# Aerospace Blockset 2

## User's Guide

MATLAB®
& SIMULINK®

The MathWorks
Accelerating the pace of engineering and science

## How to Contact The MathWorks


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

508-647-7000 (Phone)

508-647-7001 (Fax)

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

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


*Aerospace Blockset User's Guide*

**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) |

# Contents

## 1 Getting Started



## 2 Using Aerospace Blockset

# 3 Case Studies

# 4 Blocks — By Category

# 5 Blocks — Alphabetical List

# A Aerospace Units

# Index

# Getting Started

Aerospace Blockset lets you model aerospace systems for use with Simulink® and MATLAB®.

# What Is Aerospace Blockset?

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

You can use Aerospace Blockset and Simulink to develop your aerospace system concepts and to efficiently revise and test your models throughout the life cycle of your design. Animate your aerospace motion simulations with MATLAB Graphics or the optional Virtual Reality Toolbox viewer. Use Aerospace Blockset with Real-Time Workshop® to automatically generate code for real-time execution in rapid prototyping and for hardware-in-the-loop systems.

# Related Products

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


- “Requirements for Aerospace Blockset” on page 1-3
- “Other Related Products” on page 1-3


## Requirements for Aerospace Blockset

In particular, Aerospace Blockset requires current versions of these products:

- MATLAB
- Aerospace Toolbox
- Simulink

## Other Related Products

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

### For More Information About MathWorks Products

For more information about any MathWorks software products, see either

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

# Running a Demo Model

This section introduces a missile guidance model that uses blocks from Aerospace Blockset to simulate a three-degrees-of-freedom missile guidance system, in conjunction with other Simulink blocks.

The model simulates a missile guidance system with a target acquisition and interception subsystem. The model implements a nonlinear representation of the rigid body dynamics of the missile airframe, including aerodynamic forces and moments. The missile autopilot is based on the trimmed and linearized missile airframe. The missile homing guidance system regulates missile acceleration and measures the distance between the missile and its target.

For more information on this model, see Chapter 3, “Case Studies”.

The following sections describe how to work with the Aerospace Blockset demos.


- “What This Demo Illustrates” on page 1-4
- “Opening the Model” on page 1-5
- “Key Subsystems” on page 1-6
- “Running the Demo” on page 1-9
- “Modifying the Model” on page 1-12


## What This Demo Illustrates

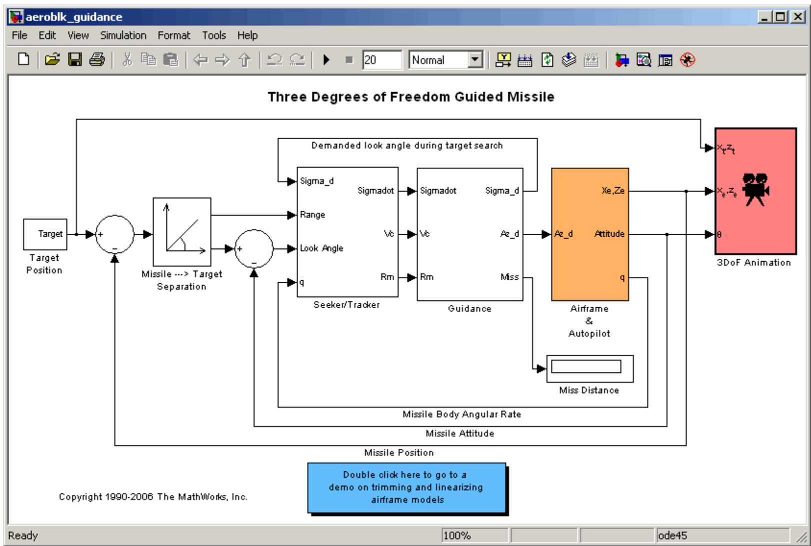
The missile guidance demo illustrates the following features of the blockset:

- Representing bodies and degrees of freedom with the Equations of Motion library blocks
- Using Aerospace Blockset with other Simulink blocks
- Using Aerospace Blockset with Stateflow®
- Feeding in and feeding out Simulink signals to and from Aerospace Blockset blocks with Actuator and Sensor blocks
- Encapsulating groups of blocks into subsystems
- Visualizing and animating an aircraft with the Animation library blocks

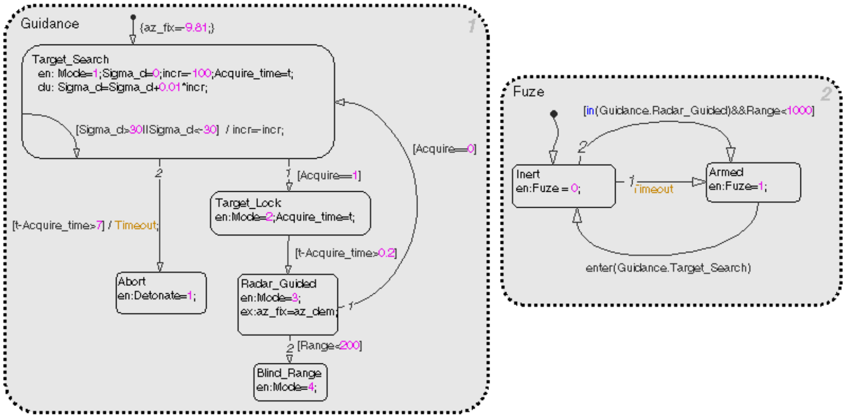
**Note** The Stateflow module in this demo is precompiled and does not require Stateflow to be installed.

## Opening the Model

Open the Demos browser, then locate and open the missile guidance demo. You can also open it by entering the demo name, `aeroblk_guidance`, at the MATLAB command line. The model opens.




A Stateflow chart for the guidance control processor also appears.

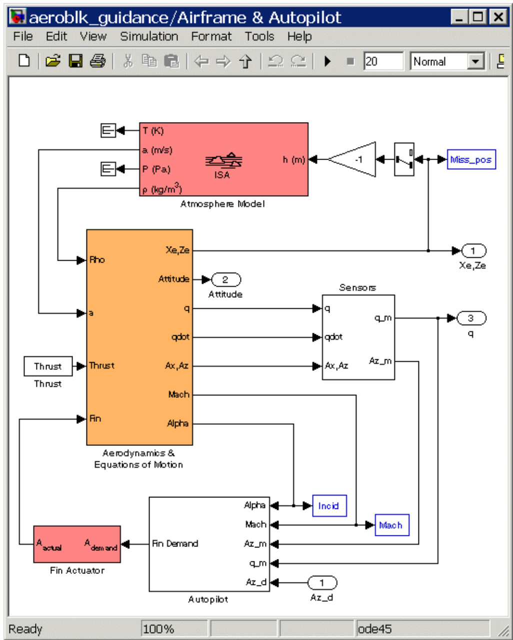



## Key Subsystems

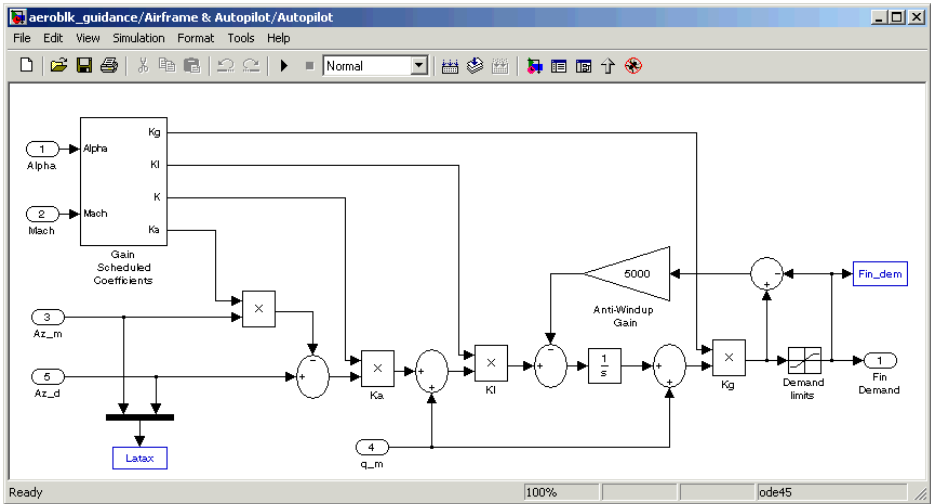
The model implements the missile environment, airframe, autopilot, and homing guidance system in subsystems.

- The Airframe & Autopilot subsystem implements the ISA Atmosphere Model block, the Incidence & Airspeed block, and the 3DoF (Body Axes) block, along with other Simulink blocks.

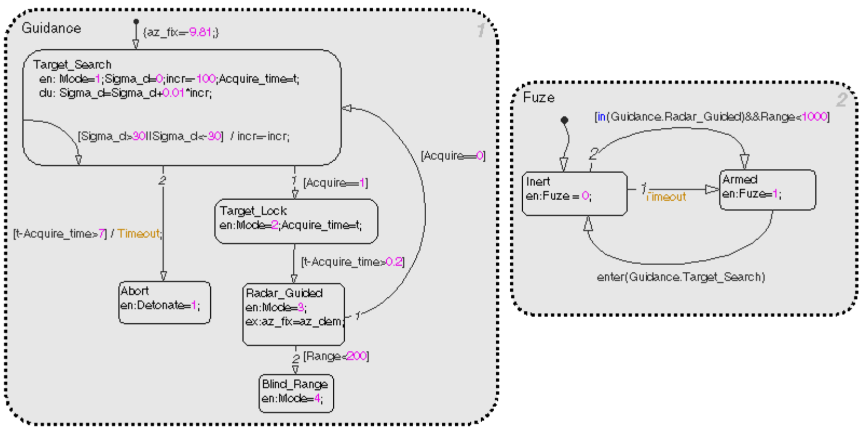
  The airframe model is a nonlinear representation of rigid body dynamics. The aerodynamic forces and moments acting on the missile body are generated from coefficients that are nonlinear functions of both incidence and Mach number.



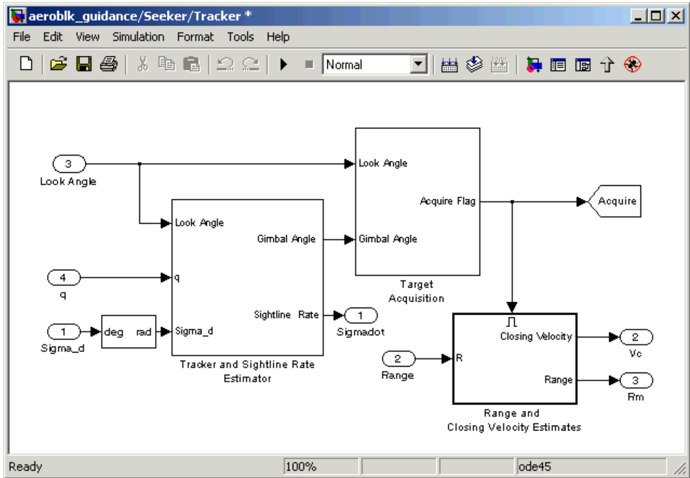

- The model implements the missile autopilot as a classical three-loop design using measurements from an accelerometer located ahead of the missile's center of gravity and from a rate gyro to provide additional damping.




- The model implements the homing guidance system as two subsystems: the Guidance subsystem and the Seeker/Tracker subsystem.
  - The Guidance subsystem uses a Stateflow state chart to control the tracker directly by sending demands to the seeker gimbals.




  - The Seeker/Tracker subsystem consists of Simulink blocks that control the seeker gimbals to keep the seeker dish aligned with the target and provide the guidance law with an estimate of the sight line rate.




## Running the Demo

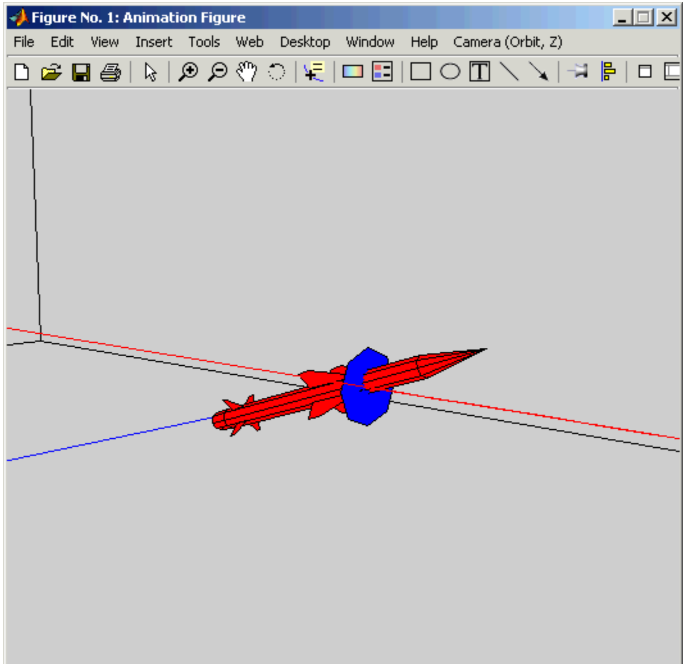
Running a demo lets you observe the model simulation in real time. After you run the demo, you can examine the resulting data in plots, graphs, and other visualization tools. To run the missile guidance model, follow these steps:

1 If it is not already open, open the `aeroblk_guidance` demo.

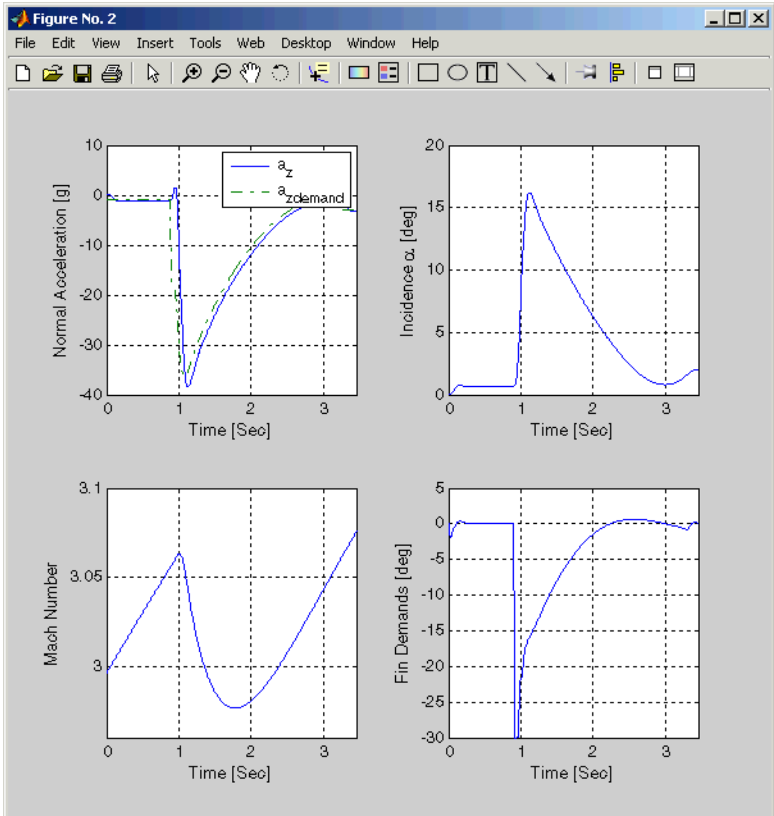
2 From the **Simulation** menu, select **Start**. In Windows, you can also click the start button in the model window toolbar.

   The simulation proceeds until the missile intercepts the target, which takes approximately 3 seconds. Once the interception has occurred, four scope figures open to display the following data:

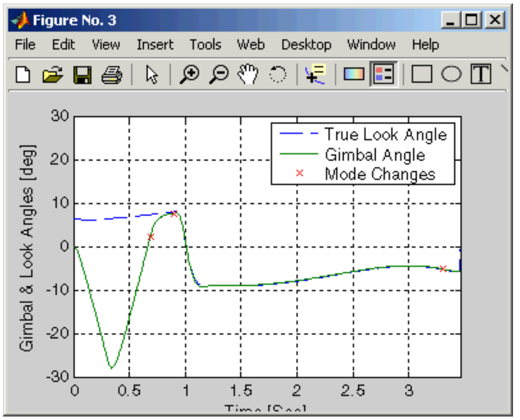
   a A three-dimensional animation of the missile and target interception course



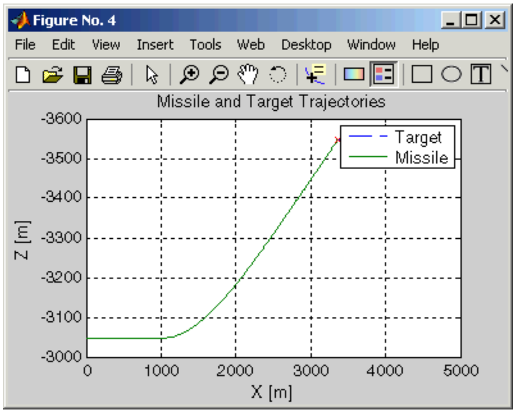

**b** Plots that measure flight parameters over time, including Mach number, fin demand, acceleration, and degree of incidence




**c** A plot that measures gimbal versus true look angles




**d** A plot that measures missile and target trajectories




## Modifying the Model

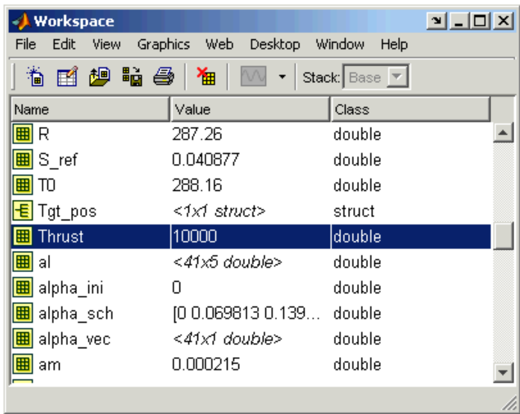
You can adjust the missile guidance model settings and examine the effects on simulation performance. Here are two modifications that you can try. The first modification adjusts the missile engine thrust (dynamic pressure). The second modification changes the camera point of view for the interception animation.

### Adjusting the Thrust

As in any Simulink model, you can adjust aerospace model parameters from the MATLAB workspace. To demonstrate this, change the `Thrust` variable in the model workspace and evaluate the results in the simulation.

1. Open the `aeroblk_guidance` model.

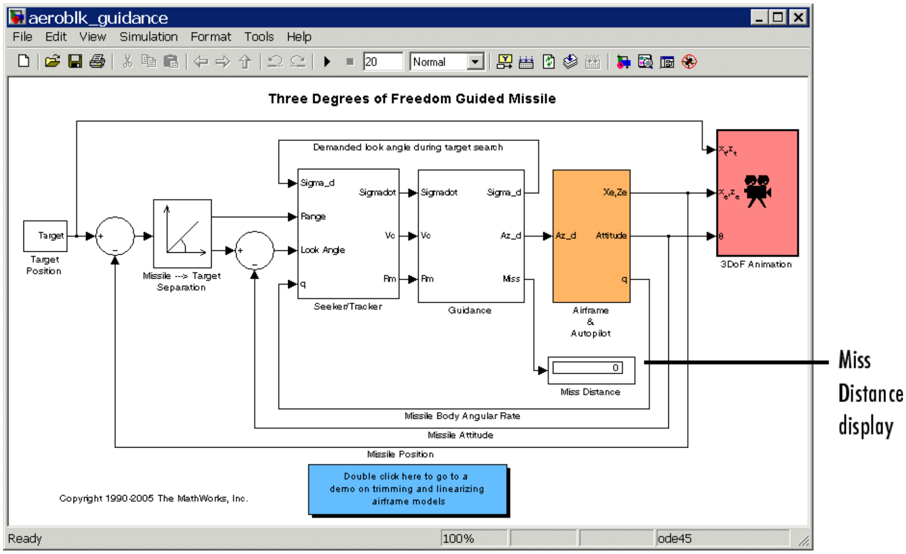
2. In the MATLAB desktop, find the `Thrust` variable in the **Workspace** pane.




   The `Thrust` variable is defined in the `aeroblk_guid_dat.m` file, which the `aeroblk_guidance` model uses to populate parameter and variable values. By default, the `Thrust` variable should be set to `10000`.

3. Single-click the `Thrust` variable to select it. To edit the value, right-click the `Thrust` variable and select **Edit Value**. Change the value to `5000`.

Before you run the demo again, locate the Miss Distance (Display) block display in the `aeroblk_guidance` model.

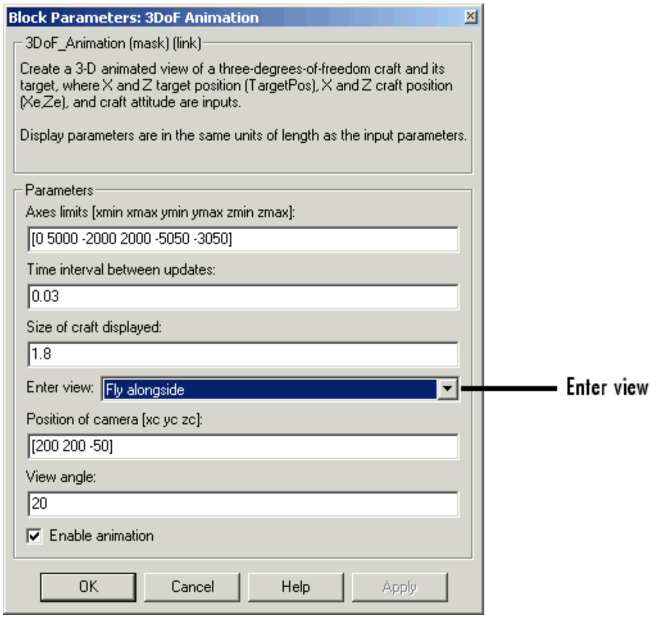



Start the demo, and after it finishes, note the miss distance display again. The miss distance should become greater than the original distance. You can experiment with different values in the `Thrust` variable and assess the effects on missile accuracy.

## Changing the Animation Point of View

By default, the missile animation view is `Fly Alongside`, which means the view tracks with the missile's flight path. You can easily change the animation point of view by adjusting a parameter of the 3DoF Animation block:

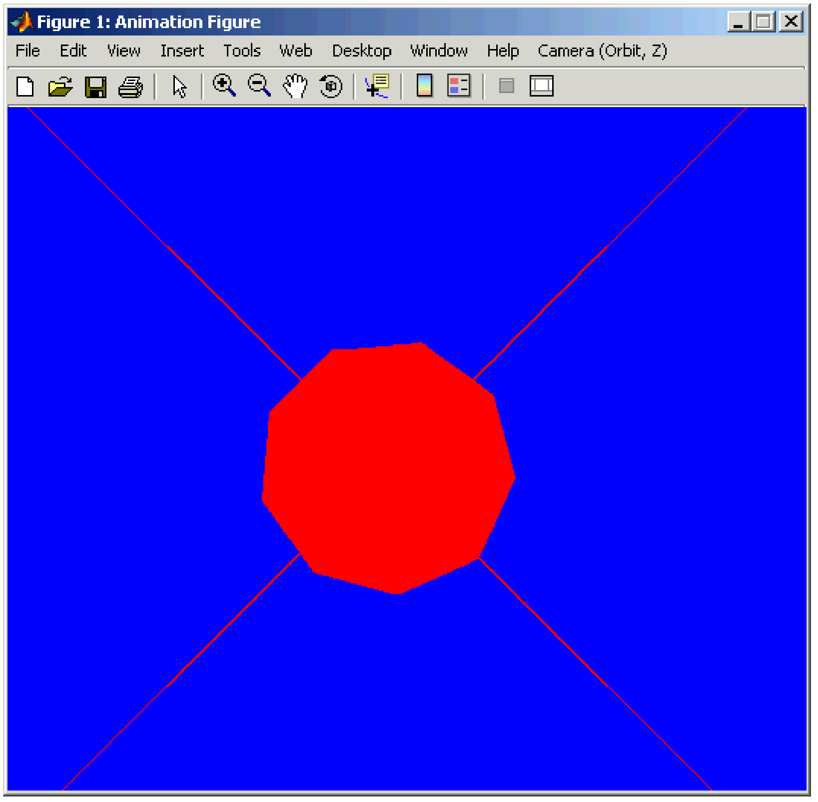
1. Open the `aeroblk_guidance` model, and double-click the 3DoF Animation block. The **Block Parameters** dialog box appears.




**2** Change the view to `Cockpit`.

**3** Click the **OK** button.

Run the demo again, and watch the animation. Instead of moving alongside the missile's flight path, the animation point of view lies in the cockpit. Upon target interception, the screen fills with blue, the target's color.




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

# Learning More

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

- "Using the MATLAB Help System for Documentation and Demos" on page 1-17
- "Finding Aerospace Blockset Help" on page 1-18

## Using the MATLAB Help System for Documentation and Demos

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

Consult the Help for Using MATLAB section of the MATLAB Desktop Tools and Development Environment documentation for more about the MATLAB help system.

### Opening Aerospace Demos

To open an Aerospace Blockset demo from the Help browser, open the Demos library in the Help browser by clicking the **Demos** tab in the **Help Navigator** pane on the left.

You can also open the Aerospace Blockset demos from the **Start** button of the MATLAB desktop:

1. Click the **Start** button.
2. Select **Blocksets**, then **Aerospace**, and then **Demos**.

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

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

## Finding Aerospace Blockset Help

This user's guide also includes a reference chapter.

- Appendix A, "Aerospace Units" explains the unit systems used by the blockset.

# Using Aerospace Blockset

Constructing a simple model with Aerospace Blockset is easy to learn if you know how to create Simulink models. If you are not familiar with Simulink, please see the Simulink documentation.

# Introducing the Aerospace Blockset Libraries

Aerospace Blockset is organized into hierarchical libraries of closely related blocks for use in Simulink. The following sections explain how to access the libraries from MATLAB and summarize the blocks in each library.

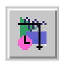


View the details for each block in Chapter 5, "Blocks — Alphabetical List".

## Opening Aerospace Blockset in Windows

You can open Aerospace Blockset from the Simulink Library Browser.

### Opening the Simulink Library Browser

To start Simulink, click the  button in the MATLAB toolbar, or enter

```
simulink
```

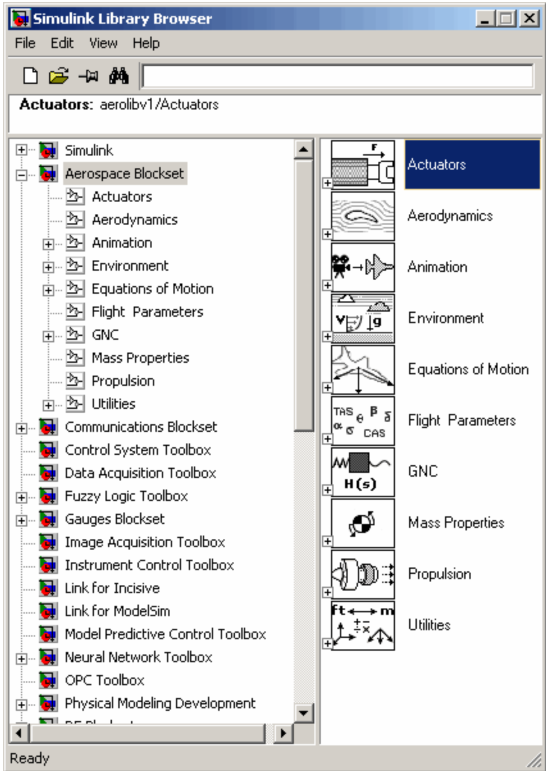
at the command line.

### Simulink Libraries

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

### Opening Aerospace Blockset

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

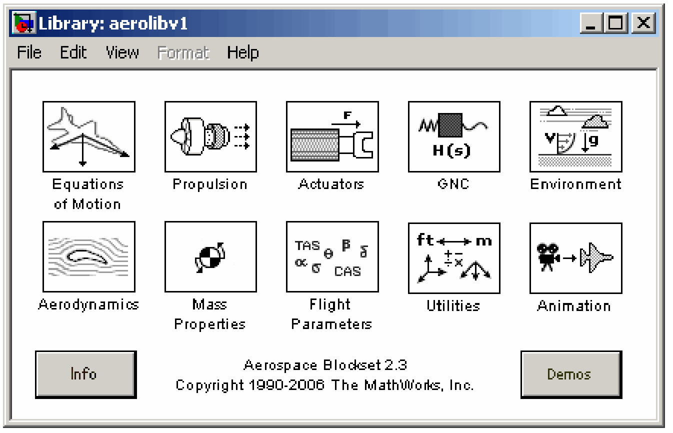



The first item in the list is Simulink itself, which is already expanded to show the available Simulink libraries. Click the ⊞ symbol to the left of any blockset name to expand the hierarchical list and display that blockset's libraries within the browser.

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

```
aerolib
```

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




For a complete list of all the blocks in Aerospace Blockset by library, see "Summary of Aerospace Block Libraries" on page 2-5.

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

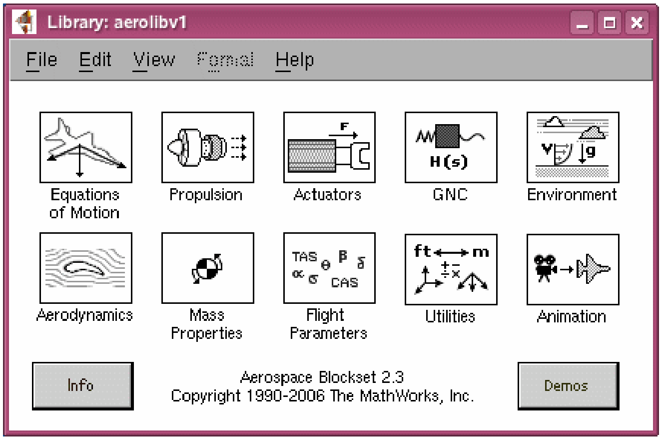
## Opening Aerospace Blockset on UNIX Platforms

On UNIX platforms, the Simulink Library window opens when you start Simulink. To open Aerospace Blockset, double-click the **Aerospace Blockset** icon to open Aerospace Blockset.

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

```
aerolib
```

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




For a complete list of all the blocks in Aerospace Blockset by library, see "Summary of Aerospace Block Libraries" on page 2-5.

## Summary of Aerospace Block Libraries

The blocks of Aerospace Blockset are organized into these libraries.

### Actuators Library

The Actuators library provides blocks for representing linear and nonlinear actuators with saturation and rate limits.

### Aerodynamics Library

The Aerodynamics library provides the Aerodynamic Forces and Moments block using the aerodynamic coefficients, dynamic pressure, center of gravity, and center of pressure.

### Animation Library

The Animation library provides the animation blocks for visualizing flight paths and trajectories and for working with a flight simulator interface. The Animation library contains the MATLAB-Based Animation, Flight Simulator Interfaces, and Animation Support Utilities sublibraries.

**MATLAB-Based Animation Sublibrary.** The MATLAB-Based Animation sublibrary provides the 3DoF Animation block and the 6DoF Animation block. Using the animation blocks, you can visualize flight paths and trajectories.

**Flight Simulator Interfaces Sublibrary.** The Flight Simulator Interfaces sublibrary provides the interface blocks to connect Aerospace Blockset to the third-party FlightGear flight simulator.

**Animation Support Utilities Sublibrary.** The Animation Support Utilities sublibrary provides additional blocks for running the FlightGear flight simulator. It contains a joystick interface for Windows platform and a block that lets you set the simulation pace.

### Environment Library

The Environment library provides blocks that simulate aspects of an aircraft and spacecraft environment, such as atmospheric conditions, gravity, magnetic fields, and wind. The Environment library contains the Atmosphere, Gravity, and Wind sublibraries.

**Atmosphere Sublibrary.** The Atmosphere sublibrary provides general atmospheric models, such as ISA and COESA, and other blocks, including nonstandard day simulations, lapse rate atmosphere, and pressure altitude.

**Gravity Sublibrary.** The Gravity sublibrary provides blocks that calculate the gravity and magnetic fields for any point on the Earth.

**Wind Sublibrary.** The Wind sublibrary provides blocks for wind-related simulations, including turbulence, gust, shear, and horizontal wind.

### Equations of Motion Library

The Equations of Motion library provides blocks for implementing the equations of motion to determine body position, velocity, attitude, and related values. The Equations of Motion library contains the 3DoF, 6DoF, and Point Mass sublibraries.

**3DoF Sublibrary.** The 3DoF sublibrary provides blocks for implementing three-degrees-of-freedom equations of motion in your simulations, including custom variable mass models.

**6DoF Sublibrary.** The 6DoF sublibrary provides blocks for implementing six-degrees-of-freedom equations of motion in your simulations, using Euler angles and quaternion representations.

**Point Mass Sublibrary.** The Point Mass sublibrary provides blocks for implementing point mass equations of motion in your simulations.

### Flight Parameters Library

The Flight Parameters library provides blocks for various parameters, including ideal airspeed correction, Mach number, and dynamic pressure.

### GNC Library

The GNC library provides blocks for creating control and guidance systems, including various controller models. The GNC library contains the Control, Guidance, and Navigation sublibraries.

**Control Sublibrary.** The Control sublibrary provides blocks for simulating various control types, such as one-dimensional, two-dimensional, and three-dimensional models.

**Guidance Sublibrary.** The Guidance sublibrary provides the Calculate Range block, which computes the range between two vehicles.

**Navigation Sublibrary.** The Navigation sublibrary provides blocks for three-axis measurement of accelerations, angular rates, and inertias.

### Mass Properties Library

The Mass Properties library provides blocks for simulating the center of gravity and inertia tensors.

### Propulsion Library

The Propulsion library provides the Turbofan Engine System block, which simulates an engine system and controller.

### Utilities Library

The Utilities library contains miscellaneous blocks useful in building models. The library contains the Axes Transformations, Math Operations, and Unit Conversions sublibraries.

**Axes Transformations Sublibrary.** The Axes Transformations sublibrary provides blocks for transforming axes of coordinate systems to different types, such as Euler angles to quaternions and vice versa.

**Math Operations Sublibrary.** The Math Operations sublibrary provides blocks for common mathematical and matrix operations, including sine and cosine generation and various 3-by-3 matrix operations.

**Unit Conversions Sublibrary.** The Unit Conversions sublibrary provides blocks for converting common measurement units from one system to another, such as converting velocity from feet per second to meters per second and vice versa.

# Creating Aerospace Models

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. For general model-building rules, see the Simulink documentation.

1. *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 Aerospace Blockset, 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 the 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.

2. *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.

3 *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.

4 *Encapsulate subsystems.* Systems made with Aerospace Blockset can function as subsystems of larger, more complex models, like subsystems in any Simulink model.

## Model Referencing Limitations

The Model block allows you to include a model as a block in another model. If you include a model that contains continuous blocks from Aerospace Blockset, the referenced model containing the Aerospace Blockset blocks will not inherit its sample time from the parent model of the Model block. The referenced model will have intrinsic sample times. See "Model Block Sample Times" in the Simulink user's guide documentation about this limitation. If you want to use Real-Time Workshop® to generate code, do not use the following Aerospace Blockset blocks in referenced models. Because they are noninlined S-functions, Real-Time Workshop cannot generate standalone executables (Real-Time Workshop targets) for referenced models that include these blocks:

- WGS84 Gravity Model
- COESA Atmosphere Model
- Non-Standard Day 210C, Non-Standard Day 310
- Pressure Altitude

If you are only interested in simulation, see "Model Referencing Limitations" in the Simulink user's guide documentation for additional limitations for simulating noninlined functions.

# Building a Simple Actuator System

In this tutorial, you drag, drop, and configure a some basic blocks to drive, simulate, and measure an aerospace actuator. The tutorial guides you through these aspects of model building:

- "Building the Model" on page 2-11
- "Running the Simulation" on page 2-19

By the end of the tutorial, you will have constructed a simple actuator model that measures the actuator's position in relation to a sine wave.

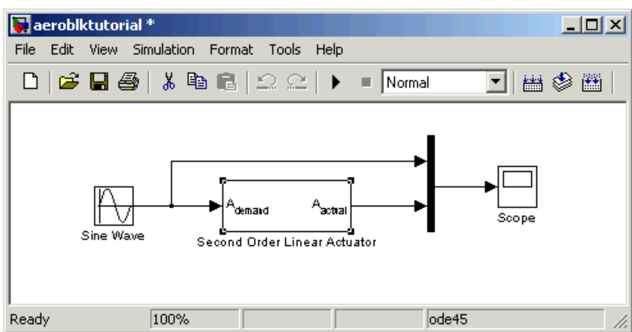
## Building the Model

Simulink 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.

---



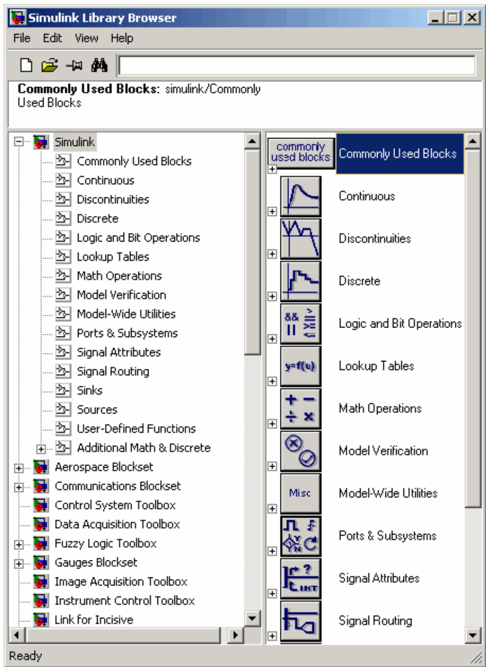

The following sections explain how to build a model on Windows and UNIX platforms:

- "Creating a Model on Windows Platforms" on page 2-12

- “Creating a Model on UNIX Platforms” on page 2-15

## Creating a Model on Windows Platforms

**1** Click the  button in the MATLAB toolbar or enter `simulink` at the MATLAB command line. The Simulink library browser appears.




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

**3** Add a Sine Wave block to the model.

   **a** Click **Sources** in the Library Browser to view the blocks in the Simulink Sources library.

   **b** Drag the Sine Wave block from the Sources library into the new model window.

**4** Add a Second Order Linear Actuator 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 Second Order Linear Actuator block into the model window.

**5** Add a Mux block to the model.

  **a** Click **Signal Routing** in the Library Browser to view the blocks in the Simulink Signals & Systems library.

  **b** Drag the Mux block from the Signal Routing library into the model window.

**6** Add a Scope block to the model.

  **a** Click **Sinks** in the Library Browser to view the blocks in the Simulink Sinks library.

  **b** Drag the Scope block from the Sinks library into the model window.

**7** Resize the Mux block in the model.

  **a** Click the Mux block to select the block.

  **b** Hold down the mouse button and drag a corner of the Mux block to change the size of the block.

**8** Connect the blocks.

  **a** Position the pointer near the output port of the Sine Wave block. Hold down the mouse button and drag the line that appears until it touches the input port of the Second Order Linear Actuator block. Release the mouse button.

