C2 in blue). Figure 10 shows the corresponding controller configuration in our Simulink model.



**Figure 9:** Structure of the longitudinal controller.



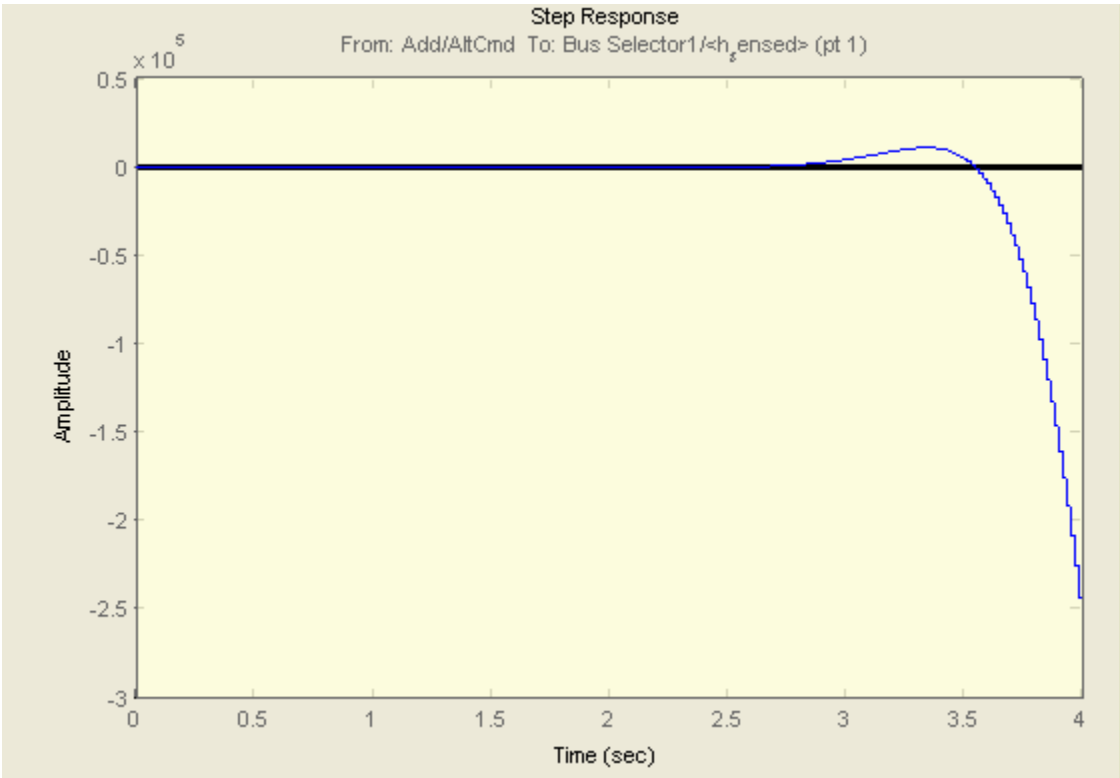
**Figure 10:** Longitudinal controller in Simulink model.

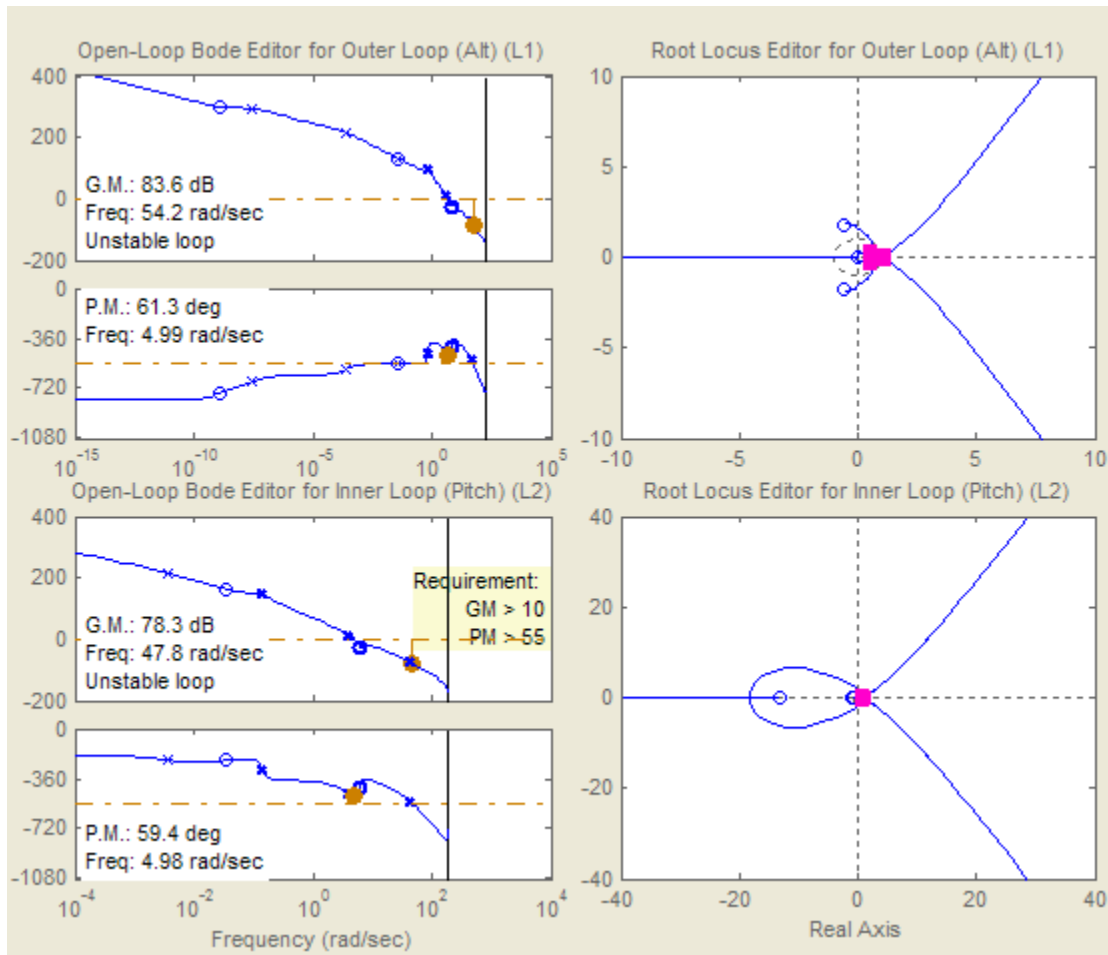
With Simulink® Control Design™ software, we can tune the controllers directly in Simulink using a range of tools and techniques.

Using the Simulink Control Design interface, we set up the control problem by specifying:

- Two controller blocks
- Closed-loop input or altitude command
- Closed-loop output signals or sensed altitude
- Steady-state or trim condition.

Using this information, Simulink Control Design software automatically computes linear approximations of the model and identifies feedback loops to be used in the design. To design the controllers for the inner and outer loops, we use root locus and bode plots for the open loops and a step response plot for the closed-loop response (Figure 11).

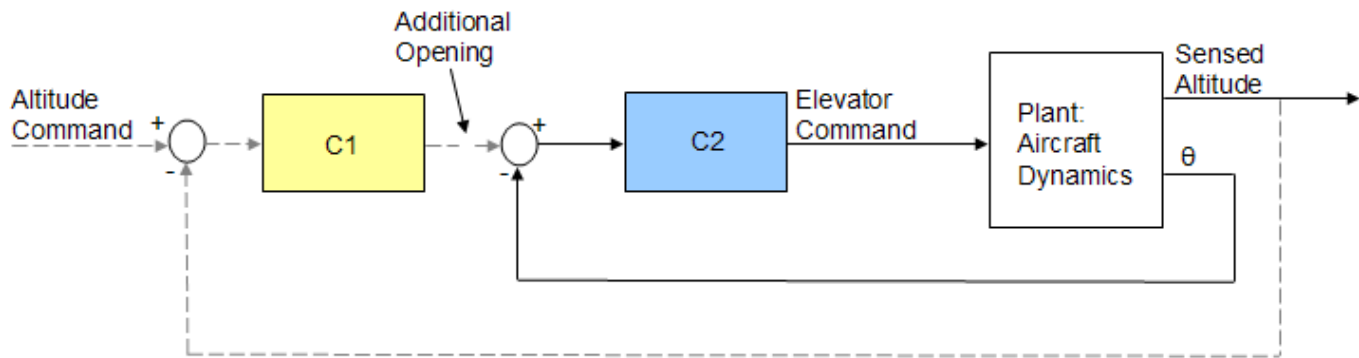




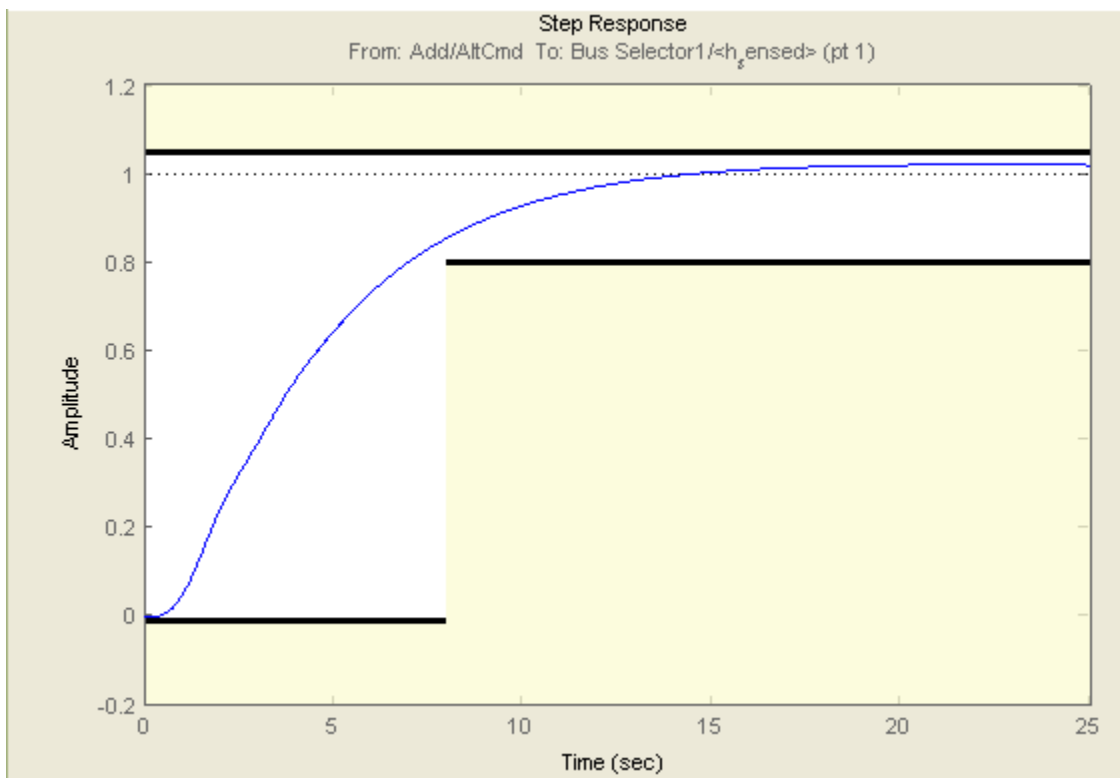
**Figure 11:** Design plots before controller tuning.

We then interactively tune the compensators for the inner and outer loops using these plots. Because the plots update in real time as we tune the compensators, we can see the coupling effects that these changes have on other loops and on the closed-loop response.

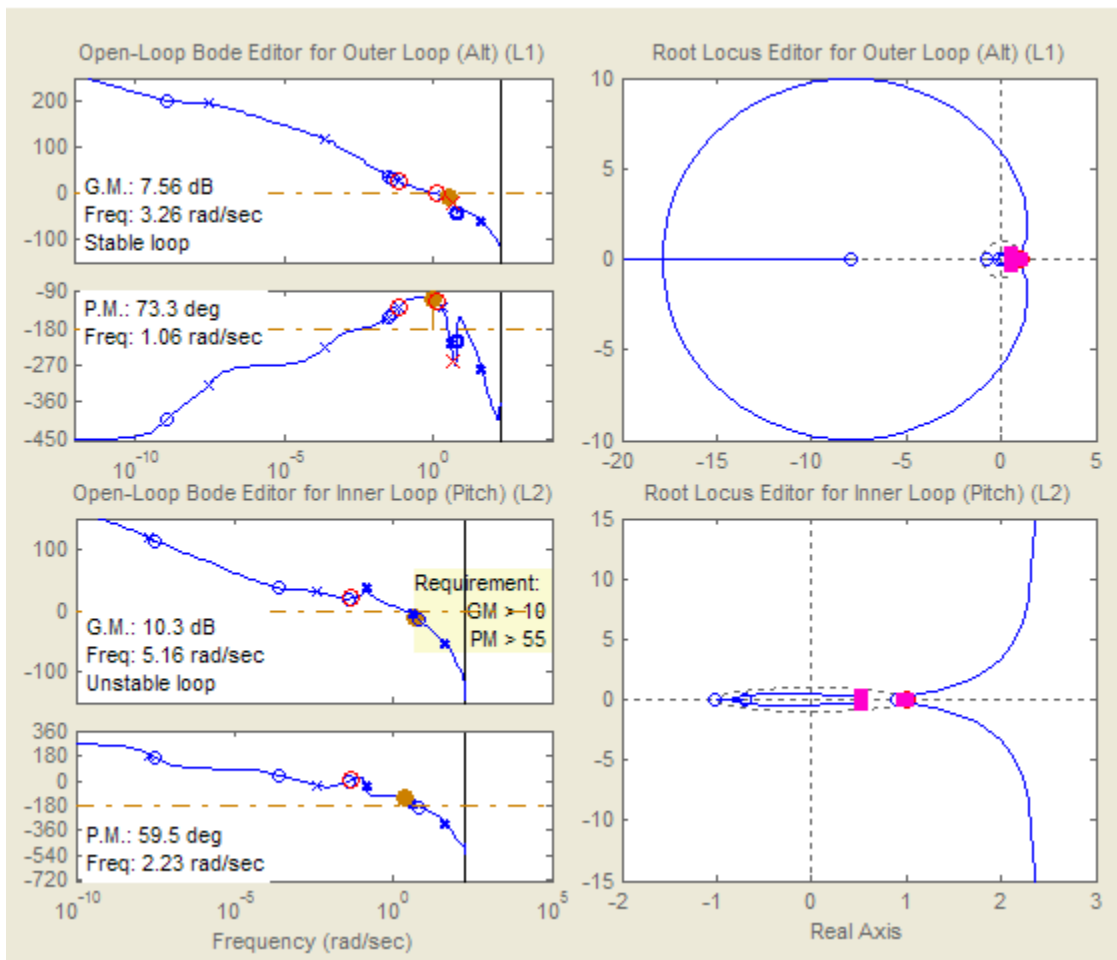
To make the multi-loop design more systematic, we use a sequential loop closure technique. This technique lets us incrementally take into account the dynamics of the other loops during the design process. With Simulink Control Design, we configure the inner loop to have an additional loop opening at the output of the outer loop controller (C1 in Figure 12). This approach decouples the inner loop from the outer loop and simplifies the inner-loop controller design. After designing the inner loop, we design the outer loop controller. Figure 13 shows the resulting tuned compensator design at the final trimmed operating point.



**Figure 12:** Block diagram of inner loop, isolated by configuring an additional loop opening.







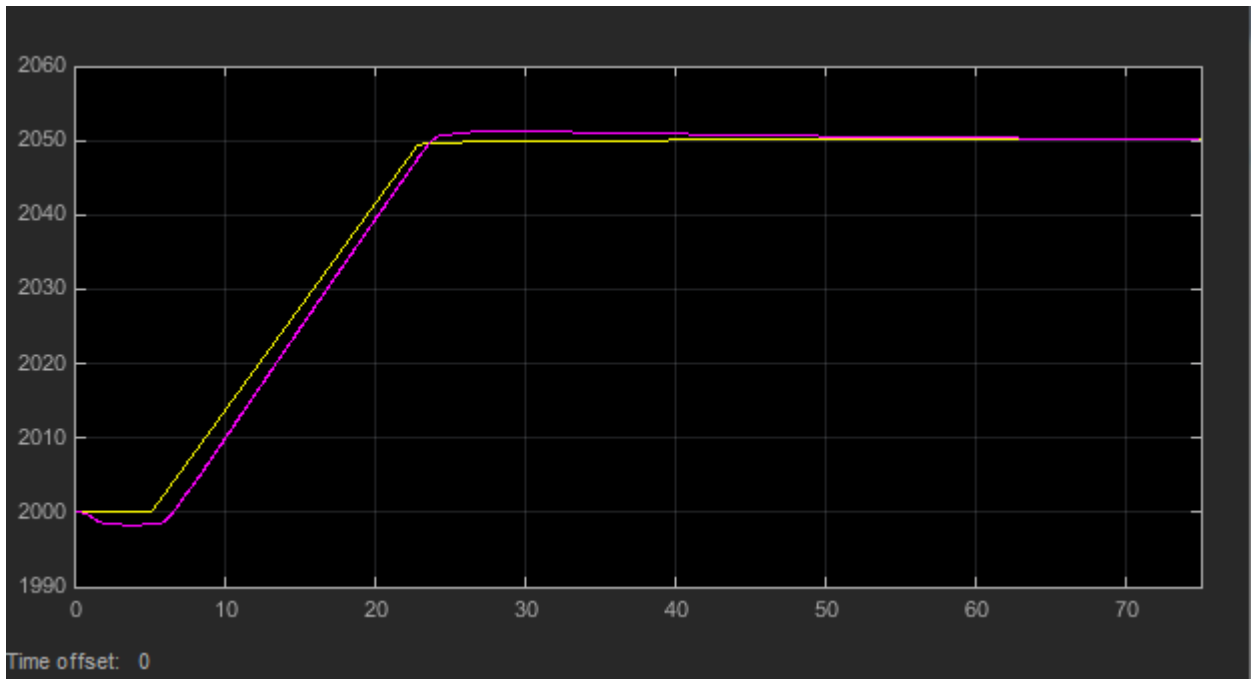
**Figure 13:** Design plots at trim condition after controller tuning.

You can tune the controller in Simulink Control Design software in several ways. For example:

- You can use a graphical approach, and interactively move controller gain, poles, and zeros until you get a satisfactory response (Figure 13).
- You can use Simulink® Design Optimization™ software within Simulink Control Design software to tune the controller automatically.

After you specify frequency domain requirements, such as gain margin and phase margin and time domain requirements, Simulink Design Optimization software automatically tunes controller parameters to satisfy those requirements. Once we have developed an acceptable controller design, the control blocks in the Simulink model are automatically updated. See the examples “Getting Started with the Control System Designer” (Control System Toolbox) in Control Systems Toolbox examples and in Simulink Control Design examples for more information on tuning controllers.

We can now run our nonlinear simulation with flight control logic and check that the controller performance is acceptable. Figure 15 shows the results from a closed-loop simulation of our nonlinear Simulink model for a requested altitude increase from 2,000 meters to 2,050 meters starting from a trimmed operating point. Although a pilot requests a step change in altitude, the actual controller altitude request rate is limited to provide a comfortable and safe ride for the passengers.



**Figure 14:** The final check is to run nonlinear simulation with our controller design and check that altitude (purple) tracks altitude request (yellow) in the stable and acceptable fashion.

We can now use these simulation results to determine whether our aircraft design meets its performance requirements. The requirement called for the climb rate to be above 2 m/s. As we can see, the aircraft climbed from 2,000 to 2,050 meters in less than 20 seconds, providing a climb rate higher than 2.5 m/s. Therefore, this particular geometric configuration and controller design meets our performance requirements.

In addition to traditional time plots, we can visualize simulation results using the Aerospace Blockset interface to FlightGear (Figure 15).



**Figure 15:** Visualizing simulation results using the Aerospace Blockset interface to FlightGear.

We can also use the Aerospace Toolbox interface to FlightGear to play back MATLAB data using either simulation results or actual flight test data.

### Completing the Design Process

The next steps involve

- Building a hardware-in-the-loop system to test real-time performance
- Building the actual vehicle hardware and software
- Conducting the flight test
- Analyzing and visualizing the flight test data.

Because these steps are not the focus of this example, we will not describe them here. Instead, we will simply mention that they can all be streamlined and simplified using the appropriate tools, such as Embedded Coder®, Simulink® Real-Time™, and Aerospace Toolbox software.

### Summary

In this example we showed how to:

- Use Digital Datcom and Aerospace Toolbox software to rapidly develop the initial design of your flight vehicle and evaluate different geometric configurations.
- Use Simulink and Aerospace Blockset software to rapidly create a flight simulation of your vehicle.
- Use Simulink Control Design software to design flight control laws.

This approach enables you to determine the optimal geometrical configuration of your vehicle and estimate its performance and handling qualities well before any hardware is built, reducing design costs and eliminating errors. In addition, using a single tool chain helps facilitate communication among different groups and accelerates design time.

## **References**

[1] Cannon, M, Gabbard, M, Meyer, T, Morrison, S, Skocik, M, Woods, D. "Swineworks D-200 Sky Hogg Design Proposal." AIAA®/General Dynamics Corporation Team Aircraft Design Competition, 1991-1992.

[2] Turvesky, A., Gage, S., and Buhr, C., "Accelerating Flight Vehicle Design", MATLAB® Digest, January 2007.

[3] Turvesky, A., Gage, S., and Buhr, C., "Model-based Design of a New Lightweight Aircraft", AIAA paper 2007-6371, AIAA Modeling and Simulation Technologies Conference and Exhibit, Hilton Head, South Carolina, Aug. 20-23, 2007.

## Multiple Aircraft with Collaborative Control

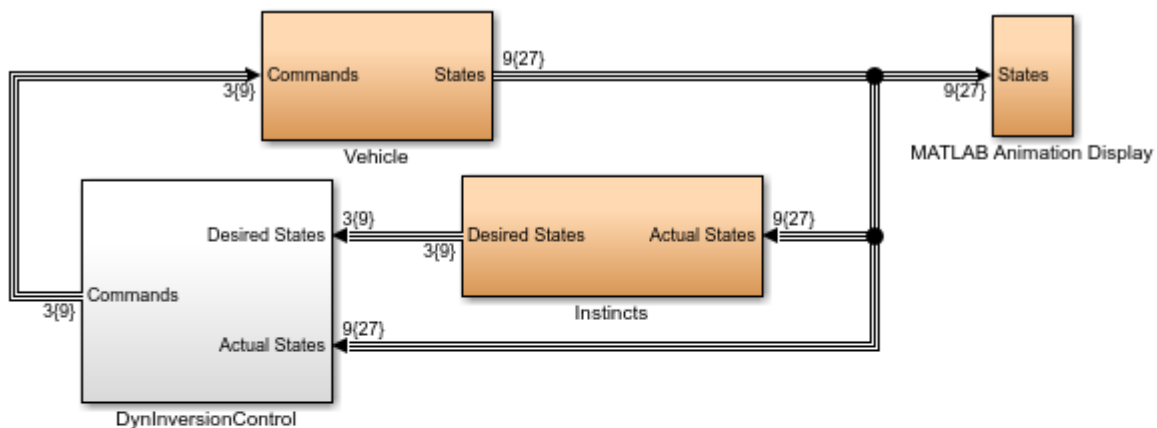
This model shows the simulation of multiple aircraft in formation flight, with emphasis on the necessary requirements and the realized benefits in making the simulation vectorized so that it can easily be updated for an arbitrary number of vehicles. To perform their avoidance task, this set of aircraft uses cooperative control.

This model uses color coding to aid in locating Aerospace Blockset™ blocks. The red blocks are Aerospace Blockset blocks, the orange blocks are subsystems that contain additional Aerospace Blockset blocks, and the white blocks are Simulink® blocks.

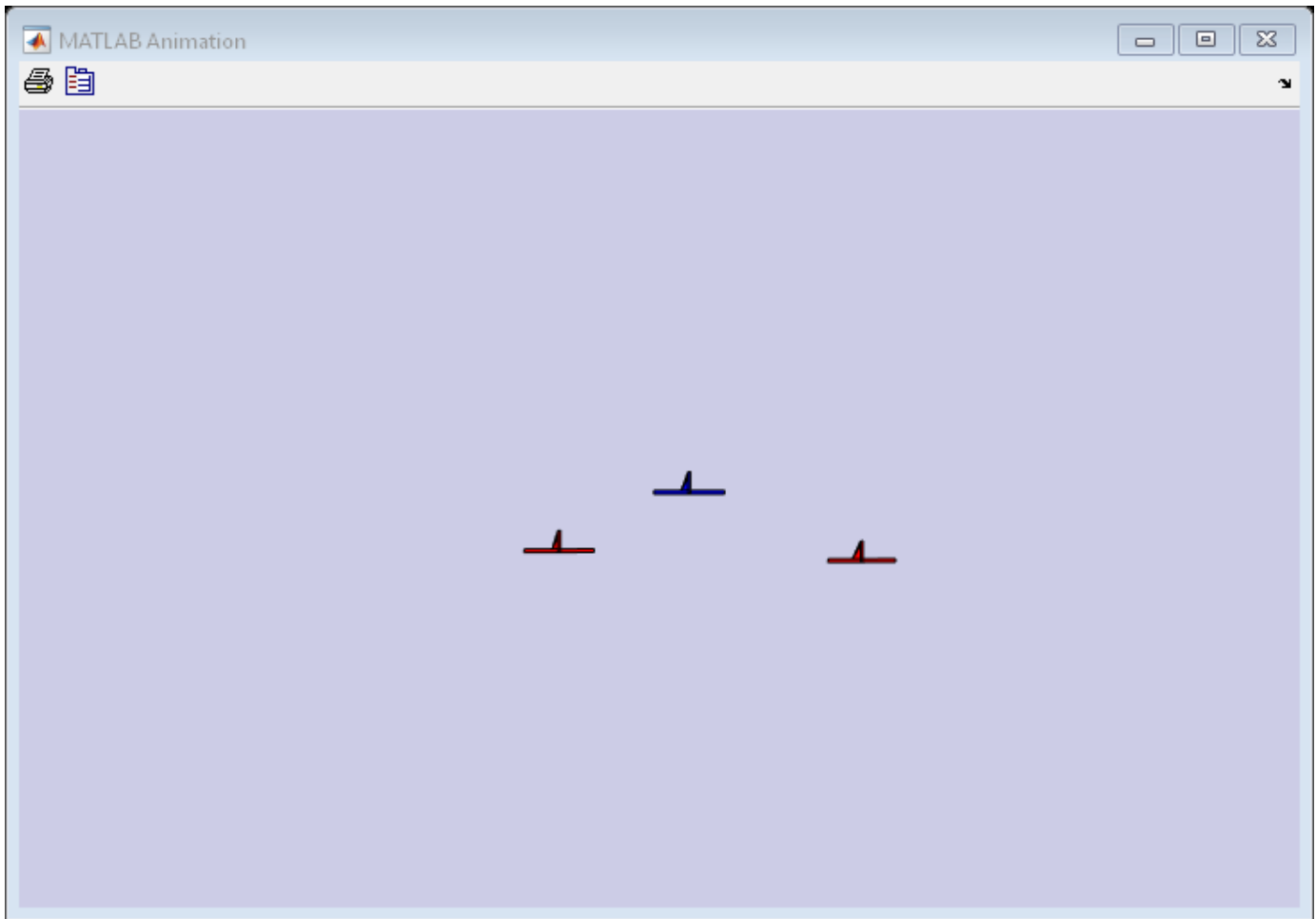
The simulation uses Simulink and Aerospace Blockset software, which allow for a hierarchal block diagram representation to include the control laws, vehicle models and visualization.

The MATLAB® Animation Display subsystem contains the MATLAB® Animation block from Aerospace Blockset to visualize the simulation. There are three types of bodies. The blue body is the first body in the formation. It is the target of the camera. The second and third bodies in the formation are red. The two black bodies represent the obstacles.

The basis of this simulation comes from previous research performed in the study of aircraft formation flight in the context of cooperative game theory and the natural aggregate motion of flocking birds, schooling fish, and the herding of land animals.



This multiple aircraft simulation was based on:  
Anderson M., Robbins D., "Formation Flight as a Cooperative Game", AIAA-98-4124, AIAA GNC, 1998.



## HL-20 with Flight Instrumentation Blocks

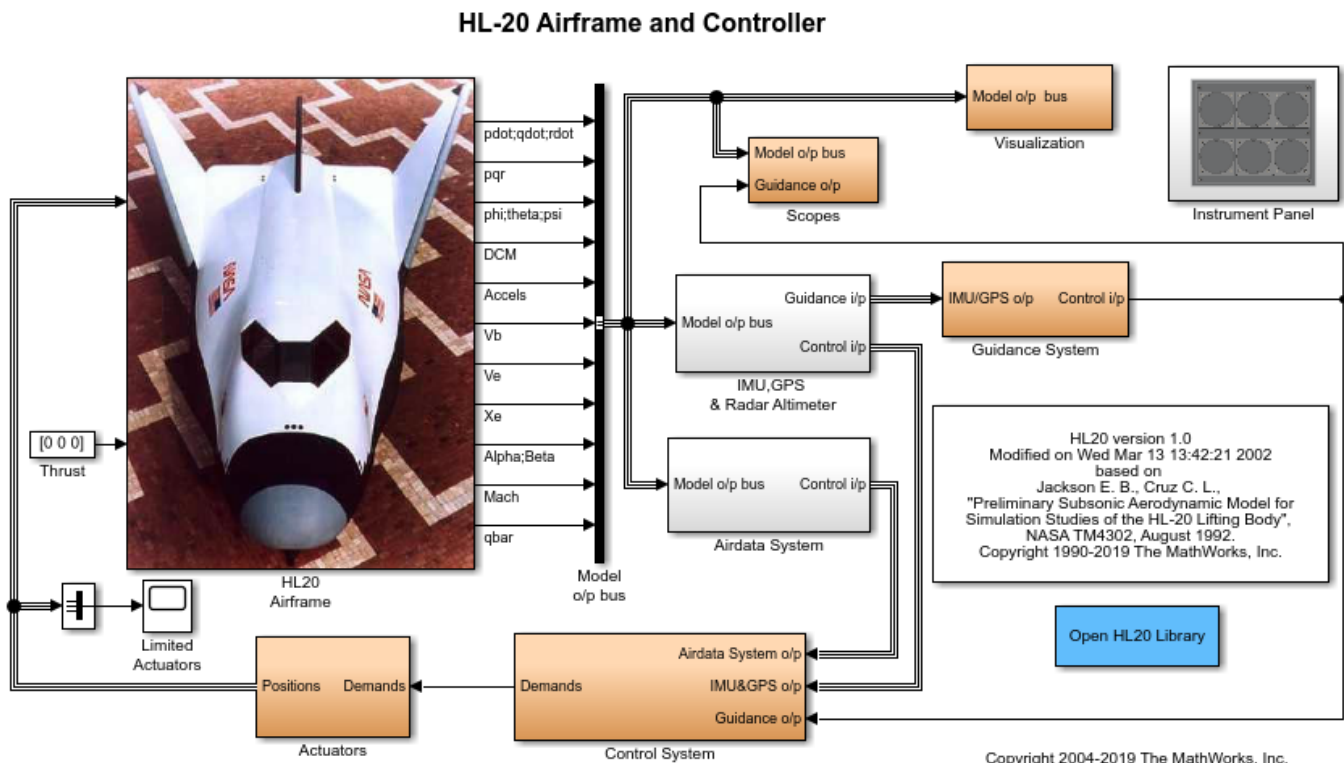
This model shows NASA's HL-20 lifting body and controller modeled in Simulink® and Aerospace Blockset™ software. This model simulates approach and landing flight phases using an auto-landing controller. The Visualization subsystem uses aircraft-specific gauges from the Aerospace Blockset™ Flight Instrumentation library.

The HL-20 also known as personnel launch system (PLS) is a lifting body re-entry vehicle that was designed to complement the Space Shuttle orbiter. Designed to carry up to ten people and very little cargo[1], the HL-20 lifting body was to be placed in orbit either launched vertically via booster rockets or transported in the payload bay of the Space Shuttle orbiter. HL-20 lifting body was designed to have a powered deorbiting accomplished with an onboard propulsion system while its reentry was to be nose-first, horizontal and unpowered.

The HL-20 lifting body was developed as a low cost solution for getting to and from low Earth orbit. The proposed benefits of the HL-20 were reduced operating costs due to rapid turnaround between landing and launch, improved flight safety, and ability to land conventionally on runways. Potential scenarios for the HL-20 were orbital rescue of stranded astronauts, International Space Station crew exchange if the Space Shuttle orbiter was not available, observation missions, and satellite servicing missions.

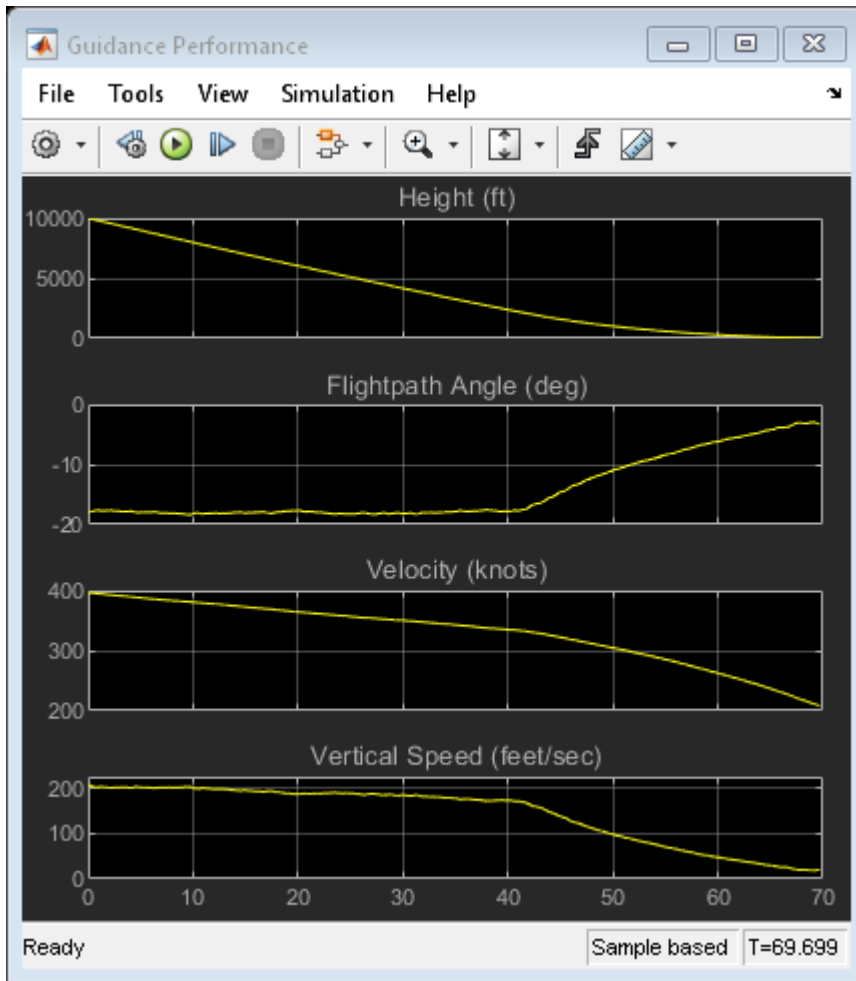
Additional information about HL-20

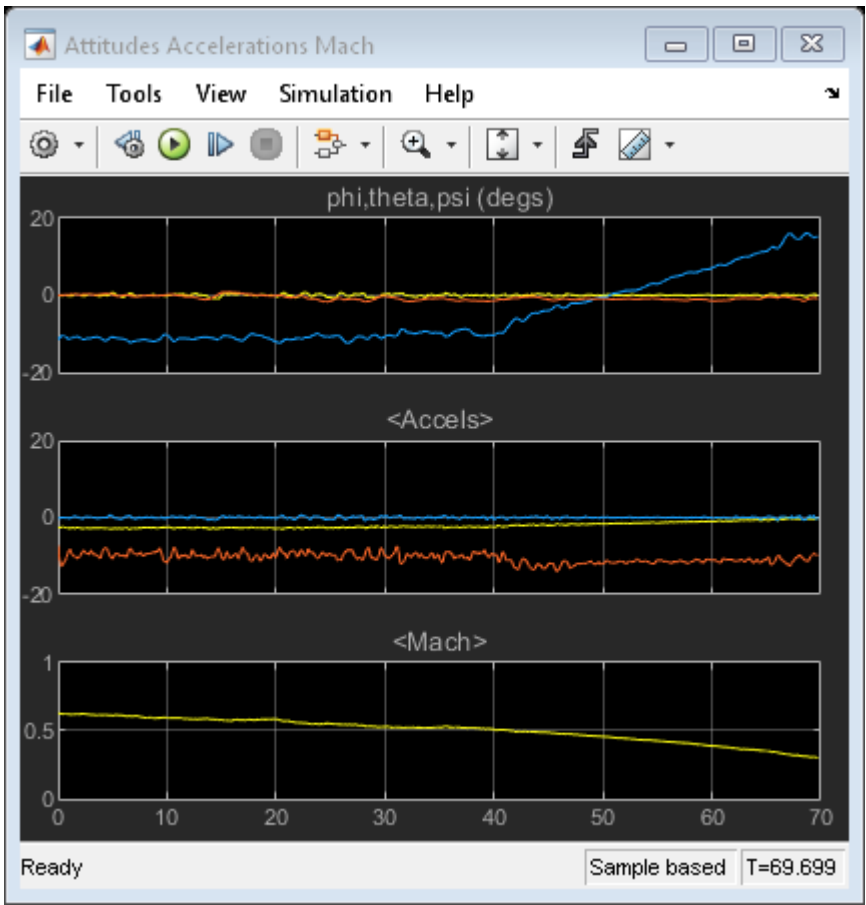
[1] Jackson E. B., Cruz C. L., "Preliminary Subsonic Aerodynamic Model for Simulation Studies of the HL-20 Lifting Body," NASA TM4302 (August 1992)

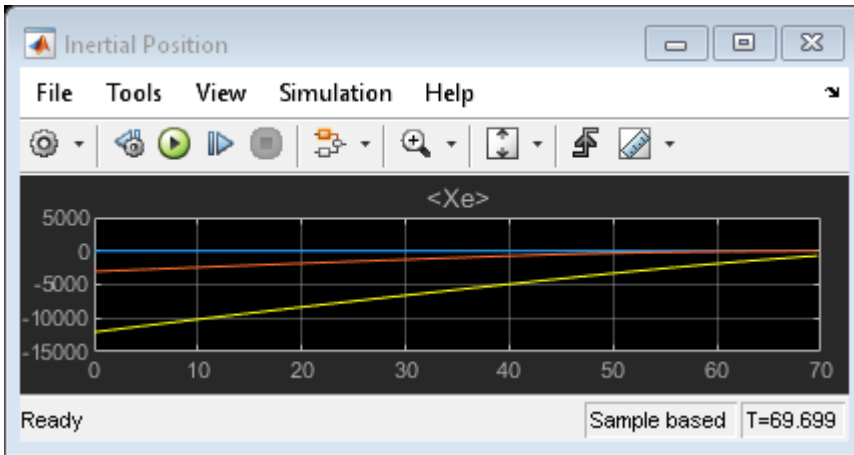
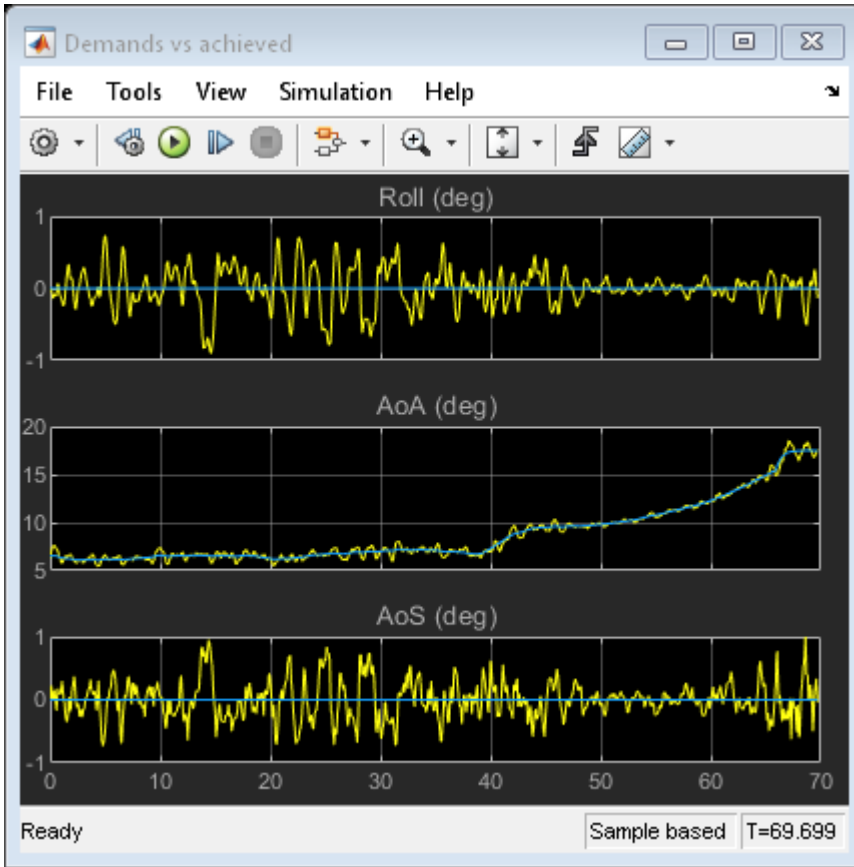