  **b** Using the same technique, connect the output of the Second Order Linear Actuator block to the second input port of the Mux block.

  **c** Using the same technique, connect the output of the Mux block to the input port of the Scope block.

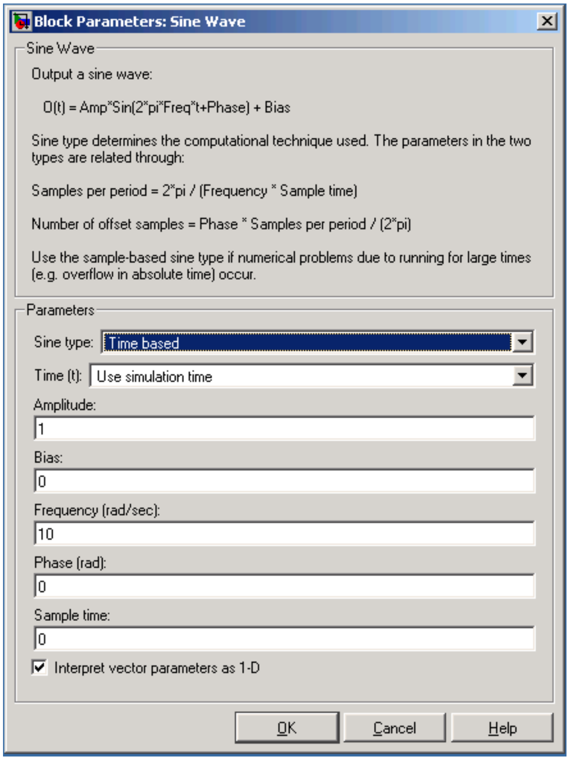
  **d** Position the pointer near the first input port of the Mux block. Hold down the mouse button and drag the line that appears over the line from the output port of the Sine Wave block until double crosshairs appear. Release the mouse button. The lines are connected when a knot is present at their intersection.

9 Set the block parameters.

   a Double-click the Sine Wave block. The dialog box that appears allows you to set the block's parameters.

   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.

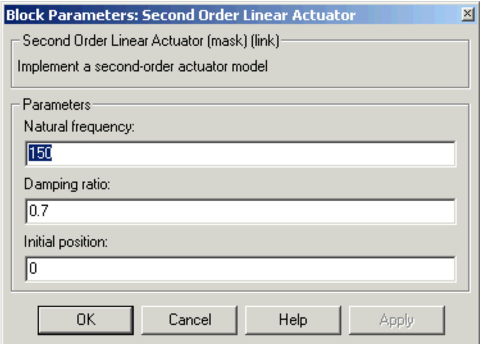
   b Click **OK**.




   c Double-click the Second Order Linear 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.

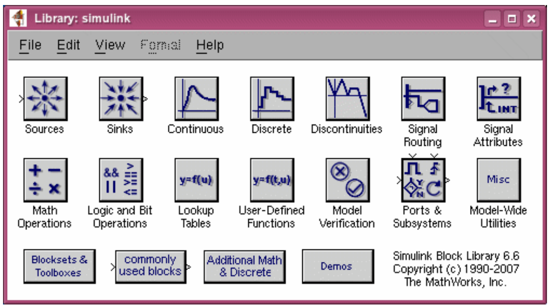
**d** Click **OK.**




## Creating a Model on UNIX Platforms

The steps for creating a model in UNIX are similar to the steps in Windows.

**1** Enter `simulink` at the MATLAB command line. The Simulink library window appears.




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

**3** Add a Sine Wave block to the model.

a. Double-click **Sources** in the Simulink Library window to view the blocks in the Simulink Sources library.

b. Drag the Sine Wave block from the Sources library into the new model window.

4. Add a Second Order Linear Actuator block to the model.

   a. Double-click **Aerospace Blockset** in the Simulink Library browser. This opens the Aerospace Blockset block libraries.

   b. In the Aerospace Blockset block libraries, click **Actuators** to view the blocks in the Actuator library.

   c. Drag the Second Order Linear Actuator block into the model window.

5. Add a Mux block to the model.

   a. Double-click **Signal Routing** in the Simulink Library to view the Signal Routing blocks.

   b. Drag the Mux block from the Signal Routing library into the model window.

6. Add a Scope block to the model.

   a. Double-click **Sinks** in the Simulink Library window to view the blocks in the Simulink Sinks library.

   b. Drag the Scope block from the Sinks library into the model window.

7. Resize the Mux block in the model.

   a. Click the Mux block to select the block.

   b. Hold down the mouse button and drag a corner of the Mux block to change the size of the block.

8. Connect the blocks.

   a. Position the pointer near the output port of the Sine Wave block. Hold down the mouse button and drag the line that appears until it touches the input port of the Second Order Linear Actuator block. Release the mouse button.

   b. Using the same technique, connect the output of the Second Order Linear Actuator block to the second input port of the Mux block.

**c** Using the same technique, connect the output of the Mux block to the input port of the Scope block.

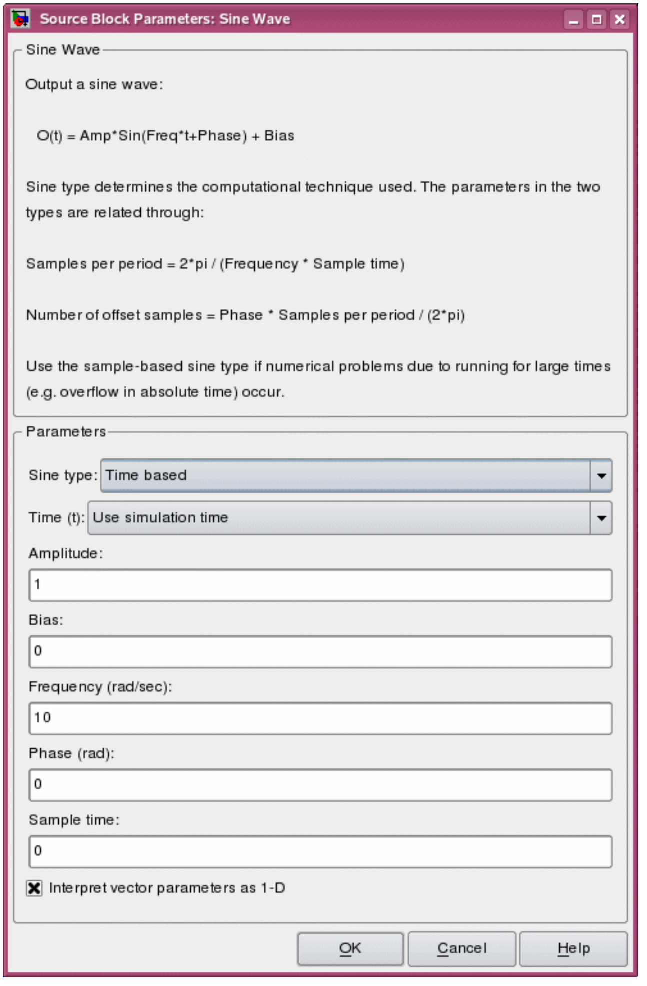
**d** Position the pointer near the first input port of the Mux block. Hold down the mouse button and drag the line that appears over the line from the output port of the Sine Wave block until double crosshairs appear. Release the mouse button. The lines are connected when a knot is present at their intersection.

**9** Set the block parameters.

**a** Double-click the Sine Wave block. The dialog box that appears allows you to set the block's parameters.

In 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.

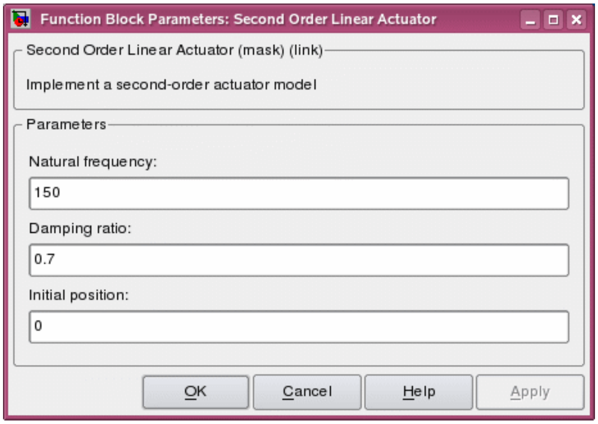
**b** Click **OK.**




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

For 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.**




## Running 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 **Start** 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. This causes the simulation to pause.

   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 and allow the simulation to continue. 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.

---

**Note** Opening a dialog box for a source block causes the simulation to pause. While the simulation is paused, you can edit the parameter values. You must close the dialog box to have the changes take effect and allow the simulation to continue.

---

### Running a Simulation from an M-File

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

# About Aerospace Coordinate Systems

Coordinate systems allow you to keep track of an aircraft or spacecraft's position and orientation in space. This section introduces important terminology and the major coordinate systems used by Aerospace Blockset.


- "Fundamental Coordinate System Concepts" on page 2-21
- "Coordinate Systems for Modeling" on page 2-23
- "Coordinate Systems for Navigation" on page 2-25
- "Coordinate Systems for Display" on page 2-28


The "References" on page 2-29 point you to further information.

## Fundamental Coordinate System Concepts

The Aerospace Blockset coordinate systems are based on these underlying concepts from geodesy, astronomy, and physics.

### Definitions

Aerospace 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: force = mass•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's 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's surface. The geographic poles are the intersection of the Earth's surface and the geopolar axis. In general, the Earth's 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

Aerospace Blockset makes three standard approximations in defining coordinate systems relative to the Earth.

- The Earth's 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's 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's rotation about its axis. This is a *rotating, geocentric* system. The blockset ignores the Earth's acceleration around the Sun, the Sun's acceleration in the Galaxy, and the Galaxy's acceleration through cosmos. In most applications, only the Earth's rotation matters.

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

### Motion with Respect to Other Planets

Aerospace 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's motion through the air is not the same as its motion relative to the ground.

See the "Equations of Motion" on page 4-6 for further details on how Aerospace 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.

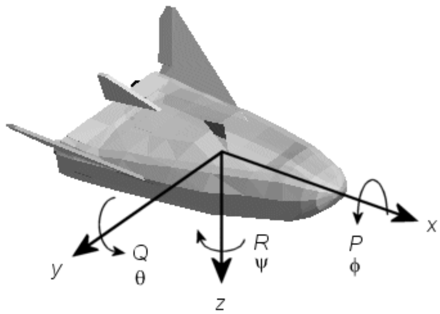
The orientation of the body coordinate axes is fixed in the shape of body.

- 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 |




### 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's velocity ***V***.

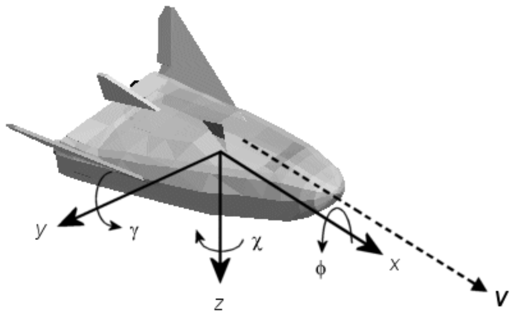
The orientation of the wind coordinate axes is fixed by the velocity ***V***.

- The *x*-axis points in the direction of ***V***.
- The *y*-axis points to the right of the *x*-axis (facing in the direction of ***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 Φ, γ, χ. They are:

| | |
|---|---|
| Φ | Bank angle about the *x*-axis |
| γ | Flight path about the *y*-axis |
| χ | Heading angle about the *z*-axis |



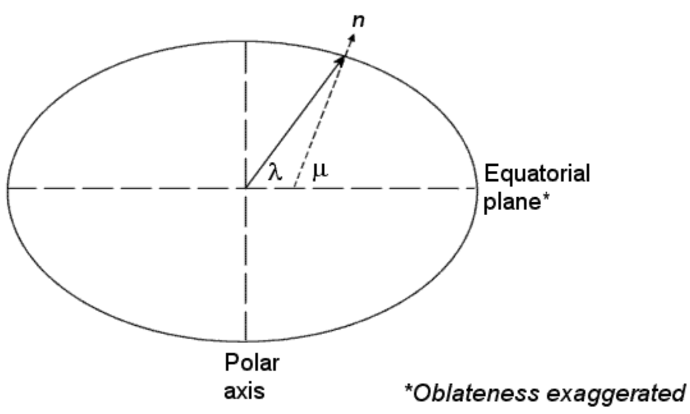

## Coordinate Systems for Navigation

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* λ on the Earth's surface is defined by the angle subtended by the radius vector from the Earth's center to the surface point with the equatorial plane.

The *geodetic latitude* μ on the Earth's surface is defined by the angle subtended by the surface normal vector n and the equatorial plane.

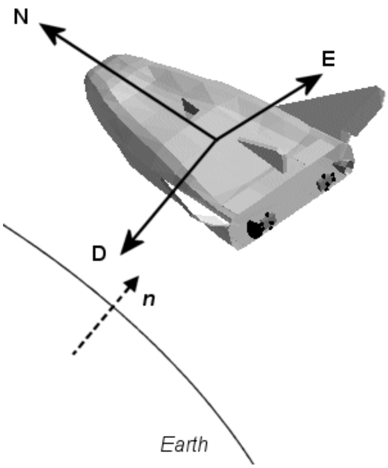


### NED Coordinates

The north-east-down (NED) system is a noninertial system with its origin fixed at the aircraft or spacecraft's center of gravity. Its axes are oriented along the geodetic directions defined by the Earth's 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's surface, antiparallel to the surface's outward normal $\boldsymbol{n}$.

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

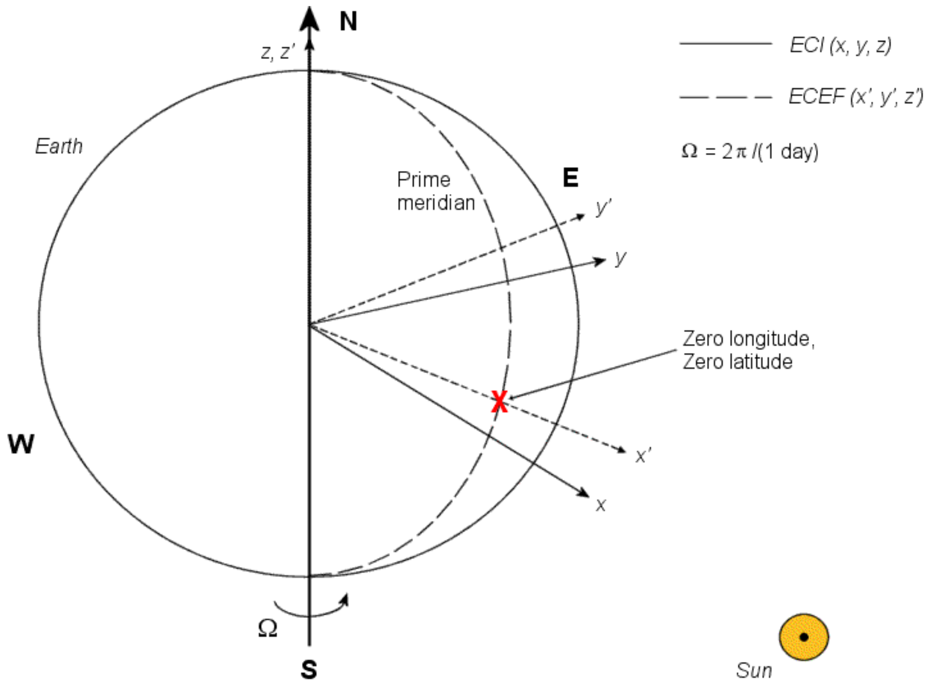



### ECI Coordinates

The Earth-centered inertial (ECI) system is a mixed inertial system. It is oriented with respect to the Sun. Its origin is fixed at the center of the Earth. (See figure following.)

- The $z$-axis points northward along the Earth's rotation axis.
- The $x$-axis points outward in the Earth's equatorial plane exactly at the Sun. (This rule ignores the Sun's oblique angle to the equator, which varies with season. The actual Sun always remains in the $xz$ plane.)

- The $y$-axis points into the eastward quadrant, perpendicular to the $xz$ plane so as to satisfy the RH rule.




**Earth-Centered Coordinates**

## ECEF Coordinates

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

- The $z'$-axis points northward along the Earth's rotation axis.
- The $x'$-axis points outward along the intersection of the Earth's equatorial plane and prime meridian.
- The $y'$-axis points into the eastward quadrant, perpendicular to the $x$-$z$ plane so as to satisfy the RH rule.

# Coordinate Systems for Display

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

## MATLAB Graphics Coordinates

See the MATLAB 3-D Visualization documentation 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.

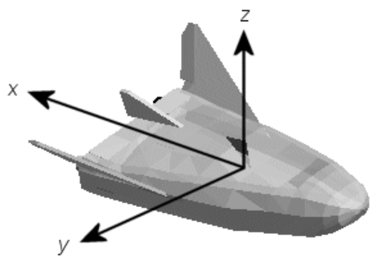
## FlightGear Coordinates

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

- "Working with the Flight Simulator Interface" on page 2-36 discusses the blockset interface to FlightGear.
- See the FlightGear documentation at `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:

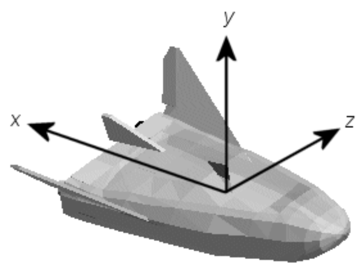
- The $x$-axis is positive toward the back of the vehicle.
- The $y$-axis is positive toward the right of the vehicle.
- The $z$-axis is positive upward, e.g., wheels typically have the lowest $z$ values.




### AC3D Coordinates

AC3D is a low-cost, widely used, geometry editor available from `www.ac3d.org`. 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

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

*Mapping Toolbox User's Guide,* The MathWorks, Inc., Natick, Massachusetts. `www.mathworks.com/access/helpdesk/help/toolbox/map/`.

Rogers, R. M., *Applied Mathematics in Integrated Navigation Systems,* AIAA, Reston, Virginia, 2000.

Sobel, D., *Longitude*, Walker & Company, New York, 1995.

Stevens, B. L., and F. L. Lewis, *Aircraft Control and Simulation,* 2nd ed., *Aircraft Control and Simulation,* Wiley-Interscience, New York, 2003.

Thomson, W. T., *Introduction to Space Dynamics,* John Wiley & Sons, New York, 1961/Dover Publications, Mineola, New York, 1986.

World Geodetic System 1984 (WGS 84), `http://earth-info.nga.mil/GandG/wgs84/`.

# Introducing the Flight Simulator Interface

Aerospace Blockset supports an interface to the third-party FlightGear flight simulator, an open source software package available through a GNU General Public License (GPL).



## About the FlightGear Interface

The FlightGear flight simulator interface included with Aerospace Blockset is a unidirectional transmission link from Simulink to FlightGear using FlightGear's published `net_fdm` binary data exchange protocol. Data is transmitted via UDP network packets to a running instance of FlightGear. Aerospace Blockset supports multiple standard binary distributions of FlightGear. See "Running FlightGear with Simulink" on page 2-41 for interface details.

FlightGear is a separate software entity neither created, owned, nor maintained by The MathWorks.

- To report bugs in or request enhancements to the Aerospace Blockset FlightGear interface, use the MathWorks contact information provided at the front of the PDF version of this User's Guide.
- To report bugs or request enhancements to FlightGear itself, visit `www.flightgear.org` and use the contact page.

## Obtaining FlightGear

You can obtain FlightGear from `www.flightgear.org` 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 customization and porting to custom environments.

## Configuring Your Computer for FlightGear

You must have a high performance graphics card with stable drivers to use FlightGear. For more information, see the FlightGear CD distribution or the hardware requirements and documentation areas of the FlightGear Web site, `www.flightgear.org`.

MathWorks tests of FlightGear's performance and stability indicate significant sensitivity to computer video cards, driver versions, and driver settings. You need OpenGL support with hardware acceleration activated. The OpenGL settings are particularly important. Without proper setup, performance can drop from about a 30 frames-per-second (fps) update rate to less than 1 fps.

### Graphics Recommendations for Windows

The MathWorks recommends the following for Windows users:

- Choose a graphics card with good OpenGL performance.
- Always use the latest tested and stable driver release for your video card. Test the driver thoroughly on a few computers before deploying to others.

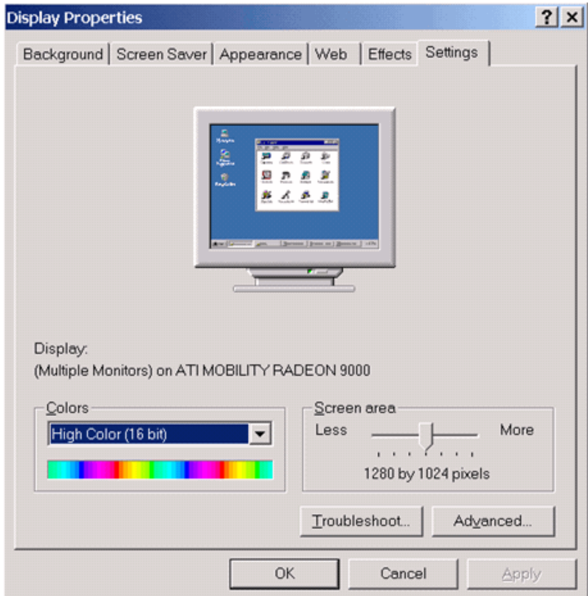
  For Microsoft Windows 2000 or XP systems running on x86 (32-bit) or AMD-64/EM64T chip architectures, the graphics card operates in the unprotected kernel space known as Ring Zero. This means that glitches in the driver can cause Windows to lock or crash. Before buying a large number of computers for 3-D applications, test, with your vendor, one or two computers to find a combination of hardware, operating system, drivers, and settings that are stable for your applications.

### Setting Up OpenGL Graphics on Windows

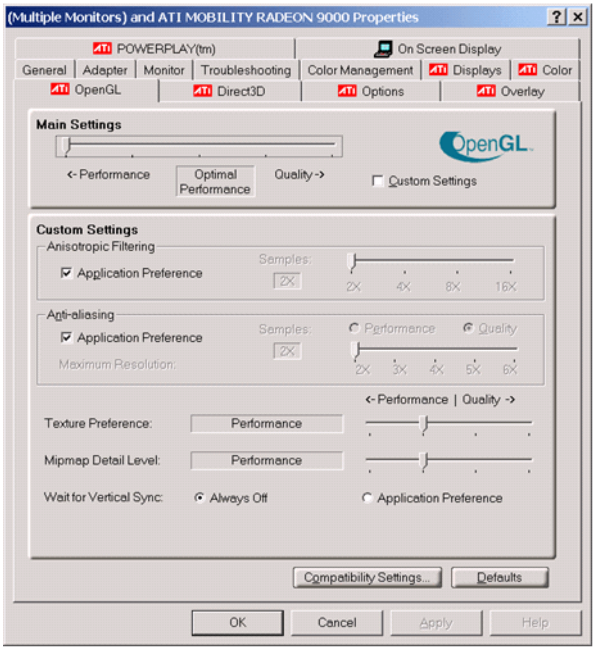
For complete information on OpenGL settings, refer to the documentation at the OpenGL Web site: `www.opengl.org`.

Follow these steps to optimize your video card settings. Your driver's panes might look different.

1. Ensure that you have activated the OpenGL hardware acceleration on your video card. On Windows, access this configuration through **Start > Settings > Control Panel > Display**, which opens the following dialog box. Select the **Settings** tab.




**2** Click the **Advanced** button in the lower right of the dialog box, which brings up the graphics card's custom configuration dialog box, and go to the **OpenGL** tab. For an ATI Mobility Radeon 9000 video card, the **OpenGL** pane looks like this:




**3** For best performance, move the **Main Settings** slider near the top of the dialog box to the **Performance** end of the slider.

**4** If stability is a problem, try other screen resolutions, other color depths in the **Displays** pane, and other OpenGL acceleration modes.

Many cards perform much better at 16 bits-per-pixel color depth (also known as 65536 color mode, 16-bit color). For example, on an ATI Mobility Radeon 9000 running a given model, 30 fps are achieved in 16-bit color mode, while 2 fps are achieved in 32-bit color mode.

## Setup on Linux, Macintosh, and Other Platforms

FlightGear distributions are available for Linux, Macintosh, and other UNIX platforms from the FlightGear Web site, `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 Web site.

### Using MATLAB Graphics Controls to Configure Your OpenGL Settings

You can also control your OpenGL rendering from the MATLAB command line with the MATLAB Graphics `opengl` command. Consult the `opengl` command reference for more information.

## Installing and Starting FlightGear

The extensive FlightGear documentation guides you through the installation in detail. Consult the documentation section of the FlightGear Web site for complete installation instructions: `www.flightgear.org`.

Keep the following points in mind:

- Generous central processor speed, system and video RAM, and virtual memory are essential for good flight simulator performance.

  The MathWorks recommends a minimum of 512 megabytes of system RAM and 128 megabytes of video RAM for reasonable performance.

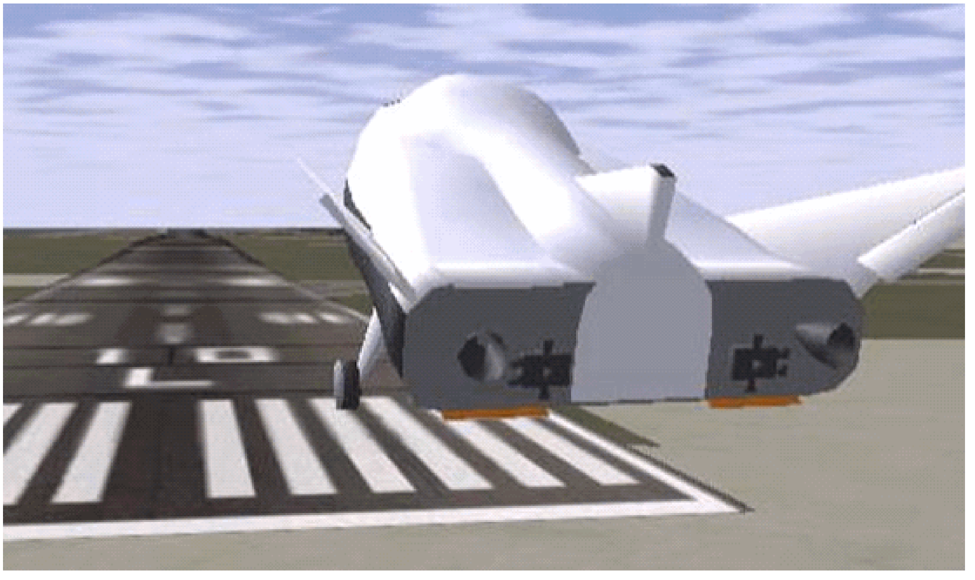
- Be sure to have sufficient disk space for the FlightGear download and installation.
- The MathWorks recommends configuring your computer's graphics card before you install FlightGear. See the preceding section, "Configuring Your Computer for FlightGear" on page 2-32.
- Shutting down all running applications (including MATLAB) before installing FlightGear is recommended.
- MathWorks tests indicate that the operational stability of FlightGear is especially sensitive during startup. It is best to not move, resize, mouse over, overlap, or cover up the FlightGear window until the initial simulation scene appears after the startup splash screen fades out.
- The current releases of FlightGear are optimized for flight visualization at altitudes below 100,000 feet. FlightGear does work not well or at all with very high altitude and orbital views.

# Working with the Flight Simulator Interface

Use this section to learn how to use the FlightGear flight simulator and Aerospace Blockset to visualize your Simulink aircraft models:


- "About Aircraft Geometry Models" on page 2-36
- "Working with Aircraft Geometry Models" on page 2-39
- "Running FlightGear with Simulink" on page 2-41
- "Running the NASA HL-20 Demo with FlightGear" on page 2-51


If you have not yet installed FlightGear, see "Introducing the Flight Simulator Interface" on page 2-31.



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

## About Aircraft Geometry Models

Before you can visualize your aircraft's 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`. Another popular 3-D editor for aircraft models is Flight Sim Design Studio, distributed by Abacus Publications at `www.abacuspub.com`.

### Aircraft Model Structure and Requirements

Aircraft models live in the *`FlightGearRoot`*`/data/Aircraft/` directory and subdirectories. A complete aircraft model must contain a directory 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 directory:

- 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://www.flightgear.org/Docs/fgfs-model-howto.html
```

### Required Flight Dynamics Model Specification

The flight dynamics model (FDM) specification is a required element for an aircraft model. To set Simulink 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>
```

### Obtaining and Modifying 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 directory 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 `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.

## Working with Aircraft Geometry Models

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

### Importing Aircraft Models into FlightGear

To install a compatible model into FlightGear:

1. Go to your installed FlightGear directory. Open the `data` directory, then the `Aircraft` directory: `/FlightGear/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/`).

### Example: Animating 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 "Running the NASA HL-20 Demo with FlightGear" on page 2-51.

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 2-21 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 demo 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>
```

## Running FlightGear with Simulink

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 Simulink:

- "Setting the Flight Dynamics Model to Network in the Aircraft Data File" on page 2-42 explains how to create the network connection you need.
- "Obtaining the Destination IP Address" on page 2-42 starts by determining the IP address of the computer running FlightGear.
- "Adding and Connecting Interface Blocks" on page 2-43 shows how to add and connect interface and pace blocks to your Simulink model.
- "Creating a FlightGear Run Script" on page 2-46 shows how to write a FlightGear run script compatible with your Simulink model.
- "Starting FlightGear" on page 2-48 guides you through the final steps to making Simulink work with FlightGear.
- "Improving Performance" on page 2-50 helps you speed your model up.

- "Running FlightGear and Simulink on Different Computers" on page 2-50 explains how to connect a simulation from Simulink running on one computer to FlightGear running on another computer.

### Setting 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>
```

### Obtaining 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's name, enter at the MATLAB command line:

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

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

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

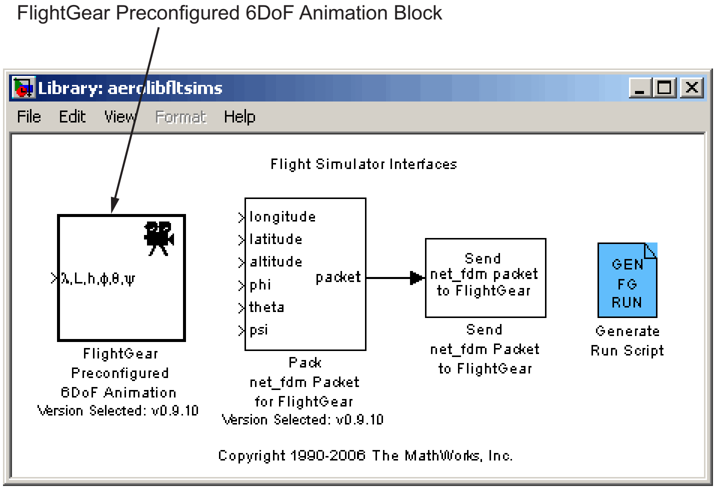
- If you are working in Windows, get your computer's IP address by entering at the DOS prompt:

```
ipconfig /all
```

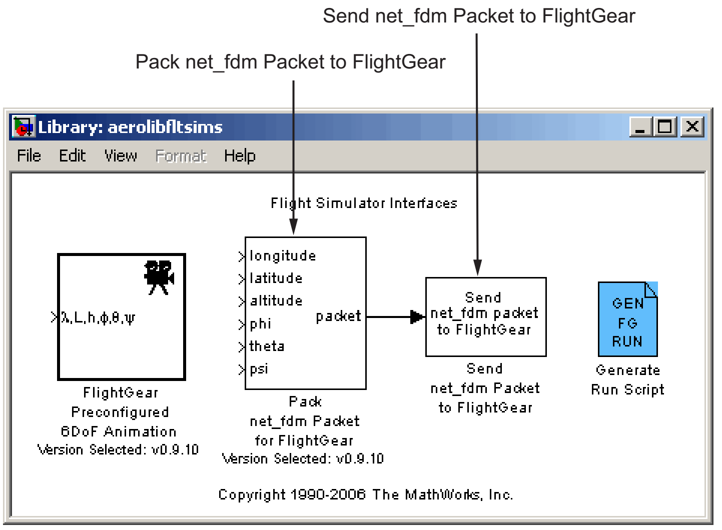
  Examine the IP address entry in the resulting output. There is one entry per Ethernet device.

## Adding and Connecting Interface Blocks

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

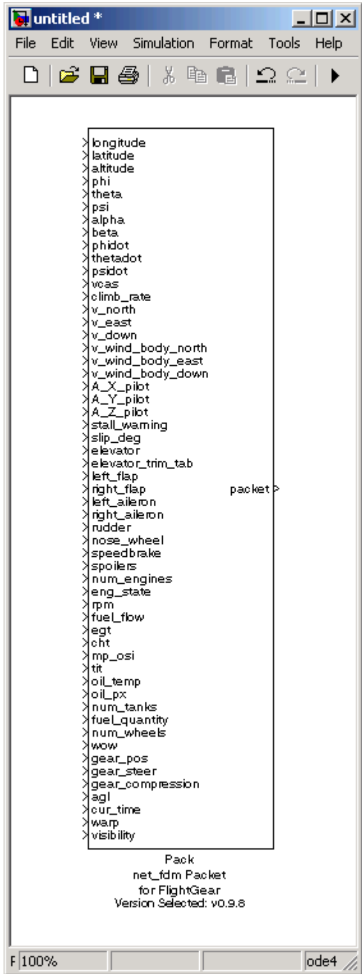



The FlightGear Preconfigured 6DoF Animation block is a subsystem containing the Pack net_fdm Packet for FlightGear and Send net_fdm Packet to FlightGear blocks:

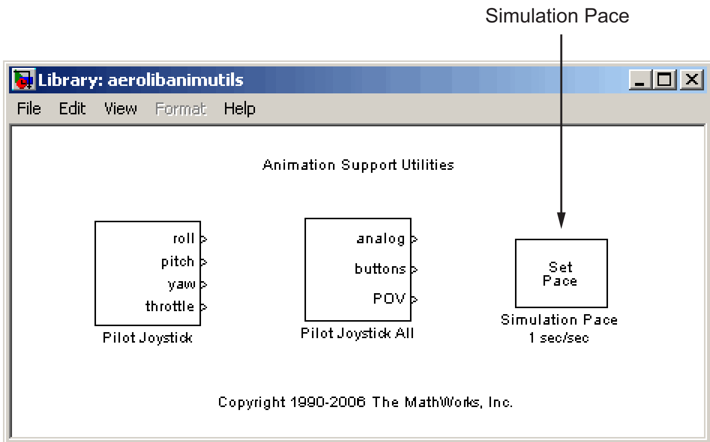



These transmit data to a FlightGear session. The blocks are separate for maximum flexibility and compatibility.

- The Pack net_fdm Packet for FlightGear block formats a binary structure compatible with FlightGear from model inputs. In its default configuration, only the 6DoF ports are shown, 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 “Obtaining the Destination IP Address” on page 2-42.
- The Simulation Pace block, available in the “Animation Support Utilities Sublibrary” on page 2-6, slows down 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.



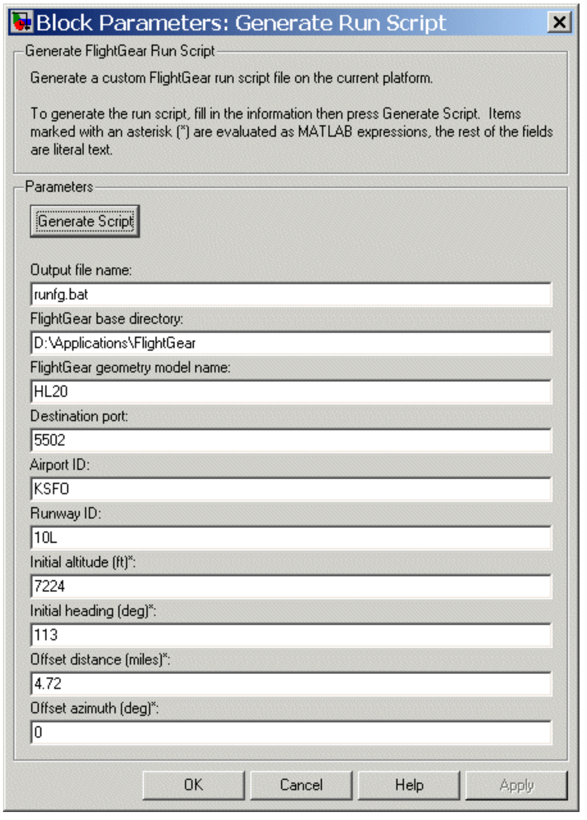

## Creating a FlightGear Run Script

To start FlightGear with the desired initial conditions (location, date, time, weather, operating modes), it is best to create a run script by using the Generate Run Script block or the interface included in FlightGear.

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 MATLAB just before starting your Simulink model.

**Using 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" on page 2-6.
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.




**5** In the **Output file name** field, type the name of the output file. This name should be the name of the command, with the `.bat` extension, you want to use to start FlightGear with these initial parameters.

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

**6** In the **FlightGear base directory** field, specify the name of your FlightGear installation directory.

**7** In the **FlightGear geometry model name** field, specify the name of the subdirectory, in the *`FlightGear`*`/data/Aircraft` directory, 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.

   Aerospace Blockset generates the run script, and saves it in your MATLAB working directory under the filename that you specified in the **Output file name** field.

10 Repeat steps 5 through 9 to generate other run scripts, if needed.

11 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 directory. This is an example of the contents of a run script file:

```
>> cd D:\Applications\FlightGear-0.9.8a
>> SET FG_ROOT=D:\Applications\FlightGear-0.9.8a\data
>> cd \bin\Win32\
>> 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-id=KSFO --runway=10L
--altitude=7224 --heading=113 --offset-distance=4.72 --offset-azimuth=0
```

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

### Starting FlightGear

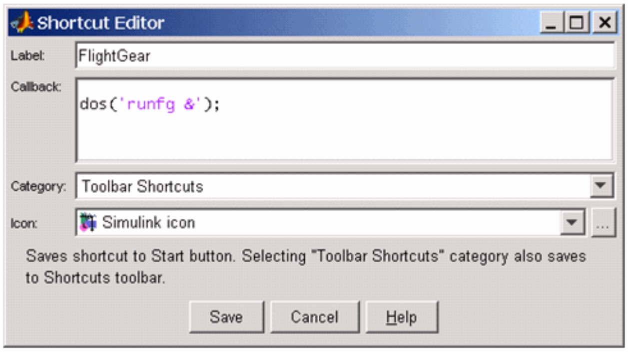
If your computer has enough computational power to run both Simulink and FlightGear at the same time, a simple way to start FlightGear is to create a MATLAB desktop button containing the following command to execute a run script like the one created above:

```
dos('runfg &')
```

To create a desktop button:

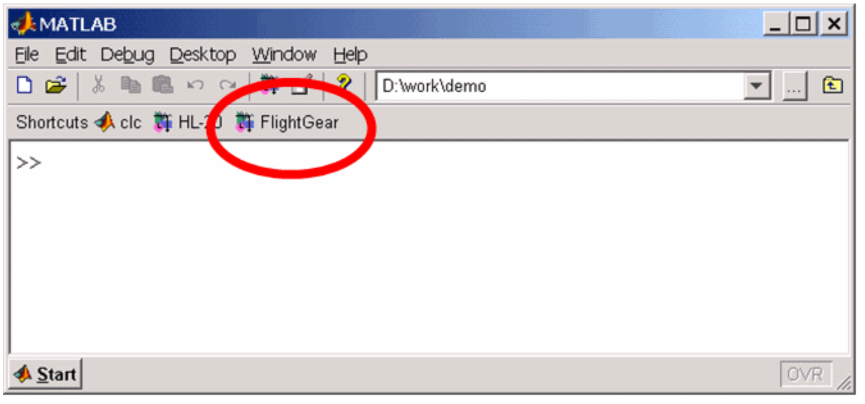
1 From the **Start** button on your MATLAB desktop, click **Shortcuts > New Shortcut**. The **Shortcut Editor** dialog opens.

**2** Set the **Label**, **Callback**, **Category**, and **Icon** fields as shown in the following figure.




**3** Click **Save**.

The **FlightGear** toolbar button appears in your MATLAB desktop. If you click it, the `runfg.bat` file runs in the current directory.




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

**1** Make sure your model is in a writable directory. 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 Simulink 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.

### Improving Performance

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

- Use the Simulink Accelerator to speed up your model execution.
- 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.

### Running FlightGear and Simulink on Different Computers

It is possible to simulate an aerospace system in Simulink 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 “Obtaining the Destination IP Address” on page 2-42.

2 Enter this target computer’s IP address in the Send net_fdm Packet to FlightGear block. See “Adding and Connecting Interface Blocks” on page 2-43.

3 Update the Generate Run Script block in your model with the target computer’s FlightGear base directory. Regenerate the run script to reflect the target computer’s separate identity.

   See “Creating a FlightGear Run Script” on page 2-46.

4 Copy the generated run script to the target computer. Start FlightGear there. See “Starting FlightGear” on page 2-48.

5 Start your Simulink model on the source computer. FlightGear running on the target displays the simulation motion.

## Running the NASA HL-20 Demo with FlightGear

Aerospace Blockset contains a demo model of the NASA HL-20 lifting body that uses the FlightGear interface.

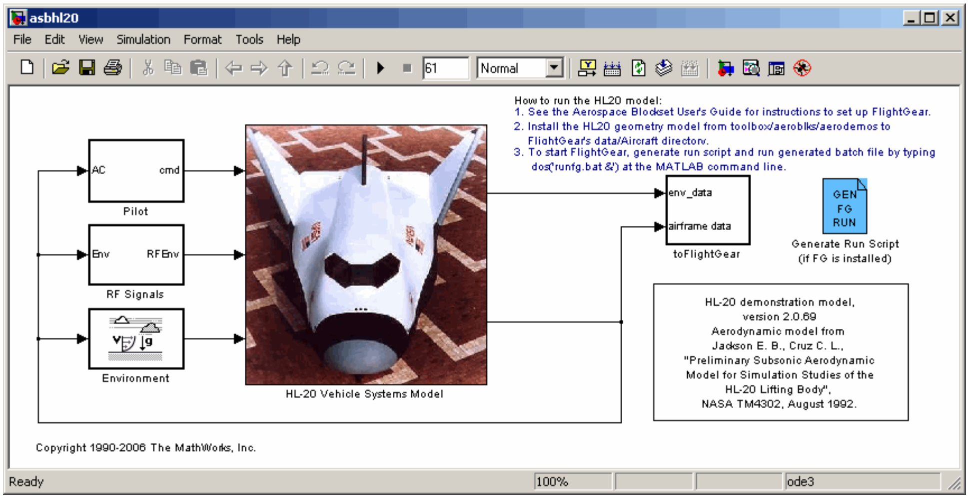
You need to have FlightGear installed and configured before attempting to simulate this model. See "Introducing the Flight Simulator Interface" on page 2-31.

To run this demo:

1 Copy the `HL20` folder from *`matlabroot`*`\toolbox\aeroblks\aerodemos\` directory to *`FlightGear`*`\data\Aircraft\` directory. This folder contains the preconfigured geometries for the HL-20 simulation and `HL20-set.xml`. The file *`matlabroot`*`\toolbox\aeroblks\aerodemos\HL20\models\HL20.xml` defines the geometry.

   For more about this step, see "Importing Aircraft Models into FlightGear" on page 2-39.

2 Start MATLAB. Open the demo either by entering `asbhl20` in the MATLAB Command Window or by finding the demo entry (NASA HL-20 with FlightGear Interface) in the Demos browser and clicking **Open this model** on its demo page. The model opens.




**3** If this is your first time running FlightGear for this model, double-click the Generate Run Script block to create a run script. Make sure to specify your FlightGear installation directory in the **FlightGear base directory** field. For more information, see "Creating a FlightGear Run Script" on page 2-46.

**4** Execute the script you just created manually by entering the following at the MATLAB command line:

```
dos('runfg &')
```

If you created a **FlightGear** desktop button, you can click it instead to start the run script and start FlightGear. For more information, see "Starting FlightGear" on page 2-48.

**5** Now start the simulation and view the animation in FlightGear.

---

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.

---

# Case Studies

These case studies illustrate how to model realistic aerospace systems with Simulink and Aerospace Blockset.

# Ideal Airspeed Correction

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

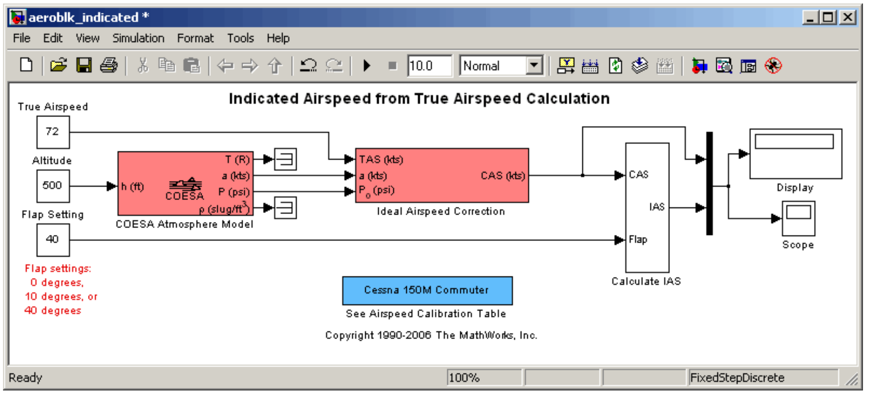
The following sections demonstrate the details:

- "Airspeed Correction Models" on page 3-2 shows how to open the models.
- "Measuring Airspeed" on page 3-3 describes the different types of airspeed.
- "Modeling Airspeed Correction" on page 3-4 describes how the Ideal Airspeed Correction block is implemented.
- "Simulating Airspeed Correction" on page 3-7 runs the model.

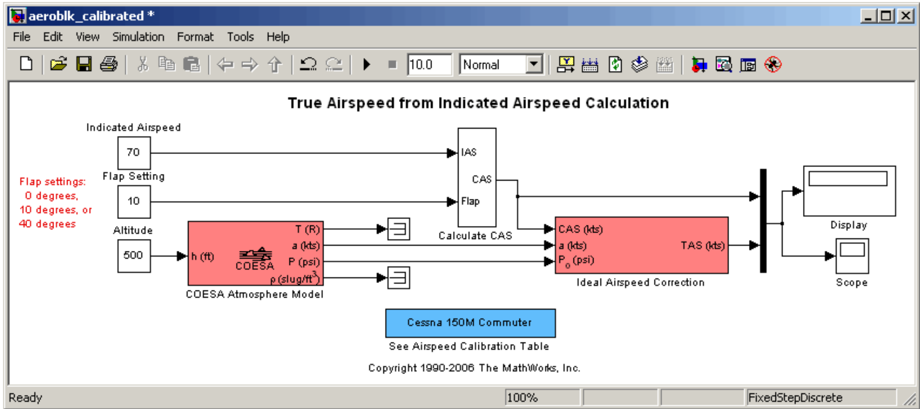
## Airspeed Correction Models

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

```
aeroblk_indicated
aeroblk_calibrated
```




**aeroblk_indicated Model**




**aeroblk_calibrated Model**

## Measuring 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.

## Modeling 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 3-5 calculates the change in atmospheric conditions with changing altitude.
- "Ideal Airspeed Correction Block" on page 3-5 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 |
| | Calibrated airspeed |
| Equivalent Airspeed | True airspeed |
| | Calibrated airspeed |
| Calibrated Airspeed | True airspeed |
| | 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 3-6
- "Calibrated Airspeed Implementation" on page 3-6

- "Equivalent Airspeed Implementation" on page 3-6

**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$^3$ |
| $P_0$ | Static pressure at mean sea level in N/m$^2$ |
| $\gamma$ | Ratio of specific heats |
| $q$ | Dynamic pressure at mean sea level in N/m$^2$ |

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

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

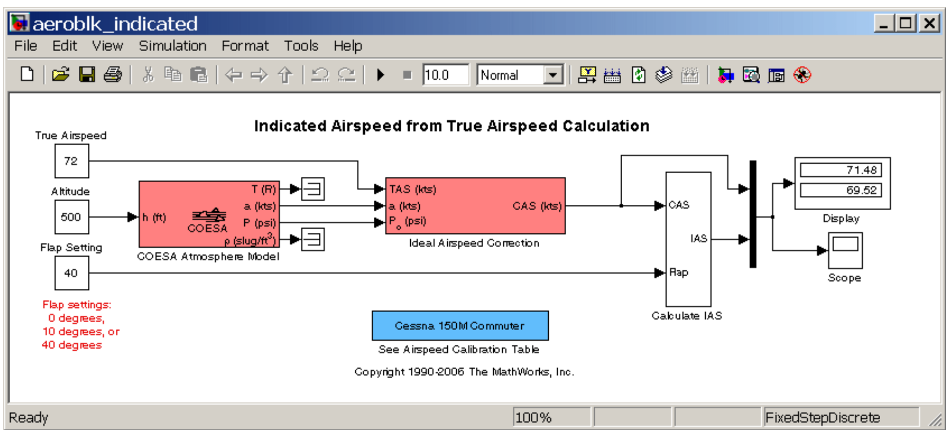
The symbols are defined as follows:

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

## Simulating 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.

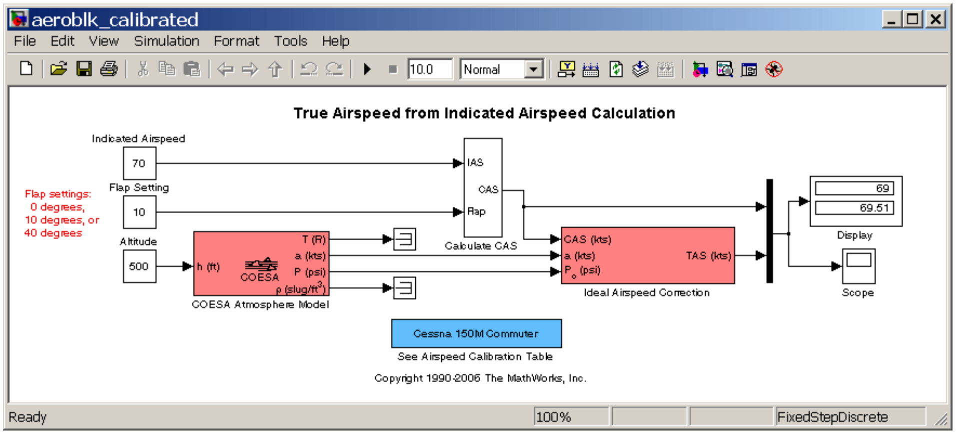
The model's Display block shows both indicated and calibrated airspeeds.




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

**Note** The final section of this study requires Virtual Reality Toolbox.

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.

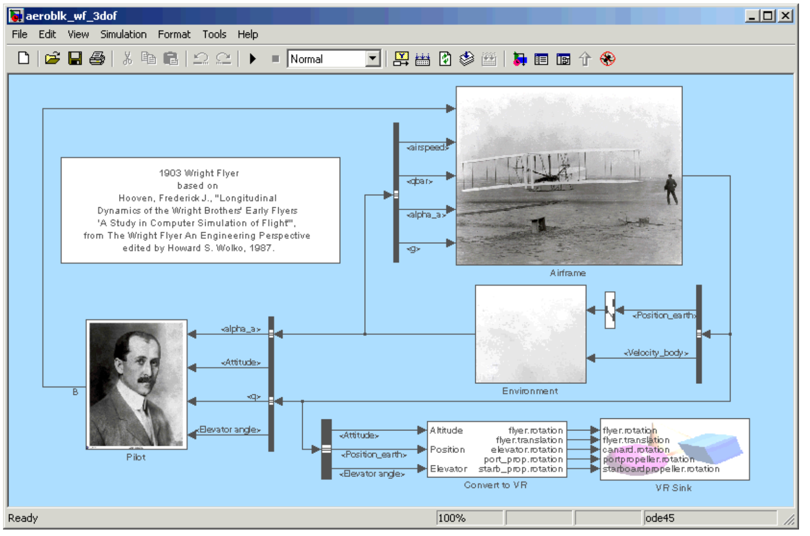
The study consists of these sections:

- "Wright Flyer Model" on page 3-10 shows how to open the model used in this case study.
- "Airframe Subsystem" on page 3-10 describes the airframe subsystem.
- "Environment Subsystem" on page 3-14 describes the environment subsystem.
- "Pilot Subsystem" on page 3-15 describes the Pilot subsystem.
- "Running the Simulation" on page 3-16 provides a demonstration of the Wright Flyer model, including a virtual world visualization.

- “References” on page 3-17 provides a selected bibliography.

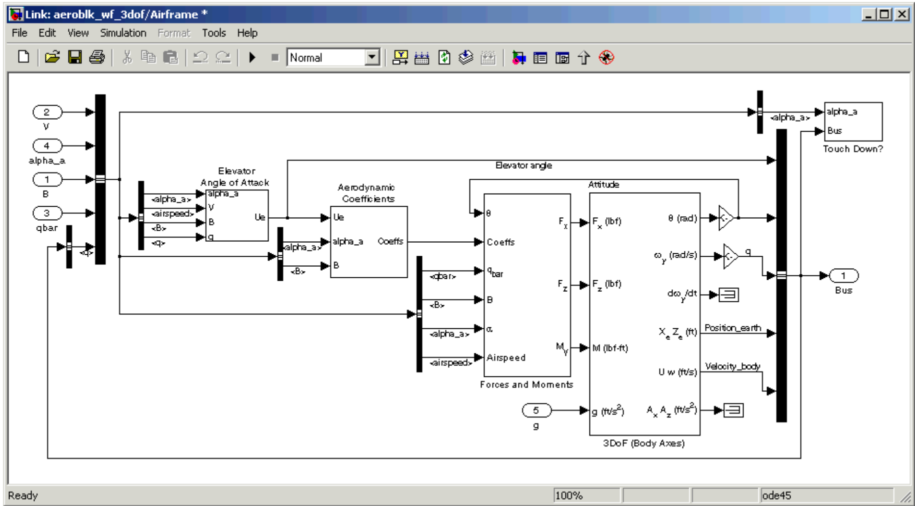
## Wright Flyer Model

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.

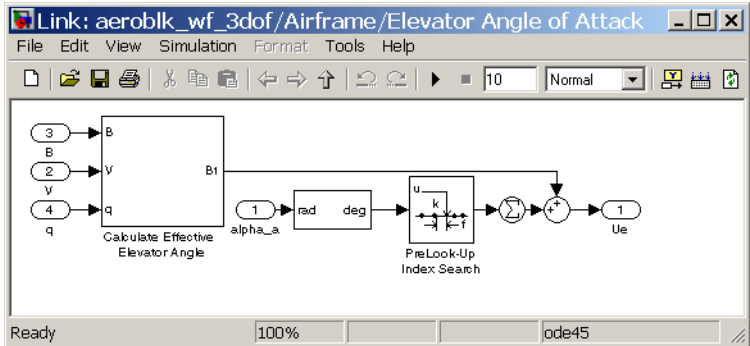



The Airframe subsystem consists of the following parts:

- “Elevator Angle of Attack Subsystem” on page 3-11
- “Aerodynamic Coefficients Subsystem” on page 3-12
- “Forces and Moments Subsystem” on page 3-12
- “3DoF (Body Axes) Block” on page 3-13

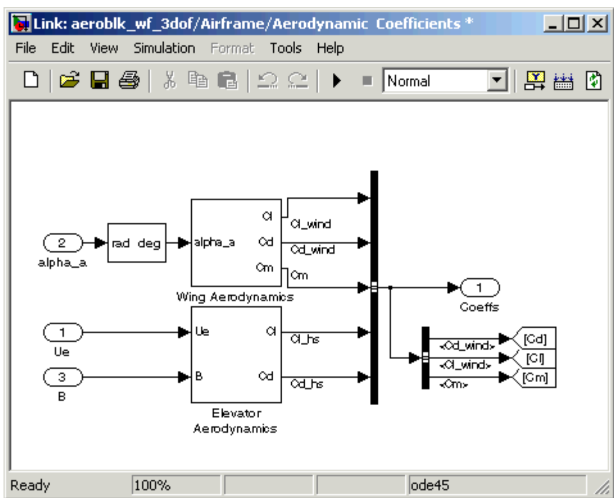
## Elevator Angle of Attack Subsystem

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



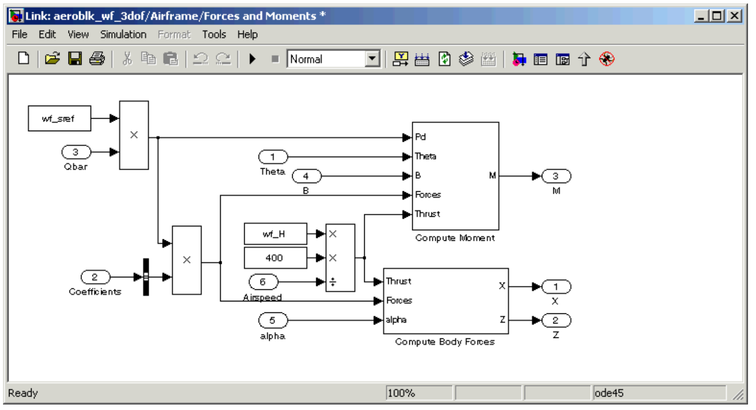

## Aerodynamic Coefficients Subsystem

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.



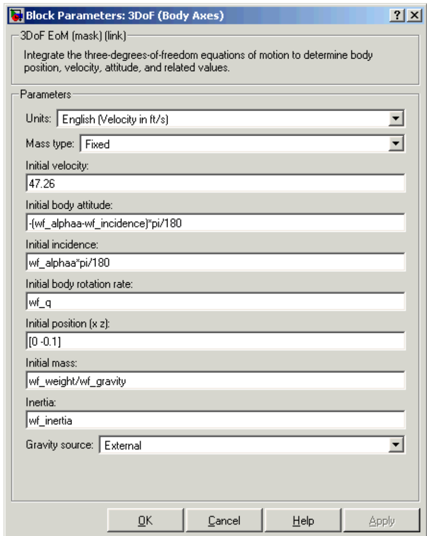

## Forces and Moments Subsystem

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**

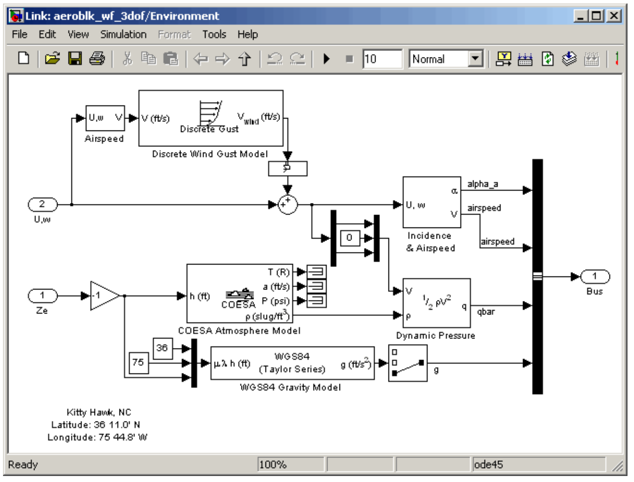
## Environment Subsystem

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 Aerospace Blockset, 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.

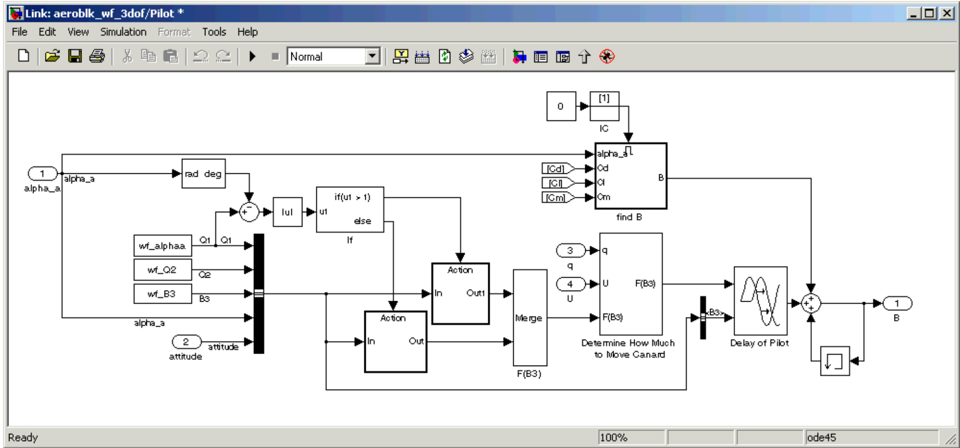
- The WGS84 Gravity Model block produces the gravity at the Wright Flyer's latitude, longitude,and height.




## Pilot Subsystem

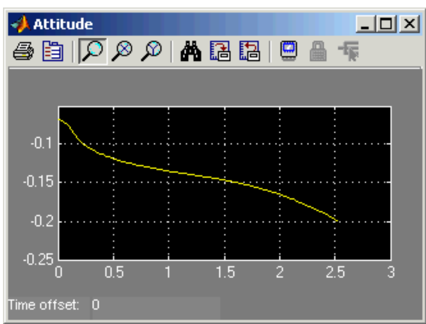
The Pilot subsystem controls the aircraft by responding to both pitch angle (attitude) and angle of attack. If the angle of attack differs from the set angle of attack by more than one degree, the Pilot subsystem responds with a correction of the elevator (canard) angle. When the angular velocity exceeds +/- 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.




# Running 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.



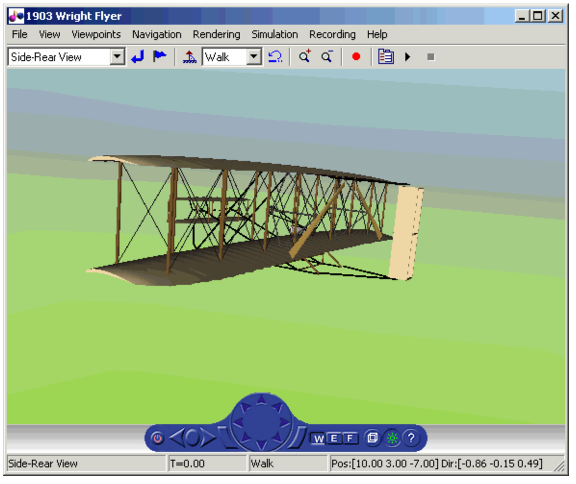

**Attitude Scope (Measured in Radians)**

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 Virtual Reality Toolbox.

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

- `http://www.wrightexperience.com`
- `http://wright.nasa.gov`

# NASA HL-20 Lifting Body Airframe

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, Aerospace Blockset efficiently models portions of the HL-20 airframe. The remaining portions, including calculation of the aerodynamic coefficients, are modeled with Simulink. This case study examines the HL-20 airframe model and touches on how the aerodynamic data are used in the model.

This study consists of these sections:

- "NASA HL-20 Lifting Body" on page 3-19 provides an overview of the history and purposes of the NASA HL-20 lifting body.
- "The HL-20 Airframe and Controller Model" on page 3-20 describes the HL-20 combined plant and controller model.
- "References" on page 3-33 provides a selected bibliography.

## 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.

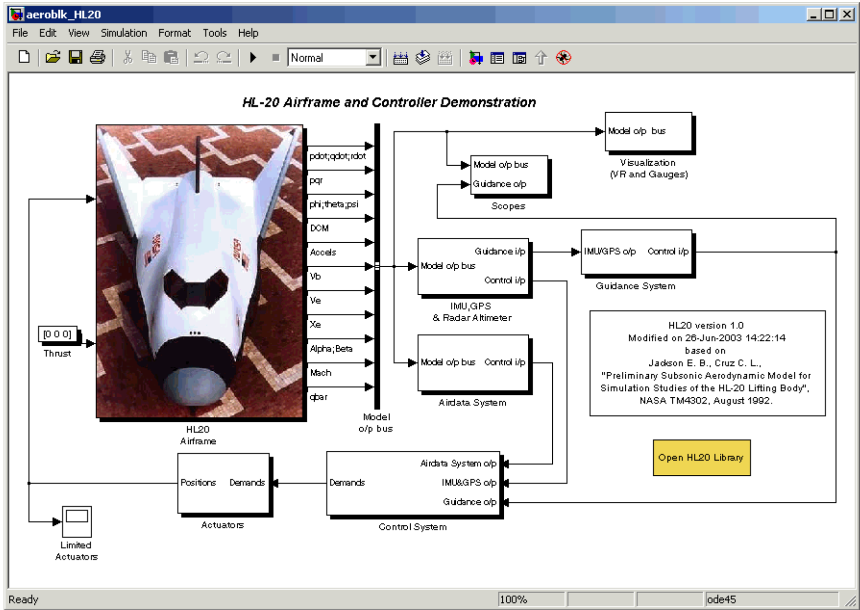
Potential uses for the HL-20 include

- 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:

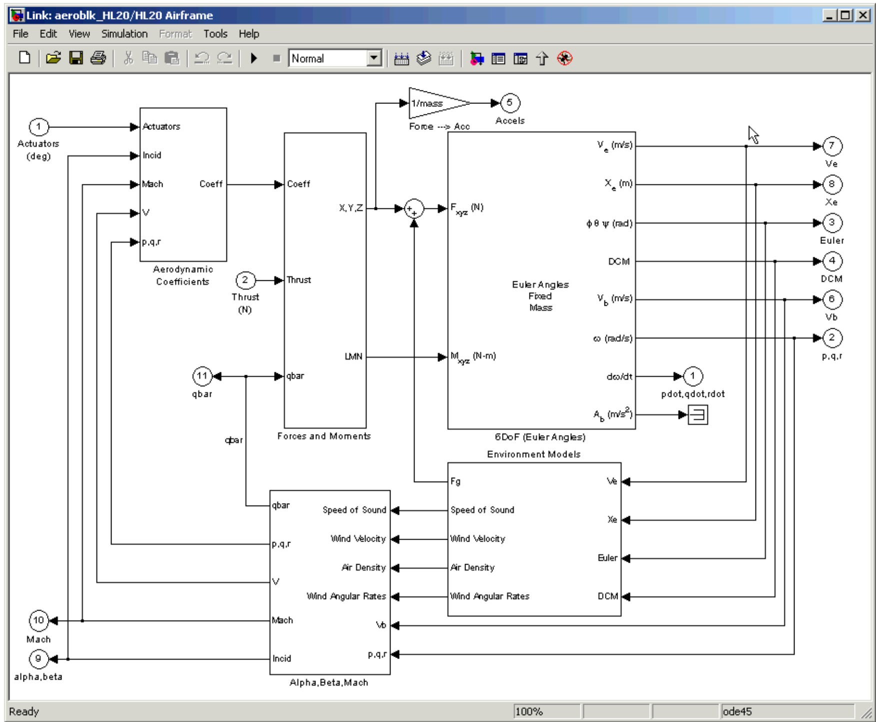
- The airframe is assumed to be rigid and have constant mass, center of gravity, and inertia, since the model represents only the unpowered reentry portion of a mission.
- HL-20 is assumed to be a laterally symmetric vehicle.
- Compressibility (Mach) effects are assumed to be negligible.
- Control effectiveness 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 3-23
- "Environmental Models Subsystem" on page 3-23
- "Alpha, Beta, Mach Subsystem" on page 3-26
- "Aerodynamic Coefficients Subsystem" on page 3-27
- "Forces and Moments Subsystem" on page 3-32




**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 Aerospace Blockset. 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.

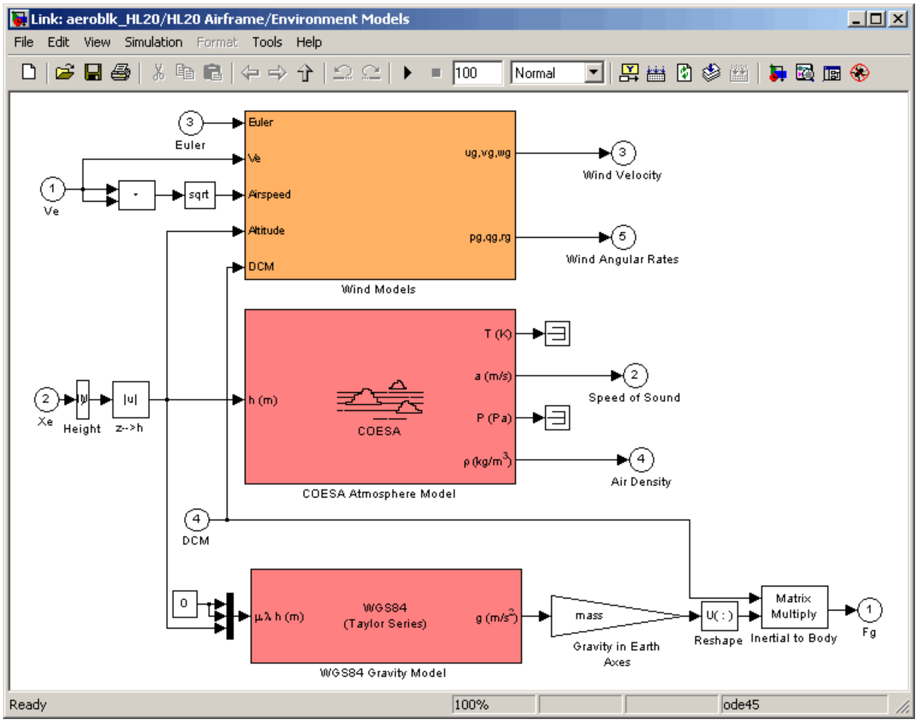
  - The Discrete Wind Gust Model block implements a wind gust of the standard “1 - cosine” shape.

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

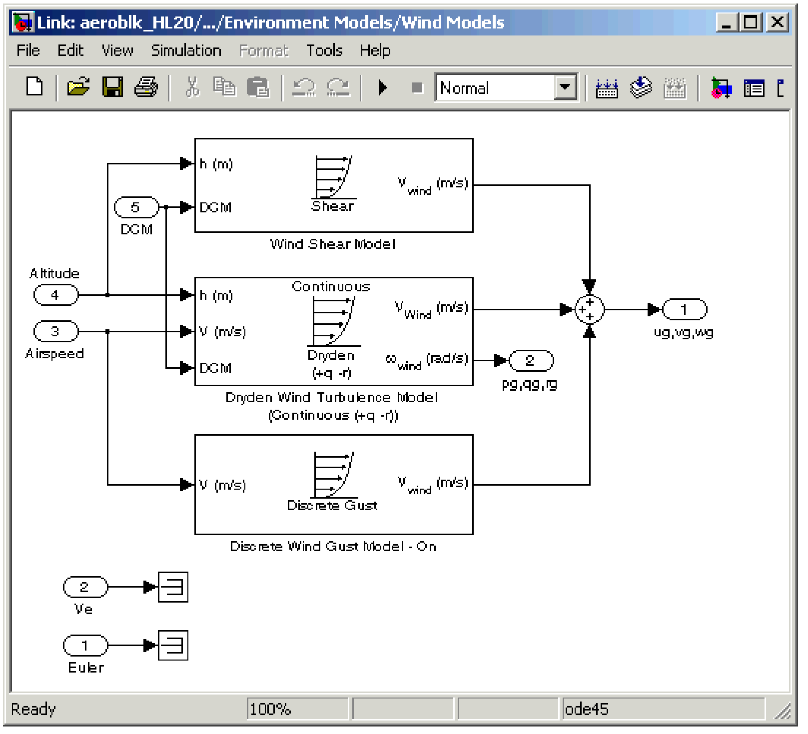
  - The Dryden Wind Turbulence Model (Continuous) block uses the Dryden spectral representation to add turbulence to the aerospace model by passing band-limited white noise through appropriate forming filters.

    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**



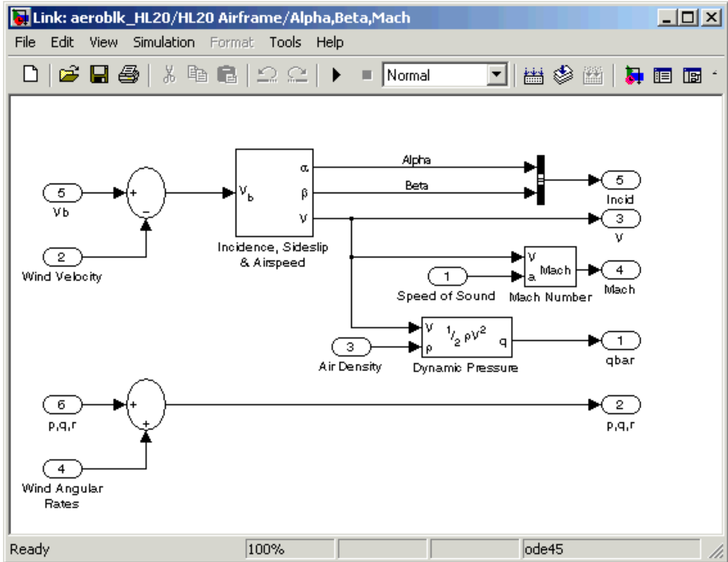

**Wind Models in HL-20 Airframe Model**

## 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)**

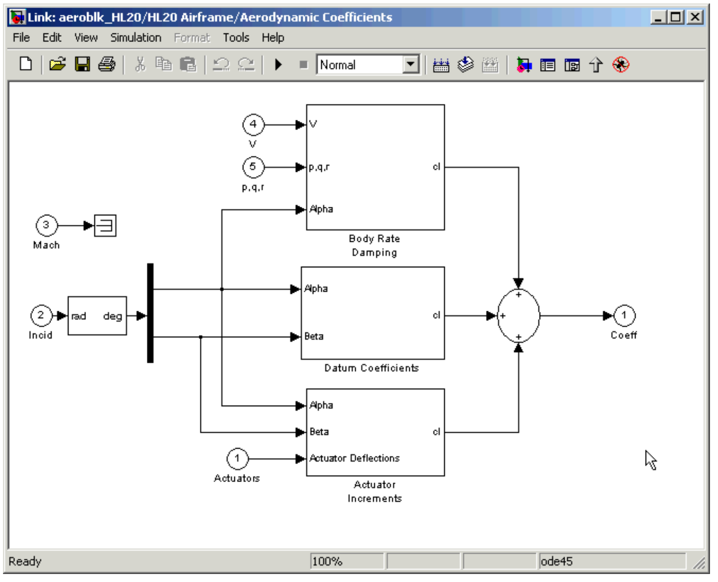
## Aerodynamic Coefficients Subsystem

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].

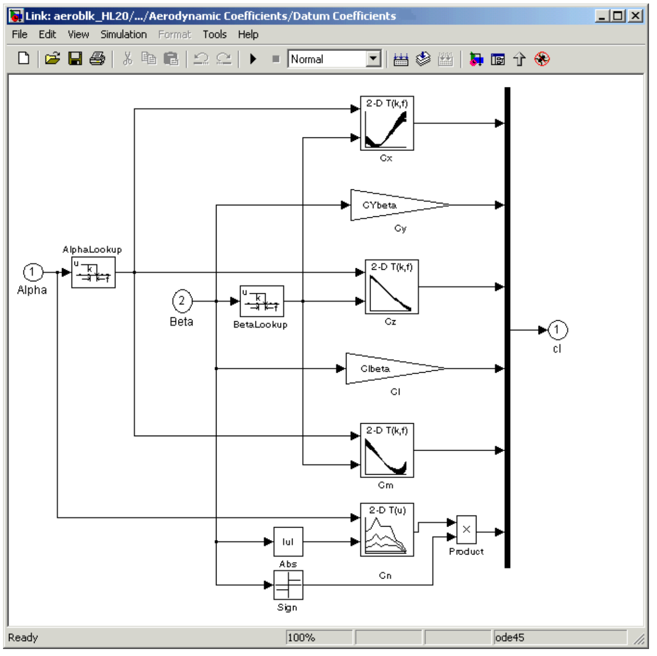
The polynomial functions contained in the M-file `aeroblk_init_hl20.m` are used to calculate lookup tables used by the model's preload function. Lookup tables substitute for polynomial functions. Depending on the order and implementation of the function, using lookup tables can be more efficient than 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:

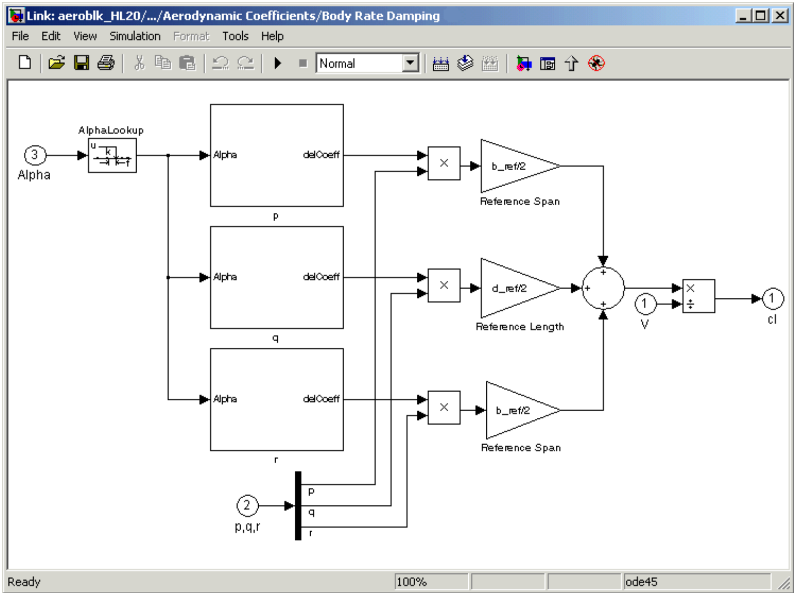

- "Datum Coefficients Subsystem" on page 3-29
- "Body Rate Damping Subsystem" on page 3-30
- "Actuator Increment Subsystem" on page 3-31


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].