## HL-20 with Simulink® 3D Animation™ and Flight Instrumentation Blocks

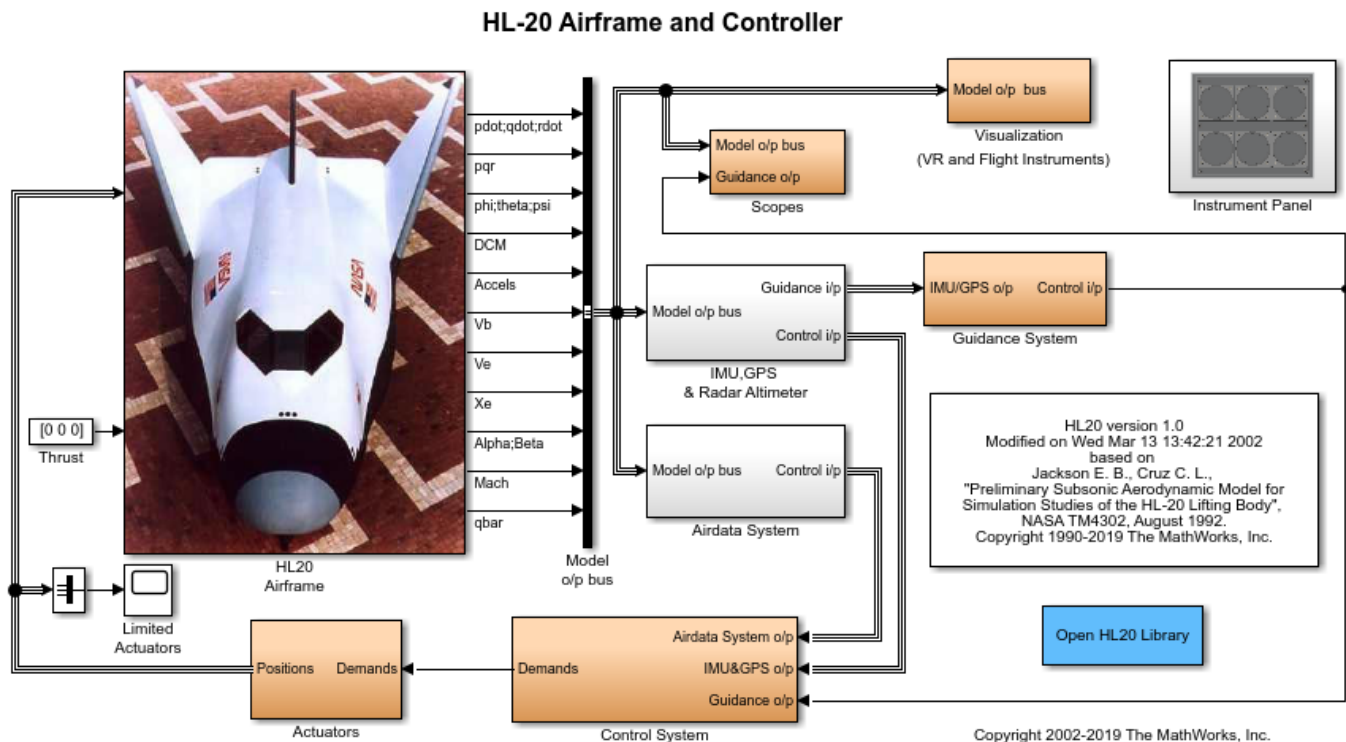
This model shows NASA's HL-20 lifting body and controller modeled in Simulink®, Aerospace Blockset™, and Simulink® 3D Animation™ software. This model simulates approach and landing flight phases using an auto-landing controller. The Visualization subsystem uses aircraft-specific gauges from the Aerospace Blockset™ Flight Instrumentation library.

The HL-20 also known as personnel launch system (PLS) is a lifting body re-entry vehicle that was designed to complement the Space Shuttle orbiter. Designed to carry up to ten people and very little cargo[1], the HL-20 lifting body was to be placed in orbit either launched vertically via booster rockets or transported in the payload bay of the Space Shuttle orbiter. HL-20 lifting body was designed to have a powered deorbiting accomplished with an onboard propulsion system while its reentry was to be nose-first, horizontal and unpowered.

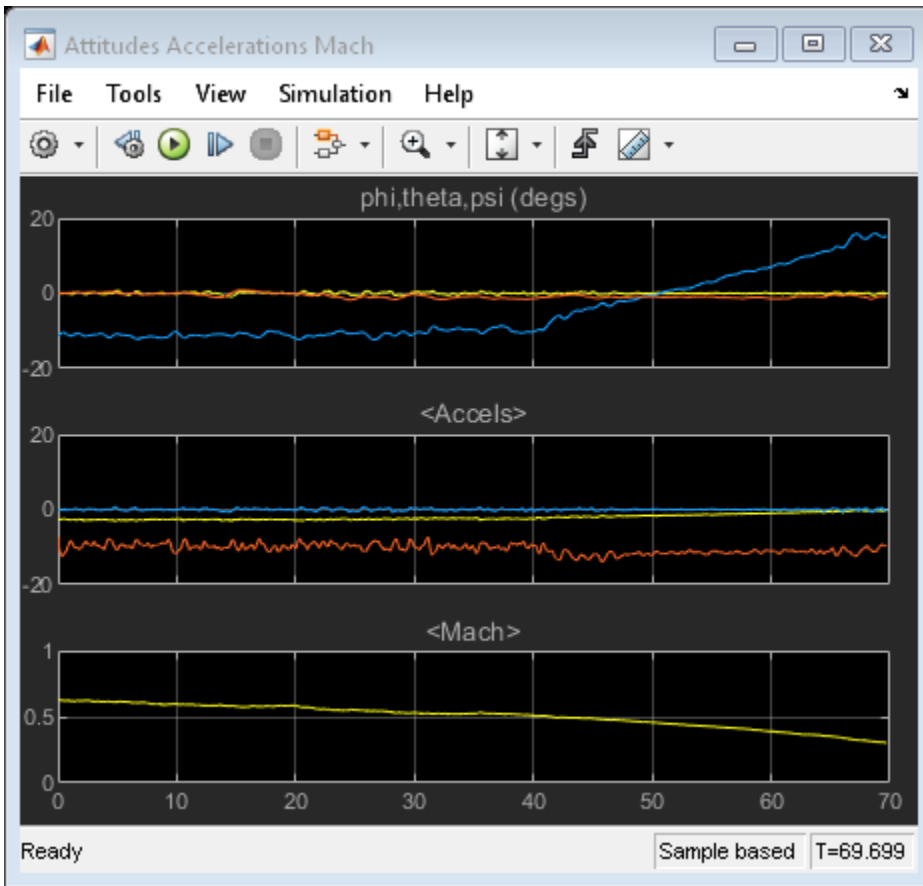
The HL-20 lifting body was developed as a low cost solution for getting to and from low Earth orbit. The proposed benefits of the HL-20 were reduced operating costs due to rapid turnaround between landing and launch, improved flight safety, and ability to land conventionally on runways. Potential scenarios for the HL-20 were orbital rescue of stranded astronauts, International Space Station crew exchange if the Space Shuttle orbiter was not available, observation missions, and satellite servicing missions.

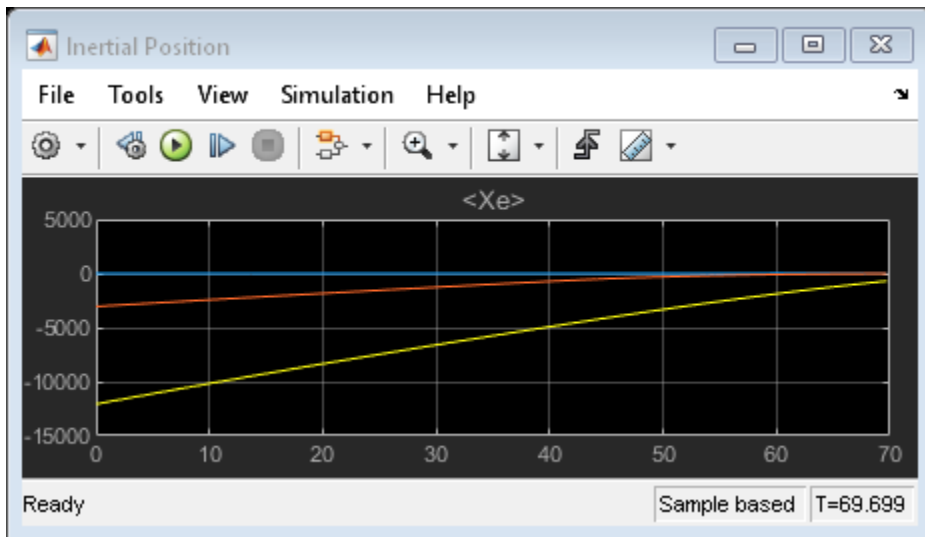
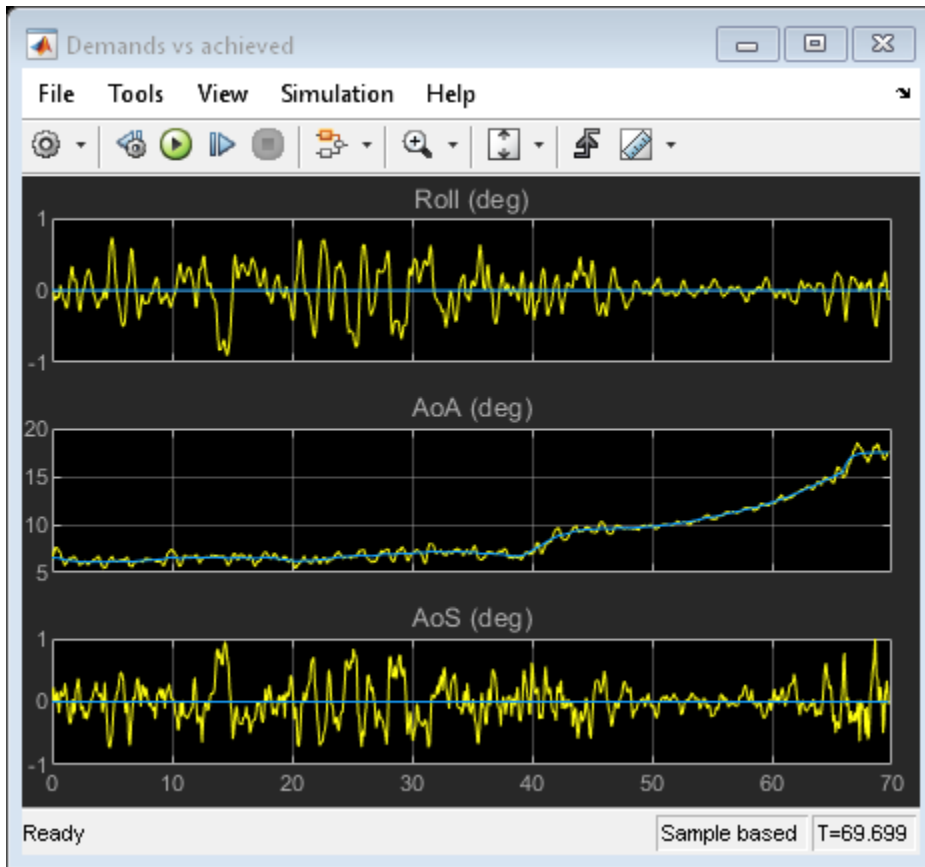
Additional information about HL-20

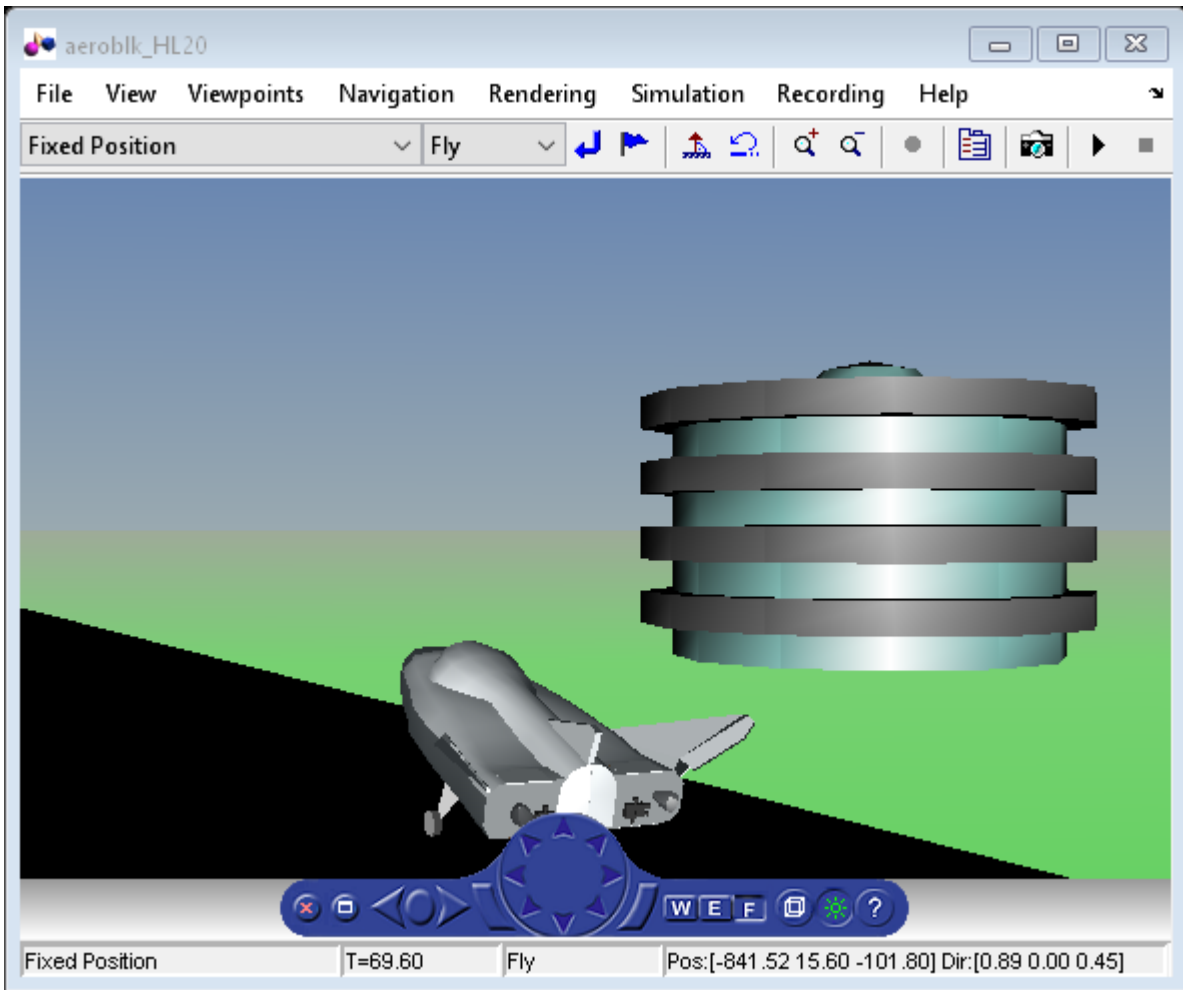
[1] Jackson E. B., Cruz C. L., "Preliminary Subsonic Aerodynamic Model for Simulation Studies of the HL-20 Lifting Body," NASA TM4302 (August 1992)













## HL-20 Project with Optional FlightGear Interface

This project shows how to model NASA's HL-20 lifting body with Simulink®, Stateflow® and Aerospace Blockset™ software. The vehicle model includes the aerodynamics, control logic, fault management systems (FDIR), and engine controls (FADEC). It also includes effects of the environment, such as wind profiles for the landing phase. The entire model simulates approach and landing flight phases using an auto-landing controller. To analyze the effects of actuator failures and wind gust variation on the stability of the vehicle, use the "Run Failure Analysis in Parallel" project shortcut. If Parallel Computing Toolbox™ is installed, the analysis is run in parallel. If Parallel Computing Toolbox™ is not installed, the analysis is run in serial. Visualization for this model is done via an interface to FlightGear, an open source flight simulator package. If the FlightGear interface is unavailable, you can simulate the model by closing the loop using the alternative data sources provided in the Variant block. In this block, you can choose a previously saved data file, a Signal Editor block, or a set of constant values. This example requires Control System Toolbox™.

### FlightGear Interface

For more information on the FlightGear interface, read these documentation topics:

- "Flight Simulator Interface" on page 2-16
- "Work with the Flight Simulator Interface" on page 2-20
- "Run the HL-20 Example with FlightGear" on page 2-29

For a more detailed description of this model components, view a recorded navigation through the model using this link:

- [Spacecraft Automated Landing System](#)

### NASA HL-20 Background

The HL-20, also known as personnel launch system (PLS), is a lifting body re-entry vehicle that was designed to complement the Space Shuttle orbiter. Designed to carry up to ten people and very little cargo[1], the HL-20 lifting body was to be placed in orbit either launched vertically via booster rockets or transported in the payload bay of the Space Shuttle orbiter. HL-20 lifting body was designed to have a powered deorbiting accomplished with an onboard propulsion system while its reentry was to be nose-first, horizontal and unpowered.

The HL-20 lifting body was developed as a low cost solution for getting to and from low Earth orbit. The proposed benefits of the HL-20 were reduced operating costs due to rapid turnaround between landing and launch, improved flight safety, and ability to land conventionally on runways. Potential scenarios for the HL-20 were orbital rescue of stranded astronauts, International Space Station crew exchange if the Space Shuttle orbiter was not available, observation missions, and satellite servicing missions.

### Opening HL-20 Project

Run the following command to create and open a working copy of the project files for this example.

```
asbh\20
```



For more information on using Simulink Projects and HL-20, see:

- Tamayo S., Gage S., Walker G., "Integrated Project Management Tool for Modeling Simulation of Complex Systems", AIAA Modeling and Simulation Technologies Conference (August 2012)

**Additional Information About NASA HL-20**

[1] Jackson E. B., Cruz C. L., "Preliminary Subsonic Aerodynamic Model for Simulation Studies of the HL-20 Lifting Body," NASA TM4302 (August 1992)

## Quaternion Estimate from Measured Rates

This model shows how to estimate a quaternion and model the equations in the following ways:

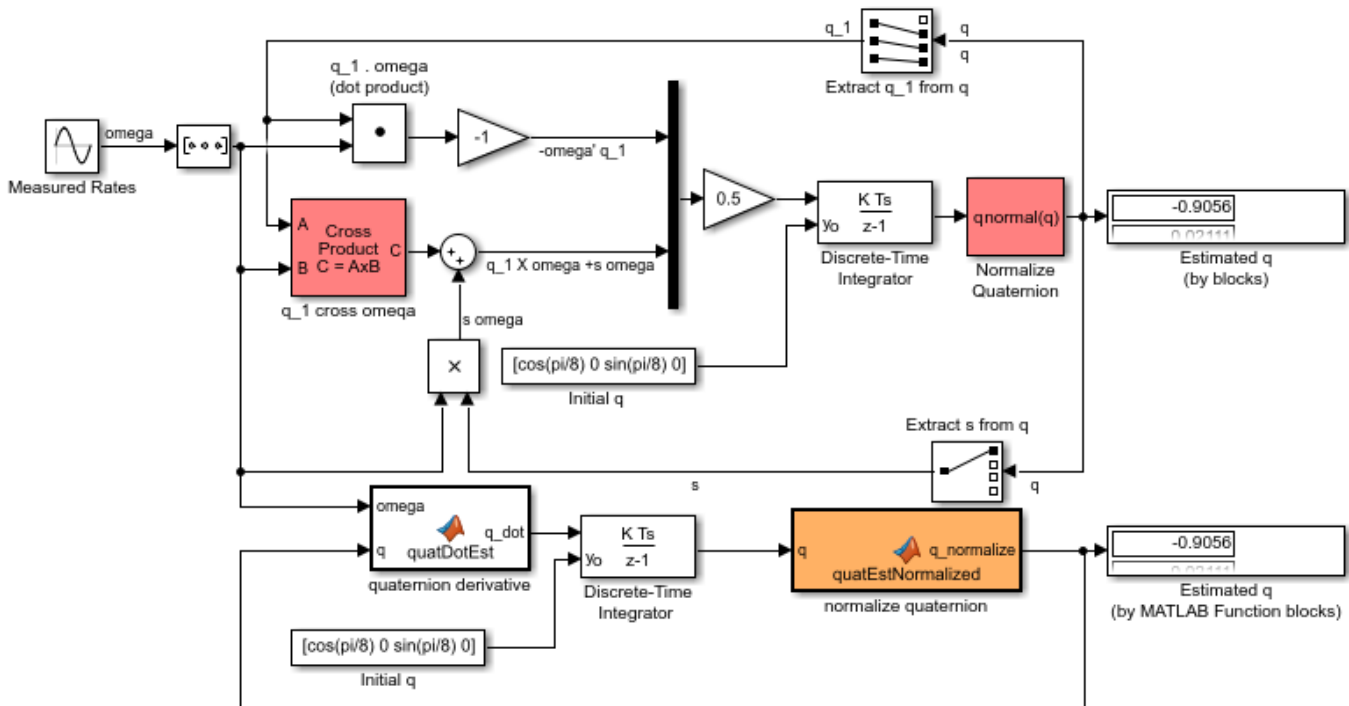
\*Using Simulink® and Aerospace Blockset™ software to implement the equations.