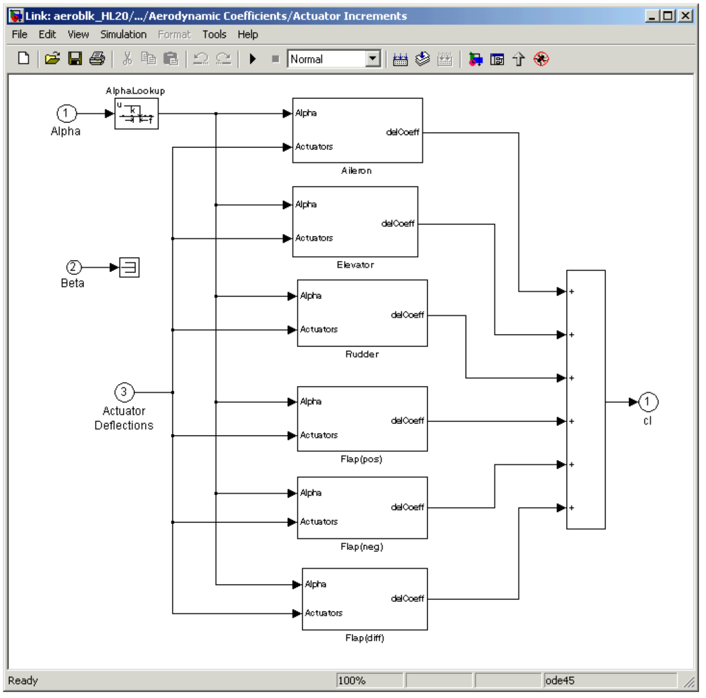
**Datum Coefficients Subsystem.** The Datum Coefficients subsystem calculates coefficients for the basic configuration without control surface deflection. These datum coefficients depend only on the incidence angles of the body.



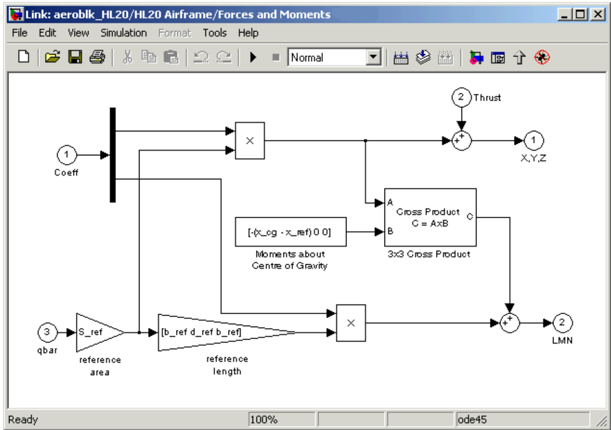

**Body Rate Damping Subsystem.** Dynamic motion derivatives are computed in the Body Rate Damping subsystem.




**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.




### Completing 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` demo.

## 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] Morring, 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/craft/hl20.htm`

`http://www.aviationnow.com/content/publication/awst/20020520/aw46.htm`
(requires subscription)

# Missile Guidance System

This case study explains the design and simulation of a guidance system for a three-degrees-of-freedom missile. The model includes all aspects of the system, from the missile airframe (plant) and environment to the controller.


- "Missile Guidance System Model" on page 3-35 shows how to open the model used in this study.
- "Modeling Airframe Dynamics" on page 3-36 describes the implementation of the atmospheric equations and equations of motion for the missile airframe.
- "Modeling a Classical Three-Loop Autopilot" on page 3-43 describes the design of the missile autopilot to control the acceleration normal to the missile body.
- "Modeling the Homing Guidance Loop" on page 3-45 describes the design of a homing guidance loop to track the target and generate the demands that are passed to the autopilot. This subsystem uses Stateflow.
- "Simulating the Missile Guidance System" on page 3-51 describes the simulation of the model and evaluation of system performance.
- "Extending the Model" on page 3-53 examines a representation of the full six-degrees-of-freedom equations of motion.
- "References" on page 3-54 provides a selected bibliography.


---

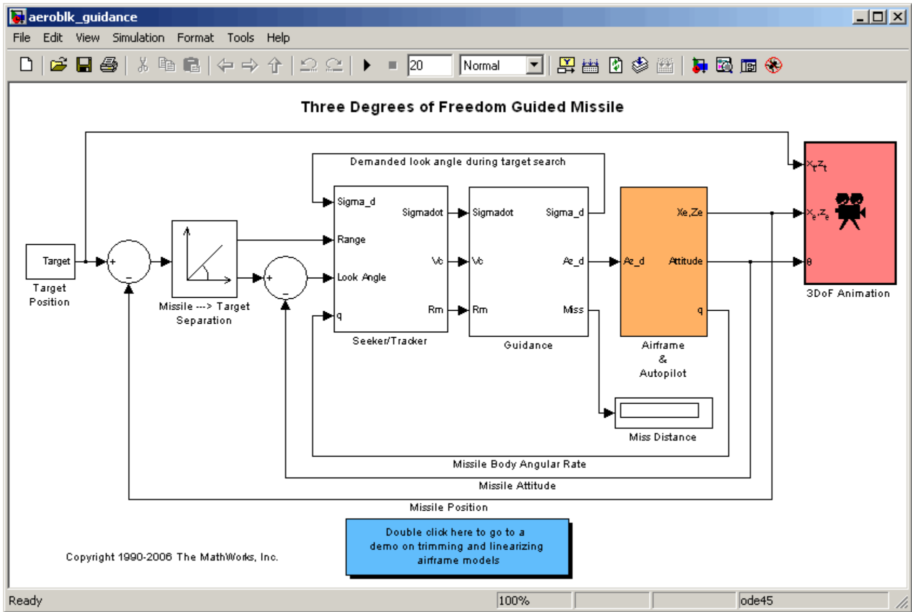
**Note** The Stateflow module in this demo is precompiled and does not require Stateflow to be installed.

---

## Missile Guidance System Model

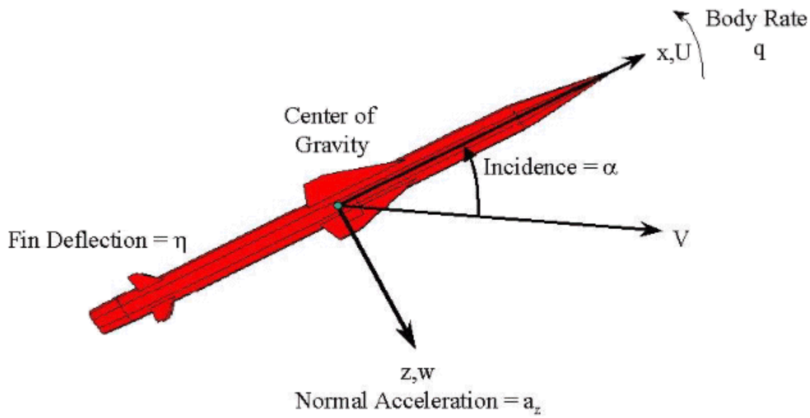
To view the missile guidance system model, enter `aeroblk_guidance` at the MATLAB command line.

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




## Modeling Airframe Dynamics

The model of the missile airframe in this demo uses advanced control methods applied to missile autopilot design [1], [2], [3]. The model represents a tail-controlled missile traveling between Mach 2 and Mach 4, at altitudes ranging between 3,050 meters (10,000 feet) and 18,290 meters (60,000 feet), and with typical angles of attack in the range of ±20 degrees.



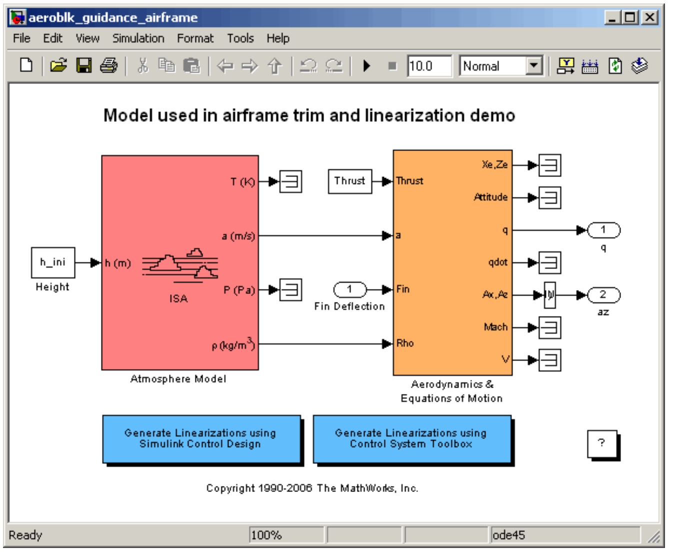

**Missile Airframe Model**

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

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

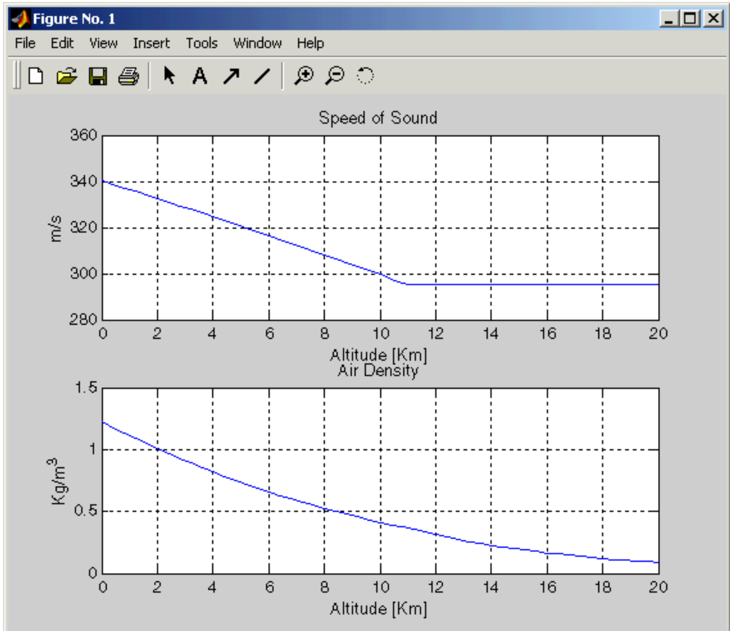
- "ISA Atmosphere Model Block" on page 3-38 calculates the change in atmospheric conditions with changing altitude.
- "Aerodynamics & Equations of Motion Subsystem" on page 3-40 calculates the magnitude of the forces and moments acting on the missile body and integrates the equations of motion.

To view the missile airframe model, enter `aeroblk_guidance_airframe` at the MATLAB command line.




## ISA Atmosphere Model Block

The ISA Atmosphere Model block is an approximation of the International Standard Atmosphere (ISA). This block implements two sets of equations. The troposphere requires one set of equations, and the lower stratosphere requires the other set. The troposphere lies between sea level and 11,000 meters (36,089 feet). The ISA model assumes a linear temperature drop with increasing altitude in the troposphere. The lower stratosphere ranges between 11,000 meters (36,089 feet) and 20,000 meters (65,617 feet). The ISA models the lower stratosphere by assuming that the temperature remains constant.




**Variation of Sound Speed and Air Density with Altitude**

The following equations define the troposphere.

$$T = T_o - Lh$$

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

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

$$a = \sqrt{\gamma R T}$$

The following equations define the lower stratosphere.

$$T = T_o - L \cdot hts$$

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

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

$$a = \sqrt{\gamma R T}$$

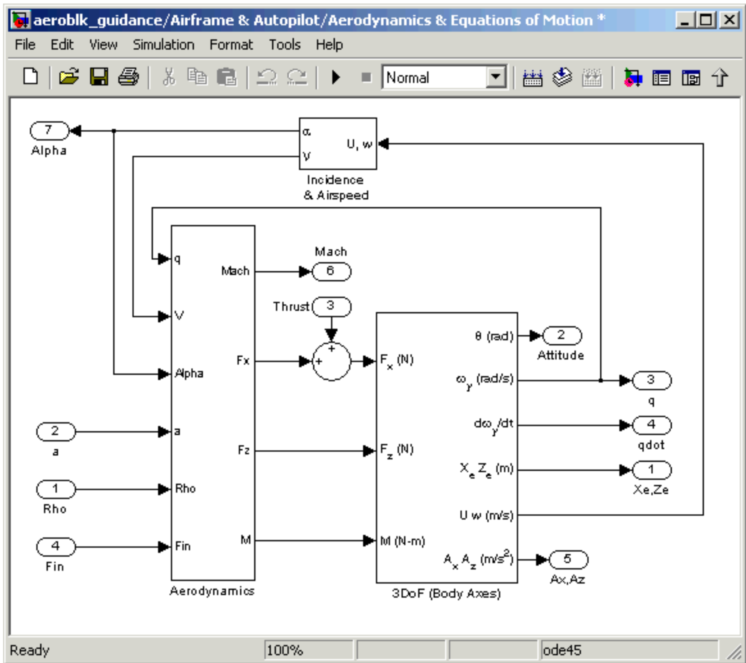
The symbols are defined as follows:

| | |
|---|---|
| $T_0$ | Absolute temperature at mean sea level in kelvin (K) |
| $\rho_0$ | Air density at mean sea level in kg/m$^3$ |
| $P_0$ | Static pressure at mean sea level in N/m$^2$ |
| $h$ | Altitude in m |
| $hts$ | Height of the troposphere in m |
| $T$ | Absolute temperature at altitude $h$ in kelvin (K) |
| $\rho$ | Air density at altitude $h$ in kg/m$^3$ |
| $P$ | Static pressure at altitude $h$ in N/m$^2$ |
| $a$ | Speed of sound at altitude $h$ in m/s$^2$ |
| $L$ | Temperature lapse rate in K/m |
| $R$ | Characteristic gas constant J/kg-K |
| $\gamma$ | Ratio of specific heats |
| $g$ | Acceleration due to gravity in m/s$^2$ |

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

## Aerodynamics & Equations of Motion Subsystem

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




These are the three-degrees-of-freedom body axis equations of motion, which are defined in the 3DoF (Body Axes) block.

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

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

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

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

These are the aerodynamic forces and moments equations, which are defined in the Aerodynamics subsystem.

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

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

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

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

These are the stability axes variables, which are calculated in the Incidence & Airspeed block.

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

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

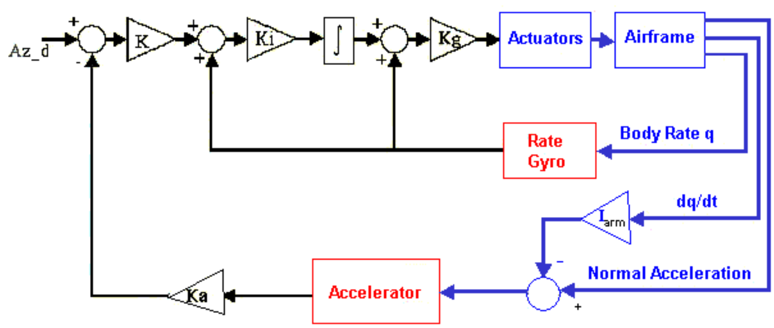
The symbols are defined as follows:

| | |
|---|---|
| $\theta$ | Attitude in radians |
| $q$ | Body rotation rate in rad/s |
| $M$ | Missile mass in kg |
| $g$ | Acceleration due to gravity in $m/s^2$ |
| $I_{yy}$ | Moment of inertia about the $y$-axis in $kg\text{-}m^2$ |
| $\dot{W}$ | Acceleration in the Z body axis in $m/s^2$ |
| $\dot{q}$ | Change in body rotation rate in $rad/s^2$ |
| $T$ | Thrust in the $X$ body axis in N |
| $\rho$ | Air density in $kg/m^3$ |
| $S_{ref}$ | Reference area in $m^2$ |
| $C_X$ | Coefficient of aerodynamic force in the $X$ body axis |
| $C_Z$ | Coefficient of aerodynamic force in the $Z$ body axis |
| $C_M$ | Coefficient of aerodynamic moment about the $Y$ body axis |

| | |
|---|---|
| $d_{ref}$ | Reference length in m |
| $\eta$ | Fin angle in rad |
| $F_X$ | Aerodynamic force in the $X$ body axis in N |
| $F_Z$ | Aerodynamic force in the $Z$ body axis in N |
| $M$ | Aerodynamic moment along the $Y$ body axis |
| $\bar{q}$ | Dynamic pressure in Pa |
| $V$ | Airspeed in m/s |
| $\alpha$ | Incidence in rad |
| $U$ | Velocity in the $X$ body axis in m/s |
| $W$ | Velocity in the $Z$ body axis in m/s |

## Modeling a Classical Three-Loop Autopilot

The missile autopilot controls the acceleration normal to the missile body. The autopilot structure of this case study is a three-loop design using measurements from an accelerometer located ahead of the missile's center of gravity and from a rate gyro to provide additional damping. The controller gains are scheduled on incidence and Mach number and tuned for robust performance at an altitude of 3,050 meters (10,000 feet).




**Classical Autopilot**

Designing an autopilot requires the following:

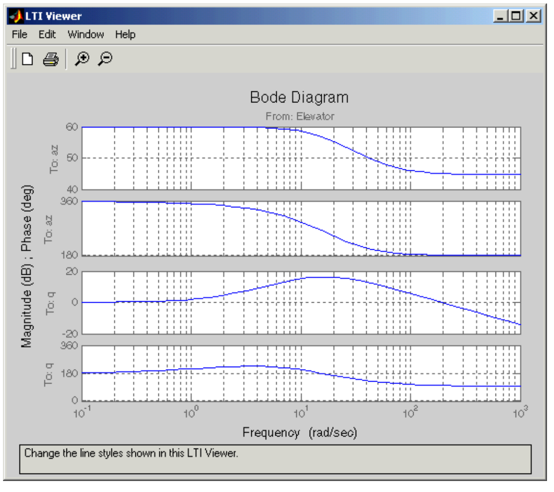
- "Trimming and Linearizing an Airframe Model" on page 3-44 explains how to model the airframe pitch dynamics for several trimmed flight conditions.
- "Autopilot Design" on page 3-45 summarizes the autopilot design process.

## Trimming and Linearizing an Airframe Model

Designing the autopilot with classical design techniques requires linear models of the airframe pitch dynamics for several trimmed flight conditions. MATLAB can determine the trim conditions and derive linear state-space models directly from the nonlinear Simulink model. This step saves time and helps to validate the model. The functions provided by Simulink Control Design or Control System Toolbox allow you to visualize the behavior of the airframe in terms of open-loop frequency or time response.

The airframe trim demos show how to trim and linearize an airframe model.

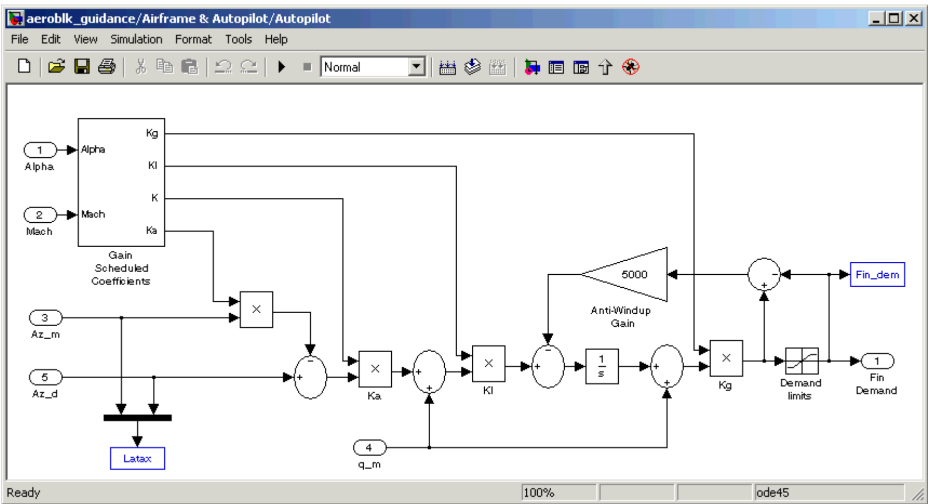
- To run the demo based on Control System Toolbox, enter `asbguidance_trimlinearize_cst`. The results of this demo are displayed as a Bode diagram in the LTI Viewer.
- The alternative demo, `asbguidance_trimlinearize`, uses Simulink Control Design instead and produces identical results.



### Autopilot Design

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

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



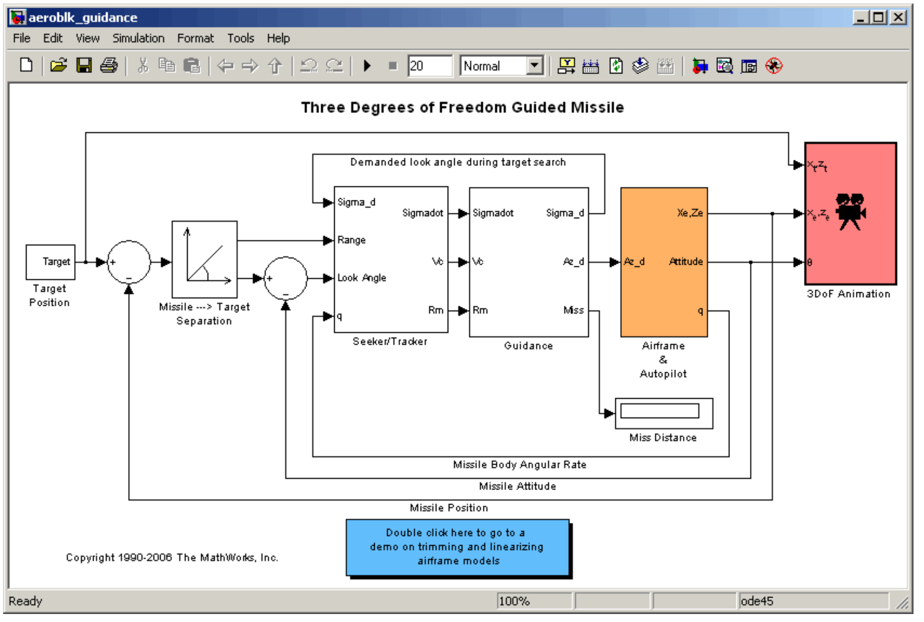

## Modeling the Homing Guidance Loop

The complete homing guidance loop consists of these two subsystems:

- The “Guidance Subsystem” on page 3-46 generates the normal acceleration demands that are passed to the autopilot and uses Stateflow.

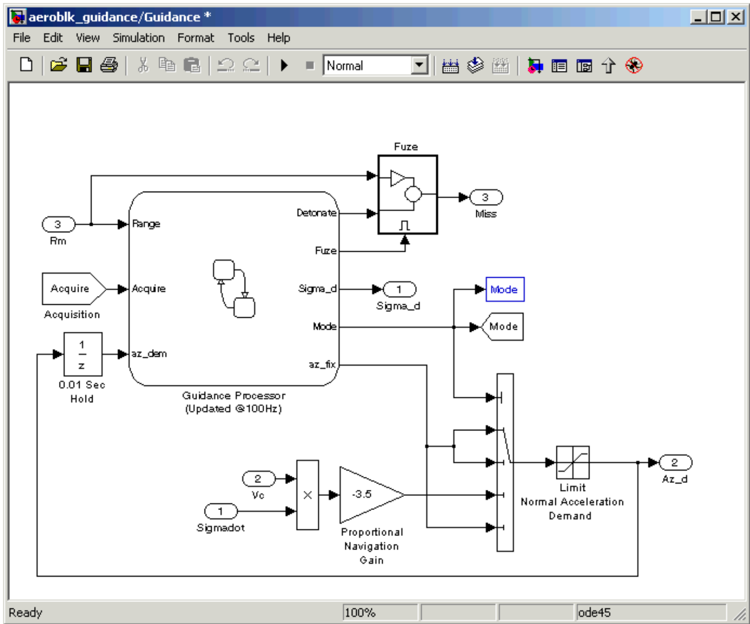
- The "Seeker/Tracker Subsystem" on page 3-49 returns measurements of the relative motion between the missile and the target.

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



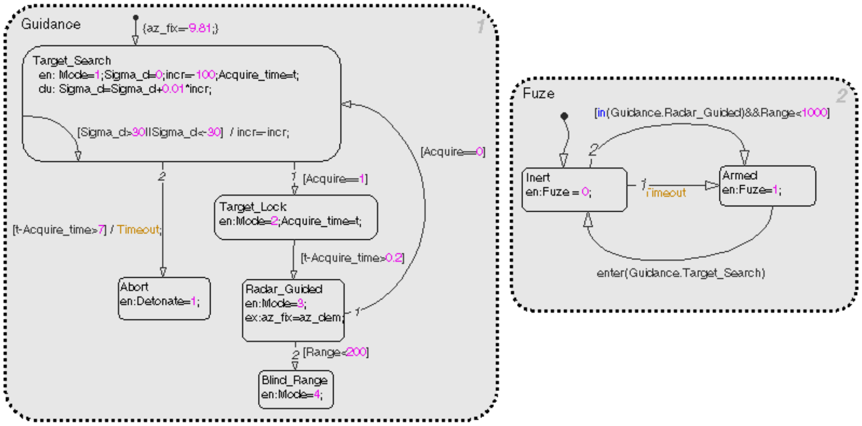

## Guidance Subsystem

Initially, the Guidance subsystem searches to locate the target's position and then generates demands during closed-loop tracking. A Stateflow chart controls the transfer between the different modes of these operations. Stateflow is the ideal tool for rapidly defining all the operational modes, both during normal operation and during unusual situations.

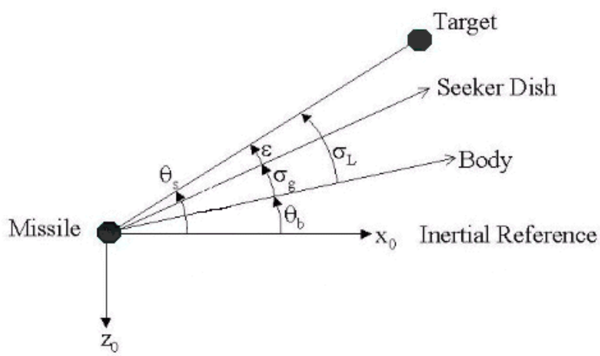



**Guidance Processor State Chart.** Mode switching is triggered by events generated in Simulink or in the Stateflow chart. The variable `Mode` is passed to Simulink and is used to control the Simulink model's behavior and response. For example, the Guidance Processor state chart, which is part of the Guidance subsystem, shows how the system reacts in response to either losing the target lock or failing to acquire the target's position during the target search.

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




**Proportional Navigation Guidance.** Once the seeker has acquired the target, a proportional navigation guidance (PNG) law guides the missile until impact. This form of guidance law is the most basic, used in guided missiles since the 1940s, and can be applied to radar-, infrared-, or television-guided missiles. The navigation law requires measurements of the closing velocity between the missile and target, which for a radar-guided missile can be obtained with a Doppler tracking device, and an estimate for the rate of change of the inertial sight line angle.



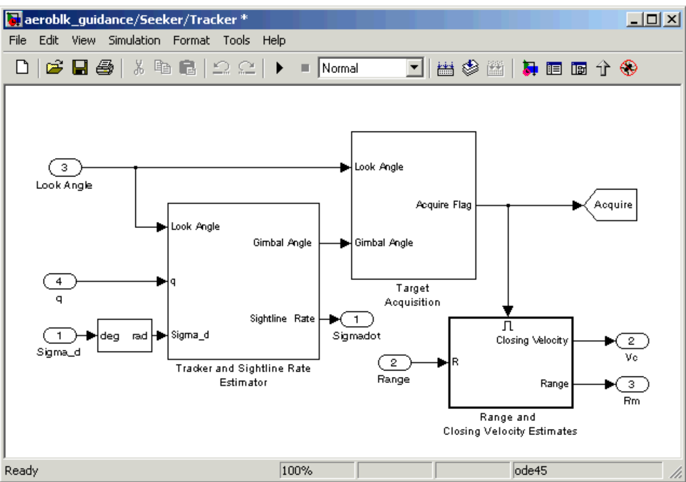

**Proportional Navigation Guidance Measurements**

The diagram symbols are defined as follows:

| | |
|---|---|
| $\lambda$ | Navigation gain (> 2) |
| $V_c$ | Closing velocity |
| $\Theta_b$ | Body attitude |
| $\dot{\theta}_s$ | Sight line rate |
| $\sigma_g$ | Gimbal angle |
| $\sigma_L$ | Look angle |
| $\sigma_d$ | Dish angle |
| $a_{z\_dem} = \lambda V_c \dot{\theta}_s$ | Demanded normal acceleration |

## Seeker/Tracker Subsystem

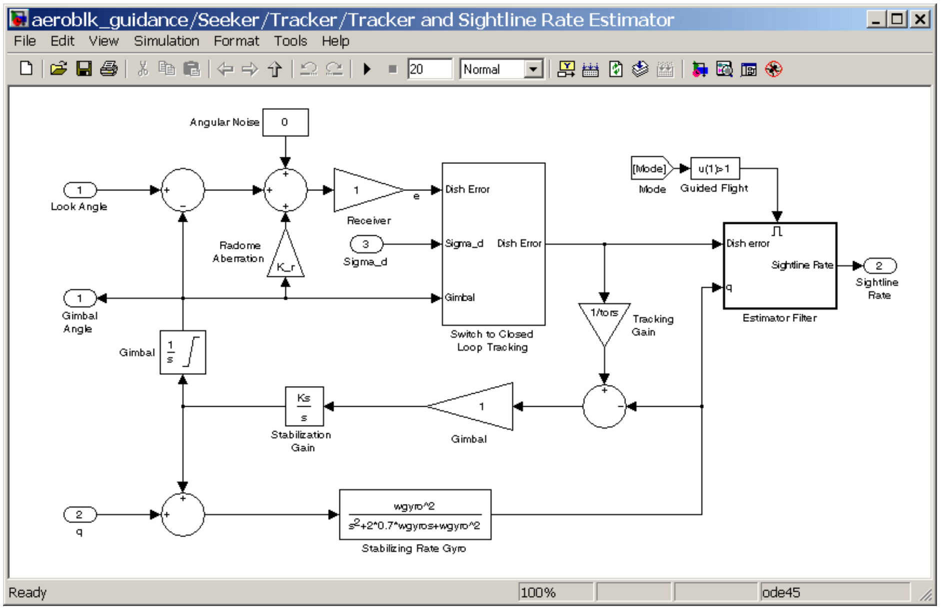
The Seeker/Tracker subsystem controls the seeker gimbals to keep the seeker dish aligned with the target and provides the guidance law with an estimate of the sight line rate.




**Tracker and Sightline Rate Estimator.** The Tracker and Sightline Rate Estimator is the most elaborate subsystem of the Seeker/Tracker subsystem because of its complex error modeling.

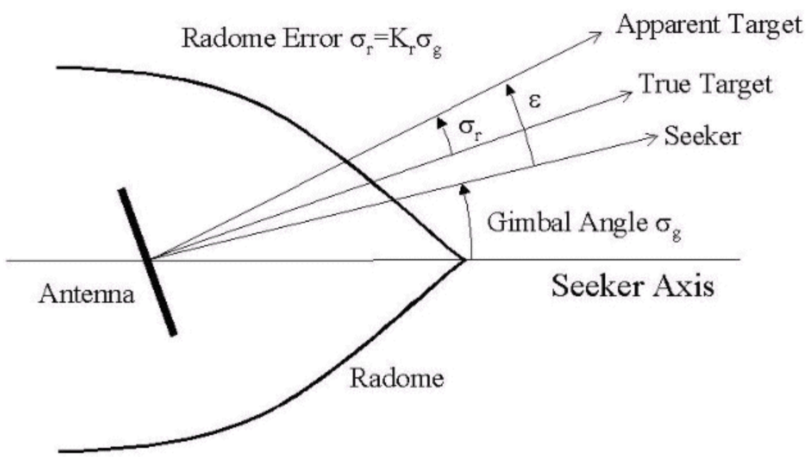
The subsystem contains a number of feedback loops, estimated parameters, and parasitic effects for the homing guidance.

- The tracker loop time constant `tors` is set to `0.05` second, a compromise between maximizing speed of response and keeping the noise transmission within acceptable levels.
- The stabilization loop compensates for body rotation rates. The gain `Ks`, which is the loop crossover frequency, is set as high as possible subject to the limitations of the stabilizing rate gyro's bandwidth.
- The sight line rate estimate is a filtered value of the sum of the rate of change of the dish angle measured by the stabilizing rate gyro and an estimated value for the rate of change of the angular tracking error (`e`) measured by the receiver. In this model, the bandwidth of the estimator filter is set to half that of the bandwidth of the autopilot.



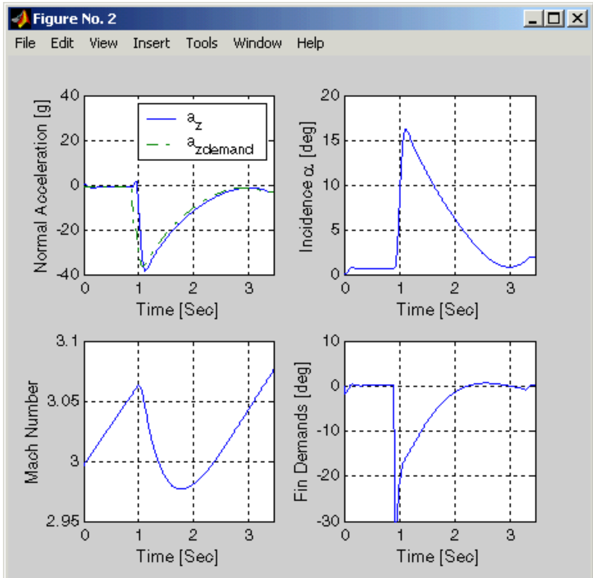

**Radome Aberration.** The Tracker and Sightline Rate Estimator subsystem also models the radome aberration.

Radome aberration is a parasitic feedback effect commonly modeled in radar-guided missile designs and occurs because the shape of the protective covering over the seeker distorts the returning signal and gives a false reading of the look angle to the target. The distortion is, in general, a nonlinear function of the current gimbal angle. A common approximation is to assume a linear relationship between the gimbal angle and the magnitude of the distortion. The approximation is valid for a limited range of angle. Other parasitic effects, such as sensitivity to normal acceleration in the rate gyros, are often modeled as well to test the robustness of the target tracker and estimator filters.

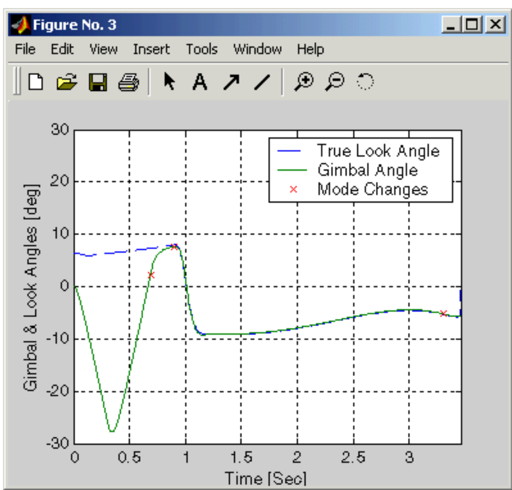



## Simulating the Missile Guidance System

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




Target acquisition occurs 0.69 second after search initiation, with closed-loop guidance starting after 0.89 second. Impact with the target occurs at 3.46 seconds, with the range to target at the point of closest approach calculated to be 0.26 meter.

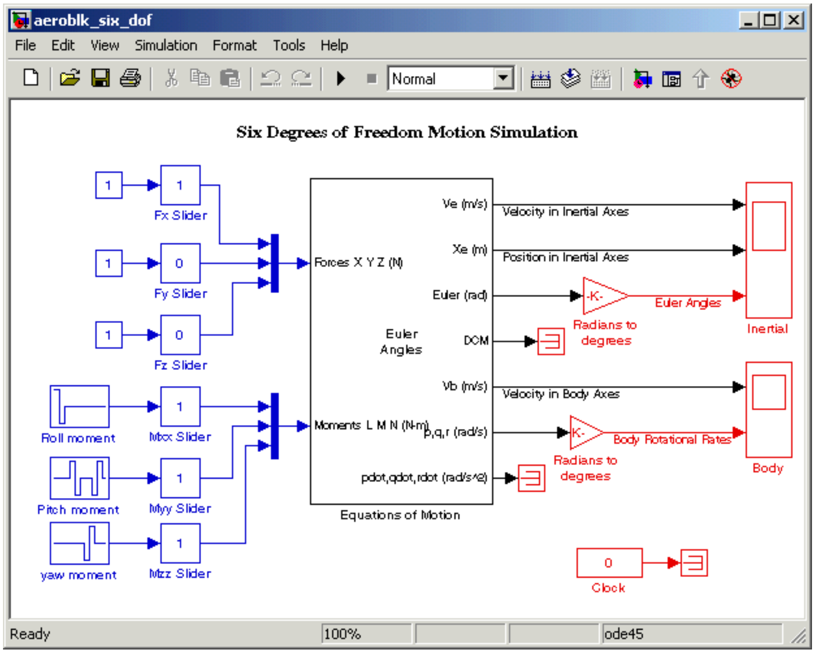


## Extending the Model

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

Six degrees of freedom can be represented using a quaternion or Euler angles.

- The first implementation uses a quaternion to represent the angular orientation of the body in space. The quaternion is appropriate when the standard Euler angle definitions become singular as the pitch attitude tends to ±90 degrees.
- The second implementation uses the standard Euler angle equations of motion. Euler angles are appropriate when obtaining trim conditions and modeling linear airframes. This model contains one of the six-degrees-of-freedom equations of motion blocks.



## References

[1] Bennani, S., D. M. C. Willemsen, and C. W. Scherer, “Robust LPV control with bounded parameter rates,” AIAA-97-3641, August 1997.

[2] Mracek, C. P. and J. R. Cloutier, “Full Envelope Missile Longitudinal Autopilot Design Using the State-Dependent Riccati Equation Method,” AIAA-97-3767, August 1997.

[3] Shamma, J. S. and J. R. Cloutier, “Gain-Scheduled Missile Autopilot Design Using Linear Parameter Varying Transformations,” *Journal of Guidance*, *Control and Dynamics*, Vol. 16, No. 2, March-April 1993.

[4] Lin, Ching-Fang, *Modern Navigation, Guidance, and Control Processing*, Vol. 2, Prentice Hall, 1991.

# Blocks — By Category

# Actuators

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

# Aerodynamics

| | |
|---|---|
| Aerodynamic Forces and Moments | Compute aerodynamic forces and moments using aerodynamic coefficients, dynamic pressure, center of gravity, center of pressure, and velocity |
| Digital DATCOM Forces and Moments | Compute aerodynamic forces and moments using Digital DATCOM static and dynamic stability derivatives |

# Animation

## MATLAB-Based Animation

| | |
|---|---|
| 3DoF Animation | Create 3-D MATLAB Graphics animation of three-degrees-of-freedom object |
| 6DoF Animation | Create 3-D MATLAB Graphics animation of six-degrees-of-freedom object |
| MATLAB Animation | Create six–degrees-of-freedom multibody custom geometry block |

## Flight Simulator Interfaces

| | |
|---|---|
| FlightGear Preconfigured 6DoF Animation | Connect model to FlightGear flight simulator |
| Generate Run Script | Generate FlightGear run script on current computer |
| Pack net_fdm Packet for FlightGear | Generate `net_fdm` packet for FlightGear |
| Send net_fdm Packet to FlightGear | Transmit `net_fdm` packet to destination IP address and port for FlightGear session |

## Animation Support Utilities

| | |
|---|---|
| Pilot Joystick | Provide joystick interface on Windows platform |
| Pilot Joystick All | Provide joystick interface on Windows platform |
| Simulation Pace | Set simulation rate for FlightGear flight simulator |

# Environment

| | |
|---|---|
| Atmosphere (p. 4-4) | Atmospheric profiles |
| Gravity & Magnetism (p. 4-5) | Gravity and magnetic fields |
| Wind (p. 4-5) | Atmospheric winds |

## Atmosphere

| | |
|---|---|
| COESA Atmosphere Model | Implement 1976 COESA lower atmosphere |
| ISA Atmosphere Model | Implement International Standard Atmosphere (ISA) |
| Lapse Rate Model | Implement lapse rate model for atmosphere |
| Non-Standard Day 210C | Implement MIL-STD-210C climatic data |
| Non-Standard Day 310 | Implement MIL-HDBK-310 climatic data |
| Pressure Altitude | Calculate pressure altitude based on ambient pressure |

## Gravity & Magnetism

| | |
|---|---|
| WGS84 Gravity Model | Implement 1984 World Geodetic System (WGS84) representation of Earth's gravity |
| World Magnetic Model 2000 | Calculate Earth's magnetic field at specific location and time using World Magnetic Model 2000 (WMM2000) |
| World Magnetic Model 2005 | Calculate Earth's magnetic field at specific location and time using World Magnetic Model 2005 (WMM2005) |

## Wind

| | |
|---|---|
| Discrete Wind Gust Model | Generate discrete wind gust |
| Dryden Wind Turbulence Model (Continuous) | Generate continuous wind turbulence with Dryden velocity spectra |
| Dryden Wind Turbulence Model (Discrete) | Generate discrete wind turbulence with Dryden velocity spectra |
| Horizontal Wind Model | Transform horizontal wind into body-axes coordinates |
| Von Karman Wind Turbulence Model (Continuous) | Generate continuous wind turbulence with Von Kármán velocity spectra |
| Wind Shear Model | Calculate wind shear conditions |

# Flight Parameters

| | |
|---|---|
| Dynamic Pressure | Compute dynamic pressure using velocity and air density |
| Ideal Airspeed Correction | Calculate equivalent airspeed (EAS), calibrated airspeed (CAS), or true airspeed (TAS) from each other |
| Incidence & Airspeed | Calculate incidence and airspeed |
| Incidence, Sideslip & Airspeed | Calculate incidence, sideslip, and airspeed |
| Mach Number | Compute Mach number using velocity and speed of sound |
| Radius at Geocentric Latitude | Estimate radius of ellipsoid planet at geocentric latitude |
| Relative Ratio | Calculate relative atmospheric ratios |
| 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 |

# Equations of Motion

| | |
|---|---|
| Three DoFs (p. 4-7) | Dynamics with one rotation and two translation axes |
| Six DoFs (p. 4-7) | Dynamics with three rotation and three translation axes |
| Point Masses (p. 4-9) | Dynamics of point masses |

## Three DoFs

| | |
|---|---|
| 3DoF (Body Axes) | Implement three-degrees-of-freedom equations of motion with respect to body axes |
| 3DoF (Wind Axes) | Implement three-degrees-of-freedom equations of motion with respect to wind axes |
| Custom Variable Mass 3DoF (Body Axes) | Implement three-degrees-of-freedom equations of motion of custom variable mass with respect to body axes |
| Custom Variable Mass 3DoF (Wind Axes) | Implement three-degrees-of-freedom equations of motion of custom variable mass with respect to wind axes |
| Simple Variable Mass 3DoF (Body Axes) | Implement three-degrees-of-freedom equations of motion of simple variable mass with respect to body axes |
| Simple Variable Mass 3DoF (Wind Axes) | Implement three-degrees-of-freedom equations of motion of simple variable mass with respect to wind axes |

## Six DoFs

| | |
|---|---|
| 6DoF (Euler Angles) | Implement Euler angle representation of six-degrees-of-freedom equations of motion |
| 6DoF (Quaternion) | Implement quaternion representation of six-degrees-of-freedom equations of motion with respect to body axes |

| | |
|---|---|
| 6DoF ECEF (Quaternion) | Implement quaternion representation of six-degrees-of-freedom equations of motion in Earth-centered Earth-fixed (ECEF) coordinates |
| 6DoF Wind (Quaternion) | Implement quaternion representation of six-degrees-of-freedom equations of motion with respect to wind axes |
| 6DoF Wind (Wind Angles) | Implement wind angle representation of six-degrees-of-freedom equations of motion |
| Custom Variable Mass 6DoF (Euler Angles) | Implement Euler angle representation of six-degrees-of-freedom equations of motion of custom variable mass |
| 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 |
| 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 |
| 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 |

| | |
|---|---|
| Custom Variable Mass 6DoF Wind (Wind Angles) | Implement wind angle representation of six-degrees-of-freedom equations of motion of custom variable mass |
| Simple Variable Mass 6DoF (Euler Angles) | Implement Euler angle representation of six-degrees-of-freedom equations of motion of simple variable mass |
| 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 |
| 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 |
| 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 |
| Simple Variable Mass 6DoF Wind (Wind Angles) | Implement wind angle representation of six-degrees-of-freedom equations of motion of simple variable mass |

## Point Masses

| | |
|---|---|
| 4th Order Point Mass (Longitudinal) | Calculate fourth-order point mass |
| 4th Order Point Mass Forces (Longitudinal) | Calculate forces used by fourth-order point mass |

| | |
|---|---|
| 6th Order Point Mass (Coordinated Flight) | Calculate sixth-order point mass in coordinated flight |
| 6th Order Point Mass Forces (Coordinated Flight) | Calculate forces used by sixth-order point mass in coordinated flight |

# Guidance, Navigation, and Control



## Control

| | |
|---|---|
| 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 |
| 1D Controller [A(v),B(v),C(v),D(v)] | Implement gain-scheduled state-space controller depending on one scheduling parameter |
| 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 |
| 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 |
| 2D Controller Blend | Implement 2-D vector of state-space controllers by linear interpolation of their outputs |

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

### Guidance

| | |
|---|---|
| Calculate Range | Calculate range between two crafts given their respective positions |

### Navigation

| | |
|---|---|
| Three-Axis Accelerometer | Implement three-axis accelerometer |
| Three-Axis Gyroscope | Implement three-axis gyroscope |
| Three-Axis Inertial Measurement Unit | Implement three-axis inertial measurement unit (IMU) |

## Mass Properties

| | |
|---|---|
| Estimate Center of Gravity | Calculate center of gravity location |
| Estimate Inertia Tensor | Calculate inertia tensor |
| Moments About CG Due to Forces | Compute moments about center of gravity due to forces applied at a point, not center of gravity |
| Symmetric Inertia Tensor | Create inertia tensor from moments and products of inertia |

## Propulsion

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

# Utilities



## Axes Transformations

| | |
|---|---|
| Besselian Epoch to Julian Epoch | Transform position and velocity components from discontinued Standard Besselian Epoch (B1950) to Standard Julian Epoch (J2000) |
| Direction Cosine Matrix Body to Wind | Convert angle of attack and sideslip angle to direction cosine matrix |
| Direction Cosine Matrix Body to Wind to Alpha and Beta | Convert direction cosine matrix to angle of attack and sideslip angle |
| Direction Cosine Matrix ECEF to NED | Convert geodetic latitude and longitude to direction cosine matrix |
| Direction Cosine Matrix ECEF to NED to Latitude and Longitude | Convert direction cosine matrix to geodetic latitude and longitude |
| Direction Cosine Matrix to Euler Angles | Convert direction cosine matrix to Euler angles |
| Direction Cosine Matrix to Quaternions | Convert direction cosine matrix to quaternion vector |
| Direction Cosine Matrix to Wind Angles | Convert direction cosine matrix to wind angles |
| ECEF Position to LLA | Calculate geodetic latitude, longitude, and altitude above planetary ellipsoid from Earth-centered Earth-fixed (ECEF) position |

| | |
|---|---|
| Euler Angles to Direction Cosine Matrix | Convert Euler angles to direction cosine matrix |
| Euler Angles to Quaternions | Convert Euler angles to quaternion |
| Flat Earth to LLA | Estimate geodetic latitude, longitude, and altitude from flat Earth position |
| Force Conversion | Convert from force units to desired force units |
| Geocentric to Geodetic Latitude | Convert geocentric latitude to geodetic latitude |
| Geodetic to Geocentric Latitude | Convert geodetic latitude to geocentric latitude |
| Julian Epoch to Besselian Epoch | Transform position and velocity components from Standard Julian Epoch (J2000) to discontinued Standard Besselian Epoch (B1950) |
| LLA to ECEF Position | Calculate Earth-centered Earth-fixed (ECEF) position from geodetic latitude, longitude, and altitude above planetary ellipsoid |
| Quaternions to Direction Cosine Matrix | Convert quaternion vector to direction cosine matrix |
| Quaternions to Euler Angles | Convert quaternion vector to Euler angles |
| Wind Angles to Direction Cosine Matrix | Convert wind angles to direction cosine matrix |

## Math Operations

| | |
|---|---|
| 3x3 Cross Product | Calculate cross product of two 3-by-1 vectors |
| Adjoint of 3x3 Matrix | Compute adjoint of matrix |

| | |
|---|---|
| Create 3x3 Matrix | Create 3-by-3 matrix from nine input values |
| Determinant of 3x3 Matrix | Compute determinant of matrix |
| Invert 3x3 Matrix | Compute inverse of 3-by-3 matrix using determinant |
| Quaternion Division | Divide quaternion by another quaternion |
| Quaternion Inverse | Calculate inverse of quaternion |
| Quaternion Modulus | Calculate modulus of quaternion |
| Quaternion Multiplication | Calculate product of two quaternions |
| Quaternion Norm | Calculate norm of quaternion |
| Quaternion Normalize | Normalize quaternion |
| Quaternion Rotation | Rotate vector by quaternion |
| SinCos | Compute sine and cosine of angle |

## Unit Conversions

| | |
|---|---|
| Acceleration Conversion | Convert from acceleration units to desired acceleration units |
| Angle Conversion | Convert from angle units to desired angle units |
| Angular Acceleration Conversion | Convert from angular acceleration units to desired angular acceleration units |
| Angular Velocity Conversion | Convert from angular velocity units to desired angular velocity units |
| Density Conversion | Convert from density units to desired density units |
| Length Conversion | Convert from length units to desired length units |

| | |
|---|---|
| Mass Conversion | Convert from mass units to desired mass units |
| Pressure Conversion | Convert from pressure units to desired pressure units |
| Temperature Conversion | Convert from temperature units to desired temperature units |
| Velocity Conversion | Convert from velocity units to desired velocity units |

# 5

# Blocks — Alphabetical List

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

## Purpose

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

## Library

GNC/Controls

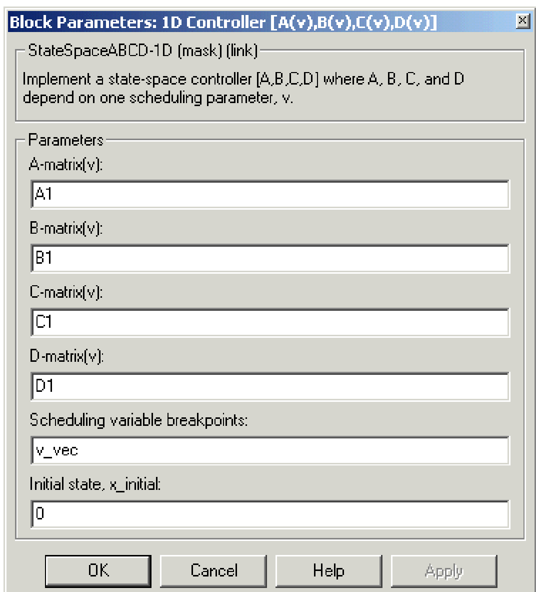
## Description


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

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

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.

## Dialog Box



**A-matrix(v)**

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

**B-matrix(v)**

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

**C-matrix(v)**

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

**D-matrix(v)**

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

**Scheduling variable breakpoints**

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

**Initial state, x_initial**

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

## Inputs and Outputs

The first input is the measurements.

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

The output is the actuator demands.

## Assumptions and Limitations

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

## Examples

See H-Infinity Controller (1 Dimensional Scheduling) in the `aeroblk_lib_HL20` demo library for an example of this block.

## See Also

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

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

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

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

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

## Purpose

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

## Library

GNC/Controls

## Description


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

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

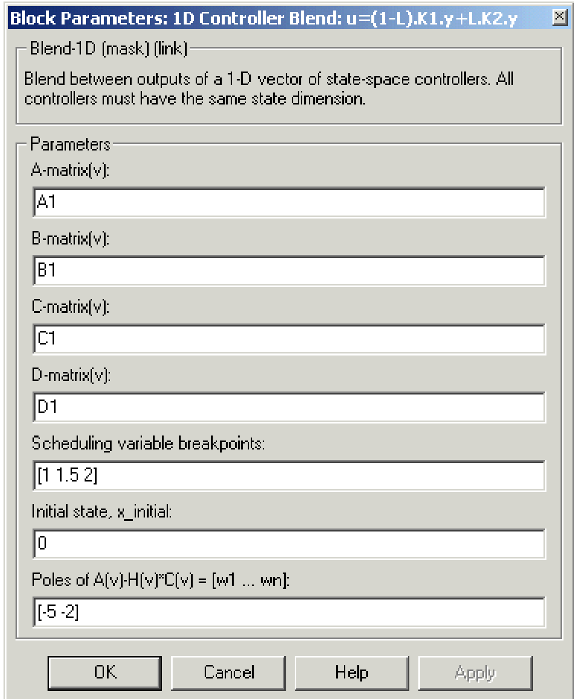
$$
\begin{aligned}
\dot{x}_1 &= A_1 x_1 + B_1 y \\
u_1 &= C_1 x_1 + D_1 y \\
\dot{x}_2 &= A_2 x_2 + B_2 y \\
u_2 &= C_2 x_2 + D_2 y \\
u &= (1-\lambda)u_1 + \lambda u_2
\end{aligned}
$$

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

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

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

## Dialog Box




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

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

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

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

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

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

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

## Inputs and Outputs

The first input is the measurements.

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

The output is the actuator demands.

## Assumptions and Limitations

---

**Note** This block requires Control System Toolbox.

---

## Reference

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

## See Also

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

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

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

2D Controller Blend

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

## Purpose

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

## Library

GNC/Controls

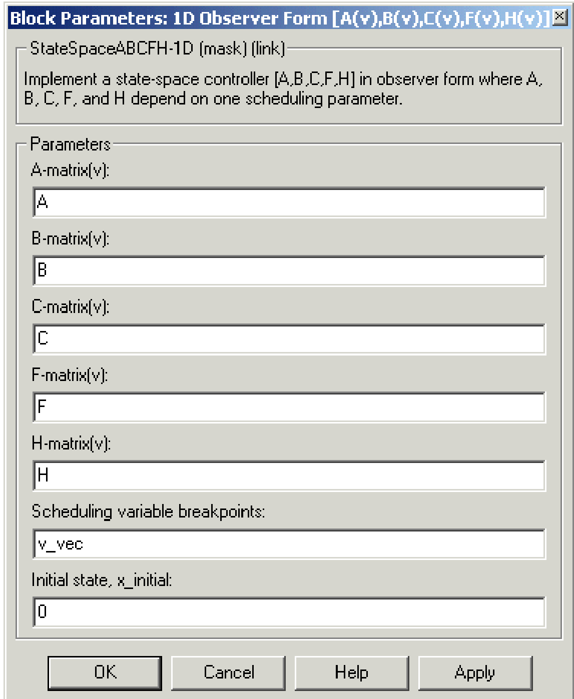
## Description


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

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

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

## Dialog Box




**A-matrix(v)**

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

**B-matrix(v)**

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

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

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

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

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

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

## Inputs and Outputs

The first input is the set-point error.

The second input is the scheduling variable.

The third input is measured actuator position.

The output is the actuator demands.

## Assumptions and Limitations

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

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

## Examples

See H-Infinity Controller (1 Dimensional Scheduling) in the `aeroblk_lib_HL20` demo library for an example of this block.

## Reference

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

## See Also

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

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

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

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

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

## Purpose

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

## Library

GNC/Controls

## Description


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

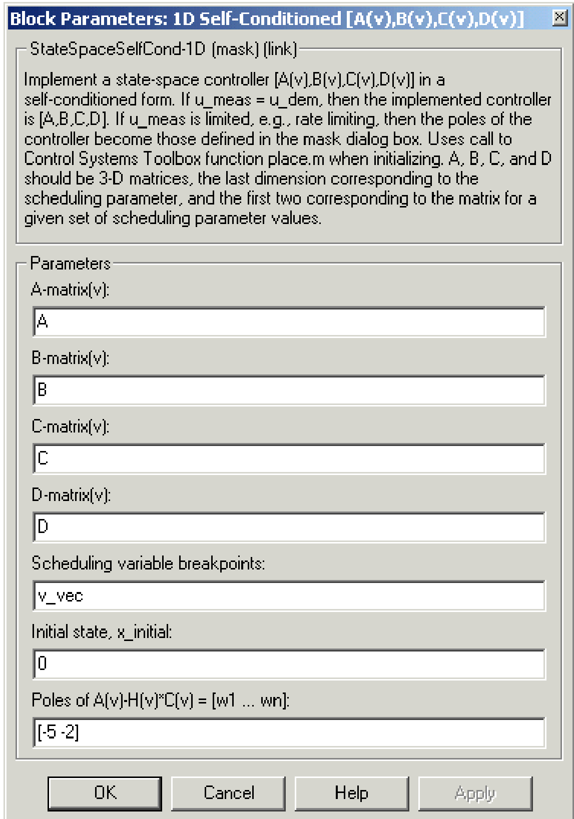
$$\dot{x} = A(v)x + B(v)y$$
$$u = C(v)x + D(v)y$$

in the self-conditioned form

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

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

## Dialog Box




**A-matrix(v)**

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

**B-matrix(v)**

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

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

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

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

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

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

## Inputs and Outputs

The first input is the measurements.

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

The third input is the measured actuator position.

The output is the actuator demands.

## Assumptions and Limitations

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

---

**Note** This block requires Control System Toolbox.

---

## Reference

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

## See Also

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

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

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

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

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

## Purpose

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

## Library

GNC/Controls

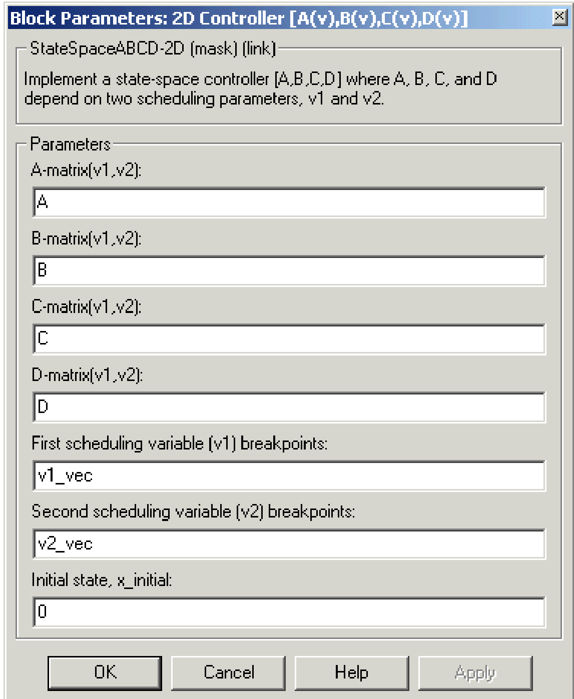
## Description


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

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

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.

## Dialog Box




**A-matrix(v1,v2)**

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. Hence, for example, if the A-matrix corresponding to the first entry of v1 and first entry of v2 is the identity matrix, then A(:,:,1,1) = [1 0;0 1];.

**B-matrix(v1,v2)**

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

**C-matrix(v1,v2)**

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

**D-matrix(v1,v2)**

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

**First scheduling variable (v1) breakpoints**

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

**Second scheduling variable (v2) breakpoints**

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

**Initial state, x_initial**

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

## Inputs and Outputs

The first input is the measurements.

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

The output is the actuator demands.

## Assumptions and Limitations

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

## Examples

See H-Infinity Controller (Two Dimensional Scheduling) in the `aeroblk_lib_HL20` demo library for an example of this block.

## See Also

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

2D Controller Blend

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

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

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

## Purpose

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

## Library

GNC/Controls

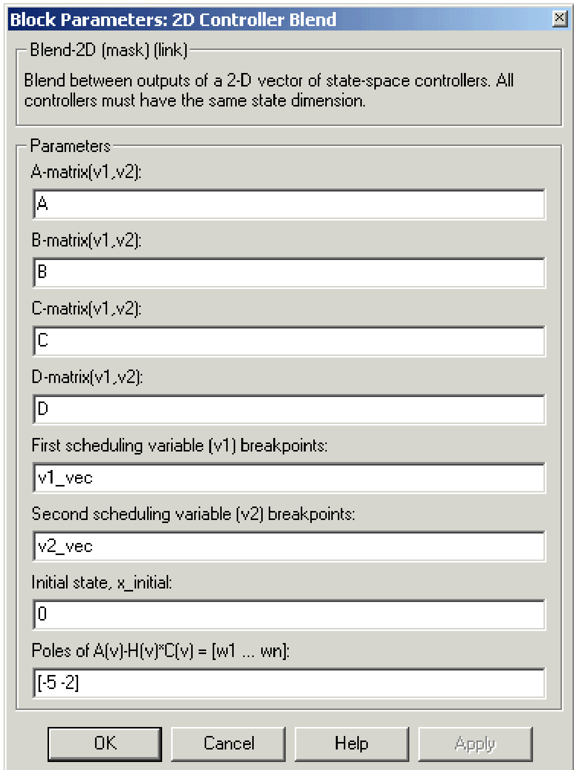
## Description


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

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

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

## Dialog Box




**A-matrix(v1,v2)**

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 v1 and v2. Hence, for example, if the A-matrix corresponding to the first entry of v1 and first entry of v2 is the identity matrix, then A(:,:,1,1) = [1 0;0 1];.

**B-matrix(v1,v2)**

B-matrix of the state-space implementation.

**C-matrix(v1,v2)**

C-matrix of the state-space implementation.

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

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

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

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

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

## Inputs and Outputs

The first input is the measurements.

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

The output is the actuator demands.

## Assumptions and Limitations

---

**Note** This block requires Control System Toolbox.

---

**Reference**

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

**See Also**

1D Controller 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)]

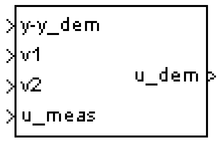
## Purpose

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

## Library

GNC/Controls

## Description

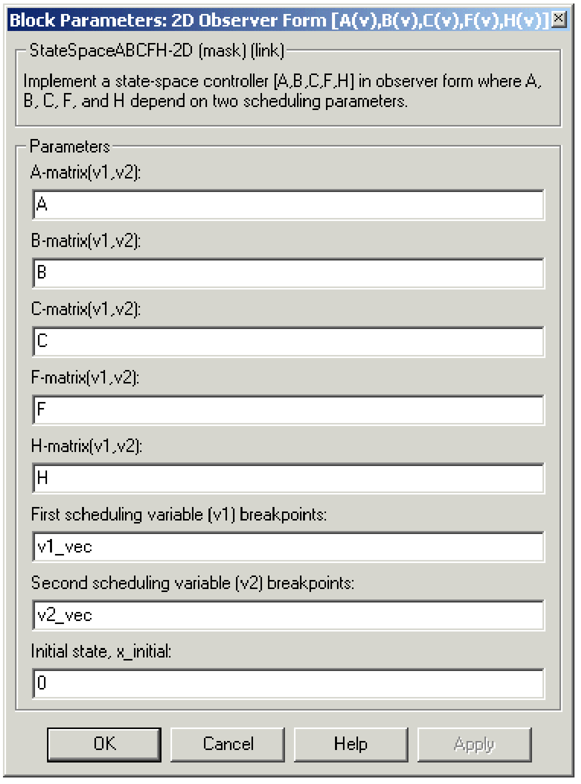



The 2D Observer Form [A(v),B(v),C(v),F(v),H(v)] block implements a gain-scheduled state-space controller defined in the following observer form:

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

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

## Dialog Box




**A-matrix(v1,v2)**

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. Hence, for example, if the A-matrix corresponding to the first entry of v1 and first entry of v2 is the identity matrix, then A(:,:,1,1) = [1 0;0 1];.

**B-matrix(v1,v2)**

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

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

**F-matrix(v1,v2)**
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. Hence, 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];.

**H-matrix(v1,v2)**
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. Hence, 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];.

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

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

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

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

## Inputs and Outputs

The first input is the set-point error.

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

The fourth input is the measured actuator position.

The output is the actuator demands.

## Assumptions and Limitations

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

## Examples

See H-Infinity Controller (Two Dimensional Scheduling) in the `aeroblk_lib_HL20` demo library for an example of this block.

## Reference

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

## See Also

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

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

2D Controller Blend

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

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

## Purpose

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

## Library

GNC/Controls

## Description


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

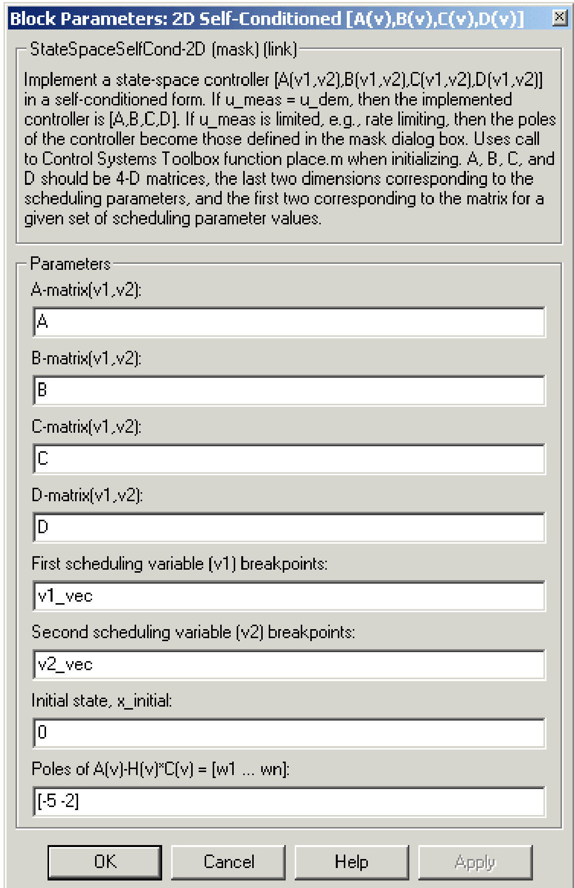
$$\dot{x} = A(v)x + B(v)y$$
$$u = C(v)x + D(v)y$$

in the self-conditioned form

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

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

## Dialog Box




**A-matrix(v1,v2)**

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. Hence, for example, if the A-matrix corresponding to the first entry of v1 and first entry of v2 is the identity matrix, then A(:,:,1,1) = [1 0;0 1];.

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

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

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

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

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

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

**Poles of A(v)-H(v)*C(v)**
Vector of the desired poles of A-HC. Note that the poles are assigned to the same locations for all values of the scheduling

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

### Inputs and Outputs

The first input is the measurements.

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

The fourth input is the measured actuator position.

The output is the actuator demands.

### Assumptions and Limitations

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

**Note** This block requires Control System Toolbox.

### Reference

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

### See Also

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

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

2D Controller Blend

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

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

## Purpose

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

## Library

GNC/Controls

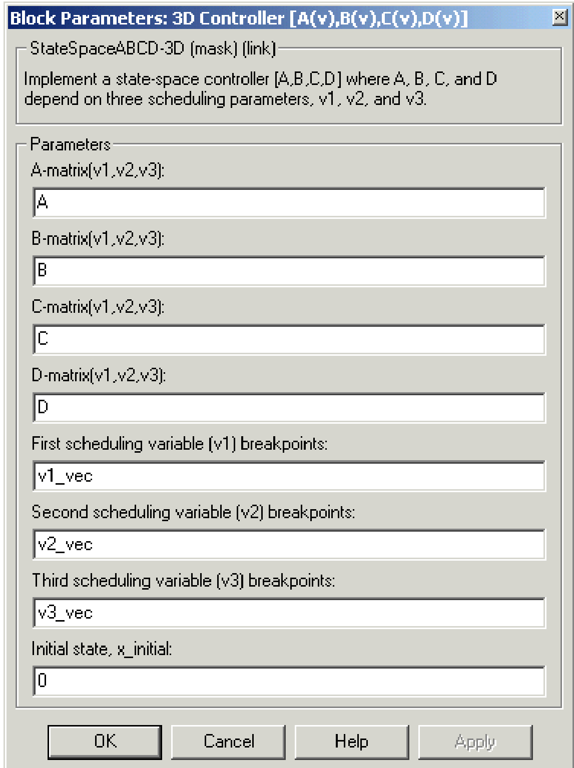
## Description


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

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

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.

## Dialog Box




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

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. Hence, for example, if the A-matrix corresponding to the first entry of v1, the first entry of v2, and the first entry of v3 is the identity matrix, then A(:,:,1,1,1) = [1 0 0;0 1 0; 0 0 1];.

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

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

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

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

**D-matrix(v1,v2,v3)**

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

**First scheduling variable (v1) breakpoints**

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

**Second scheduling variable (v2) breakpoints**

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

**Third scheduling variable (v3) breakpoints**

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

**Initial state, x_initial**

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

## Inputs and Outputs

The first input is the measurements.

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

The output is the actuator demands.

## Assumptions and Limitations

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

## See Also

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

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

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

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

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

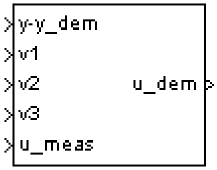
## Purpose

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

## Library

GNC/Controls

## Description



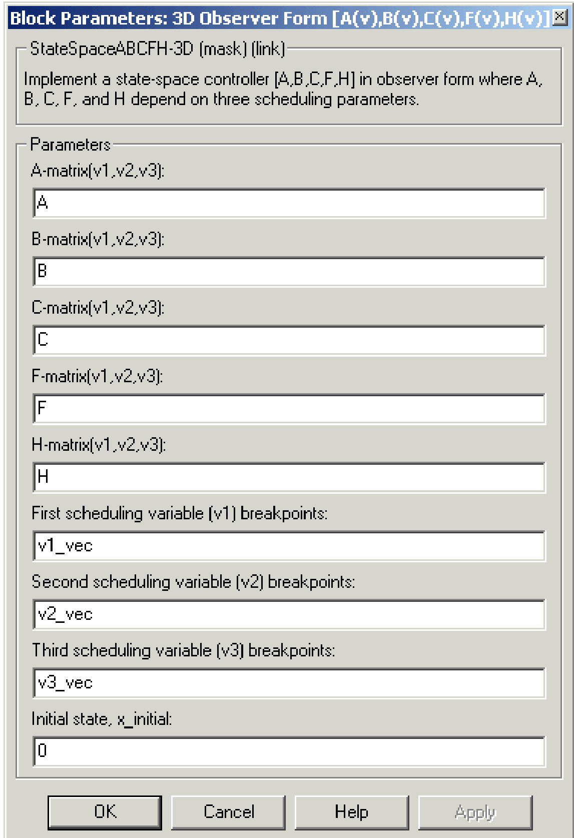

The 3D Observer Form [A(v),B(v),C(v),F(v),H(v)] block implements a gain-scheduled state-space controller defined in the following observer form:

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

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

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

## Dialog Box




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

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. Hence, for example, if the A-matrix corresponding to the first entry of v1, the first entry of v2, and the first entry of v3 is the identity matrix, then A(:,:,1,1,1) = [1 0;0 1];.

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

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. Hence, for example, if the B-matrix corresponding to the first entry of v1, the first entry of v2, and the first entry of v3 is the identity matrix, then B(:,:,1,1,1) = [1 0;0 1];.

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

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

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

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

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

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

**First scheduling variable (v1) breakpoints**

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

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

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

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

## Inputs and Outputs

The first input is the set-point error.

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

The fifth input is measured actuator position.

The output is the actuator demands.

## Assumptions and Limitations

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

## Reference

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

## See Also

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

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

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

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

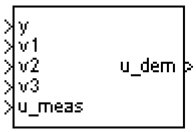
## Purpose

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

## Library

GNC/Controls

## Description




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

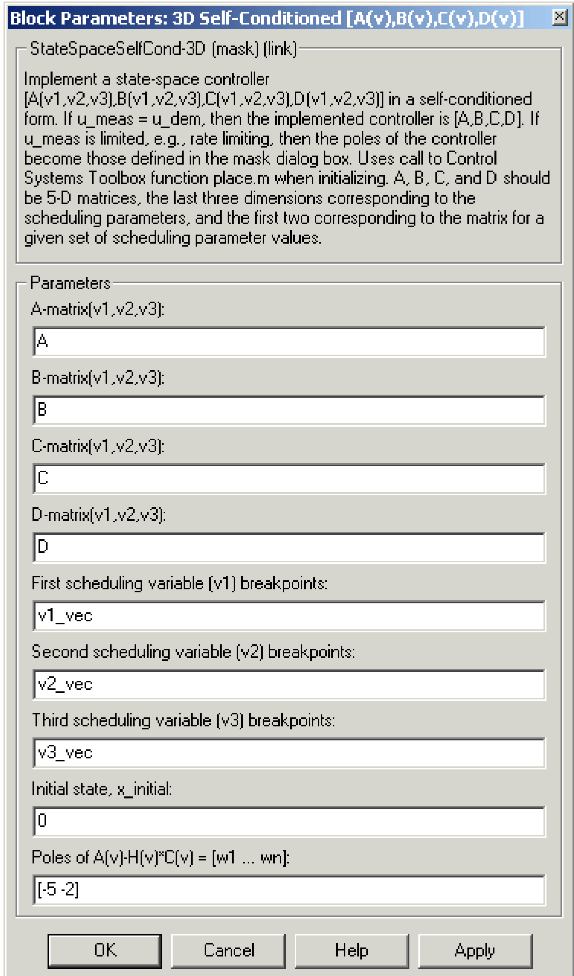
$$\dot{x} = A(v)x + B(v)y$$
$$u = C(v)x + D(v)y$$

in the self-conditioned form

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

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

## Dialog Box




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

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

**B-matrix(v1,v2,v3)**
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. Hence, for example, if the B-matrix corresponding to the first entry of v1, the first entry of v2, and the first entry of v3 is the identity matrix, then B(:,:,1,1,1) = [1 0;0 1];.

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

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

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

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

**Third scheduling variable (v3) breakpoints**

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

**Initial state, x_initial**

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

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

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

## Inputs and Outputs

The first input is the measurements.

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

The fifth input is the measured actuator position.

The output is the actuator demands.

## Assumptions and Limitations

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

**Note** This block requires Control System Toolbox.

## Reference

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

## See Also

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

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

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

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

# 3DoF Animation

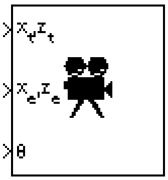
## Purpose

Create 3-D MATLAB Graphics animation of three-degrees-of-freedom object

## Library

Animation

## Description

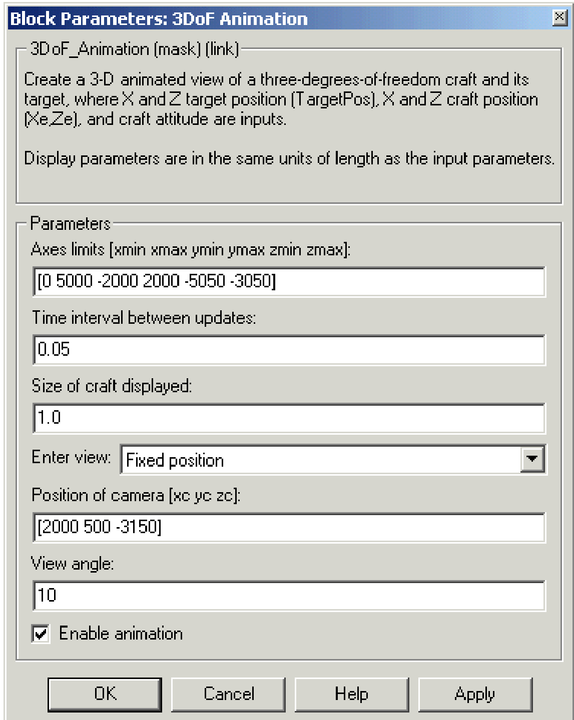


The 3DoF Animation block displays a 3-D animated view of a three-degrees-of-freedom (3DoF) craft, its trajectory, and its target using MATLAB Graphics.

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

This block does not produce deployable code, but can be used with Real-Time Workshop external mode as a SimViewingDevice.

## Dialog Box




**Axes limits [xmin xmax ymin ymax zmin zmax]**
Specifies the three-dimensional space to be viewed.

**Time interval between updates**
Specifies the time interval at which the animation is redrawn.

**Size of craft displayed**
Scale factor to adjust the size of the craft and target.

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

- Fixed position
- Cockpit
- Fly alongside

**Position of camera [xc yc zc]**
Specifies the MATLAB Graphics parameter **CameraPosition** for the figure axes. Used in all cases except for the Cockpit view.