\*Using MATLAB® Function block to incorporate an Aerospace Toolbox quaternion function.

This model has been color coded to aid in locating Aerospace Blockset blocks.

The red blocks are Aerospace Blockset blocks, the orange block is a MATLAB Function block containing a function with MATLAB function block support provided by Aerospace Blockset and the white blocks are Simulink blocks.

Quaternion Estimate from Measured Rates



Copyright 2007-2019 The MathWorks, Inc.

## Indicated Airspeed from True Airspeed Calculation

This model shows how to compute the indicated airspeed from true airspeed using the Ideal Airspeed Correction block. The Aerospace Blockset™ blocks are indicated in red.

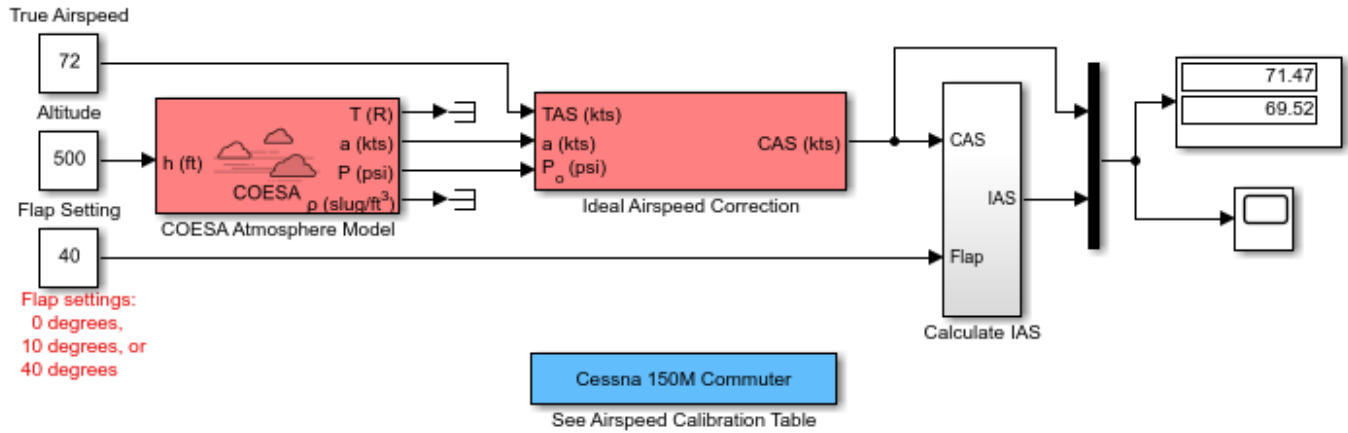
True airspeed is the airspeed that we would read ideally (and the airspeed value easily calculated within a simulation). However there are errors introduced through the pitot-static airspeed indicators used to determine airspeed. These measurement errors are density error, compressibility error and calibration error. Applying these errors to true airspeed will result in indicated airspeed. (the ideal airspeed correction block can handle the density error and compressibility error)

Density Error -- It is a fact that an airspeed indicator reads lower than true airspeed at higher altitudes. This is due to lower air density at altitude. When the difference or error in air density at altitude from air density on a standard day at sea level is applied to true airspeed, it results in equivalent airspeed (EAS). Equivalent airspeed is true airspeed modified with the changes in atmospheric density which affect the airspeed indicator.

Compressibility Error -- Air has a limited ability to resist compression. This ability is reduced by an increase in altitude, an increase in speed, or a restricted volume. Within the airspeed indicator, there is a certain amount of trapped air. When flying at high altitudes and higher airspeeds, calibrated airspeed (CAS) is always higher than equivalent airspeed. Calibrated airspeed is equivalent airspeed modified with compressibility effects of air which affect the airspeed indicator.

Calibration Error -- The airspeed indicator has static vent(s) to maintain a pressure equal to atmospheric pressure inside the instrument. Position and placement of the static vent along with angle of attack and velocity of the aircraft will determine the pressure inside the airspeed indicator and thus the amount of calibration error of the airspeed indicator. Needless to say, calibration error is specific to a given aircraft design. A calibration table is usually given in the pilot operating handbook (POH) or in other aircraft specifications. Using this calibration table, the indicated airspeed (IAS) is determined from calibrated airspeed by modifying it with calibration error of the airspeed indicator. Indicated airspeed is displayed in the cockpit instrumentation.

### Indicated Airspeed from True Airspeed Calculation

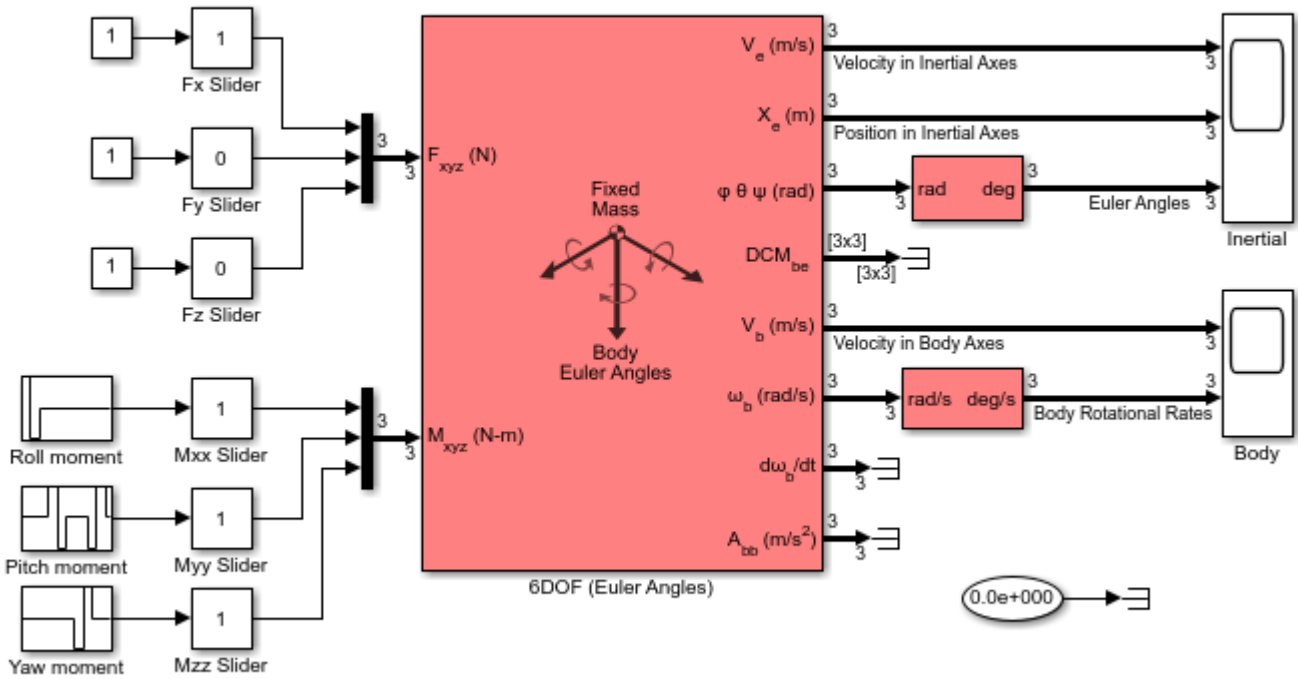


Copyright 1990-2019 The MathWorks, Inc.

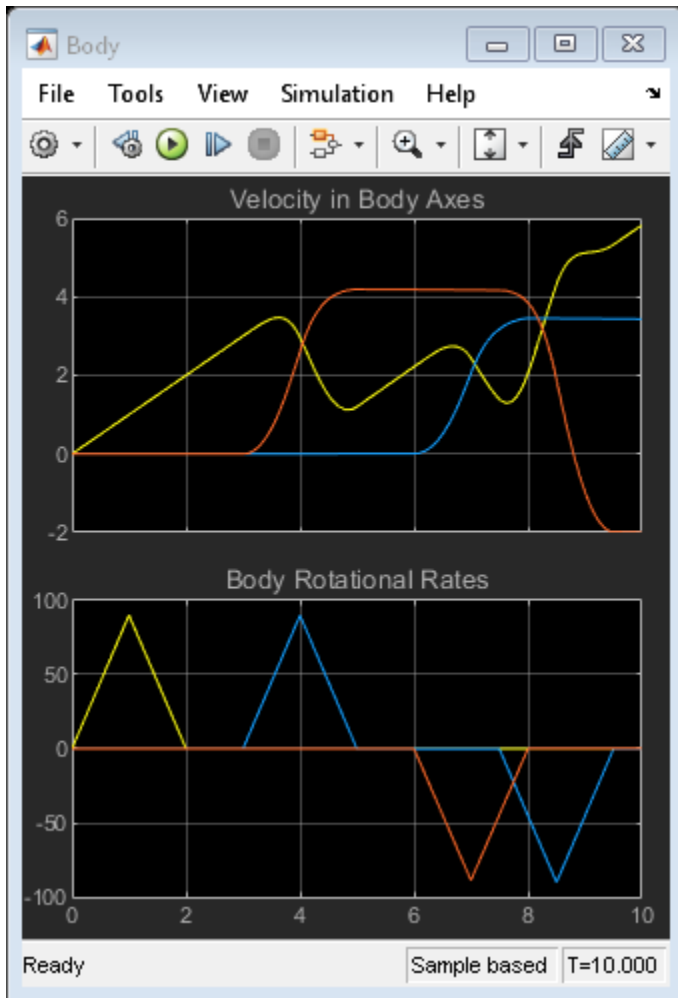
## Six Degree of Freedom Motion Platform

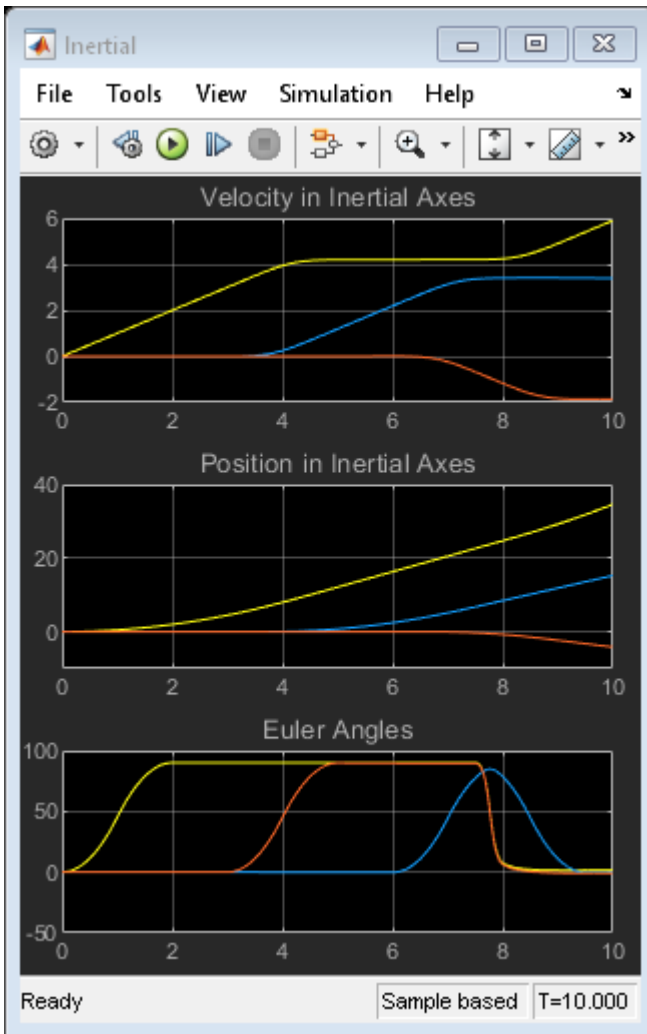
This model shows how to connect an Aerospace Blockset™ six degree of freedom equation of motion block.

### Six Degree of Freedom Motion Platform



Copyright 1990-2019 The MathWorks, Inc.

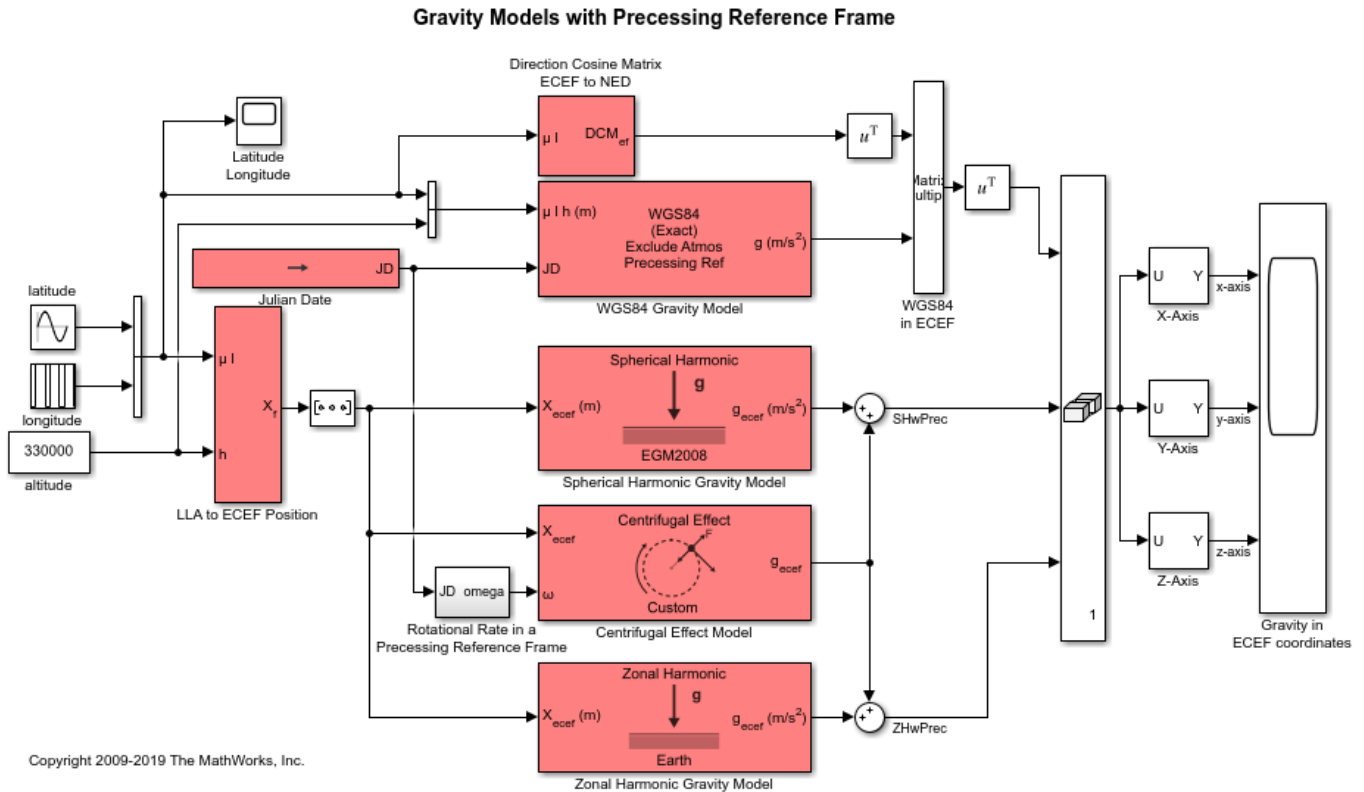


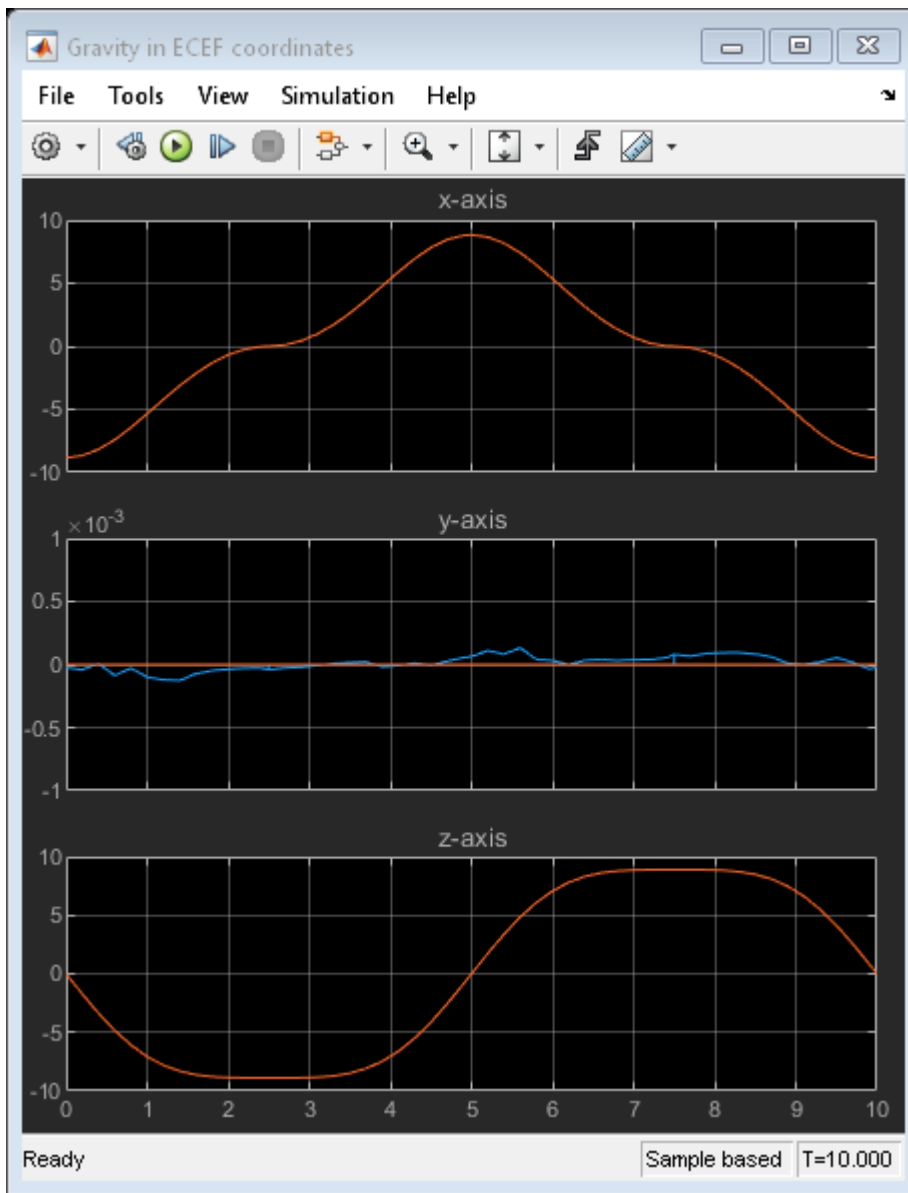


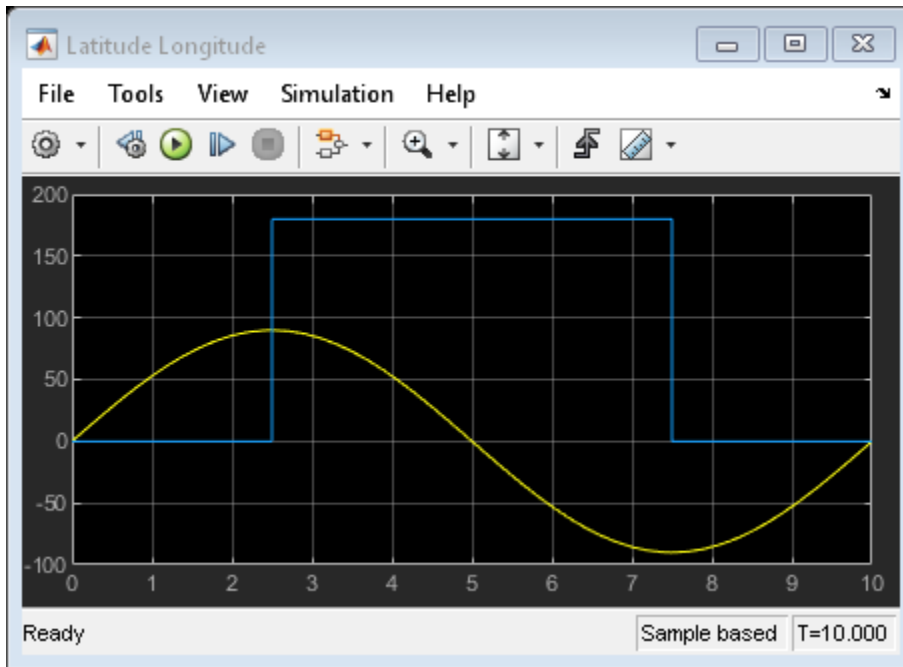


# Gravity Models with Precessing Reference Frame

This model shows how to implement various gravity models with precessing reference frames using Aerospace Blockset™ blocks. The Aerospace Blockset blocks are shown in red.







## True Airspeed from Indicated Airspeed Calculation

This model shows how to compute true airspeed from indicated airspeed using the Ideal Airspeed Correction block. The Aerospace Blockset™ blocks are indicated in red.

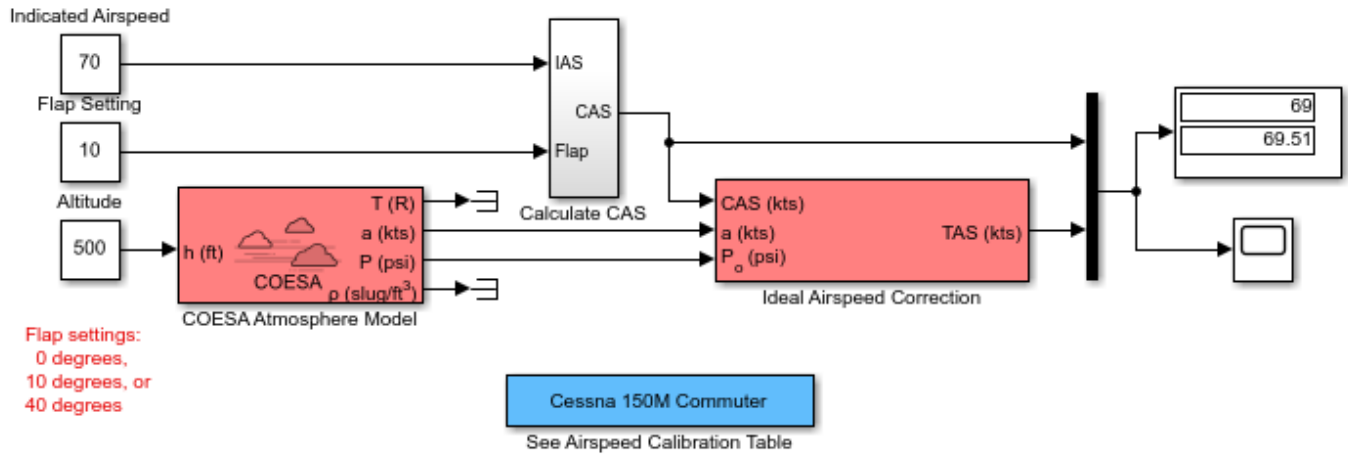
True airspeed is the airspeed that we would read ideally (and the airspeed value easily calculated within a simulation). However there are errors introduced through the pitot-static airspeed indicators used to determine airspeed. These measurement errors are density error, compressibility error and calibration error. Applying these errors to true airspeed will result in indicated airspeed. (the ideal airspeed correction block can handle the density error and compressibility error)

Density Error -- It is a fact that an airspeed indicator reads lower than true airspeed at higher altitudes. This is due to lower air density at altitude. When the difference or error in air density at altitude from air density on a standard day at sea level is applied to true airspeed, it results in equivalent airspeed (EAS). Equivalent airspeed is true airspeed modified with the changes in atmospheric density which affect the airspeed indicator.

Compressibility Error -- Air has a limited ability to resist compression. This ability is reduced by an increase in altitude, an increase in speed, or a restricted volume. Within the airspeed indicator, there is a certain amount of trapped air. When flying at high altitudes and higher airspeeds, calibrated airspeed (CAS) is always higher than equivalent airspeed. Calibrated airspeed is equivalent airspeed modified with compressibility effects of air which affect the airspeed indicator.

Calibration Error -- The airspeed indicator has static vent(s) to maintain a pressure equal to atmospheric pressure inside the instrument. Position and placement of the static vent along with angle of attack and velocity of the aircraft will determine the pressure inside the airspeed indicator and thus the amount of calibration error of the airspeed indicator. Needless to say, calibration error is specific to a given aircraft design. A calibration table is usually given in the pilot operating handbook (POH) or in other aircraft specifications. Using this calibration table, the indicated airspeed (IAS) is determined from calibrated airspeed by modifying it with calibration error of the airspeed indicator. Indicated airspeed is displayed in the cockpit instrumentation.

### True Airspeed from Indicated Airspeed Calculation



Copyright 1990-2019 The MathWorks, Inc.

## Airframe Trim and Linearize with Simulink® Control Design™

This example shows how to trim and linearize an airframe using Simulink® Control Design™ software

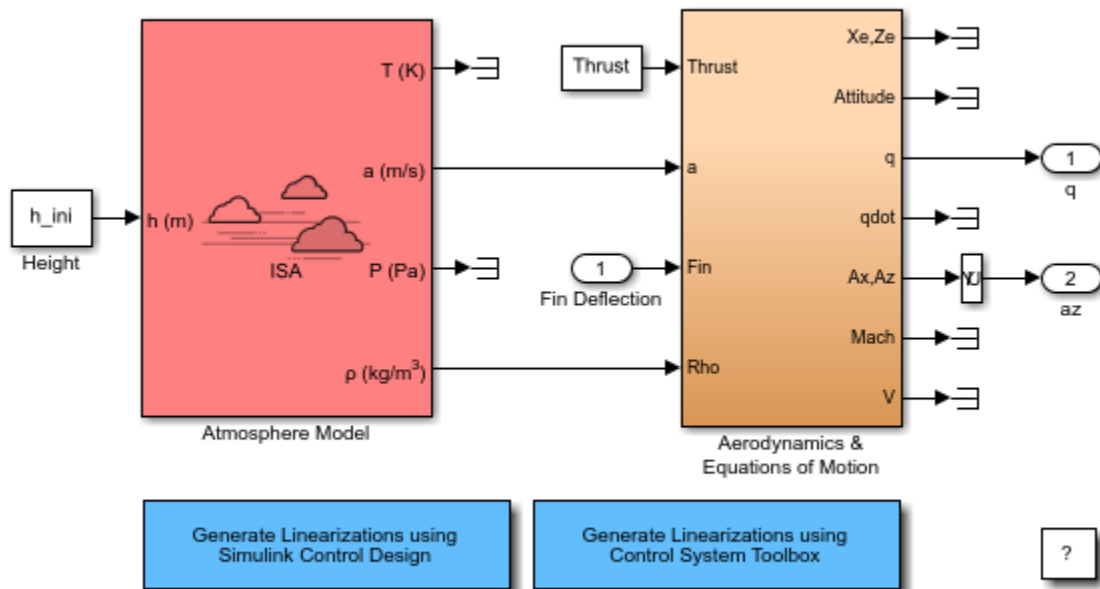
Designing an autopilot with classical design techniques requires linear models of the airframe pitch dynamics for several trimmed flight conditions. The MATLAB® technical computing environment can determine the trim conditions and derive linear state-space models directly from the nonlinear Simulink® and Aerospace Blockset™ model. This step saves time and helps to validate the model. The Simulink Control Design functions allow you to visualize the motion of the airframe in terms of open-loop frequency or time responses.

### Initialize Guidance Model

The first problem is to find the elevator deflection, and the resulting trimmed body rate ( $q$ ), which will generate a given incidence value when the missile is traveling at a set speed. Once the trim condition is found, a linear model can be derived for the dynamics of the perturbations in the states around the trim condition.

```
open_system('aeroblk_guidance_airframe');
```

### Model used in airframe trim and linearization model examples



Copyright 1990-2019 The MathWorks, Inc.

### Define State Values

```
h_ini    = 10000/m2ft;    % Trim Height [m]
M_ini    = 3;             % Trim Mach Number
alpha_ini = -10*d2r;     % Trim Incidence [rad]
theta_ini = 0*d2r;       % Trim Flightpath Angle [rad]
```

```
v_ini = M_ini*(340+(295-340)*h_ini/11000);    % Total Velocity [m/s]
q_ini = 0;                                   % Initial pitch Body Rate [rad/sec]
```

### Set Operating Point and State Specifications

The first state specifications are Position states. The second state specification is Theta. Both are known, but not at steady state. The third state specifications are body axis angular rates, of which the variable  $w$  is at steady state.

```
opspec = operspec('aeroblk_guidance_airframe');
opspec.State(1).Known = [1;1];
opspec.State(1).SteadyState = [0;0];
opspec.State(2).Known = 1;
opspec.State(2).SteadyState = 0;
opspec.State(3).Known = [1 1];
opspec.State(3).SteadyState = [0 1];
```

### Search for Operating Point, Set I/O, Then Linearize

```
op = findop('aeroblk_guidance_airframe',opspec);

io(1) = linio('aeroblk_guidance_airframe/Fin Deflection',1,'input');
io(2) = linio('aeroblk_guidance_airframe/Selector',1,'output');
io(3) = linio(sprintf(['aeroblk_guidance_airframe/Aerodynamics &\n', ...
    'Equations of Motion']),3,'output');

sys = linearize('aeroblk_guidance_airframe',op,io);
```

Operating point search report:

-----

Operating point search report for the Model aeroblk\_guidance\_airframe.  
(Time-Varying Components Evaluated at time t=0)

Operating point specifications were successfully met.  
States:

-----

```
(1.) aeroblk_guidance_airframe/Aerodynamics & Equations of Motion/3DOF (Body Axes)/Position
    x:          0      dx:          968
    x:   -3.05e+03    dx:         -171
(2.) aeroblk_guidance_airframe/Aerodynamics & Equations of Motion/3DOF (Body Axes)/Theta
    x:          0      dx:         -0.216
(3.) aeroblk_guidance_airframe/Aerodynamics & Equations of Motion/3DOF (Body Axes)/U,w
    x:          968    dx:         -14.1
    x:         -171    dx:   -7.44e-08 (0)
(4.) aeroblk_guidance_airframe/Aerodynamics & Equations of Motion/3DOF (Body Axes)/q
    x:         -0.216    dx:   3.36e-08 (0)
```

Inputs:

-----

```
(1.) aeroblk_guidance_airframe/Fin Deflection
    u:          0.136    [-Inf Inf]
```

Outputs:

-----

```
(1.) aeroblk_guidance_airframe/q
```

```

y:          -0.216    [-Inf Inf]
(2.) aeroblk_guidance_airframe/az
y:          -7.44e-08 [-Inf Inf]

```

### Select Trimmed States, Create LTI Object, and Plot Bode Response

```

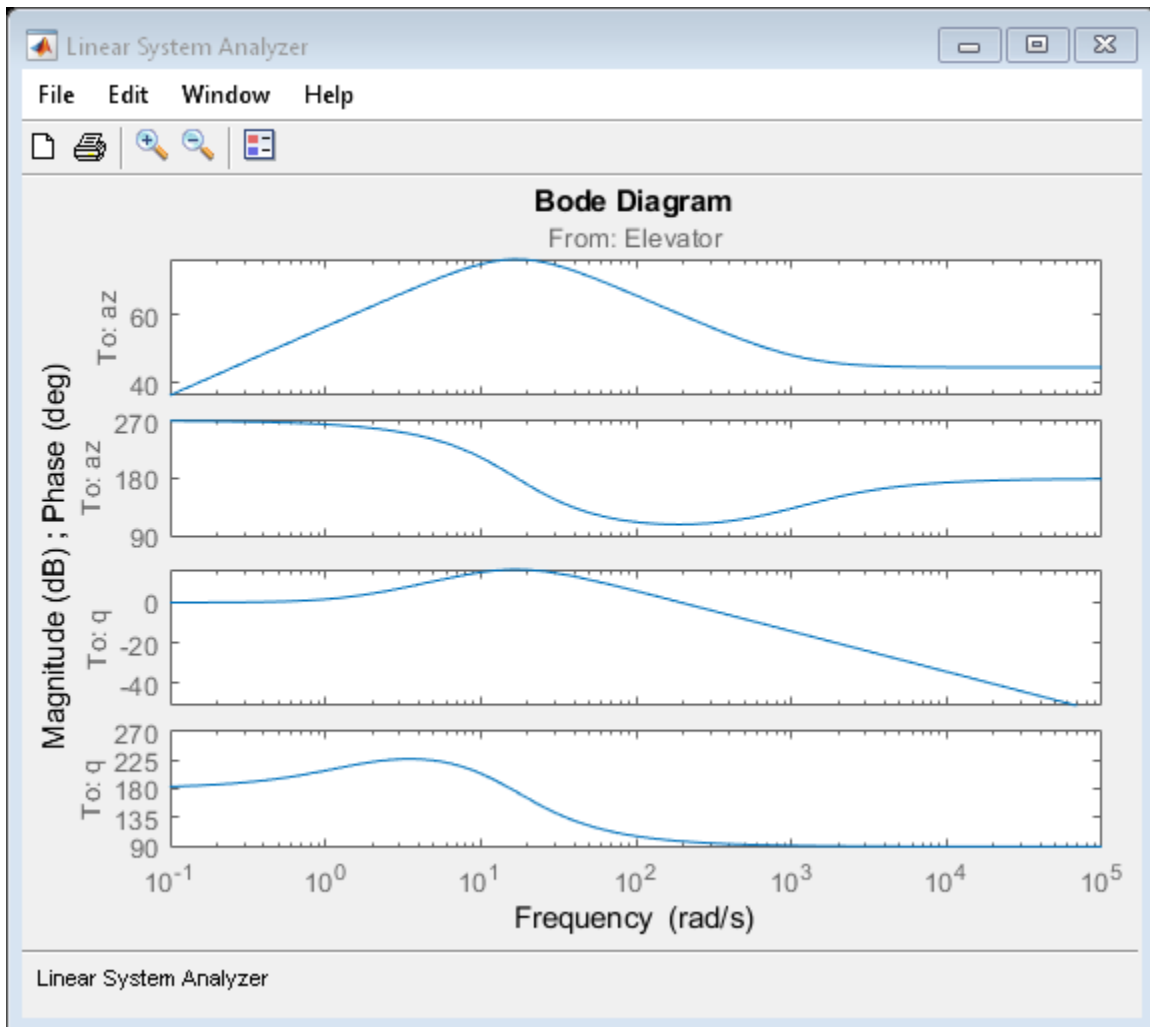
% find index of desired states in the state vector
names = sys.StateName;
q_idx = find(strcmp('q',names));
az_idx = find(strcmp('U,w(2)',names));

airframe = ss(sys.A([az_idx q_idx],[az_idx q_idx]),sys.B([az_idx q_idx],:),sys.C(:,[az_idx q_idx]),[]);

set(airframe,'inputname',{'Elevator'}, ...
     'outputname',[{'az'} {'q'}]);

ltiview('bode',airframe);

```





## Airframe Trim and Linearize with Control System Toolbox™

This example shows how to trim and linearize an airframe in the Simulink® environment using the Control System Toolbox™ software

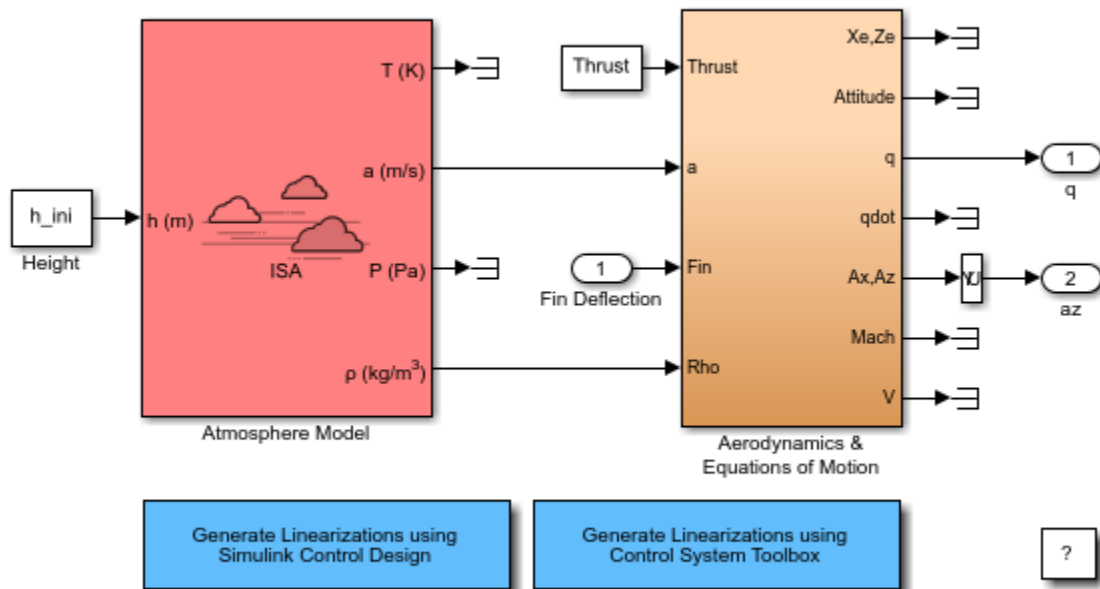
Designing an autopilot with classical design techniques requires linear models of the airframe pitch dynamics for several trimmed flight conditions. The MATLAB® technical computing environment can determine the trim conditions and derive linear state-space models directly from the nonlinear Simulink and Aerospace Blockset™ model. This step saves time and helps to validate the model. The Control System Toolbox functions allow you to visualize the motion of the airframe in terms of open-loop frequency or time responses.

### Initialize Guidance Model

The first problem is to find the elevator deflection, and the resulting trimmed body rate ( $q$ ), which will generate a given incidence value when the missile is traveling at a set speed. Once the trim condition is found, a linear model can be derived for the dynamics of the perturbations in the states around the trim condition.

```
open_system('aeroblk_guidance_airframe');
```

### Model used in airframe trim and linearization model examples



Copyright 1990-2019 The MathWorks, Inc.