**View angle**
Specifies the MATLAB Graphics parameter **CameraViewAngle** for the figure axes in degrees.

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

## Inputs

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

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

The third input is the attitude of the craft.

## Examples

See the `aero_guidance` demo for an example of this block.

## See Also

6DoF Animation

FlightGear Preconfigured 6DoF Animation

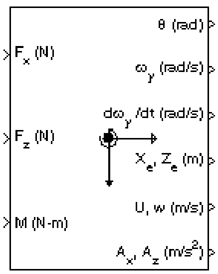
## Purpose

Implement three-degrees-of-freedom equations of motion with respect to body axes

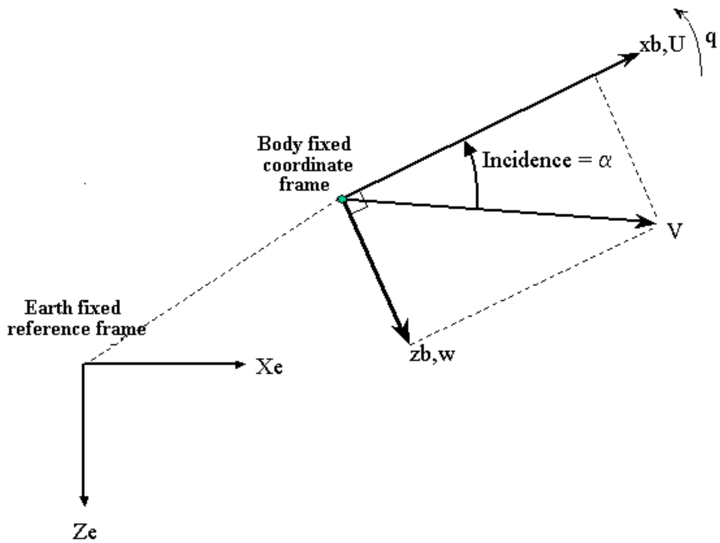
## Library

Equations of Motion/3DoF

## Description




The 3DoF (Body Axes) block considers the rotation in the vertical plane of a body-fixed coordinate frame about an Earth-fixed reference frame.




The equations of motion are

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

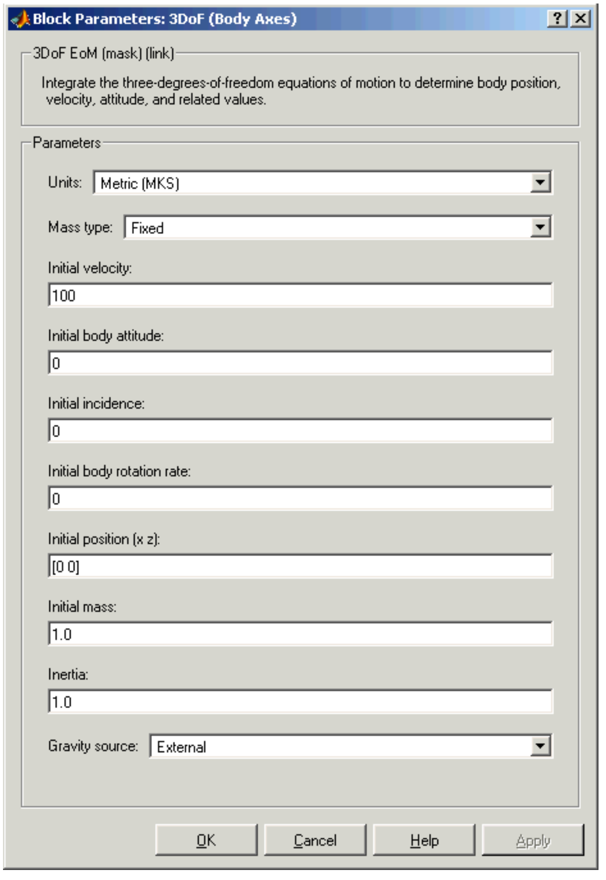
$$\dot{w} = \frac{F_z}{m} + qu + g\cos\theta$$

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

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

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

## Dialog Box




**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 Earth-fixed 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.

## Inputs and Outputs

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

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

The third input to the block is the applied pitch moment, ($M$).

The fourth optional input to the block is gravity in the selected units.

The first output from the block is the pitch attitude, in radians $(\theta)$.

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

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

The fourth output is a two-element vector containing the location of the body, in the Earth-fixed reference frame, (*Xe,Ze*).

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

The sixth output is a two-element vector containing the acceleration of the body resolved into the body-fixed coordinate frame, $(Ax,Az)$.

## Examples

See the `aero_guidance` demo 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)

# 3DoF (Wind Axes)

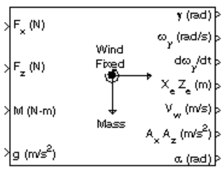
## Purpose

Implement three-degrees-of-freedom equations of motion with respect to wind axes

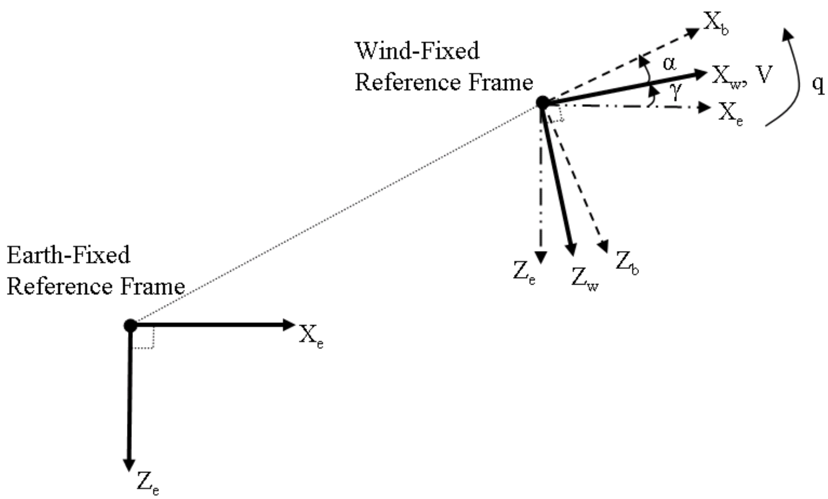
## Library

Equations of Motion/3DoF

## Description




The 3DoF (Wind Axes) block considers the rotation in the vertical plane of a wind-fixed coordinate frame about an Earth-fixed reference frame.




The equations of motion are

$$\dot{V} = \frac{F_{x_{wind}}}{m} - g\sin\gamma$$

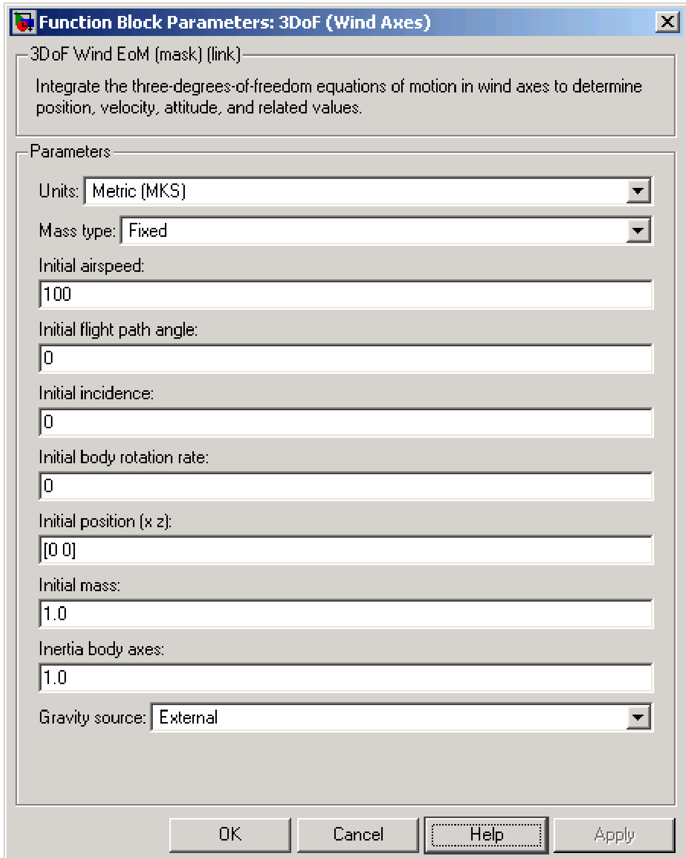
$$\dot{\alpha} = \frac{F_{z_{wind}}}{mV\cos\beta} + q + \frac{g}{V\cos\beta}\cos\gamma$$

$$\dot{q} = \dot{\theta} = \frac{M_{y_{body}}}{I_{yy}}$$

$$\dot{\gamma} = q - \dot{\alpha}$$

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

## Dialog Box




**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 Earth-fixed 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.

## Inputs and Outputs

The first input to the block is the force acting along the wind $x$-axis, $(F_x)$.

The second input to the block is the force acting along the wind $z$-axis, $(F_z)$.

The third input to the block is the applied pitch moment in body axes, $(M)$.

The fourth optional input to the block is gravity in the selected units.

The first output from the block is the flight path angle, in radians $(\gamma)$.

The second output is the pitch angular rate, in radians per second $(\omega_y)$.

The third output is the pitch angular acceleration, in radians per second squared $(d\omega_y/dt)$.

The fourth output is a two-element vector containing the location of the body, in the Earth-fixed reference frame, $(Xe,Ze)$.

The fifth output is a two-element vector containing the velocity of the body resolved into the wind-fixed coordinate frame, ($V$,0).

The sixth output is a two-element vector containing the acceleration of the body resolved into the body-fixed coordinate frame, ($Ax,Az$).

The seventh output is a scalar containing the angle of attack, $(\alpha)$.

## 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)

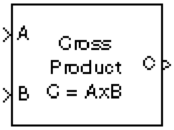
## Purpose

Calculate cross product of two 3-by-1 vectors

## Library

Utilities/Math Operations

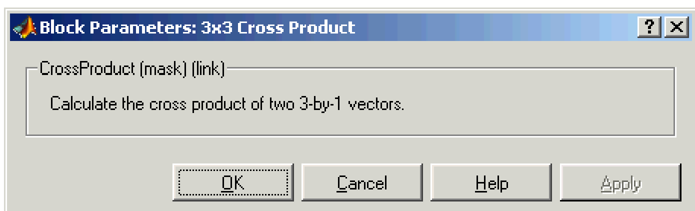
## Description




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

$$
\begin{aligned}
A &= a_1\boldsymbol{i} + a_2\boldsymbol{j} + a_3\boldsymbol{k} \\
B &= b_1\boldsymbol{i} + b_2\boldsymbol{j} + b_3\boldsymbol{k} \\
C &= A \times B = \begin{vmatrix} \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix} \\
&= (a_2 b_3 - a_3 b_2)\boldsymbol{i} + (a_3 b_1 - a_1 b_3)\boldsymbol{j} + (a_1 b_2 - a_2 b_1)\boldsymbol{k}
\end{aligned}
$$

## Dialog Box




## Inputs and Outputs

The inputs are two 3-by-1 vectors.

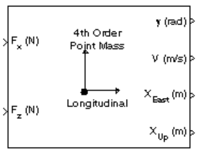
The output is a 3-by-1 vector.

# 4th Order Point Mass (Longitudinal)

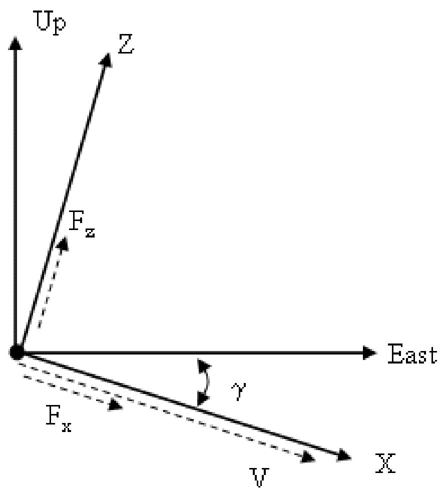
**Purpose** Calculate fourth-order point mass

**Library** 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.




The translational motions of the point mass $[X_{East}\ X_{Up}]^T$ are functions of airspeed $(V)$ and flight path angle $(\gamma)$,

$$F_x = m\dot{V}$$

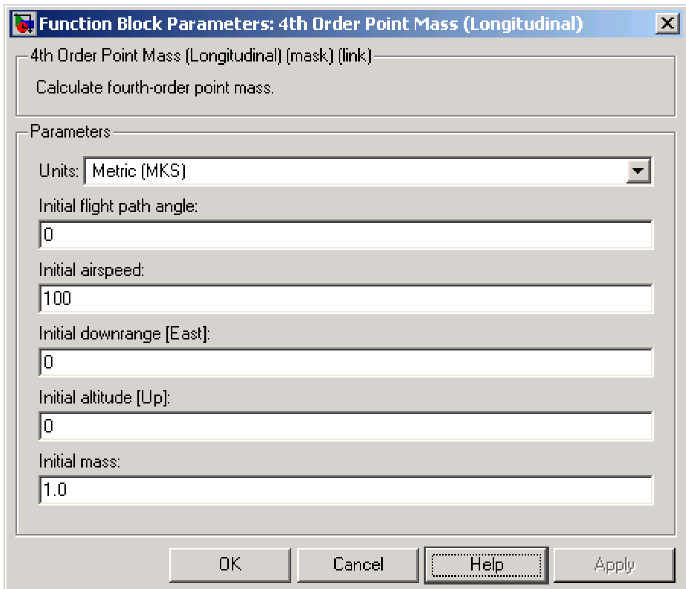
$$F_z = mV\dot{\gamma}$$

$$\dot{X}_{East} = V\cos\gamma$$

$$X_{Up} = V\sin\gamma$$

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.

## Dialog Box




**Units**

Specifies the input and output units:

| Units | Forces | Velocity | Position |
|---|---|---|---|
| `Metric (MKS)` | Newton | Meters per second | Meters |

| Units | Forces | Velocity | Position |
|---|---|---|---|
| `English (Velocity in ft/s)` | Pound | Feet per second | Feet |
| `English (Velocity in kts)` | Pound | Knots | Feet |

**Initial flight path angle**
The scalar or vector containing the initial flight path angle of the point mass(es).

**Initial airspeed**
The scalar or vector containing the initial airspeed of the point mass(es).

**Initial downrange**
The scalar or vector containing the initial downrange of the point mass(es).

**Initial altitude**
The scalar or vector containing the initial altitude of the point mass(es).

**Initial mass**
The scalar or vector containing the mass of the point mass(es).

## Inputs and Outputs

The first input is force in $x$-axis in selected units.

The second input is force in $z$-axis in selected units.

The first output is flight path angle in radians.

The second output is airspeed in selected units.

The third output is the downrange or amount traveled East in selected units.

The fourth output is the altitude or amount traveled Up in selected units.

## Assumptions and Limitations

The flat Earth-fixed 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.

## See Also

4th Order Point Mass Forces (Longitudinal)

3DoF (Body Axes)

3DoF (Wind Axes)

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)

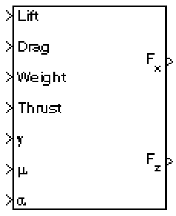
Simple Variable Mass 3DoF (Wind Axes)

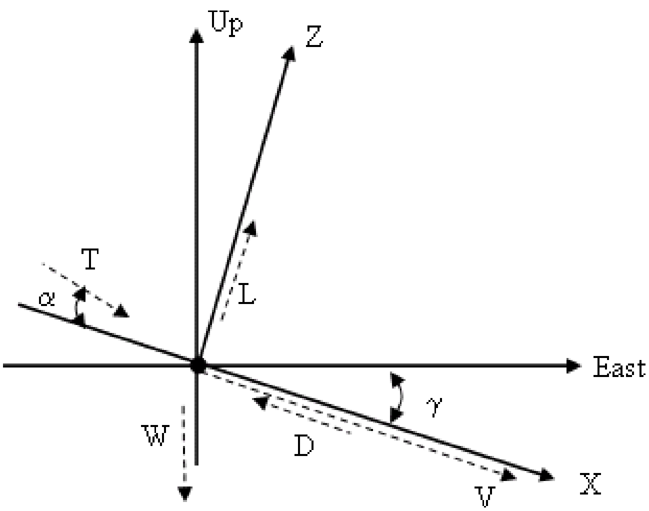
# 4th Order Point Mass Forces (Longitudinal)

**Purpose** Calculate forces used by fourth-order point mass

**Library** 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.



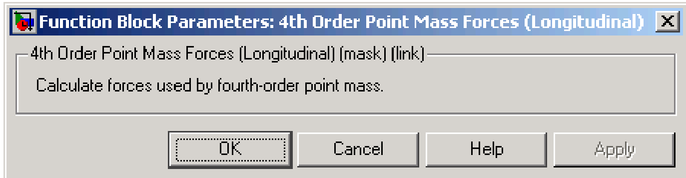




The applied forces $[F_x\ F_z]^T$ are in a system defined as follows: *x*-axis is in the direction of vehicle velocity relative to air, *z*-axis is upward, and *y*-axis completes the right-handed frame. They are functions of lift $(L)$, drag $(D)$, thrust $(T)$, weight $(W)$, flight path angle $(\gamma)$, angle of attack $(\alpha)$, and bank angle $(\mu)$.

$$F_z = (L + T\sin\alpha)\cos\mu - W\cos\gamma$$

$$F_x = T\cos\alpha - D - W\sin\gamma$$

## Dialog Box




## Inputs and Outputs

The first input is lift in units of force.

The second input is drag in units of force.

The third input is weight in units of force.

The fourth input is thrust in units of force.

The fifth input is flight path angle in radians.

The sixth input is bank angle in radians.

The seventh input is angle of attack in radians.

The first output is force in $x$-axis in units of force.

The second output is force in $z$-axis in units of force.

## Assumptions and Limitations

The flat Earth-fixed 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.

## See Also

4th Order Point Mass (Longitudinal)

6th Order Point Mass (Coordinated Flight)

6th Order Point Mass Forces (Coordinated Flight)

# 6DoF Animation

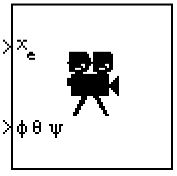
## Purpose

Create 3-D MATLAB Graphics animation of six-degrees-of-freedom object

## Library

Animation

## Description

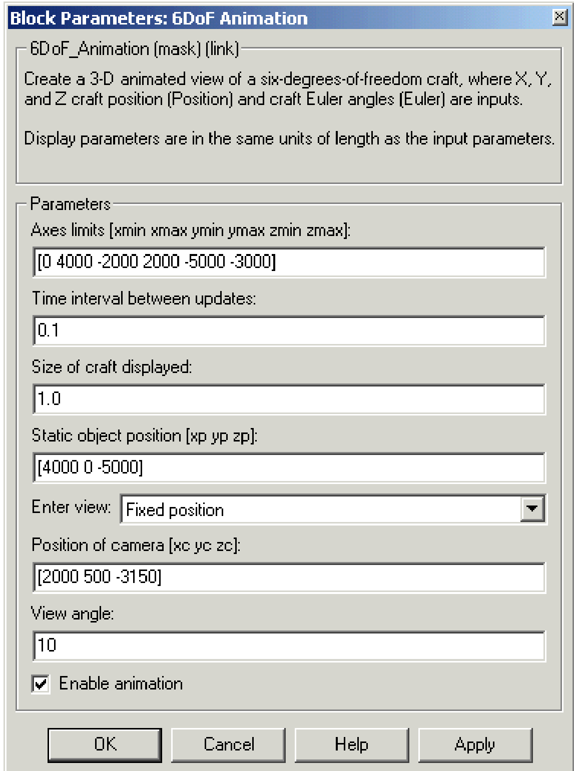



The 6DoF Animation block displays a 3-D animated view of a six-degrees-of-freedom (6DoF) craft, its trajectory, and its target using MATLAB Graphics.

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

This block does not produce deployable code, but can be used with Real-Time Workshop external mode as a SimViewingDevice.

## Dialog Box




**Axes limits [xmin xmax ymin ymax zmin zmax]**
: Specifies the three-dimensional space to be viewed.

**Time interval between updates**
: Specifies the time interval at which the animation is redrawn.

**Size of craft displayed**
: Scale factor to adjust the size of the craft and target.

**Static object position**
: Specifies the altitude, the crossrange position, and the downrange position of the target.

**Enter view**

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

- `Fixed position`
- `Cockpit`
- `Fly alongside`

**Position of camera [xc yc zc]**

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

**View angle**

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

**Enable animation**

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

## Inputs

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

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

## Examples

See the `aeroblk_vmm` demo for an example of this block.

## See Also

3DoF Animation

FlightGear Preconfigured 6DoF Animation

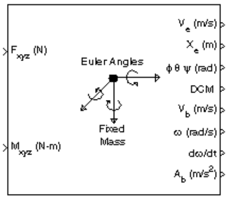
## Purpose

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

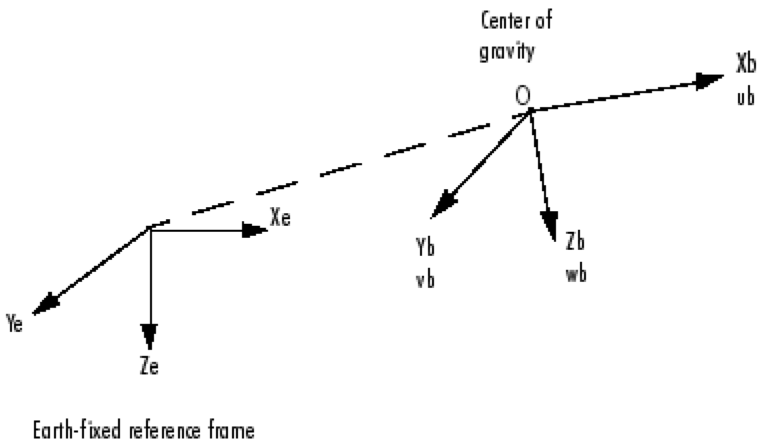
## Library

Equations of Motion/6DoF

## Description




The 6DoF (Euler Angles) block considers the rotation of a body-fixed coordinate frame $(X_b, Y_b, Z_b)$ about an Earth-fixed reference frame $(X_e, Y_e, Z_e)$. The origin of the body-fixed coordinate frame is the center of gravity of the body, and the body is assumed to be rigid, an assumption that eliminates the need to consider the forces acting between individual elements of mass. The Earth-fixed reference frame is considered inertial, 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 body-fixed coordinate frame is given below, where the applied forces $[F_x\ F_y\ F_z]^T$ are in the body-fixed frame, and the mass of the body $m$ is assumed constant.

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

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

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

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

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

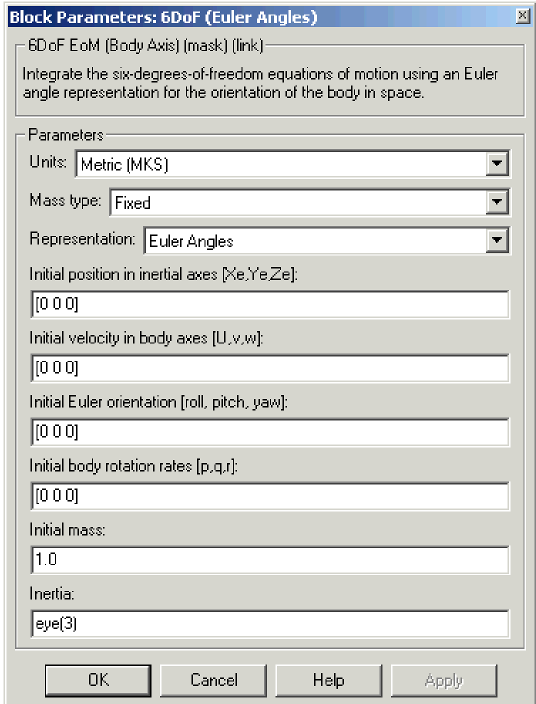
The relationship between the body-fixed angular velocity vector, $[p\ q\ r]^T$, and the rate of change of the Euler angles, $[\dot{\phi}\ \dot{\theta}\ \dot{\psi}]^T$, can be determined by resolving the Euler rates into the body-fixed coordinate frame.

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

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

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

## Dialog Box




**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 (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 `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 Earth-fixed 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**
The 3-by-3 inertia tensor matrix $I$.

## Inputs and Outputs

The first input to the block is a vector containing the three applied forces.

The second input is a vector containing the three applied moments.

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

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

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

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

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

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

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

The eighth output is a three-element vector containing the accelerations in body-fixed axes.

## 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.

## Examples

See the `aeroblk_six_dof` airframe in the `aeroblk_HL20` demo and the `asbhl20` demo for examples of this block.

## Reference

Mangiacasale, L., *Flight Mechanics of a* μ*-Airplane with a MATLAB Simulink Helper,* Edizioni Libreria CLUP, Milan, 1998.

## See Also

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)

Simple Variable Mass 6DoF Wind (Wind Angles)

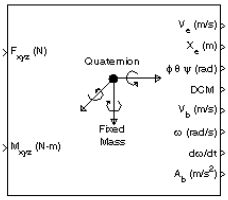
## Purpose

Implement quaternion representation of six-degrees-of-freedom equations of motion with respect to body axes

## Library

Equations of Motion/6DoF

## Description



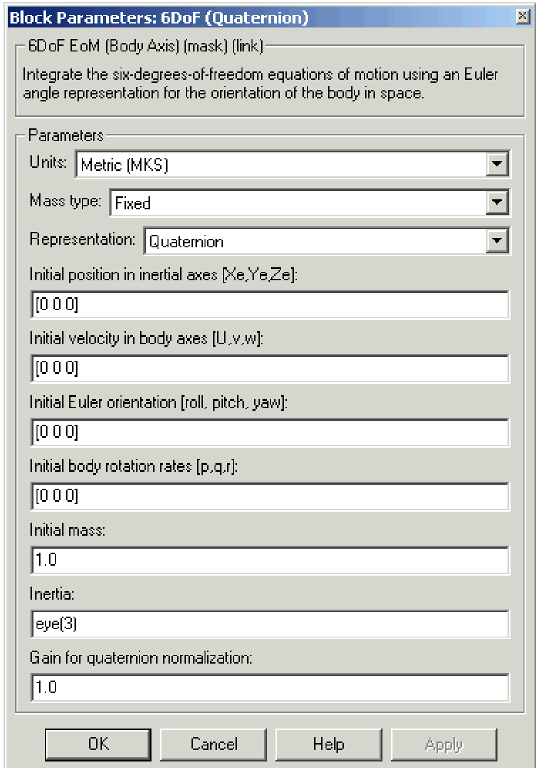

For a description of the coordinate system employed and the translational dynamics, see the block description for the 6DoF (Euler Angles) block.

The integration of the rate of change of the quaternion vector is given below. The gain $K$ drives the norm of the quaternion state vector to 1.0 should $\epsilon$ 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.

$$\begin{bmatrix} \dot{q}_0 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 0 & -p & -q & -r \\ p & 0 & r & -q \\ q & -r & 0 & p \\ r & q & -p & 0 \end{bmatrix} \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} + K\varepsilon \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix}$$

$$\varepsilon = 1 - (q_0^2 + q_1^2 + q_3^2 + q_4^2)$$

## Dialog Box




**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 (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 Earth-fixed 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.

## Inputs and Outputs

The first input to the block is a vector containing the three applied forces.

The second input is a vector containing the three applied moments.

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

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

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

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

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

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

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

The eighth output is a three-element vector containing the accelerations in body-fixed axes.

## 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

Mangiacasale, L., *Flight Mechanics of a μ-Airplane with a MATLAB Simulink Helper,* Edizioni Libreria CLUP, Milan, 1998.

## 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)

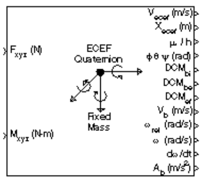
## Purpose

Implement quaternion representation of six-degrees-of-freedom equations of motion in Earth-centered Earth-fixed (ECEF) coordinates

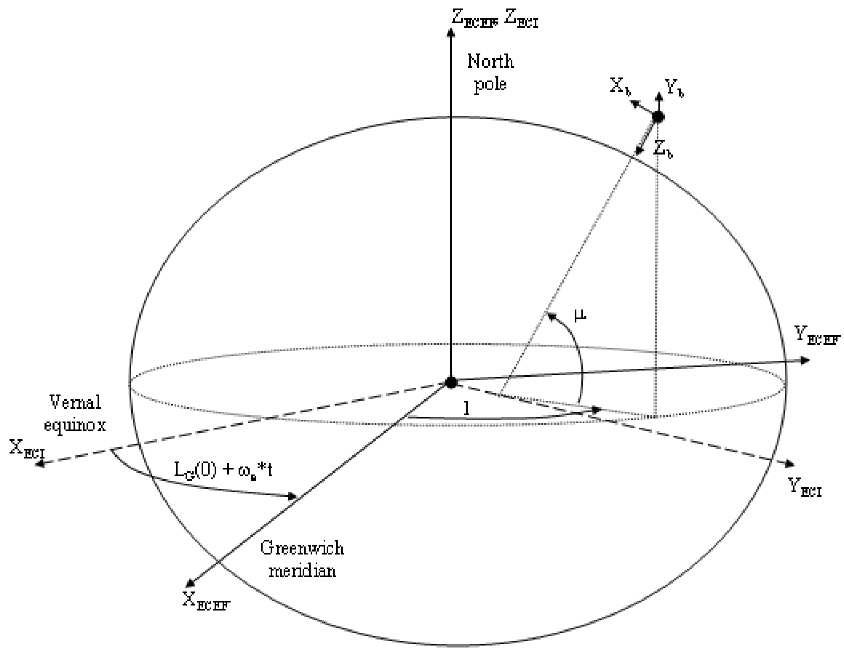
## Library

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.

$$\bar{F}_b = \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} = m(\dot{\bar{V}}_b + \bar{\omega}_b \times \bar{V}_b + DCM_{bf}\bar{\omega}_e \times \bar{V}_b) + DCM_{bf}(\bar{\omega}_e \times (\bar{\omega}_e \times \bar{X}_f))$$

where the change of position in ECEF $\underline{\dot{x}_f}$ is calculated by

$$\dot{x}_f = DCM_{fb}\bar{V}_b$$

and the velocity of the body with respect to ECEF frame, expressed in body frame($\underline{V_b}$), angular rates of the body with respect to ECI frame, expressed in body frame $(\underline{\omega}_b)$. Earth rotation rate $(\underline{\omega}_e)$, and relative angular rates of the body with respect to north-east-down (NED) frame, expressed in body frame $(\underline{\omega}_{rel})$ are defined as

$$\bar{V}_b = \begin{bmatrix} u \\ v \\ \omega \end{bmatrix}, \bar{\omega}_{rel} = \begin{bmatrix} p \\ q \\ r \end{bmatrix}, \bar{\omega}_e = \begin{bmatrix} 0 \\ 0 \\ \omega_e \end{bmatrix}, \bar{\omega}_b = \bar{\omega}_{rel} + DCM_{bf}\bar{\omega}_e + DCM_{be}\bar{\omega}_{ned}$$

$$\bar{\omega}_{ned} = \begin{bmatrix} \dot{l}\cos\mu \\ -\dot{\mu} \\ -\dot{l}\sin\mu \end{bmatrix} = \begin{bmatrix} V_E/(N+h) \\ -V_N/(M+h) \\ V_E \bullet \tan\mu/(N+h) \end{bmatrix}$$

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.

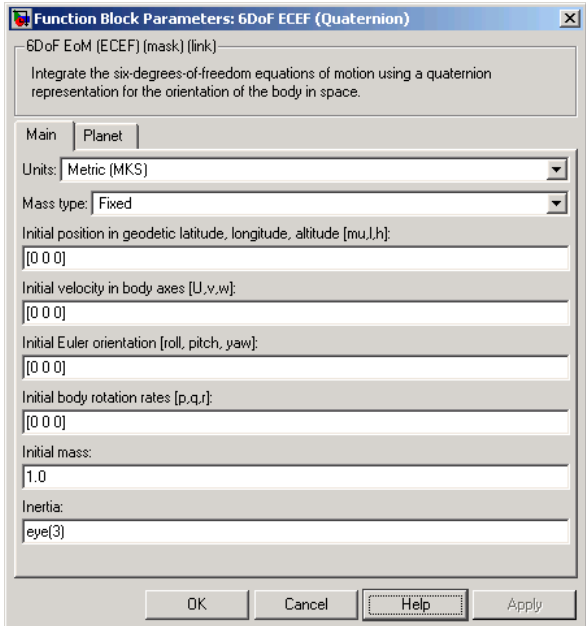
$$\underline{M}_b = \begin{bmatrix} L \\ M \\ N \end{bmatrix} = I\dot{\underline{\omega}}_b + \underline{\omega}_b \times (I\underline{\omega}_b)$$

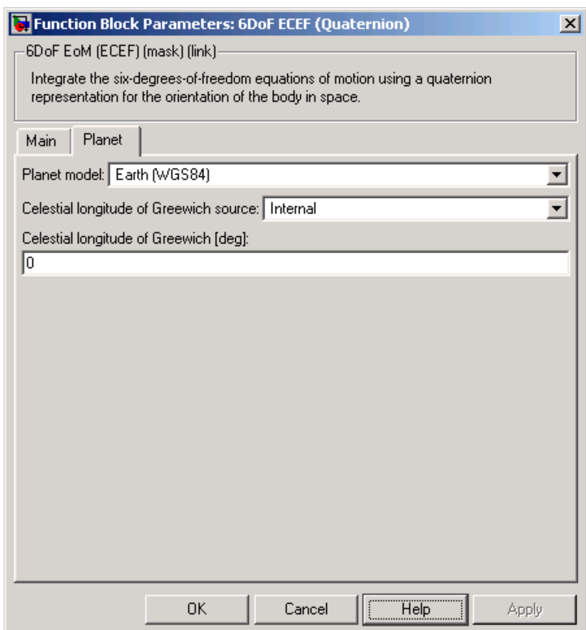
$$I = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{yx} & I_{yy} & -I_{yz} \\ -I_{zx} & -I_{zy} & I_{zz} \end{bmatrix}$$

The integration of the rate of change of the quaternion vector is given below.

$$\begin{bmatrix} \dot{q}_0 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} = -\frac{1}{2} \begin{bmatrix} 0 & \omega_b(1) & \omega_b(2) & \omega_b(3) \\ -\omega_b(1) & 0 & -\omega_b(3) & \omega_b(2) \\ -\omega_b(2) & \omega_b(3) & 0 & -\omega_b(1) \\ -\omega_b(3) & -\omega_b(2) & \omega_b(1) & 0 \end{bmatrix} \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix}$$

## Dialog Box


Function Block Parameters: 6DoF ECEF (Quaternion)
6DoF EoM (ECEF) (mask) (link)
Integrate the six-degrees-of-freedom equations of motion using a quaternion representation for the orientation of the body in space.
Main
Planet
Units: Metric (MKS)
Mass type: Fixed
Initial position in geodetic latitude, longitude, altitude [mu,l,h]:
[0 0 0]
Initial velocity in body axes [U,v,w]:
[0 0 0]
Initial Euler orientation [roll, pitch, yaw]:
[0 0 0]
Initial body rotation rates [p,q,r]:
[0 0 0]
Initial mass:
1.0
Inertia:
eye(3)
OK
Cancel
Help
Apply


Function Block Parameters: 6DoF ECEF (Quaternion)
6DoF EoM (ECEF) (mask) (link)
Integrate the six-degrees-of-freedom equations of motion using a quaternion representation for the orientation of the body in space.
Main
Planet
Planet model: Earth (WGS84)
Celestial longitude of Greewich source: Internal
Celestial longitude of Greewich [deg]:
0
OK
Cancel
Help
Apply

**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 (see Simple Variable Mass 6DoF ECEF (Quaternion)). |
| `Custom Variable` | Mass and inertia variations are customizable (see Custom Variable Mass 6DoF ECEF (Quaternion)). |

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.

**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.

**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.

## Inputs and Outputs

| Input | Dimension Type | Description |
|---|---|---|
| First input | Vector | Contains the three applied forces in body-fixed axes. |
| Second input | Vector | Contains the three applied moments in body-fixed axes. |

| Output | Dimension Type | Description |
|---|---|---|
| First output | Vector | Contains the velocity of the body with respect to ECEF frame, expressed in ECEF frame. |
| Second output | Three-element vector | Contains the position in ECEF reference frame. |
| Third output | Three-element vector | Contains the position in geodetic latitude, longitude and altitude, in degrees, degrees and selected units of length respectively. |

| Output | Dimension Type | Description |
|---|---|---|
| Fourth output | 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 output | 3-by-3 matrix | Applies to the coordinate transformation from ECI axes to body-fixed axes |
| Sixth output | 3-by-3 matrix | Applies to the coordinate transformation from geodetic axes to body-fixed axes. |
| Seventh output | 3-by-3 matrix | Applies to the coordinate transformation from ECEF axes to geodetic axes. |
| Eighth output | Three-element vector | Contains the velocity of the body with respect to ECEF frame, expressed in the body frame.. |
| Ninth output | 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. |
| Tenth output | Three-element vector | Contains the angular rates of the body with respect to the ECI frame, expressed in body frame, in radians per second. |

| Output | Dimension Type | Description |
|---|---|---|
| Eleventh output | Three-element vector | Contains the angular accelerations of the body with respect to ECI frame, expressed in the body frame, in radians per second. |
| Twelfth output | Three-element vector | Contains the accelerations in body-fixed axes. |

## 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 through the center of the Sun 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)

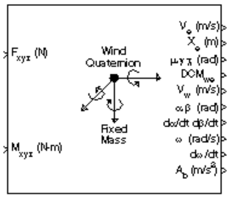
## Purpose

Implement quaternion representation of six-degrees-of-freedom equations of motion with respect to wind axes

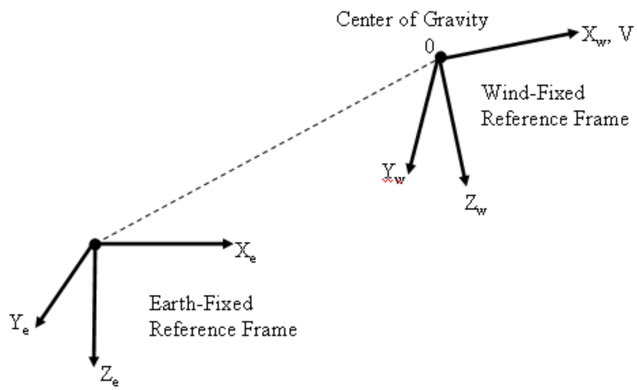
## Library

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 Earth-fixed 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 Earth-fixed 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.

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

$$\underline{V}_w = \begin{bmatrix} V \\ 0 \\ 0 \end{bmatrix}, \underline{\omega}_w = \begin{bmatrix} p_w \\ q_w \\ r_w \end{bmatrix} = DMC_{wb} \begin{bmatrix} p_b - \dot{\beta}\sin\alpha \\ q_b - \dot{\alpha} \\ r_b + \dot{\beta}\cos\alpha \end{bmatrix}, \underline{\omega}_b = \begin{bmatrix} p_b \\ q_b \\ r_b \end{bmatrix}$$

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

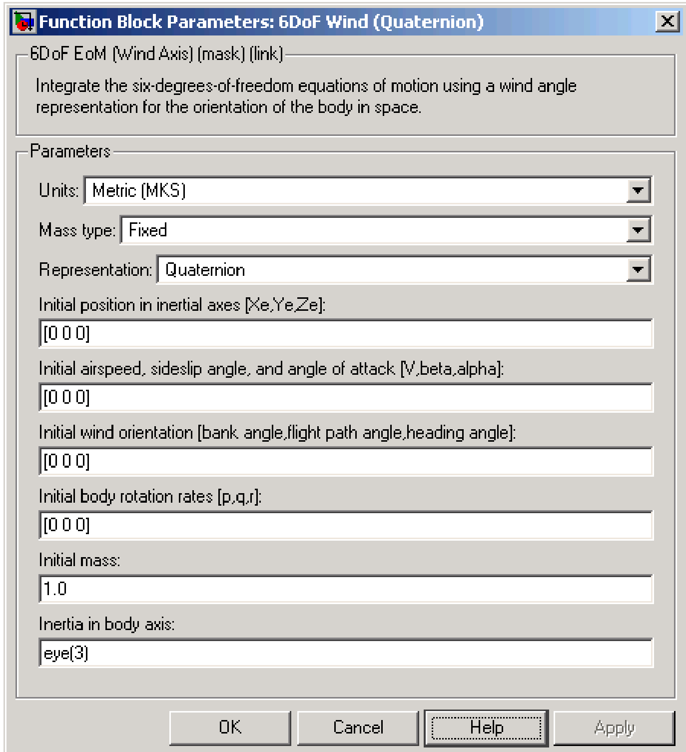
$$\underline{M}_b = \begin{bmatrix} L \\ M \\ N \end{bmatrix} = I\dot{\underline{\omega}}_b + \underline{\omega}_b \times (I\underline{\omega}_b)$$

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

The integration of the rate of change of the quaternion vector is given below.

$$\begin{bmatrix} \dot{q}_0 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} = -\frac{1}{2} \begin{bmatrix} 0 & p & q & r \\ -p & 0 & -r & q \\ -q & r & 0 & -p \\ -r & -q & p & 0 \end{bmatrix} \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix}$$

## Dialog Box




**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 (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 `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 Earth-fixed 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.

## Inputs and Outputs

The first input to the block is a vector containing the three applied forces in wind-fixed axes.

The second input is a vector containing the three applied moments in body-fixed axes.

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

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

The third output is a three-element vector containing the wind rotation angles [bank, flight path, heading], in radians.

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

The fifth output is a three-element vector containing the velocity in the wind-fixed frame.

The sixth output is a two-element vector containing the angle of attack and sideslip angle, in radians.

The seventh output is a two-element vector containing the rate of change of angle of attack and rate of change of sideslip angle, in radians per second.

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

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

The tenth output is a three-element vector containing the accelerations in body-fixed axes.

## 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)

# 6DoF Wind (Wind Angles)

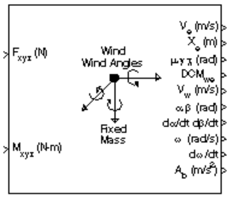
## Purpose

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.

$$\begin{bmatrix} \dot{\mu} \\ \dot{\gamma} \\ \dot{\chi} \end{bmatrix} = J \begin{bmatrix} p_w \\ q_w \\ r_w \end{bmatrix} = \begin{bmatrix} 1 & (\sin\mu\tan\gamma) & (\cos\mu\tan\gamma) \\ 0 & \cos\mu & -\sin\mu \\ 0 & \dfrac{\sin\mu}{\cos\gamma} & \dfrac{\cos\mu}{\cos\gamma} \end{bmatrix} \begin{bmatrix} p_w \\ q_w \\ r_w \end{bmatrix}$$

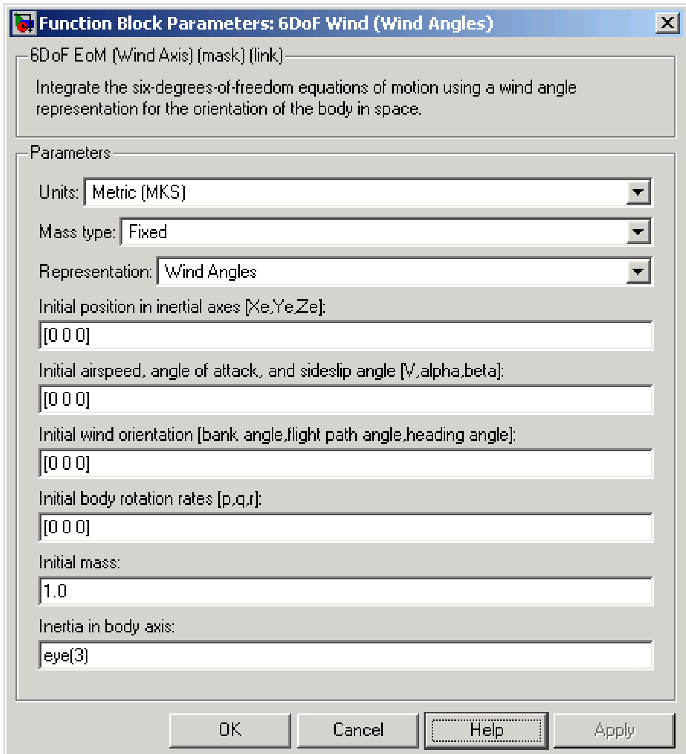
The body-fixed angular rates are related to the wind-fixed angular rate by the following equation.

$$\begin{bmatrix} p_w \\ q_w \\ r_w \end{bmatrix} = DMC_{wb} \begin{bmatrix} p_b - \dot{\beta}\sin\alpha \\ q_b - \dot{\alpha} \\ r_b + \dot{\beta}\cos\alpha \end{bmatrix}$$

Using this relationship in the wind rate vector equations, gives the relationship between the wind rate vector and the body-fixed angular rates.

$$\begin{bmatrix} \dot{\mu} \\ \dot{\gamma} \\ \dot{\chi} \end{bmatrix} = J \begin{bmatrix} p_w \\ q_w \\ r_w \end{bmatrix} = \begin{bmatrix} 1 & (\sin\mu\tan\gamma) & (\cos\mu\tan\gamma) \\ 0 & \cos\mu & -\sin\mu \\ 0 & \dfrac{\sin\mu}{\cos\gamma} & \dfrac{\cos\mu}{\cos\gamma} \end{bmatrix} DMC_{wb} \begin{bmatrix} p_b - \dot{\beta}\sin\alpha \\ q_b - \dot{\alpha} \\ r_b + \dot{\beta}\cos\alpha \end{bmatrix}$$

## Dialog Box




**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 Earth-fixed 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.

## Inputs and Outputs

The first input to the block is a vector containing the three applied forces in wind-fixed axes.

The second input is a vector containing the three applied moments in body-fixed axes.

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

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

The third output is a three-element vector containing the wind rotation angles [bank, flight path, heading], in radians.

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

The fifth output is a three-element vector containing the velocity in the wind-fixed frame.

The sixth output is a two-element vector containing the angle of attack and sideslip angle, in radians.

The seventh output is a two-element vector containing the rate of change of angle of attack and rate of change of sideslip angle, in radians per second.

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

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

The tenth output is a three-element vector containing the accelerations in body-fixed axes.

## 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)

# 6th Order Point Mass (Coordinated Flight)

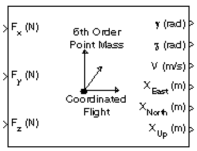
## Purpose

Calculate sixth-order point mass in coordinated flight

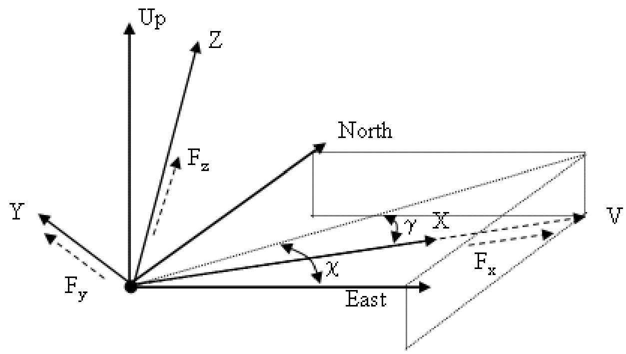
## Library

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.




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 $(\chi)$,

$$F_x = m\dot{V}$$

$$F_y = (mV\cos\gamma)\dot{\chi}$$

$$F_z = mV\dot{\gamma}$$

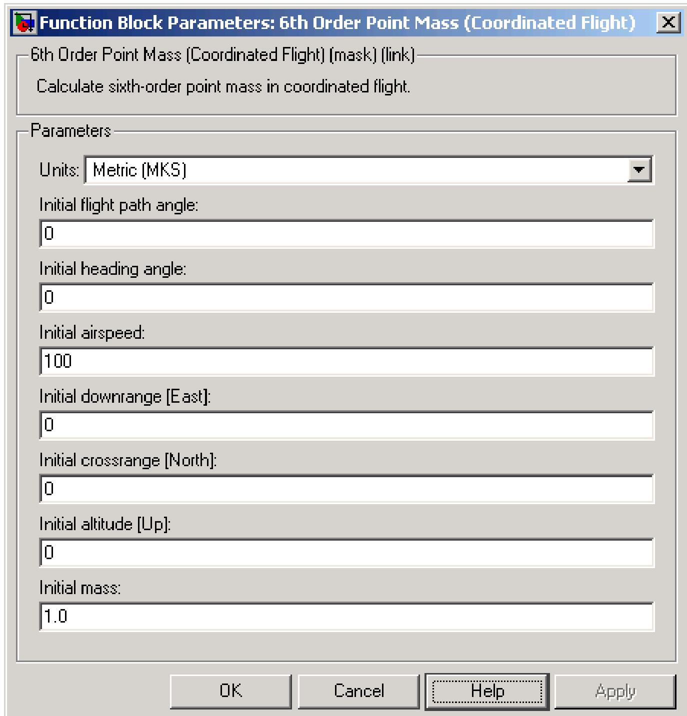
$$\dot{X}_{East} = V\cos\chi\cos\gamma$$

$$\dot{X}_{North} = V\sin\chi\cos\gamma$$

$$\dot{X}_{Up} = V\sin\gamma$$

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.

## Dialog Box




**Units**

Specifies the input and output units:

| Units | Forces | Velocity | Position |
|---|---|---|---|
| `Metric (MKS)` | Newton | Meters per second | Meters |

| Units | Forces | Velocity | Position |
|---|---|---|---|
| `English (Velocity in ft/s)` | Pound | Feet per second | Feet |
| `English (Velocity in kts)` | Pound | Knots | Feet |

**Initial flight path angle**
The scalar or vector containing initial flight path angle of the point mass(es).

**Initial heading angle**
The scalar or vector containing initial heading angle of the point mass(es).

**Initial airspeed**
The scalar or vector containing initial airspeed of the point mass(es).

**Initial downrange [East]**
The scalar or vector containing initial downrange of the point mass(es).

**Initial crossrange [North]**
The scalar or vector containing initial crossrange of the point mass(es).

**Initial altitude [Up]**
The scalar or vector containing initial altitude of the point mass(es).

**Initial mass**
The scalar or vector containing mass of the point mass(es).

## Inputs and Outputs

The first input is force in $x$-axis in selected units.

The second input is force in $y$-axis in selected units.

The third input is force in $z$-axis in selected units.

The first output is flight path angle in radians.

The second output is heading angle in radians.

The third output is airspeed in selected units.

The fourth output is the downrange or amount traveled East in selected units.

The fifth output is the crossrange or amount traveled North in selected units.

The sixth output is the altitude or amount traveled Up in selected units.

## Assumptions and 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-fixed 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.

## See Also

4th Order Point Mass (Longitudinal)

4th Order Point Mass Forces (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)

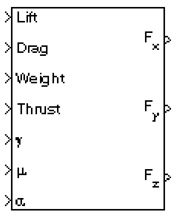
## Purpose

Calculate forces used by sixth-order point mass in coordinated flight

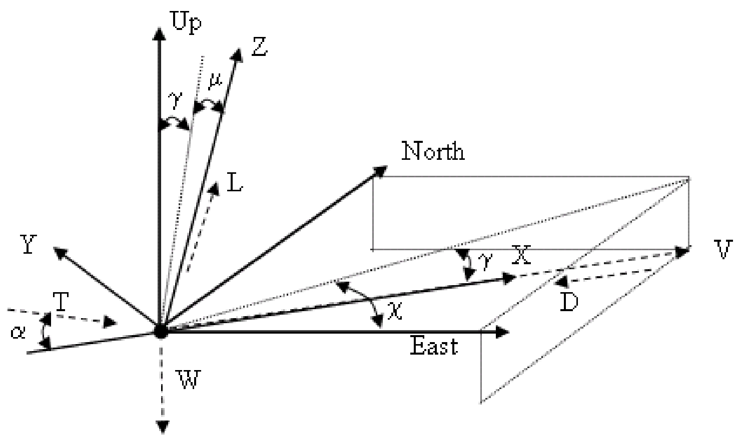
## Library

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.




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 $(\mu)$.

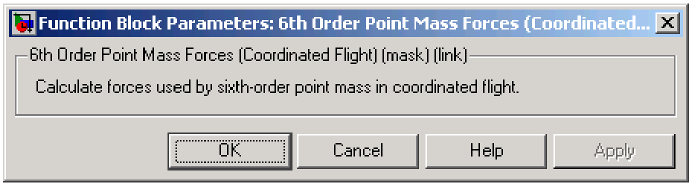
$$F_x = T\cos\alpha - D - W\sin\gamma$$

$$F_y = (L + T\sin\alpha)\sin\mu$$

$$F_z = (L + T\sin\alpha)\cos\mu - W\cos\gamma$$

# 6th Order Point Mass Forces (Coordinated Flight)

## Dialog Box




## Inputs and Outputs

The first input is lift in units of force.

The second input is drag in units of force.

The third input is weight in units of force.

The fourth input is thrust in units of force.

The fifth input is flight path angle in radians.

The sixth input is bank angle in radians.

The seventh input is angle of attack in radians.

The first output is force in $x$-axis in units of force.

The second output is force in $y$-axis in units of force.

The third output is force in $z$-axis in units of force.

## Assumptions and 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-fixed 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.

## See Also

4th Order Point Mass (Longitudinal)

4th Order Point Mass Forces (Longitudinal)

6th Order Point Mass (Coordinated Flight)

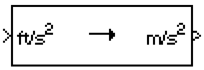
## Purpose

Convert from acceleration units to desired acceleration units

## Library

Utilities/Unit Conversions

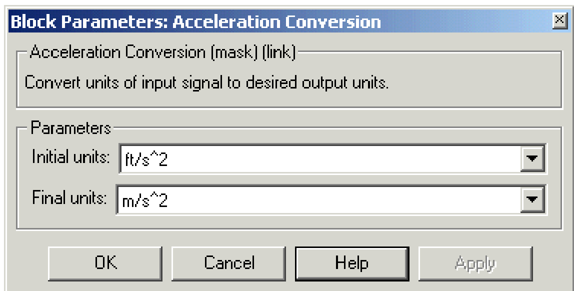
## Description




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 icon displays the input and output units selected from the **Initial units** and **Final units** lists.

## Dialog Box




**Initial units**
Specifies the input units.

**Final units**
Specifies the output units.

The following conversion units are available:

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

| | |
|---|---|
| `mph-s` | Miles per hour per second |
| `G's` | g-units |

### Inputs and Outputs

The input is acceleration in initial acceleration units.

The output is acceleration in final acceleration units.

### See Also

Angle Conversion

Angular Acceleration Conversion

Angular Velocity Conversion

Density Conversion

Force Conversion

Length Conversion

Mass Conversion

Pressure Conversion

Temperature Conversion

Velocity Conversion

# Adjoint of 3x3 Matrix

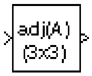
## Purpose

Compute adjoint of matrix

## Library

Utilities/Math Operations

## Description




The Adjoint of 3x3 Matrix block computes the adjoint matrix for the input matrix.

The input matrix has the form of

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix}$$

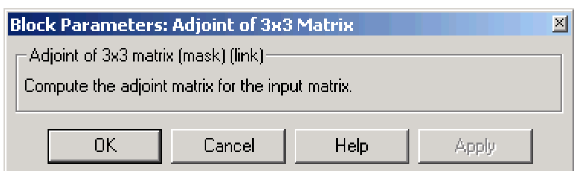
The adjoint of the matrix has the form of

$$adj(A) = \begin{bmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & M_{22} & M_{23} \\ M_{31} & M_{32} & M_{33} \end{bmatrix}$$

where

$$M_{ij} = (-1)^{i+j}$$

## Dialog Box




## Inputs and Outputs

The input is a 3-by-3 matrix.

The output of the block is 3-by-3 adjoint matrix of input matrix.

## See Also

Create 3x3 Matrix

Determinant of 3x3 Matrix

Invert 3x3 Matrix

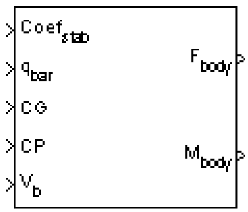
## Purpose

Compute aerodynamic forces and moments using aerodynamic coefficients, dynamic pressure, center of gravity, center of pressure, and velocity

## Library

Aerodynamics

## Description




The Aerodynamic Forces and Moments block computes the aerodynamic forces and moments about the center of gravity. By default, the inputs and outputs are represented in the body axes.

Let α be the angle of attack and β 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:

$$C_{w \leftarrow b} = \begin{bmatrix} \cos(\alpha)\cos(\beta) & \sin(\beta) & \sin(\alpha)\cos(\beta) \\ -\cos(\alpha)\sin(\beta) & \cos(\beta) & -\sin(\alpha)\sin(\beta) \\ -\sin(\alpha) & 0 & \cos(\alpha) \end{bmatrix}$$

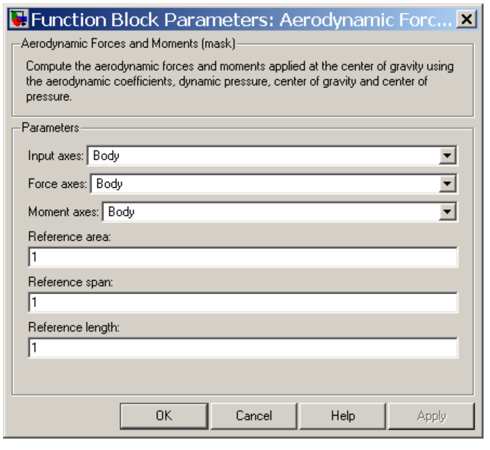
Moment coefficients have the same notation in all systems. Force coefficients are given below. Note there are no specific symbols for stability-axes force components. However, the stability axes have two components that are unchanged from the other axes.

$$\mathbf{F}_A^w \equiv \begin{bmatrix} -D \\ -C \\ -L \end{bmatrix} = C_{w \leftarrow b} \cdot \begin{bmatrix} X_A \\ Y_A \\ Z_A \end{bmatrix} \equiv C_{w \leftarrow b} \cdot \mathbf{F}_A^b$$

| 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 ($C_N$ = -$C_Z$).

## Dialog Box




**Input Axes**

Specifies coordinate system for input coefficients: `Body` (default), `Stability`, or `Wind`.

**Force Axes**
Specifies coordinate system for aerodynamic force: `Body` (default), `Stability`, or `Wind`.

**Moment Axes**
Specifies coordinate system for aerodynamic moment: `Body` (default), `Stability`, or `Wind`.

**Reference area**
Specifies the reference area for calculating aerodynamic forces and moments.

**Reference span**
Specifies the reference span for calculating aerodynamic moments in *x*-axes and *z*-axes.

**Reference length**
Specifies the reference length for calculating aerodynamic moment in the *y*-axes.

## Inputs and Outputs

The first input consists of aerodynamic coefficients (in the chosen input axes) for forces and moments. 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 $C_n$) |

The second input is the dynamic pressure.

The third input is the center of gravity.

The fourth input is the center of pressure. 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.

The fifth input (for inputs or outputs in stability or wind axes) is a three-element vector containing the velocity in the body axes.

The first output consists of the aerodynamic forces (in the chosen output axes) at the center of gravity in *x*-, *y*-, and *z*-axes.

The second output consists of the aerodynamic moments (in the chosen output axes) at the center of gravity in *x*-, *y*-, and *z*-axes.

## Assumptions and 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.

## Examples

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

## Reference

Stevens, B. L., and F. L. Lewis, *Aircraft Control and Simulation,* John Wiley & Sons, New York, 1992

## See Also

Digital DATCOM Forces and Moments

Dynamic Pressure

Estimate Center of Gravity

Moments About CG Due to Forces

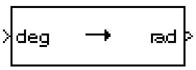
## Purpose

Convert from angle units to desired angle units

## Library

Utilities/Unit Conversions

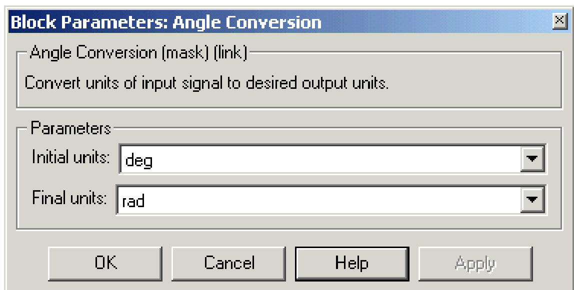
## Description




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 icon displays the input and output units selected from the **Initial units** and the **Final units** lists.

## Dialog Box




**Initial units**
: Specifies the input units.

**Final units**
: Specifies the output units.

The following conversion units are available:

| | |
|---|---|
| `deg` | Degrees |
| `rad` | Radians |
| `rev` | Revolutions |

## Inputs and Outputs

The input is angle in initial angle units.

The output is angle in final angle units.

### See Also

Acceleration Conversion

Angular Acceleration Conversion

Angular Velocity Conversion

Density Conversion

Force Conversion

Length Conversion

Mass Conversion

Pressure Conversion

Temperature Conversion

Velocity Conversion

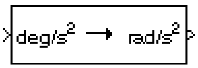
## Purpose

Convert from angular acceleration units to desired angular acceleration units

## Library

Utilities/Unit Conversions

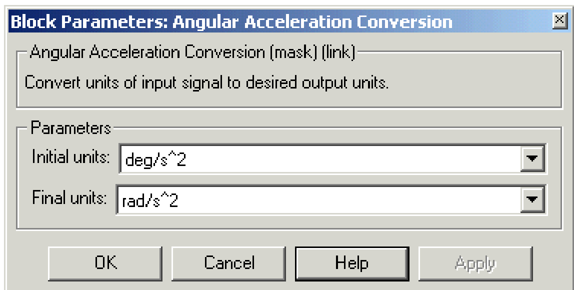
## Description




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 icon displays the input and output units selected from the **Initial units** and the **Final units** lists.

## Dialog Box




**Initial units**
Specifies the input units.

**Final units**
Specifies the output units.

The following conversion units are available:

| | |
|---|---|
| `deg/s`$^2$ | Degrees per second squared |
| `rad/s`$^2$ | Radians per second squared |
| `rpm/s` | Revolutions per minute per second |

## Inputs and Outputs

The input is angular acceleration in initial angular acceleration units.

The output is angular acceleration in final angular acceleration units.

## See Also

Acceleration Conversion

Angle Conversion

Angular Velocity Conversion

Density Conversion

Force Conversion

Length Conversion

Mass Conversion

Pressure Conversion

Temperature Conversion

Velocity Conversion

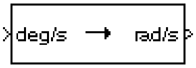
## Purpose

Convert from angular velocity units to desired angular velocity units

## Library

Utilities/Unit Conversions

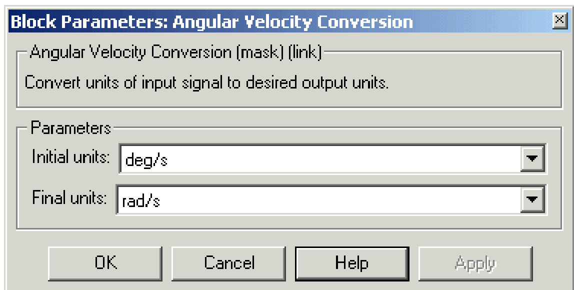
## Description




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 icon displays the input and output units selected from the **Initial units** and the **Final units** lists.

## Dialog Box




**Initial units**
: Specifies the input units.

**Final units**
: Specifies the output units.

The following conversion units are available:

| | |
|---|---|
| `deg/s` | Degrees per second |
| `rad/s` | Radians per second |
| `rpm` | Revolutions per minute |

## Inputs and Outputs

The input is angular velocity in initial angular velocity units.

The output is angular velocity in final angular velocity units.

## See Also

Acceleration Conversion

Angle Conversion

Angular Acceleration Conversion

Density Conversion

Force Conversion

Length Conversion

Mass Conversion

Pressure Conversion

Temperature Conversion

Velocity Conversion

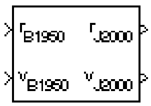
## Purpose

Transform position and velocity components from discontinued Standard Besselian Epoch (B1950) to Standard Julian Epoch (J2000)

## Library

Utilities/Axes Transformations

## Description




The Besselian Epoch to Julian Epoch block transforms two 3-by-1 vectors of Besselian Epoch position $(\underline{r}_{B1950})$, and Besselian Epoch velocity $(\underline{v}_{B1950})$ into Julian Epoch position $(\underline{r}_{J2000})$, and Julian Epoch velocity $(\underline{v}_{J2000})$. The transformation is calculated using:

$$\begin{bmatrix} \underline{r}_{J2000} \\ \underline{v}_{J2000} \end{bmatrix} = \begin{bmatrix} \underline{M}_{rr} & \underline{M}_{vr} \\ \underline{M}_{rv} & \underline{M}_{vv} \end{bmatrix} \begin{bmatrix} \underline{r}_{B1950} \\ \underline{v}_{B1950} \end{bmatrix}$$

where $(\underline{M}_{rr}, \underline{M}_{vr}, \underline{M}_{rv}, \underline{M}_{vv})$ are defined as:

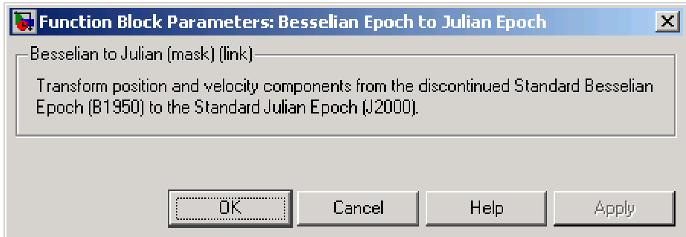
$$\underline{M}_{rr} = \begin{bmatrix} 0.9999256782 & -0.0111820611 & -0.0048579477 \\ 0.0111820610 & 0.9999374784 & -0.0000271765 \\ 0.0048579479 & -0.0000271474 & 0.9999881997 \end{bmatrix}$$

$$\underline{M}_{vr} = \begin{bmatrix} 0.00000242395018 & -0.00000002710663 & -0.00000001177656 \\ 0.00000002710663 & 0.00000242397878 & -0.00000000006587 \\ 0.00000001177656 & -0.00000000006582 & 0.00000242410173 \end{bmatrix}$$

$$\underline{M}_{rv} = \begin{bmatrix} -0.000551 & -0.238565 & 0.435739 \\ 0.238514 & -0.002667 & -0.008541 \\ -0.435623 & 0.012254 & 0.002117 \end{bmatrix}$$

$$\underline{M}_{vv} = \begin{bmatrix} 0.99994704 & -0.01118251 & -0.00485767 \\ 0.01118251 & 0.99995883 & -0.00002718 \\ 0.00485767 & -0.00002714 & 1.00000956 \end{bmatrix}$$

## Dialog Box




## Inputs and Outputs

The first input is a 3-by-1 vector containing the position in Standard Besselian Epoch (B1950).

The second input is a 3-by-1 vector containing the velocity in Standard Besselian Epoch (B1950).

The first output is a 3-by-1 vector containing the position in Standard Julian Epoch (J2000).

The second output is a 3-by-1 vector containing the velocity in Standard Julian Epoch (J2000).

## Reference

“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

Julian Epoch to Besselian Epoch

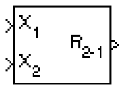
## Purpose

Calculate range between two crafts given their respective positions

## Library

GNC/Guidance

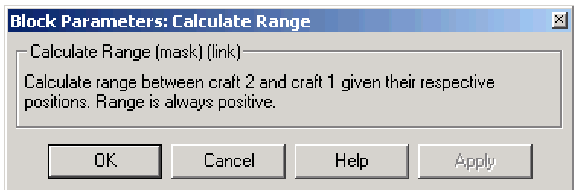
## Description




The Calculate Range block computes the range between two crafts. The equation used for the range calculation is

$$Range = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

## Dialog Box




## Inputs and Outputs

The first input is the (x, y and z) position of craft 1.

The second input is the (x, y and z) position of craft 2.

The output is the range from craft 2 and craft 1.

## Limitation

The calculated range is the magnitude of the distance, but not the direction. Therefore it is always positive or zero.

Craft positions are real values.

# COESA Atmosphere Model

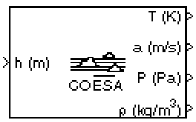
## Purpose

Implement 1976 COESA lower atmosphere

## Library

Environment/Atmosphere

## Description

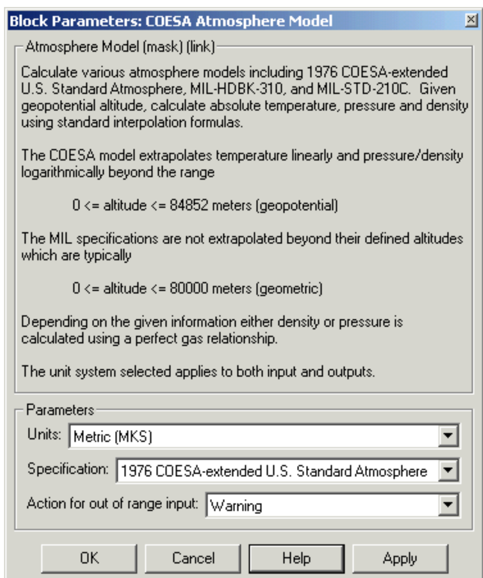



The COESA Atmosphere Model block implements the mathematical representation of the 1976 Committee on Extension to the Standard Atmosphere (COESA) United States standard lower atmospheric values for absolute temperature, pressure, density, and speed of sound for the input geopotential altitude.

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 icon displays the input and output units selected from the **Units** list.

## Dialog Box



**Units**

Specifies the input and output units:

| 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 |

**Specification**

Specify the atmosphere model type from one of the following atmosphere models. The default is `1976 COESA-extended U.S. Standard Atmosphere`.

`MIL-HDBK-310`

This selection is linked to the Non-Standard Day 310 block. See the block reference for more information.

`MIL-STD-210C`

This selection is linked to the Non-Standard Day 210C block. See the block reference for more information.

**Action for out of range input**

Specify if out-of-range input invokes a warning, error, or no action.

## Inputs and Outputs

The input is geopotential height.

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

## Assumptions and Limitations

Below the geopotential altitude of 0 m (0 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.

## Examples

See the `aeroblk_calibrated` model, the `aeroblk_indicated` model, and the airframe in the `aeroblk_HL20` demo for examples of this block.

## Reference

*U.S. Standard Atmosphere*, 1976, U.S. Government Printing Office, Washington, D.C.

## See Also

ISA Atmosphere Model

Non-Standard Day 210C

Non-Standard Day 310

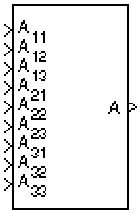
## Purpose

Create 3-by-3 matrix from nine input values

## Library

Utilities/Math Operations

## Description

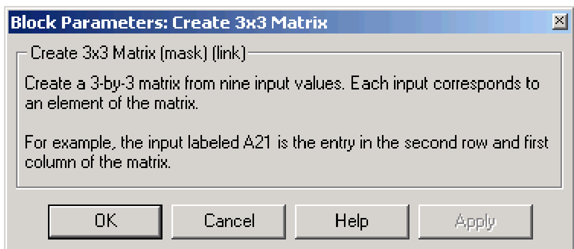



The Create 3x3 Matrix block creates a 3-by-3 matrix from nine input values where each input corresponds to an element of the matrix.

The output matrix has the form of

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix}$$

## Dialog Box




## Inputs and Outputs

The first input is the entry of the first row and first column of the matrix.

The second input is the entry of the first row and second column of the matrix.

The third input is the entry of the first row and third column of the matrix.

The fourth input is the entry of the second row and first column of the matrix.

The fifth input is the entry of the second row and second column of the matrix.

The sixth input is the entry of the second row and third column of the matrix.

The seventh input is the entry of the third row and first column of the matrix.

The eighth input is the entry of the third row and second column of the matrix.

The ninth input is the entry of the third row and third column of the matrix.

The output of the block is a 3-by-3 matrix.

## See Also

Adjoint of 3x3 Matrix

Determinant of 3x3 Matrix

Invert 3x3 Matrix

Symmetric Inertia Tensor

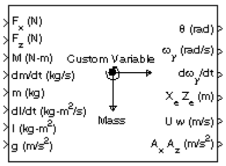
## Purpose

Implement three-degrees-of-freedom equations of motion of custom variable mass with respect to body axes

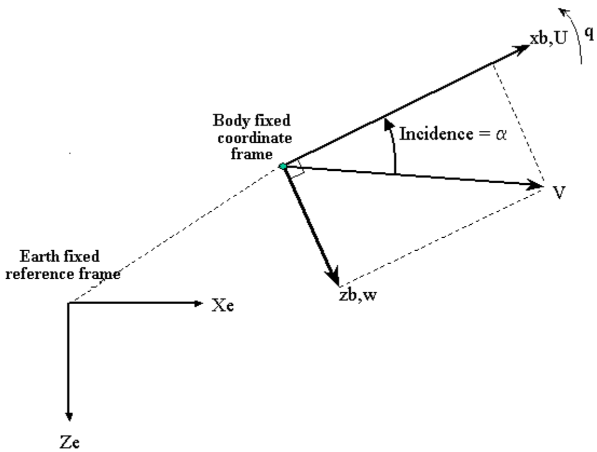
## Library

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 an Earth-fixed reference frame.




The equations of motion are

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

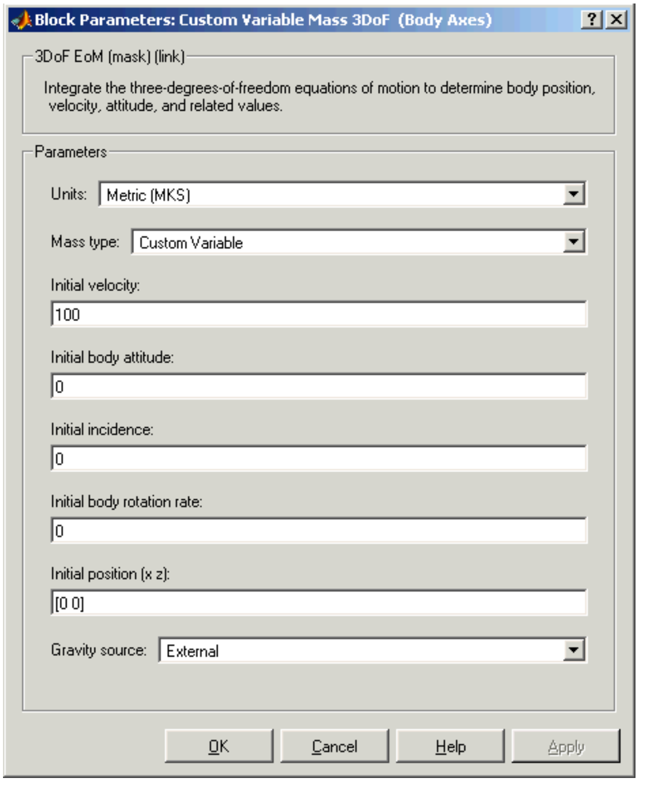
$$\dot{w} = \frac{F_z}{m} - \frac{\dot{m}w}{m} + qu + g\cos\theta$$

$$\dot{q} = \frac{M - \dot{I}_{yy}q}{I_{yy}}$$

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

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

## Dialog Box




**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 Earth-fixed reference frame.

**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`.

## Inputs and Outputs

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

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

The third input to the block is the applied pitch moment, ($M$).

The fourth input to the block is the rate of change of mass, $(\dot{m})$.

The fifth input to the block is the mass, (m).

The sixth input to the block is the rate of change of inertia tensor matrix, $(\dot{I}_{yy})$.

The seventh input to the block is the inertia tensor matrix, ($I_{yy}$).

The eighth optional input to the block is gravity in the selected units.

The first output from the block is the pitch attitude, in radians $(\theta)$.

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

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

The fourth output is a two-element vector containing the location of the body, in the Earth-fixed reference frame, (*Xe,Ze*).

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

The sixth output is a two-element vector containing the acceleration of the body resolved into the body-fixed coordinate frame, (*Ax,Az*).

## See Also

3DoF (Body Axes)

Incidence & Airspeed

Simple Variable Mass 3DoF (Body Axes)

# Custom Variable Mass 3DoF (Wind Axes)

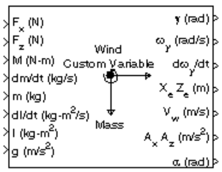
## Purpose

Implement three-degrees-of-freedom equations of motion of custom variable mass with respect to wind axes

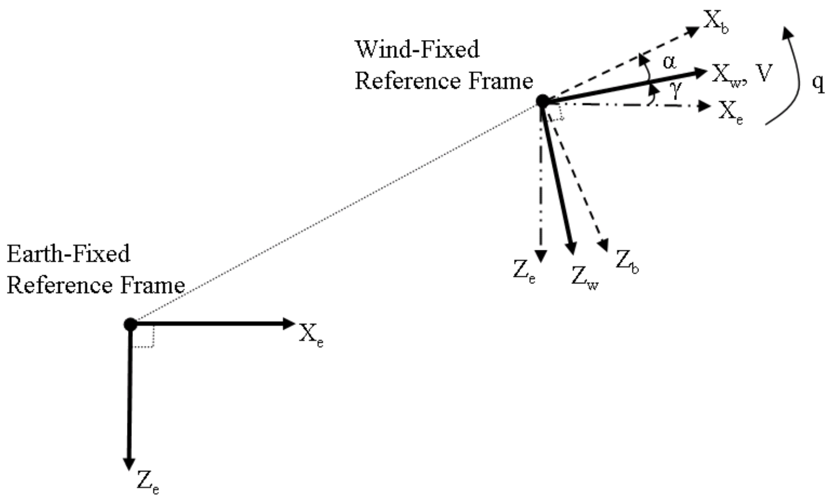
## Library

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 an Earth-fixed reference frame.