### Define State Values

```
h_ini    = 10000/m2ft;    % Trim Height [m]
M_ini    = 3;             % Trim Mach Number
alpha_ini = -10*d2r;     % Trim Incidence [rad]
theta_ini = 0*d2r;       % Trim Flightpath Angle [rad]
```

```
v_ini = M_ini*(340+(295-340)*h_ini/11000);    % Total Velocity [m/s]
```

```
q_ini = 0;                                % Initial pitch Body Rate [rad/sec]
```

### Find Names and Ordering of States from Simulink® Model

```
[sizes,x0,names]=aeroblk_guidance_airframe([],[],[],'sizes');
```

```
state_names = cell(1,numel(names));
```

```
for i = 1:numel(names)
    n = max(strfind(names{i},'/'));
    state_names{i}=names{i}(n+1:end);
end
```

### Specify Which States to Trim and Which States Remain Fixed

```
fixed_states      = [{'U,w'} {'Theta'} {'Position'}];
fixed_derivatives = [{'U,w'} {'q'}];                % w and q
fixed_outputs     = [];                             % Velocity
fixed_inputs      = [];
```

```
n_states=[];n_deriv=[];
```

```
for i = 1:length(fixed_states)
    n_states=[n_states find(strcmp(fixed_states{i},state_names))]; %#ok<AGROW>
end
```

```
for i = 1:length(fixed_derivatives)
    n_deriv=[n_deriv find(strcmp(fixed_derivatives{i},state_names))]; %#ok<AGROW>
end
```

```
n_deriv=n_deriv(2:end);                    % Ignore U
```

### Trim Model

```
[X_trim,U_trim,Y_trim,DX]=trim('aeroblk_guidance_airframe',x0,0,[0 0 v_ini]', ...
    n_states,fixed_inputs,fixed_outputs, ...
    [],n_deriv) %#ok<NOPTS>
```

```
X_trim =
```

```
1.0e+03 *
-0.0002
    0
 0.9677
-0.1706
    0
-3.0480
```

```
U_trim =
```

```
0.1362
```

```
Y_trim =
```

```
-0.2160
    0
```

```
DX =
     0
    -0.2160
   -14.0977
     0
    967.6649
   -170.6254
```

### Derive Linear Model and View Frequency Response

```
[A,B,C,D]=linmod('aeroblk_guidance_airframe',X_trim,U_trim);
if exist('control','dir')
    airframe = ss(A(n_deriv,n_deriv),B(n_deriv,:),C([2 1],n_deriv),D([2 1],:));
    set(airframe,'StateName',state_names(n_deriv), ...
        'InputName',{'Elevator'}, ...
        'OutputName',[{'az'} {'q'}]);

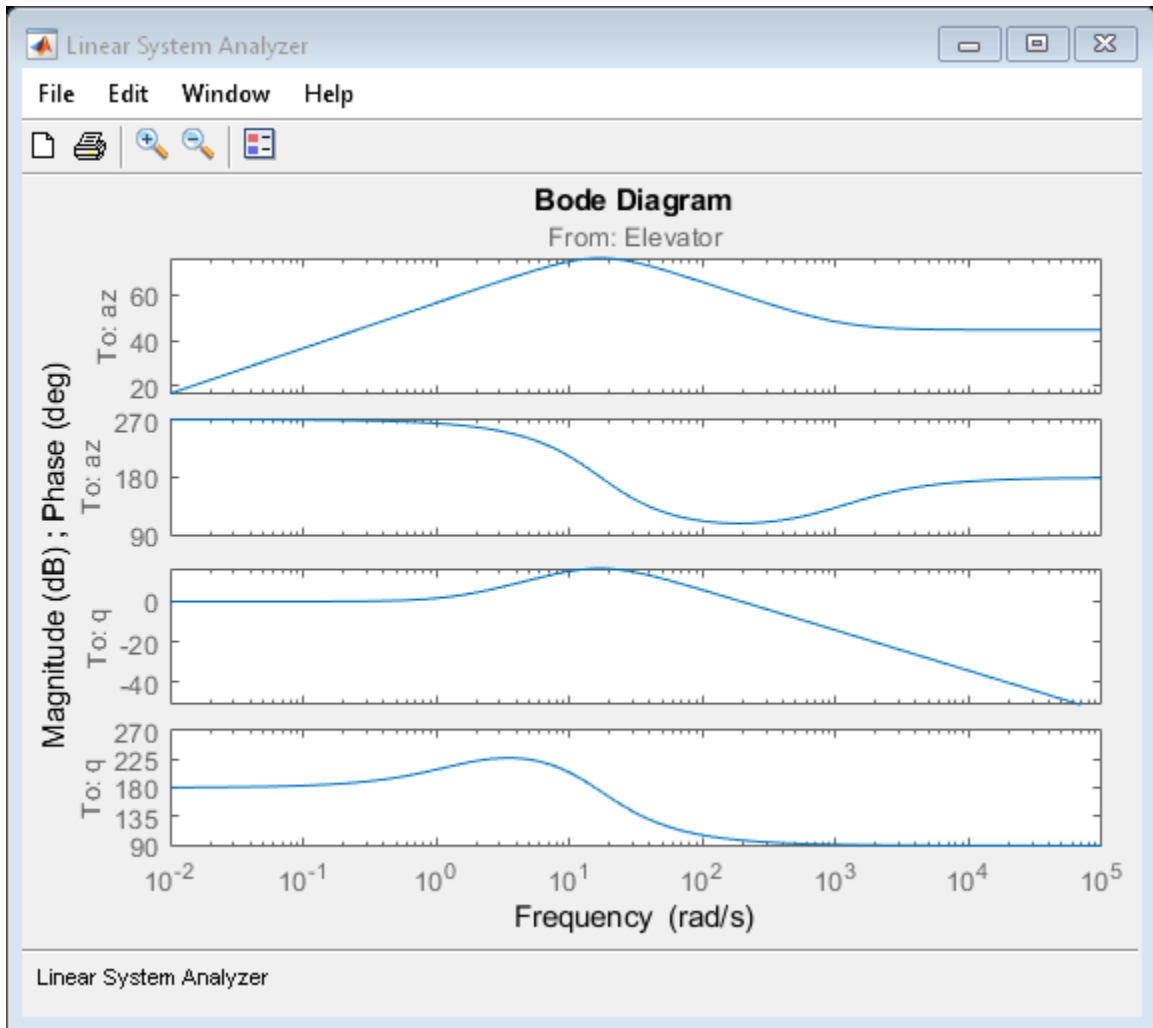
    zpk(airframe)
    ltiview('bode',airframe)
end
```

```
ans =
```

```
From input "Elevator" to output...
      -170.45 s (s+1133)
az:  -----
      (s^2 + 30.04s + 288.9)

      -194.66 (s+1.475)
q:  -----
      (s^2 + 30.04s + 288.9)
```

```
Continuous-time zero/pole/gain model.
```

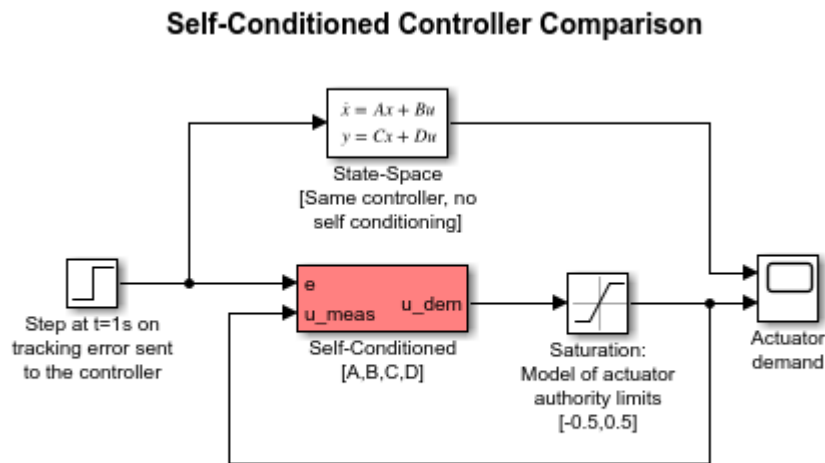


## Self-conditioned Controller Comparison

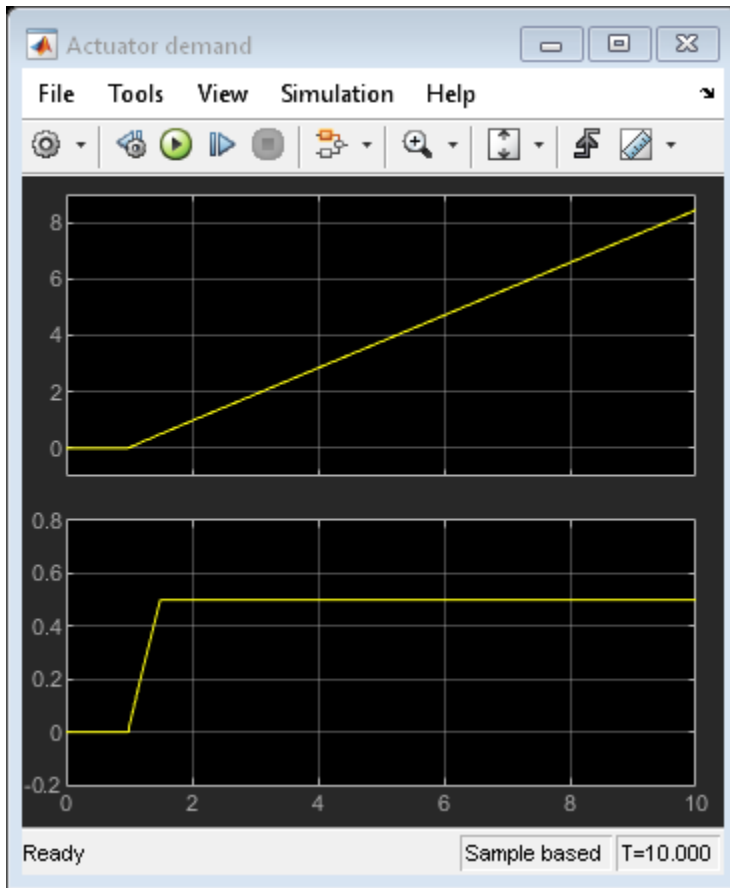
This model shows how to compare the implementation of a state-space controller  $[A,B,C,D]$  in a self-conditioned form versus a typical state-space controller  $[A,B,C,D]$ . This model requires Control System Toolbox™ software.

For the self-conditioned state-space controller, if measured control value is equal to the demanded control value ( $u_{\text{meas}} = u_{\text{dem}}$ ), then the controller implementation is the typical state-space controller  $[A,B,C,D]$ . However if measured control value ( $u_{\text{meas}}$ ) is limited, e.g., rate limiting, then the poles of the controller become those defined in the mask dialog box.

The results of a typical state-space controller  $[A,B,C,D]$  and a self-conditioned state-space controller with a limited measured control value are shown below.



Copyright 1990-2019 The MathWorks, Inc.



## Quadcopter Project

This example shows how to use Simulink® to model a quadcopter, based on the PARROT® series of mini-drones.

- To manage the model and source files, it uses “Project Management” (Simulink).
- To show the quadcopter in a three-dimensional environment, it uses “” (Simulink 3D Animation).
- For the collaborative development of a flight simulation application, it provides an implementation of the Flight Simulation application template.

This example works with the Simulink Support Package for PARROT Minidrones.

**Note:** To successfully run this example you must have a C/C++ compiler installed.

### Open the Quadcopter Project

Run the following command to create and open a working copy of the project files for this example:

```
asbQuadcopterStart
```

### Quadcopter Physical Characteristics

The following schematic shows the quadcopter physical characteristics:

- Axis
- Mass and Inertia
- Rotors



#### Axis

The quadcopter body axis is centered in the center of gravity.

- The x-axis starts at the center of gravity and points in the direction along the nose of the quadcopter.

- The y-axis starts at the center of gravity and points to the right of the quadcopter.
- The z-axis starts at the center of gravity and points downward from the quadcopter, following the right-hand rule.

### **Mass and Inertia**

We assume that the whole body works as a particle. The file `vehicleVars` contains the values for the inertia and mass.

### **Rotors**

- Rotor #1 rotates positively with respect to the z-axis. It is located parallel to the xy-plane, -45 degrees from the x-axis.
- Rotor #2 rotates negatively with respect to the body's z-axis. It is located parallel to the xy-plane, -135 degrees from the x-axis.
- Rotor #3 has the same rotation direction as rotor #1. It is located parallel to the xy-plane, 135 degrees from the x-axis.
- Rotor #4 has the same rotation direction as rotor #2. It is located parallel to the xy-plane, 45 degrees from the x-axis.

This example uses the approach defined by Prouty[1] and adapted to a heavy-lift quadcopter by Ponds et al[2].

### **Control**

For control, the quadcopter uses a complementary filter to estimate attitude, and Kalman filters to estimate position and velocity. The example implements:

- A PID controller for pitch/roll control
- A PD controller for yaw
- A PD controller for position control in North-East-Down coordinates

The `controllerVars` file contains variables pertinent to the controller. The `estimatorVars` file contains variables pertinent to the estimator.

The example implements the controller and estimators as model subsystems, enabling several combinations of estimators and controllers to be evaluated for design.

To provide inputs to the quadcopter (in pitch, roll, yaw, North (X), East (Y), Down (Z) coordinates ), use one of the following and change the `VSS_COMMAND` variable in the workspace:

- A Signal Editor block
- A joystick
- Previously saved data
- Spreadsheet data

### **Sensors**

The example uses a set of sensors to determine its states:

- An Inertial Measurement Unit (IMU) to measure the angular rates and translational accelerations.



- A camera for optical flow estimation.
- A sonar for altitude measurement.

The example stores the characteristics for the sensors in the file `sensorVars`. To include sensor dynamics with these measurements, you can change the `VSS_SENSORS` variable in the workspace.

### Environment

The models implement several Aerospace Blockset™ environment blocks, including those for atmosphere and gravity models. To include these models, you can change the `VSS_ENVIRONMENT` variable in the workspace to toggle between variable and fixed environment models.

### Linearization

The model uses the `trimLinearizeOpPoint` to linearize the nonlinear model of the quadcopter using Simulink Control Design (R).

### Testing

To make sure that the trajectory generation tool works properly, the example implements a test in the `trajectoryTest` file. For more information on how to do this, see the Simulink Control Design “Get Started with Simulink Control Design” (Simulink Control Design).

### Visualization

You can visualize the variables for the quadcopter in one of the following ways:

- Using Simulation Data Inspector.
- Using the flight instrument blocks.
- Toggling between the different visualization variant subsystems. You can toggle between the different variant subsystems by changing the `VSS_VISUALIZATION` variable. Note that one of these variants is a FlightGear animation. To use this animation, you must add a FlightGear compatible model of the quadcopter to the project. The software does not include this model.

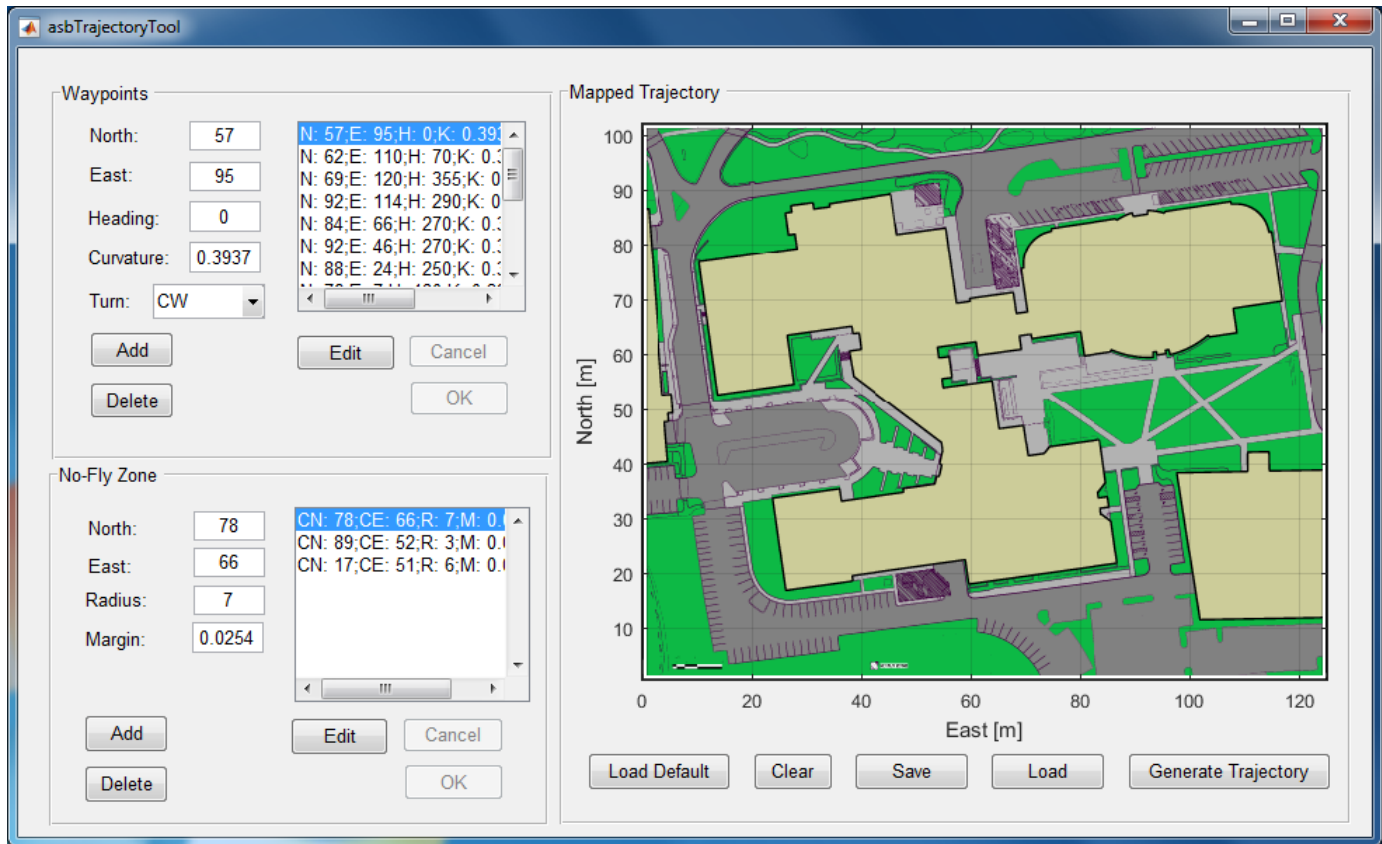
### Trajectory Generation

A trajectory generation tool, using the Dubin method, creates a set of navigational waypoints. To create a trajectory with a set of waypoints this method uses a set of poses defined by position, heading, turn curvature, and turn direction.

To start the tool, ensure that the project is open and run:

```
asbTrajectoryTool
```

The following interface displays:



The interface has several panels:

### Waypoints

This panel describes the poses the trajectory tool requires. To define these poses, the panel uses text boxes:

- **North** and **East** (position in meters)
- **Heading** (degrees from North)
- **Curvature** (turning curvature in meters<sup>-1</sup>)
- **Turn** (direction clockwise or counter-clockwise)

A list of poses appears in the waypoint list to the right of the text boxes.

To add a waypoint, enter pose values in the edit boxes and click **Add**. The new waypoint appears in the waypoint list in the same panel.

To edit the characteristics of a waypoint, select the waypoint in the list and click **Edit**. The characteristics of the waypoints display in the edit boxes. Edit the characteristics as desired, then click **OK**. To cancel the changes click **Cancel**.

To delete a waypoint, in the waypoint list, select the waypoint and click **Delete**.

### No-Fly Zone

The panel defines the location and characteristics of the no-fly zones. To define the no-fly zone, the panel uses text boxes:

- **North** and **East** (position in meters)
- **Radius** (distance in meters)
- **Margin** (safety margin in meters)

Use the **Add**, **Delete**, **Edit**, **OK**, and **Cancel** buttons in the same way as for the Waypoints panel.

### **Mapped Trajectory**

This panel plots the trajectory over the Apple Hill campus aerial schematic based on the waypoints and no-fly zone characteristics.

To generate the trajectory, add the waypoint and no-fly zone characteristics to the respective panels, then click **Generate Trajectory**.

To save the trajectory that is currently in your panel, click the **Save** button. This button only saves your last trajectory.

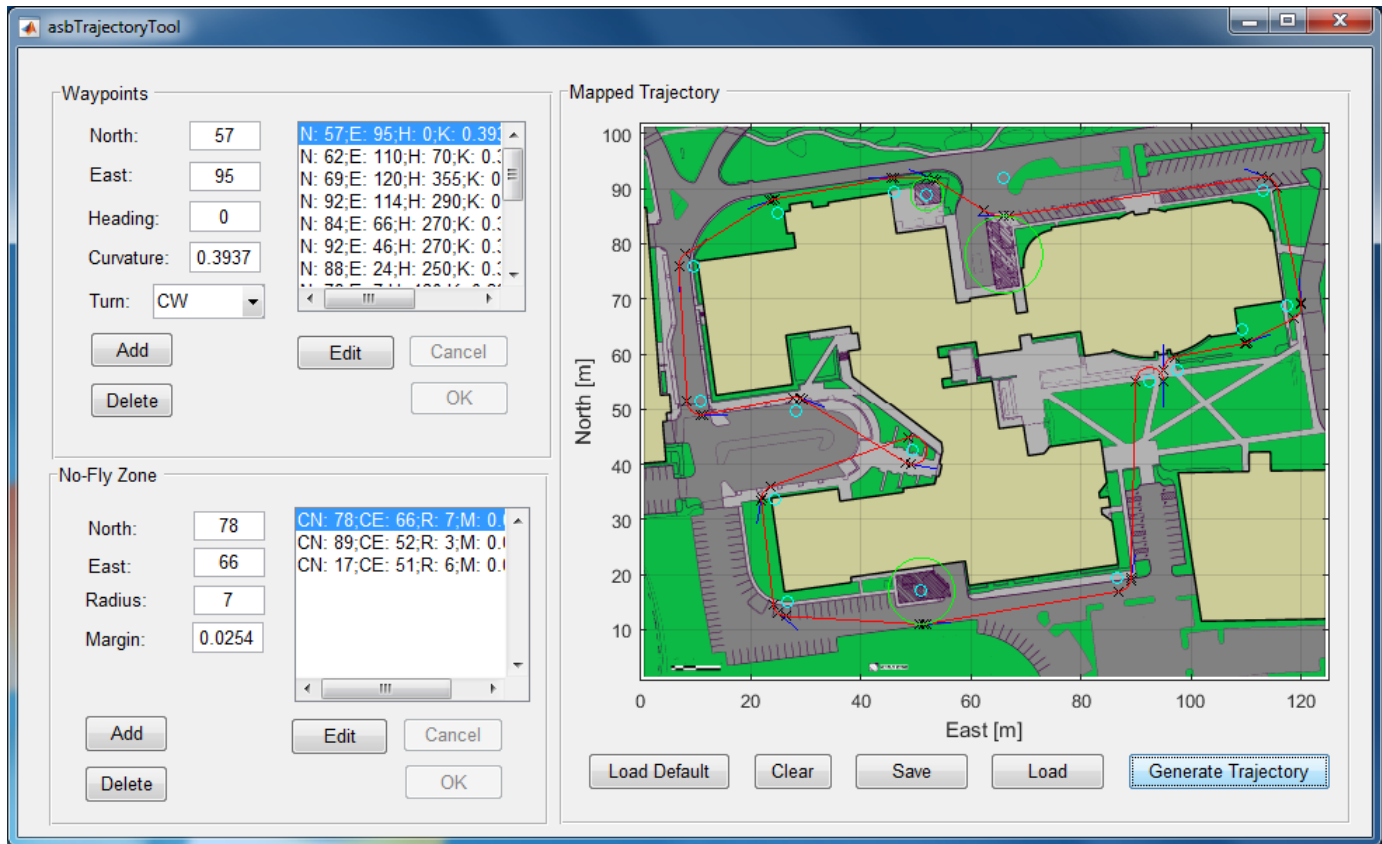
To load the last saved trajectory, click **Load**.

To load the default trajectory, press the **Load Default** button.

To clear the values in the waypoint and no-fly zone panel, click **Clear**.

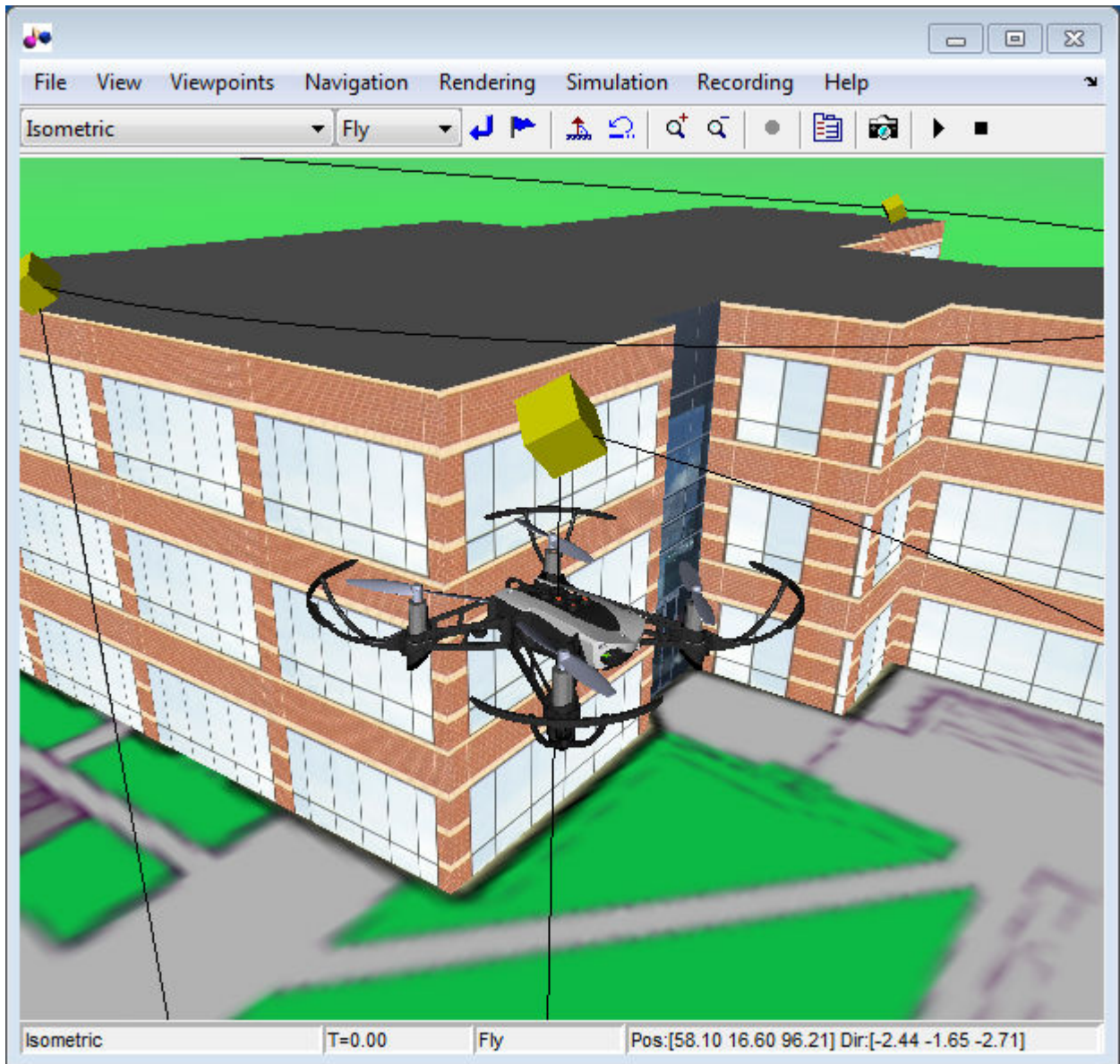
The default data contains poses for specific locations at which the toy quadcopter uses its cameras so the pilot on the ground can estimate the height of the snow on the roof. Three no-fly zones were defined for each of the auxiliary power generators, so in case there is a failure in the quadcopter, it does not cause any damage to the campus infrastructure.

When the example generates the trajectory for the default data, the plot should appear as follows:



The red line represents the trajectory, black x markers determine either a change in the trajectory or a specific pose. Blue lines that represent the heading for that specific waypoint accompany specific poses. No-fly zones are represented as green circles.

If you have a Simulink 3D Animation license, you can also view the trajectory in a 3-D representation of the Apple Hill campus:



**Note:** For visualization reasons the 3D representation of the quadcopter is not at the same scale as the environment.

### References

- [1] Prouty, R. Helicopter Performance, Stability, and Control. PWS Publishers, 2005.
- [2] Ponds, P., Mahony, R., Corke, P. Modelling and control of a large quadrotor robot. Control Engineering Practice. 2010.

## Electrical Component Analysis for Hybrid and Electric Aircraft

This example illustrates how to use modeling for rapid exploration of design space in the hybrid and electric aircraft area and compare the results to design criteria. This process can reduce the number of design iterations and ensure that the final design meets system-level requirements.

Hybrid and electric aircraft are areas of aggressive development in the aerospace industry. To accelerate the process of choosing between hybrid and pure electric power systems, select power network architectures, and size electrical components, consider using simulation with MathWorks products.

Using preconfigured simulation configurations, this example shows the tradeoffs between battery sizes for a pure electric or hybrid electric power system, with and without payloads. It includes the Pipistrel Alpha Electro and NASA X-57 Maxwell aircraft. Implemented as a project, the example provides various shortcuts that you can use to experiment with simulation.

### Model Components

The Aircraft block specifies the aircraft compared in the project:

- Pipistrel Alpha Electro[1], a preconfigured model of the world's first two-seat pure electric training aircraft.
- NASA X-57 Maxwell[2], a preconfigured model of the NASA X-57 Maxwell aircraft, an experimental electric aircraft.
- Custom, an aircraft that you can model according to your specifications.

In the Aircraft block, each aircraft is modeled as a 4th Order Point Mass (Longitudinal) in flight, with the required thrust output as a load on the motor. This abstract model assumes that the pilot takes the actions necessary to follow the mission.

You can specify several aerodynamic characteristics for the selected aircraft, including empty and maximum mass, wing area and lift curve values, a drag coefficient, and target speeds for the climb, cruise, and descent portions of the mission. The block uses these values, along with the mission altitudes, to create lookup tables for angle of attack ( $\alpha$ ) and thrust, given atmospheric density, target speed, and flight profile angle ( $\gamma$ ). At every point along the flight, the lookup tables return  $\alpha$  and thrust for input into the 4th-order Point Mass block, which calculates the accelerations. By holding  $\alpha$  and thrust at the calculated, steady-state values, the actual speed quickly reaches the desired speed. The Aircraft block also defines the climb and descent rates for the mission.

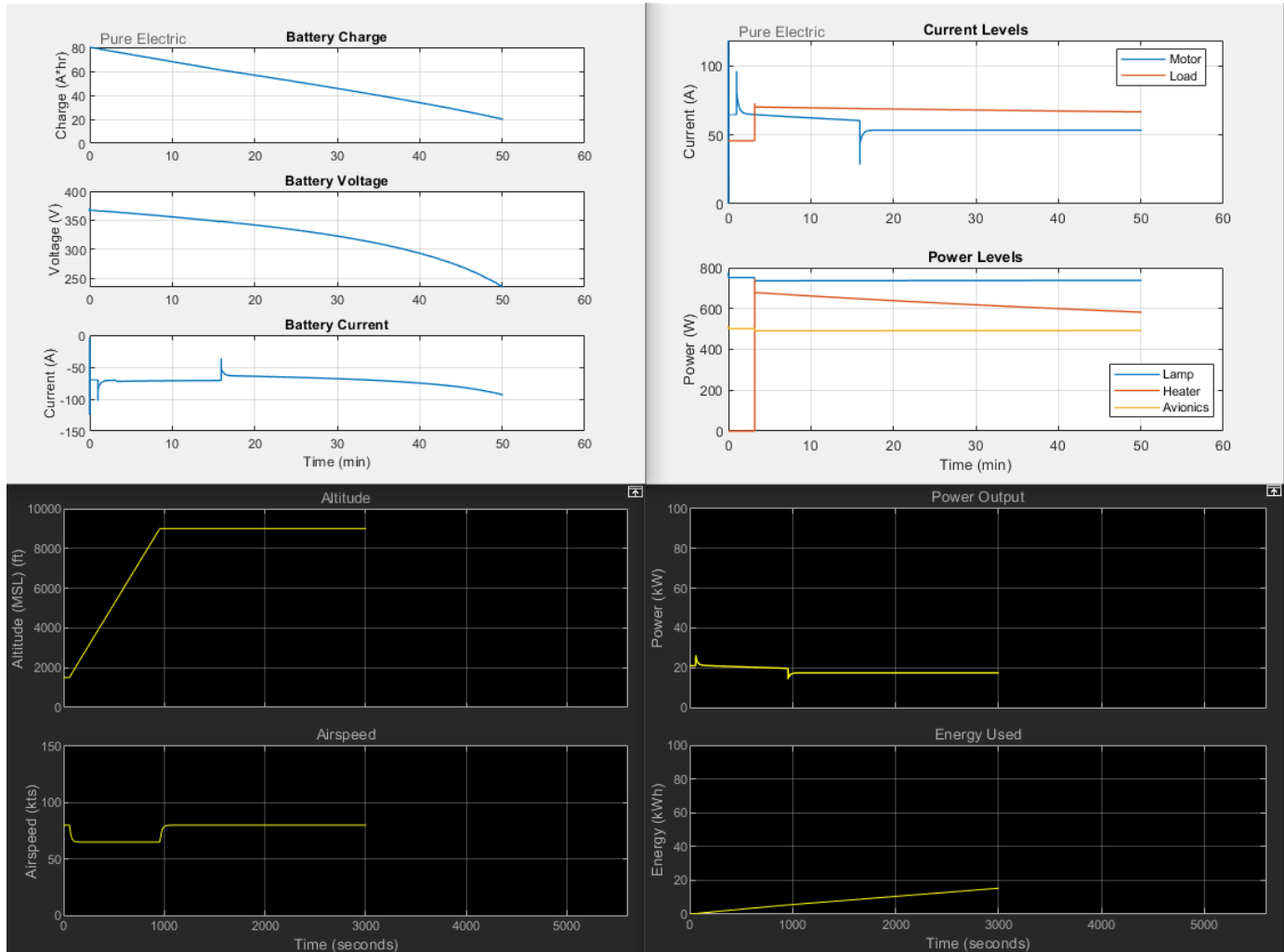
The Mission Profile block sets the airport and cruise altitudes and the total flight distance. If entered values are not feasible, such as a distance too short for the aircraft to climb to and descend from the requested cruise altitude, then the values are adjusted, and a message describes the change. To see how far the aircraft can fly until it runs out of battery power, enter a long total flight distance.

The Environment block calculates the air density at the chosen altitudes using the COESA Atmosphere Model.

### Run the Model with Default Settings

By default, the Aircraft block is configured for the Pipistrel Alpha Electro aircraft. To see the performance of this aircraft with default settings, run the model using the "Single Run" project shortcut. The Pipistrel is run with no payload.

Two figures display showing the battery states, current, and power levels during the flight. Two scope windows show the mission progress (altitude and airspeed), power output, and energy used. Note, the battery runs out of capacity (reaches 20 amp-hours) before the entire mission is completed.



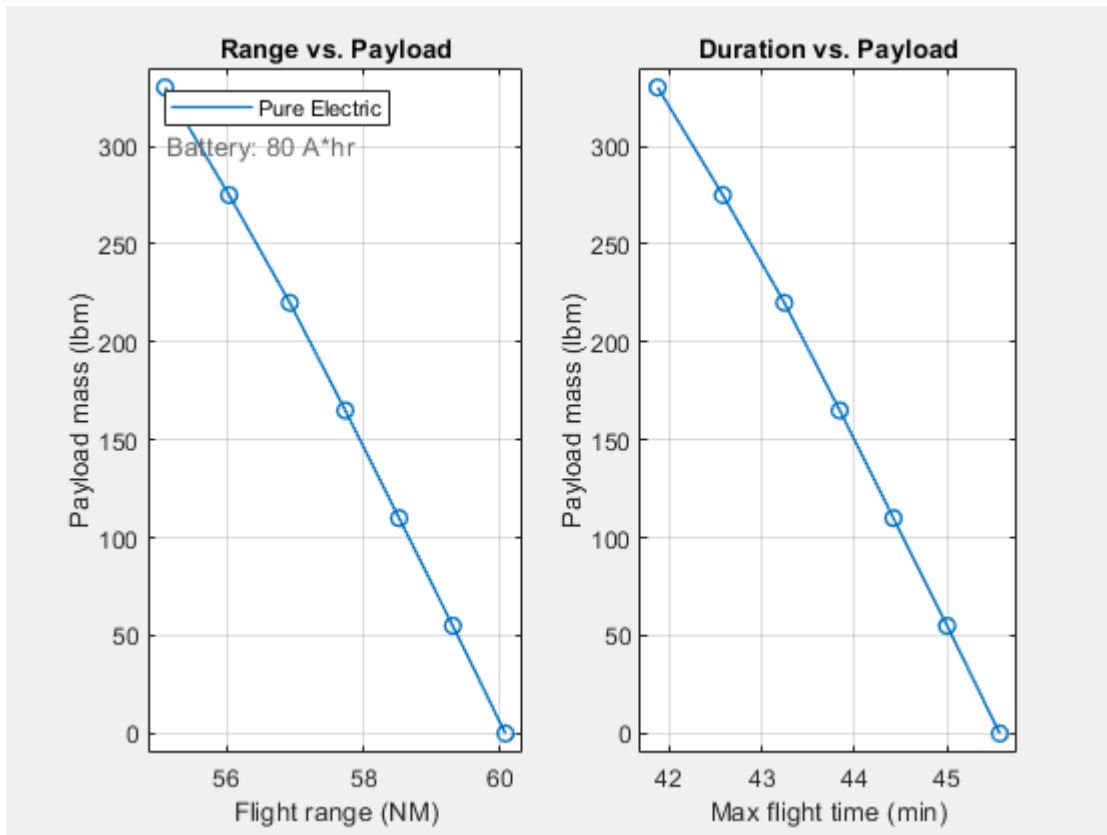
To capture the data created by a simulation, the example uses the Simscape “Data Logging” (Simscape) capability. The various simulation cases provided by the project shortcuts run scripts that run the desired cases and then extract the results from the Simscape log to create the figures. For aircraft with more than one electric motor, the total required torque calculated by the Aircraft subsystem is divided by the number of motors before being passed to the Power subsystem. The criteria to stop the simulation, `batteryCapacityMin`, is adjusted up from 20 amp-hours accordingly.

### Run the Model with Cruise at 3000 Feet

The Pipistrel is designed as a primary flight trainer. The default mission configured for the model might not be the typical mission for the Pipistrel. Modify the mission to cruise at 3000 instead of 9000 feet, and then make a single run to see the effect of this change (duration decreases).

### Run the Model with a Payload

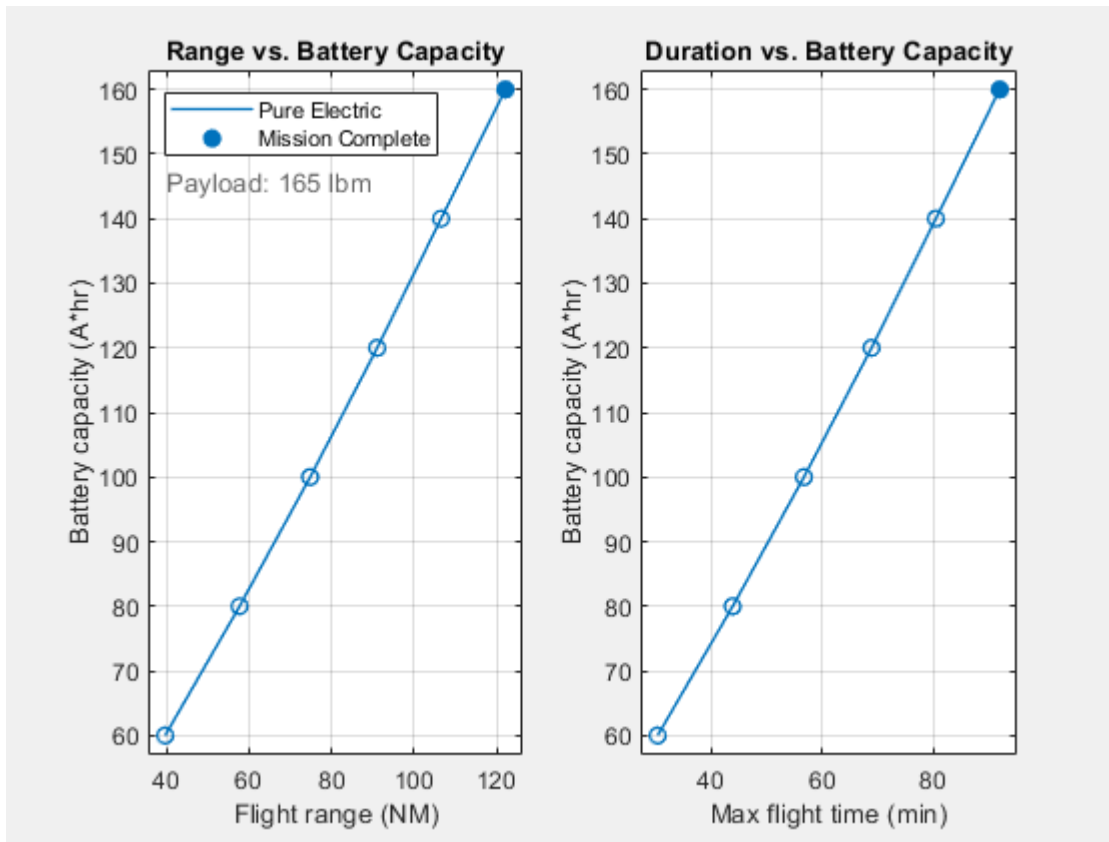
To run the Pipistrel with a 165 pound payload, use the "Set Payload Mass" shortcut, then run the model again. To see the effect of a range of payload values, use the "Sweep Payload Mass" shortcut. This shortcut varies the payload from 0 to 330 pounds. The sweeps produce different figures, showing the flight time and range for the swept parameter. Each marker represents one simulation. Hover your mouse over a marker to see its payload mass ("X") and flight range or duration ("Y") values.



### Run the Model with Various Battery Sizes

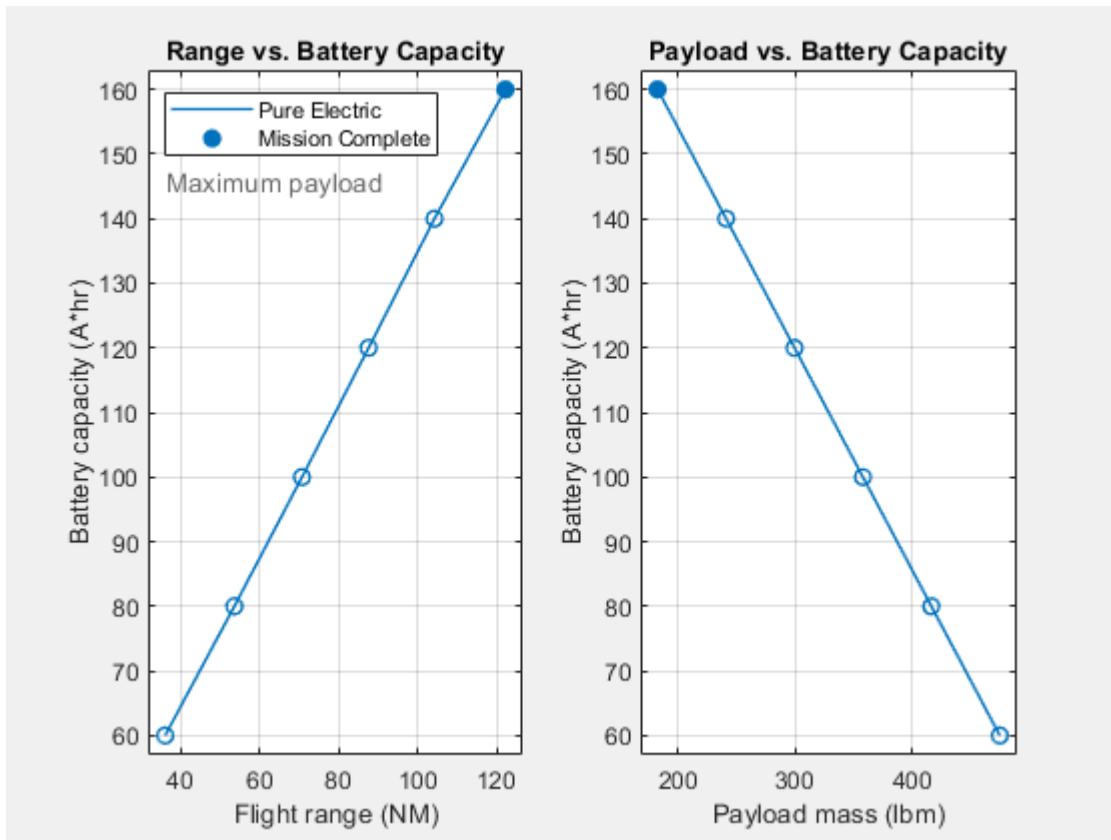
To see how the flight range is affected by the battery size, use the "Sweep Battery Size" shortcut. The shortcut varies the capacity from 60 to 160 amp-hours (or 100 to 200 amp-hours if the X-57 is selected). The example assumes the battery mass to be linearly proportional to its capacity, so increasing its capacity increases its mass as well. If the payload is set large enough (over 183 pounds for the Pipistrel), this increase in battery mass can cause the largest battery in the sweep to put the aircraft over its maximum mass value (e.g. 1212 pounds for the Pipistrel). The `total_mass` variable in the base workspace stores the total mass of each case in the sweep. If the battery capacity is enough to complete the mission, a filled marker indicates this. Note that pure electric aircraft, such as the Pipistrel, have no fuel burn, resulting in no change in mass throughout a flight.





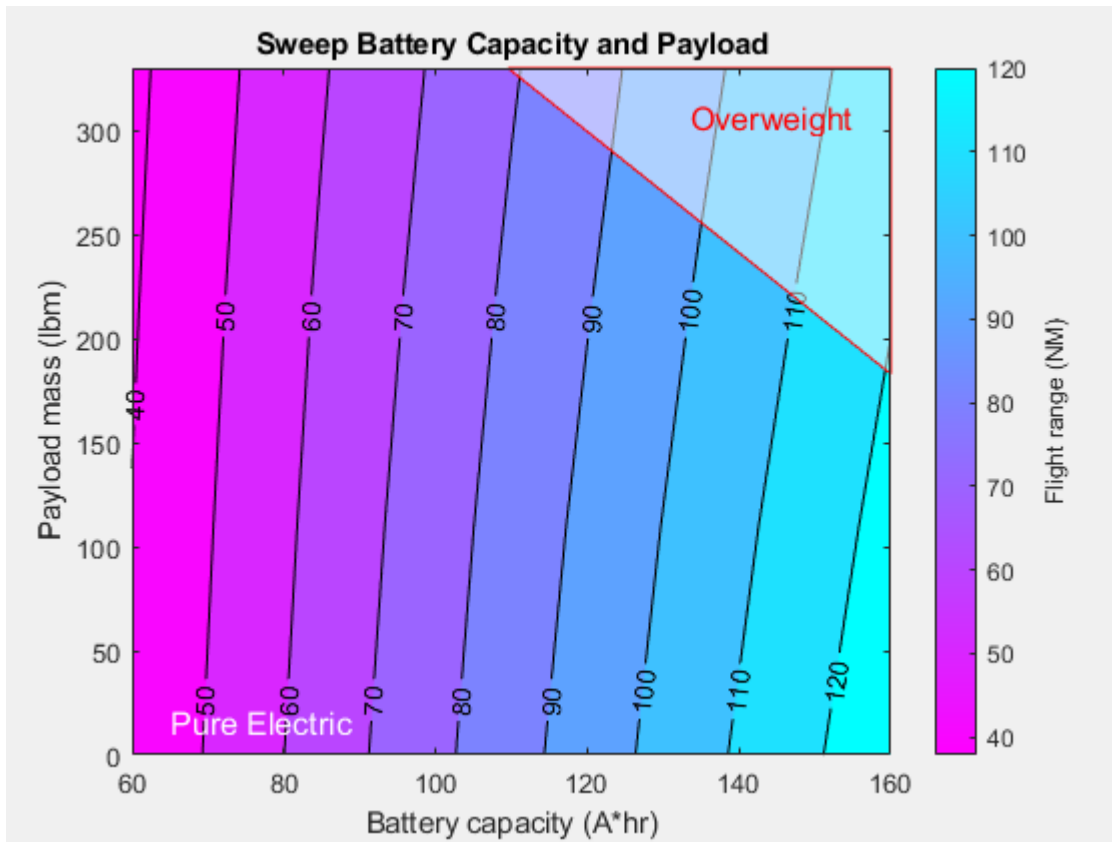
### Run the Model with Maximum Payload

The battery capacity sweep in "Run the Model with Various Battery Sizes" yields the flight range for a fixed payload. To find the flight range for the maximum payload, use the "Sweep Range at Max Payload" shortcut. This sweeps the battery capacity with the payload set such that the total mass equals the maximum mass in every case (unless payload becomes negative, in which case payload is set to zero and the model goes overweight). From these results, you can choose the maximum battery size for a given payload requirement.



### Run the Model with Various Battery and Payload Sizes

To simultaneously see the flight range distance dependence on both battery size and payload mass, use the "Sweep Battery & Payload" shortcut, which produces a contour plot. Areas where the aircraft is over the maximum weight are denoted with a red and white "Overweight" overlay.



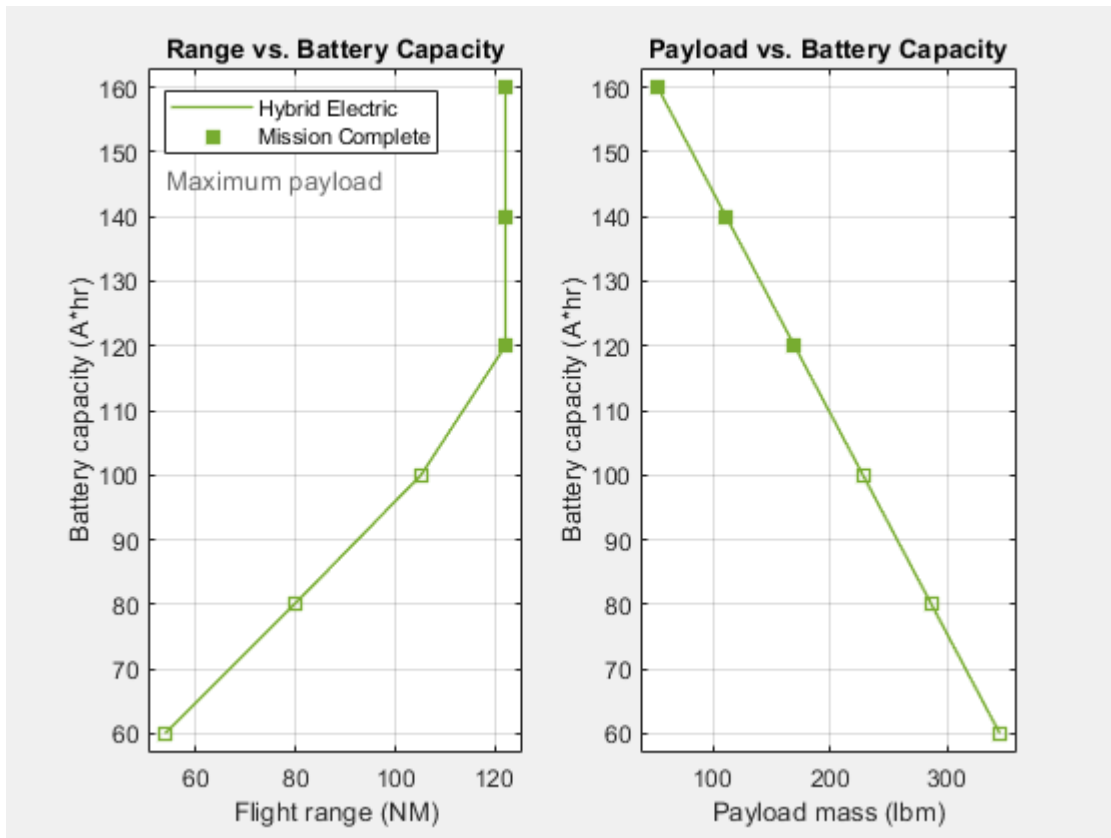
### Run the Model with the Hybrid Electric Option

Since battery power density is much less than that of aviation fuel, pure electric aircraft have less distance range than their fuel-powered counterparts. To bridge this gap, consider a hybrid electric power system. To recharge the batteries, the hybrid power subsystem variant in this example adds a 130 pound, 50 kW, two-stroke piston engine and generator to the pure electric power subsystem components.

To try a hybrid power system, perform one of these workflows:

### Change Power Subsystem Variant and Run the Battery Range Sweep

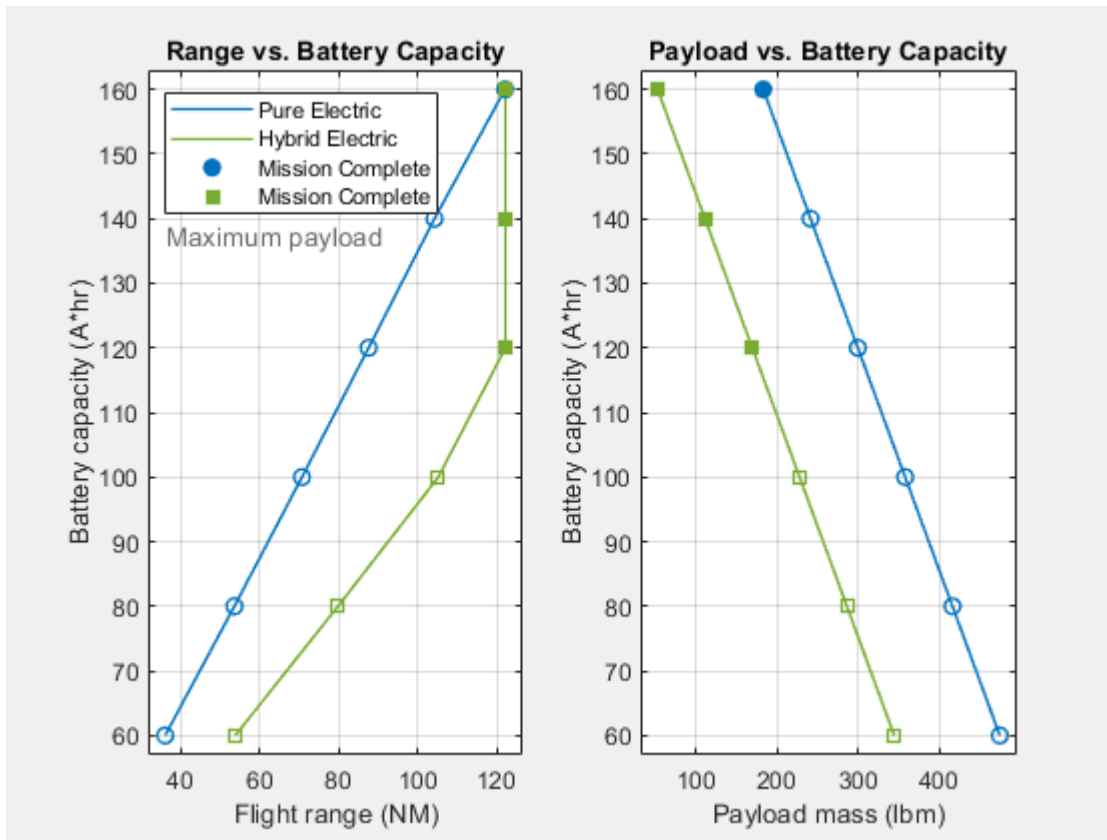
1. Use the "Hybrid Electric" shortcut. This shortcut changes the Power subsystem variant.
2. To repeat the sweep previously done for pure electric power, use "Sweep Range at Max Payload".



In this case the larger battery sizes are capable of longer missions than what is currently defined (120 NM). Try entering a longer total flight distance (e.g. 200 NM) in the Mission Profile and then rerun this sweep.

### Run the Battery Range Sweep for Both Power Variants

1. Use the "Hybrid/Electric Range Comparison" shortcut to run the sweep for both power variants.
2. Compare the results in a single figure. The results show that a hybrid power system can improve range, but at the expense of payload.



### Explore How the Mission Affects Range and Endurance

To explore how the mission affects the range and endurance, select the Custom aircraft model in the Aircraft block. The default configuration values for this aircraft are the same as for the Pipistrel, with the exception of the speeds. Adjust the speeds and mission altitudes as desired, then run and compare the results to those for the Pipistrel with default settings.

If the custom aircraft being evaluated is significantly different from the Pipistrel, then accordingly adjust the CustomAircraft values in the "asbHybridAircraftDefaults.m" file.

### Additional Model Details

#### Power Subsystem

The power subsystem is modeled with two variant models: Pure Electric and Hybrid Electric, controlled by the variable `POWER_MODE` in the base workspace.

The Pure Electric model includes a battery, high- and low-voltage DC networks, and a mechanical model of the aircraft. The mechanical model acts as a load on the high-voltage DC network. The low-voltage DC network includes a set of loads that turn on and off during the flight mission.

The series Hybrid Electric model contains all the components in the Pure Electric model, plus a 50 kW engine, a generator, and fuel. The Generic Engine drives a generator that supplements the power available from the battery. The generator recharges the battery during flight. The mass of the fuel consumed by the engine is included in the simulation. The low-voltage DC network includes a set of loads that turn on and off during the flight cycle, including the fuel pump for the combustion engine.

These two variant models are composed of three or four subsystems for load torque, the motor, the generator, and the DC power distribution.

### **Load Torque Subsystem**

This subsystem converts the required mechanical power into the load torque on the motor shaft. This model assumes that a specified amount of the motor mechanical power is converted into thrust. Dividing the required power to maintain thrust by the motor speed results in the load torque on the motor shaft. The motor control system adjusts to maintain the required shaft speed under the varying load.

### **Motor Subsystem**

This subsystem represents the electric motor and drive electronics operating in torque-control mode, or equivalently, current-control mode. The motor permissible range of torques and speeds is defined by a torque-speed envelope.

### **Fuel Pump Subsystem**

This subsystem models the fuel pump. An electric motor drives a pump that pushes fuel through a valve. The opening of the valve varies during the flight, which changes the current that the motor draws from the DC network.

### **Generator Subsystem**

This subsystem represents the generator and drive electronics operating in torque-control mode, or equivalently, current-control mode. It is driven by the combustion engine to supply additional electrical power to the aircraft network.

### **DC Power Distribution Subsystem**

This subsystem models the breakers that open and close to connect and disconnect loads from the low-voltage DC network. The varying conditions affect the power drawn from the network, the range of the aircraft, and the power requirements for the power lines in the aircraft.

### **References**

[1] <https://www.pipistrel-aircraft.com/aircraft/electric-flight/alpha-electro/>

[2] <https://www.nasa.gov/specials/X57/index.html>

# **Aerospace Units Appendix**

## Aerospace Units

The main blocks of the Aerospace Blockset library support standard measurement systems. The Unit Conversion blocks support all units listed in this table.

| Quantity             | Metric (MKS)  | English  |
|----------------------|---|--|
| Acceleration         | meters/second <sup>2</sup> (m/s <sup>2</sup> ), kilometers/second <sup>2</sup> (km/s <sup>2</sup> ), (kilometers/hour)/second (km/h-s), g-unit ( <i>g</i> ) | inches/second <sup>2</sup> (in/s <sup>2</sup> ), feet/second <sup>2</sup> (ft/s <sup>2</sup> ), (miles/hour)/second (mph/s), g-unit ( <i>g</i> )         |
| Angle                | radian (rad), degree (deg), revolution  | radian (rad), degree (deg), revolution   |
| Angular acceleration | radians/second <sup>2</sup> (rad/s <sup>2</sup> ), degrees/second <sup>2</sup> (deg/s <sup>2</sup> ), revolutions/minute (rpm), revolutions/second (rps)    | radians/second <sup>2</sup> (rad/s <sup>2</sup> ), degrees/second <sup>2</sup> (deg/s <sup>2</sup> ), revolutions/minute (rpm), revolutions/second (rps) |
| Angular velocity     | radians/second (rad/s), degrees/second (deg/s), revolutions/minute (rpm)  | radians/second (rad/s), degrees/second (deg/s), revolutions/minute (rpm)   |
| Density              | kilogram/meter <sup>3</sup> (kg/m <sup>3</sup> )  | pound mass/foot <sup>3</sup> (lbm/ft <sup>3</sup> ), slug/foot <sup>3</sup> (slug/ft <sup>3</sup> ), pound mass/inch <sup>3</sup> (lbm/in <sup>3</sup> ) |
| Force                | newton (N)  | pound (lb)   |
| Inertia              | kilogram-meter <sup>2</sup> (kg-m <sup>2</sup> )  | slug-foot <sup>2</sup> (slug-ft <sup>2</sup> )   |
| Length               | meter (m)   | inch (in), foot (ft), mile (mi), nautical mile (nm)  |
| Mass                 | kilogram (kg)   | slug (slug), pound mass (lbm)  |
| Pressure             | Pascal (Pa)   | pound/inch <sup>2</sup> (psi), pound/foot <sup>2</sup> (psf), atmosphere (atm)   |
| Temperature          | kelvin (K), degrees Celsius (°C)  | degrees Fahrenheit (°F), degrees Rankine (°R)  |
| Torque               | newton-meter (N-m)  | pound-feet (lb-ft)   |
| Velocity             | meters/second (m/s), kilometers/second (km/s), kilometers/hour (km/h)   | inches/second (in/s), feet/second (ft/s), feet/minute (ft/min), miles/hour (mph), knots  |