The equations of motion are

$$\dot{V} = \frac{F_{x_{wind}}}{m} - \frac{\dot{m}V}{m} - g\sin\gamma$$

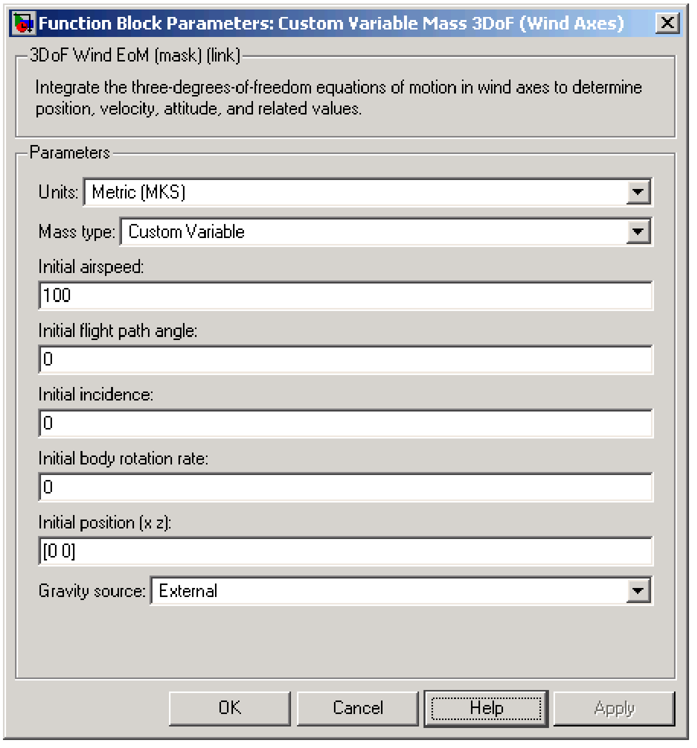
$$\dot{\alpha} = \frac{F_{z_{wind}}}{mV} + q + \frac{g}{V}\cos\gamma$$

$$\dot{q} = \ddot{\theta} = \frac{M_{y_{body}} - \dot{I}_{yy}q}{I_{yy}}$$

$$\dot{\gamma} = q - \dot{\alpha}$$

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

## Dialog Box




**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 Earth-fixed reference frame.

**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`.

## Inputs and Outputs

The first input to the block is the force acting along the wind $x$-axis, $(F_x)$.

The second input to the block is the force acting along the wind $z$-axis, $(F_z)$.

The third input to the block is the applied pitch moment in body axes, $(M)$.

The fourth input to the block is the rate of change of mass, $(\dot{m})$.

The fifth input to the block is the mass, (m).

The sixth input to the block is the rate of change of inertia tensor matrix, $(\dot{I}_{yy})$.

The seventh input to the block is the inertia tensor matrix, $(I_{yy})$.

The eighth optional input to the block is gravity in the selected units.

The first output from the block is the flight path angle, in radians $(\gamma)$.

The second output is the pitch angular rate, in radians per second $(\omega_y)$.

The third output is the pitch angular acceleration, in radians per second squared $(d\omega_y/dt)$.

The fourth output is a two-element vector containing the location of the body, in the Earth-fixed reference frame, (*Xe,Ze*).

The fifth output is a two-element vector containing the velocity of the body resolved into the wind-fixed coordinate frame, (*V,0*).

The sixth output is a two-element vector containing the acceleration of the body resolved into the body-fixed coordinate frame, (*Ax,Az*).

The seventh output is a scalar containing the angle of attack, $(\alpha)$.

## 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)

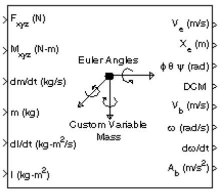
## Purpose

Implement Euler angle representation of six-degrees-of-freedom equations of motion of custom variable mass

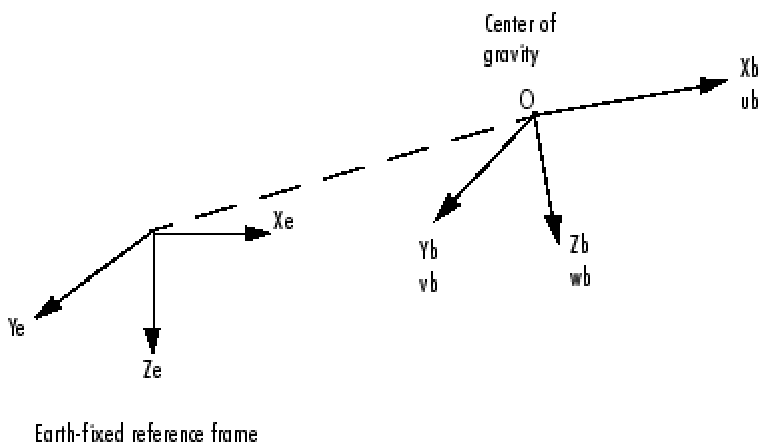
## Library

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 an Earth-fixed reference frame $(X_e, Y_e, Z_e)$. The origin of the body-fixed coordinate frame is the center of gravity of the body, and the body is assumed to be rigid, an assumption that eliminates the need to consider the forces acting between individual elements of mass. The Earth-fixed reference frame is considered inertial, 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 body-fixed coordinate frame is given below, where the applied forces $[F_x\ F_y\ F_z]^T$ are in the body-fixed frame.

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

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

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

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

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

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

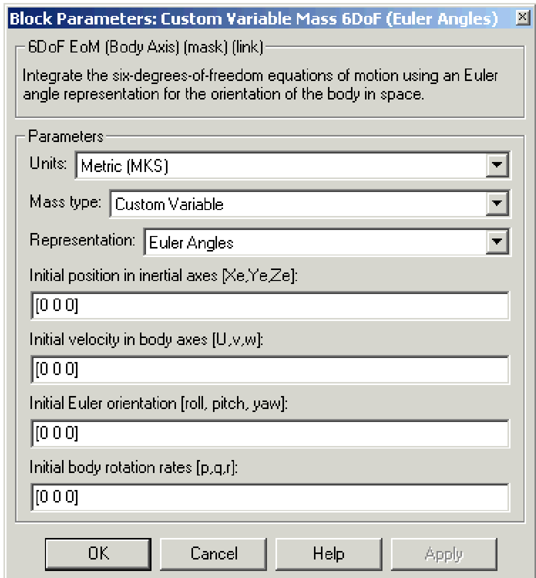
The relationship between the body-fixed angular velocity vector, $[p\ q\ r]^T$, and the rate of change of the Euler angles, $[\dot{\phi}\ \dot{\theta}\ \dot{\psi}]^T$, can be determined by resolving the Euler rates into the body-fixed coordinate frame.

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

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

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

## Dialog Box




**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 (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 Earth-fixed 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.

## Inputs and Outputs

The first input to the block is a vector containing the three applied forces.

The second input is a vector containing the three applied moments.

The third input is a scalar containing the rate of change of mass.

The fourth input is a scalar containing the mass

The fifth input is a 3-by-3 matrix for the rate of change of inertia tensor matrix.

The sixth input is a 3-by-3 matrix for the inertia tensor matrix.

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

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

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

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

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

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

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

The eighth output is a three-element vector containing the accelerations in body-fixed axes.

## Assumptions and Limitations

The block assumes that the applied forces are acting at the center of gravity of the body.

## Reference

Mangiacasale, L., *Flight Mechanics of a μ-Airplane with a MATLAB Simulink Helper,* Edizioni Libreria CLUP, Milan, 1998.

## 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)

# Custom Variable Mass 6DoF (Quaternion)

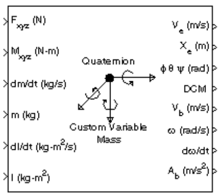
## Purpose

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



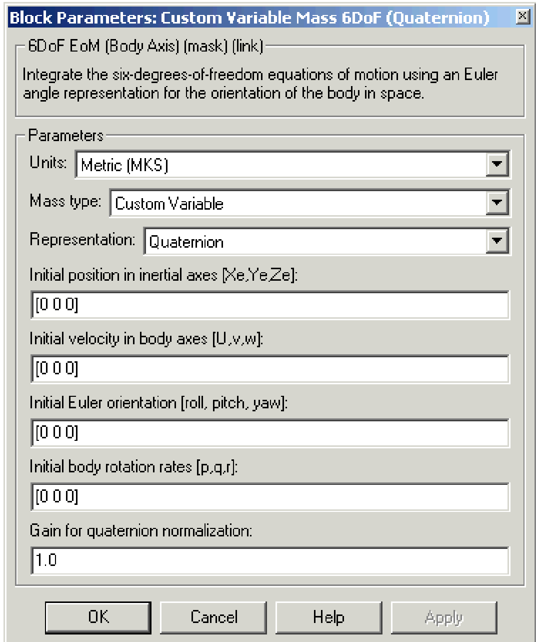

For a description of the coordinate system employed and the translational dynamics, see the block description for the Custom Variable Mass 6DoF (Euler Angles) block.

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 $\epsilon$ 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.

$$\begin{bmatrix} \dot{q}_0 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} = \frac{1}{2}\begin{bmatrix} 0 & -p & -q & -r \\ p & 0 & r & -q \\ q & -r & 0 & p \\ r & q & -p & 0 \end{bmatrix}\begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} + K\varepsilon\begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix}$$

$$\varepsilon = 1 - (q_0{}^2 + q_1{}^2 + q_3{}^2 + q_4{}^2)$$

## Dialog Box




**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 Earth-fixed 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.

## Inputs and Outputs

The first input to the block is a vector containing the three applied forces.

The second input is a vector containing the three applied moments.

The third input is a scalar containing the rate of change of mass.

The fourth input is a scalar containing the mass

The fifth input is a 3-by-3 matrix for the rate of change of inertia tensor matrix.

The sixth input is a 3-by-3 matrix for the inertia tensor matrix.

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

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

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

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

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

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

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

The eighth output is a three-element vector containing the accelerations in body-fixed axes.

## Assumptions and Limitations

The block assumes that the applied forces are acting at the center of gravity of the body.

## Reference

Mangiacasale, L., *Flight Mechanics of a u-Airplane with a MATLAB Simulink Helper*, Edizioni Libreria CLUP, Milan, 1998.

## 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)

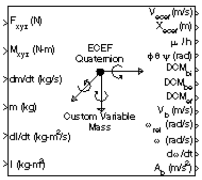
## Purpose

Implement quaternion representation of six-degrees-of-freedom equations of motion of custom variable mass in Earth-centered Earth-fixed (ECEF) coordinates

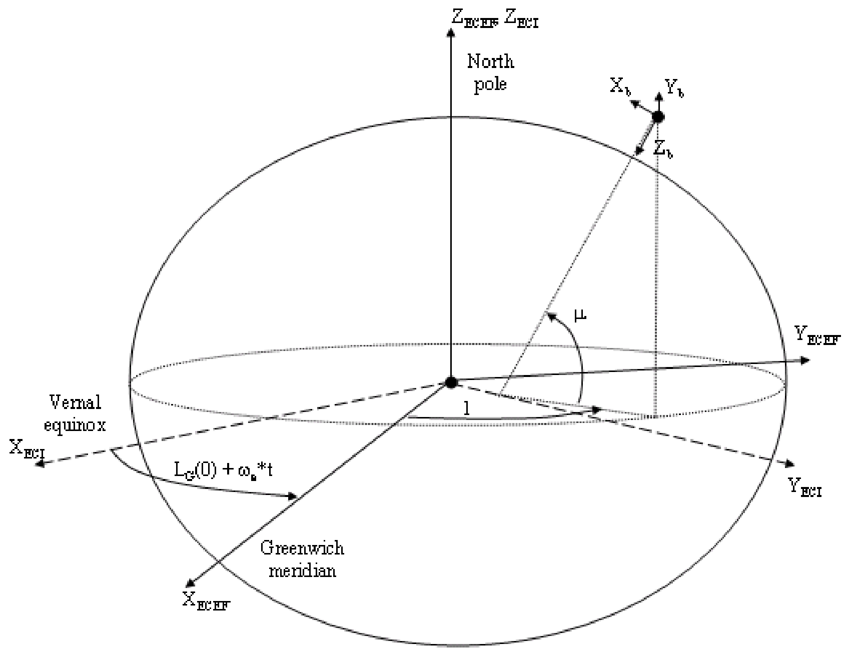
## Library

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.

$$\bar{F}_b = \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} = m(\dot{\bar{V}}_b + \bar{\omega}_b \times \bar{V}_b + DCM_{bf}\bar{\omega}_e \times \bar{V}_b) + DCM_{bf}(\bar{\omega}_e \times (\bar{\omega}_e \times \bar{X}_f)) + \dot{m}(\bar{V}_b + DCM_{bf}(\bar{\omega}_e \times \bar{X}_f))$$

where the change of position in ECEF $\dot{\underline{x}}_f$ 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$(\underline{V}_b)$, angular rates of the body with respect to ECI frame, expressed in body frame$(\underline{\omega}_b)$. Earth rotation rate $(\underline{\omega}_e)$, 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

$$\bar{V}_b = \begin{bmatrix} u \\ v \\ \omega \end{bmatrix}, \bar{\omega}_{rel} = \begin{bmatrix} p \\ q \\ r \end{bmatrix}, \bar{\omega}_e = \begin{bmatrix} 0 \\ 0 \\ \omega_e \end{bmatrix}, \bar{\omega}_b = \bar{\omega}_{rel} + DCM_{bf}\bar{\omega}_e + DCM_{be}\bar{\omega}_{ned}$$

$$\bar{\omega}_{ned} = \begin{bmatrix} \dot{l}\cos\mu \\ -\dot{\mu} \\ -\dot{l}\sin\mu \end{bmatrix} = \begin{bmatrix} V_E/(N+h) \\ -V_N/(M+h) \\ V_E \bullet \tan\mu/(N+h) \end{bmatrix}$$

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.

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

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

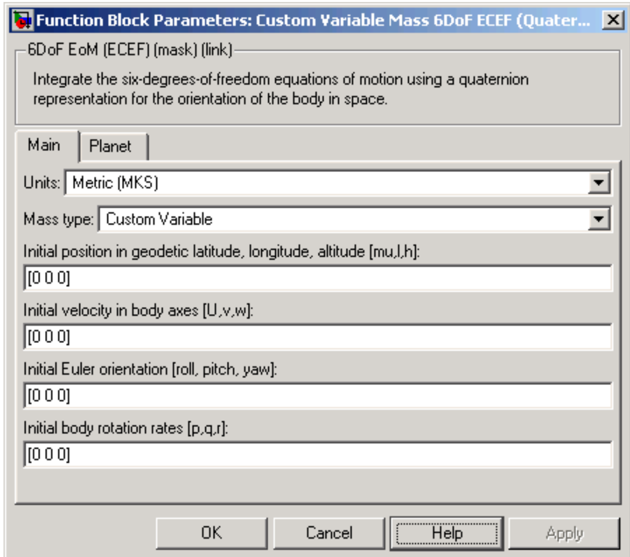
The rate of change of the inertia tensor is defined by the following equation.

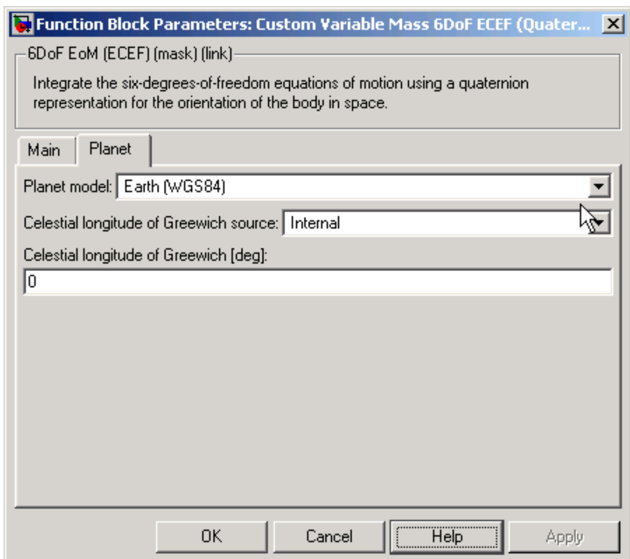
$$\dot{I} = \begin{bmatrix} \dot{I}_{xx} & -\dot{I}_{xy} & -\dot{I}_{xz} \\ -\dot{I}_{yx} & \dot{I}_{yy} & -\dot{I}_{yz} \\ -\dot{I}_{zx} & -\dot{I}_{zy} & \dot{I}_{zz} \end{bmatrix}$$

The integration of the rate of change of the quaternion vector is given below.

$$\begin{bmatrix} \dot{q}_0 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} = -1/2 \begin{bmatrix} 0 & \omega_b(1) & \omega_b(2) & \omega_b(3) \\ -\omega_b(1) & 0 & -\omega_b(3) & \omega_b(2) \\ -\omega_b(2) & \omega_b(3) & 0 & -\omega_b(1) \\ -\omega_b(3) & -\omega_b(2) & \omega_b(1) & 0 \end{bmatrix} \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix}$$

## Dialog Box


Function Block Parameters: Custom Variable Mass 6DoF ECEF (Quater...
6DoF EoM (ECEF) (mask) (link)
Integrate the six-degrees-of-freedom equations of motion using a quaternion representation for the orientation of the body in space.
Main
Planet
Units: Metric (MKS)
Mass type: Custom Variable
Initial position in geodetic latitude, longitude, altitude [mu,l,h]:
[0 0 0]
Initial velocity in body axes [U,v,w]:
[0 0 0]
Initial Euler orientation [roll, pitch, yaw]:
[0 0 0]
Initial body rotation rates [p,q,r]:
[0 0 0]
OK
Cancel
Help
Apply


Function Block Parameters: Custom Variable Mass 6DoF ECEF (Quater...
6DoF EoM (ECEF) (mask) (link)
Integrate the six-degrees-of-freedom equations of motion using a quaternion representation for the orientation of the body in space.
Main
Planet
Planet model: Earth (WGS84)
Celestial longitude of Greewich source: Internal
Celestial longitude of Greewich [deg]:
0
OK
Cancel
Help
Apply

**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.

**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.

**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.

**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.

## Inputs and Outputs

| Input | Dimension Type | Description |
|---|---|---|
| First input | Vector | Contains the three applied forces in body-fixed axes. |
| Second input | Vector | Contains the three applied moments in body-fixed axes. |
| Third input | Scalar | Contains the rate of change of mass. |
| Fourth input | Scalar | Contains the mass. |
| Fifth input | 3-by-3 matrix | Applies to the rate of change of inertia tensor matrix. |
| Sixth input | 3-by-3 matrix | Applies to the inertia tensor matrix. |

| Output | Dimension Type | Description |
|---|---|---|
| First output | Three-element vector | Contains the velocity of the body with respect to ECEF frame, expressed in ECEF frame. |
| Second output | Three-element vector | Contains the position in the ECEF reference frame. |
| Third output | Three-element vector | Contains the position in geodetic latitude, longitude and altitude, in degrees, degrees and selected units of length respectively. |

| Output | Dimension Type | Description |
|---|---|---|
| Fourth output | 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 output | 3-by-3 matrix | Applies to the coordinate transformation from ECI axes to body-fixed axes. |
| Sixth output | a 3-by-3 matrix | Applies to the coordinate transformation from geodetic axes to body-fixed axes. |
| Seventh output | 3-by-3 matrix | Applies to the coordinate transformation from ECEF axes to geodetic axes. |
| Eighth output | Three-element vector | Contains the velocity of the body with respect to ECEF frame, expressed in body frame. |
| Ninth output | 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 output | Three-element vector | Contains the angular rates of the body with respect to ECI frame, expressed in body frame, in radians per second. |

| Output | Dimension Type | Description |
|---|---|---|
| Eleventh output | Three-element vector | Contains the angular accelerations of the body with respect to ECI frame, expressed in body frame, in radians per second. |
| Twelfth output | Three-element vector | Contains the accelerations in body-fixed axes. |

## 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 ±90 degrees, and longitude that lies between ±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 through the center of the Sun 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)

# Custom Variable Mass 6DoF Wind (Quaternion)

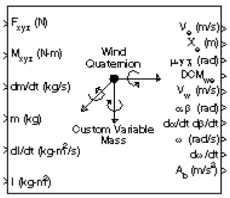
## Purpose

Implement quaternion representation of six-degrees-of-freedom equations of motion of custom variable mass with respect to wind axes

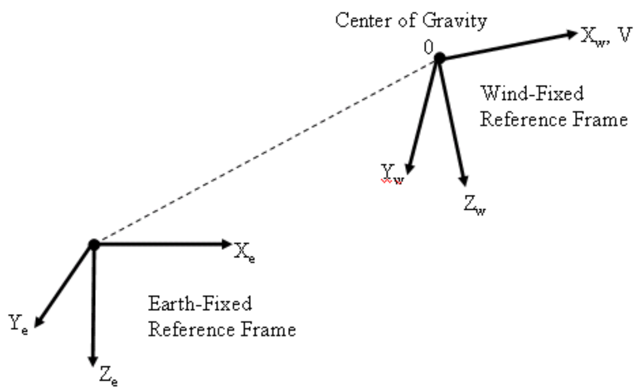
## Library

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 Earth-fixed 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 Earth-fixed 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.

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

$$\underline{V}_w = \begin{bmatrix} V \\ 0 \\ 0 \end{bmatrix}, \underline{\omega}_w = \begin{bmatrix} p_w \\ q_w \\ r_w \end{bmatrix} = DMC_{wb} \begin{bmatrix} p_b - \dot{\beta}\sin\alpha \\ q_b - \dot{\alpha} \\ r_b + \dot{\beta}\cos\alpha \end{bmatrix}, \underline{w}_b = \begin{bmatrix} p_b \\ q_b \\ r_b \end{bmatrix}$$

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

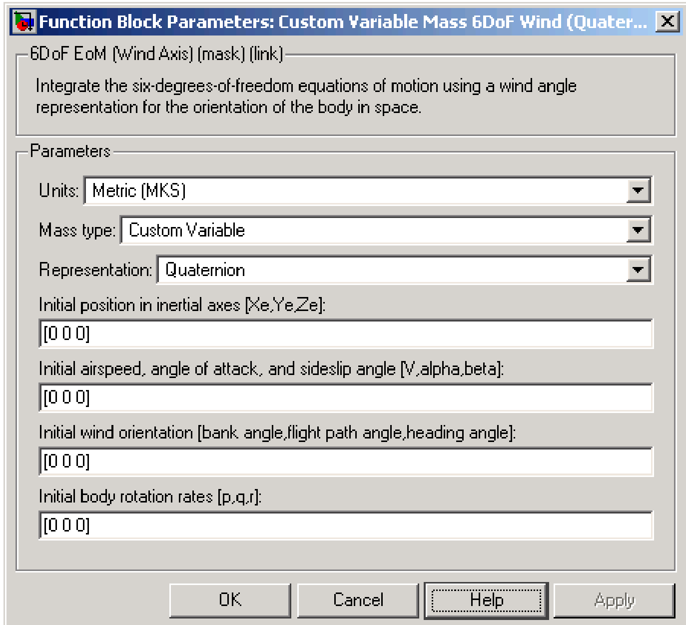
$$\underline{M}_b = \begin{bmatrix} L \\ M \\ N \end{bmatrix} = I\dot{\underline{\omega}}_b + \underline{\omega}_b \times (I\,\underline{\omega}_b) + \dot{I}\underline{\omega}_b$$

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

The integration of the rate of change of the quaternion vector is given below.

$$\begin{bmatrix} \dot{q}_0 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} = -\frac{1}{2} \begin{bmatrix} 0 & p & q & r \\ -p & 0 & -r & q \\ -q & r & 0 & -p \\ -r & -q & p & 0 \end{bmatrix} \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix}$$

## Dialog Box




**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 Earth-fixed 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.

## Inputs and Outputs

The first input to the block is a vector containing the three applied forces in wind-fixed axes.

The second input is a vector containing the three applied moments in body-fixed axes.

The third input is a scalar containing the rate of change of mass.

The fourth input is a scalar containing the mass

The fifth input is a 3-by-3 matrix for the rate of change of inertia tensor matrix in body-fixed axes.

The sixth input is a 3-by-3 matrix for the inertia tensor matrix in body-fixed axes.

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

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

The third output is a three-element vector containing the wind rotation angles [bank, flight path, heading], in radians.

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

The fifth output is a three-element vector containing the velocity in the wind-fixed frame.

The sixth output is a two-element vector containing the angle of attack and sideslip angle, in radians.

The seventh output is a two-element vector containing the rate of change of angle of attack and rate of change of sideslip angle, in radians per second.

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

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

The tenth output is a three-element vector containing the accelerations in body-fixed axes.

## Assumptions and Limitations

The block assumes that the applied forces are acting at the center of gravity of the body.

## References

Mangiacasale, L., *Flight Mechanics of a u-Airplane with a MATLAB Simulink Helper*, Edizioni Libreria CLUP, Milan, 1998.

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)

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)

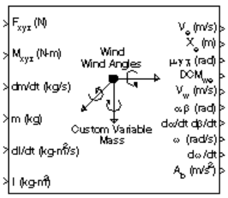
## Purpose

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.

$$\begin{bmatrix} \dot{\mu} \\ \dot{\gamma} \\ \dot{\chi} \end{bmatrix} = J \begin{bmatrix} p_w \\ q_w \\ r_w \end{bmatrix} = \begin{bmatrix} 1 & (\sin\mu\tan\gamma) & (\cos\mu\tan\gamma) \\ 0 & \cos\mu & -\sin\mu \\ 0 & \dfrac{\sin\mu}{\cos\gamma} & \dfrac{\cos\mu}{\cos\gamma} \end{bmatrix} \begin{bmatrix} p_w \\ q_w \\ r_w \end{bmatrix}$$

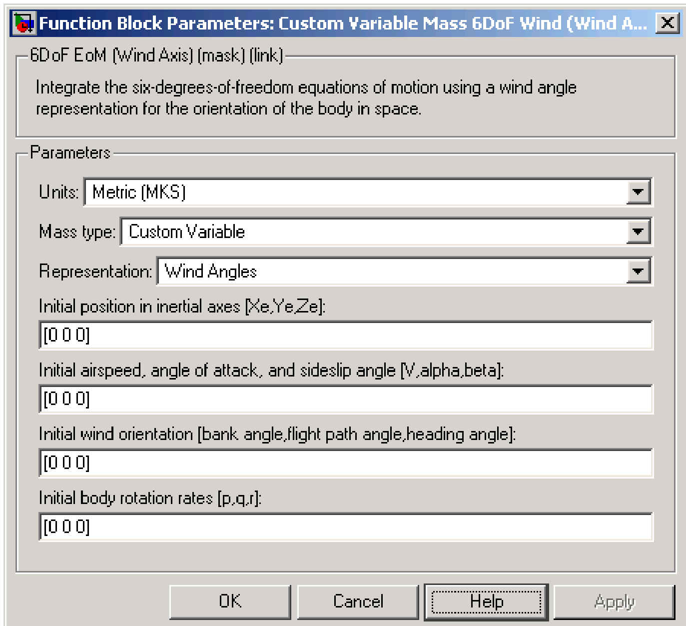
The body-fixed angular rates are related to the wind-fixed angular rate by the following equation.

$$\begin{bmatrix} p_w \\ q_w \\ r_w \end{bmatrix} = DMC_{wb} \begin{bmatrix} p_b - \dot{\beta}\sin\alpha \\ q_b - \dot{\alpha} \\ r_b + \dot{\beta}\cos\alpha \end{bmatrix}$$

Using this relationship in the wind rate vector equations, gives the relationship between the wind rate vector and the body-fixed angular rates.

$$\begin{bmatrix} \dot{\mu} \\ \dot{\gamma} \\ \dot{\chi} \end{bmatrix} = J \begin{bmatrix} p_w \\ q_w \\ r_w \end{bmatrix} = \begin{bmatrix} 1 & (\sin\mu\tan\gamma) & (\cos\mu\tan\gamma) \\ 0 & \cos\mu & -\sin\mu \\ 0 & \dfrac{\sin\mu}{\cos\gamma} & \dfrac{\cos\mu}{\cos\gamma} \end{bmatrix} DMC_{wb} \begin{bmatrix} p_b - \dot{\beta}\sin\alpha \\ q_b - \dot{\alpha} \\ r_b + \dot{\beta}\cos\alpha \end{bmatrix}$$

## Dialog Box




**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 (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 Earth-fixed 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.

## Inputs and Outputs

The first input to the block is a vector containing the three applied forces in wind-fixed axes.

The second input is a vector containing the three applied moments in body-fixed axes.

The third input is a scalar containing the rate of change of mass.

The fourth input is a scalar containing the mass

The fifth input is a 3-by-3 matrix for the rate of change of inertia tensor matrix in body-fixed axes.

The sixth input is a 3-by-3 matrix for the inertia tensor matrix in body-fixed axes.

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

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

The third output is a three-element vector containing the wind rotation angles [bank, flight path, heading], in radians.

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

The fifth output is a three-element vector containing the velocity in the wind-fixed frame.

The sixth output is a two-element vector containing the angle of attack and sideslip angle, in radians.

The seventh output is a two-element vector containing the rate of change of angle of attack and rate of change of sideslip angle, in radians per second.

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

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

The tenth output is a three-element vector containing the accelerations in body-fixed axes.

## Assumptions and Limitations

The block assumes that the applied forces are acting at the center of gravity of the body.

## References

Mangiacasale, L., *Flight Mechanics of a μ-Airplane with a MATLAB Simulink Helper,* Edizioni Libreria CLUP, Milan, 1998.

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)

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)

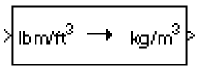
## Purpose

Convert from density units to desired density units

## Library

Utilities/Unit Conversions

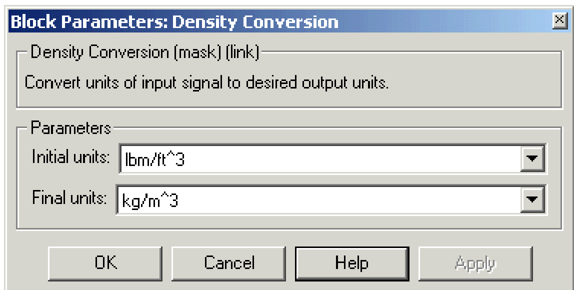
## Description




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 icon displays the input and output units selected from the **Initial units** and the **Final units** lists.

## Dialog Box




**Initial units**
Specifies the input units.

**Final units**
Specifies the output units.

The following conversion units are available:

| | |
|---|---|
| `lbm/ft`$^3$ | Pound mass per cubic foot |
| `kg/m`$^3$ | Kilograms per cubic meter |
| `slug/ft`$^3$ | Slugs per cubic foot |
| `lbm/in`$^3$ | Pound mass per cubic inch |

### Inputs and Outputs

The input is density in initial density units.

The output is density in final density units.

### See Also

Acceleration Conversion

Angle Conversion

Angular Acceleration Conversion

Angular Velocity Conversion

Force Conversion

Length Conversion

Mass Conversion

Pressure Conversion

Temperature Conversion

Velocity Conversion

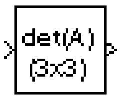
## Purpose

Compute determinant of matrix

## Library

Utilities/Math Operations

## Description




The Determinant of 3x3 Matrix block computes the determinant for the input matrix.

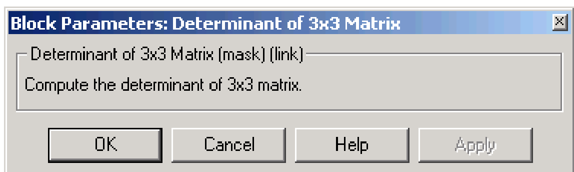
The input matrix has the form of

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix}$$

The determinant of the matrix has the form of

$$det(A) = A_{11}(A_{22}A_{33} - A_{23}A_{32}) - A_{12}(A_{21}A_{33} - A_{23}A_{31}) + A_{13}(A_{21}A_{32} - A_{22}A_{31})$$

## Dialog Box




## Inputs and Outputs

The input is a 3-by-3 matrix.

The output of the block is the determinant of input matrix.

## See Also

Adjoint of 3x3 Matrix

Create 3x3 Matrix

Invert 3x3 Matrix

# Digital DATCOM Forces and Moments

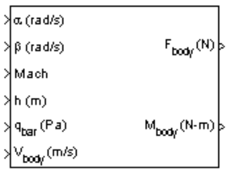
## Purpose

Compute aerodynamic forces and moments using Digital DATCOM static and dynamic stability derivatives

## Library

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.

Algorithms for calculating forces and moments build up the overall aerodynamic forces and moments ($\boldsymbol{F}$ and $\boldsymbol{M}$) from data contained in the **Digital DATCOM structure** parameter:

$$\boldsymbol{F} = \boldsymbol{F}_{\text{static}} + \boldsymbol{F}_{\text{dyn}}$$

$$\boldsymbol{M} = \boldsymbol{M}_{\text{static}} + \boldsymbol{M}_{\text{dyn}}$$

$\boldsymbol{F}_{\text{static}}$ and $\boldsymbol{M}_{\text{static}}$ are the static contribution, and $\boldsymbol{F}_{\text{dyn}}$ and $\boldsymbol{M}_{\text{dyn}}$ 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 |
|---|---|
| $C_{\text{D}}$ | Matrix of drag coefficients. These coefficients are defined positive for an aft-acting load. |
| $C_{\text{L}}$ | Matrix of lift coefficients. These coefficients are defined positive for an up-acting load. |
| $C_{\text{m}}$ | Matrix of pitching-moment coefficients. These coefficients are defined positive for a nose-up rotation. |
| $C_{\text{Y}\beta}$ | Matrix of derivatives of side-force coefficients with respect to sideslip angle |

| Coefficient | Meaning |
|---|---|
| $C_{n\beta}$ | Matrix of derivatives of yawing-moment coefficients with respect to sideslip angle |
| $C_{l\beta}$ | Matrix of derivatives of rolling-moment coefficients with respect to sideslip angle |

These are the static contributions to the aerodynamic coefficients in stability axes.

$$C_{D\ static} = C_D$$

$$C_{y\ static} = C_{Y\beta}\beta$$

$$C_{L\ static} = C_L$$

$$C_{l\ static} = C_{l\beta}\beta$$

$$C_{m\ static} = C_M$$

$$C_{n\ static} = C_{n\beta}\beta$$

### Dynamic Stability Characteristics

Dynamic stability characteristics include the following.

| Coefficient | Meaning |
|---|---|
| $C_{lq}$ | Matrix of rolling-moment 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 |

| Coefficient | Meaning |
|---|---|
| $C_{\mathrm{Yp}}$ | Matrix of lateral force derivatives due to roll rate |
| $C_{\mathrm{np}}$ | Matrix of yawing-moment derivatives due to roll rate |
| $C_{\mathrm{nr}}$ | Matrix of yawing-moment derivatives due to yaw rate |
| $C_{\mathrm{lr}}$ | Matrix of rolling-moment derivatives due to yaw rate |

These are the dynamic contributions to the aerodynamic coefficients in stability axes.

$$C_{\mathrm{D\ dyn}} = 0$$

$$C_{\mathrm{y\ dyn}} = C_{\mathrm{yp}}\, p(b_{\mathrm{ref}}/2V)$$

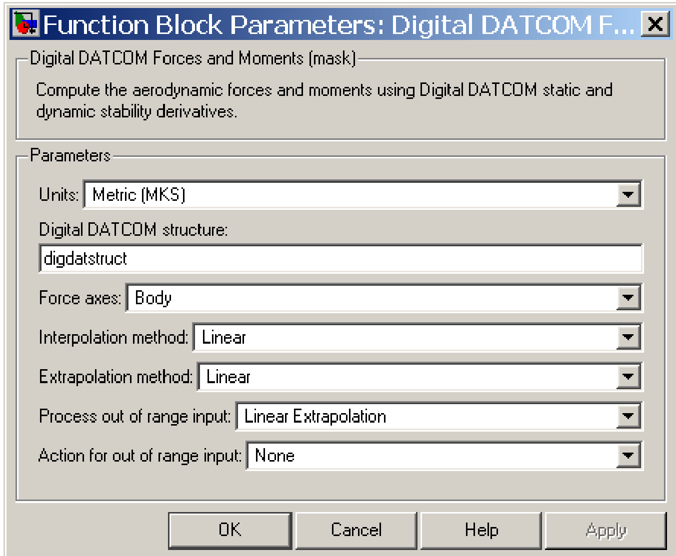
$$C_{\mathrm{L\ dyn}} = C_{\mathrm{L}\dot{\alpha}}\, \dot{\alpha}\,(c_{\mathrm{bar}}/2V)$$

$$C_{\mathrm{l\ dyn}} = (C_{\mathrm{lp}}p + C_{\mathrm{lq}}q + C_{\mathrm{lr}}r)\,(b_{\mathrm{ref}}/2V)$$

$$C_{\mathrm{m\ dyn}} = (C_{\mathrm{mq}}q + C_{\mathrm{m}\dot{\alpha}}\,\dot{\alpha})\,(c_{\mathrm{bar}}/2V)$$

$$C_{\mathrm{n\ dyn}} = (C_{\mathrm{np}}p + C_{\mathrm{nr}}r)\,(b_{\mathrm{ref}}/2V)$$

## Dialog Box




**Units**

Specifies the input and output units:

| 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 |

**Digital DATCOM structure**

Specifies the MATLAB structure containing the digital DATCOM data. This structure is generated by the Aerospace Toolbox function `datcomimport`.

**Force axes**

Specifies coordinate system for aerodynamic force: `Body` or `Wind`.

**Interpolation method**

`None (flat)` or `Linear`

**Extrapolation method**

`None (clip)` or `Linear`

**Process out of range input**

Specifies how to handle out-of-range input: `Linear Extrapolation` or `Clip to Range`.

**Action for out of range input**

Specifies if out-of-range input invokes a warning, an error, or no action.

## Inputs and Outputs

The first input is the angle of attack, in radians.

The second input is the sideslip angle, in radians.

The third input is the Mach number.

The fourth input is the altitude, in selected length units.

The fifth input is the dynamic pressure, in selected pressure units.

The sixth input is the velocity in selected velocity units and selected force axes.

The seventh (optional) input is the angle of attack rate, in radians per second.

The eighth (optional) input is the body angular rates, in radians per second.

The ninth (optional) input is the ground height, in selected units of length.

The tenth (optional) input is the control surface deflections, in radians.

The first output consists of the aerodynamic forces at the center of gravity in selected coordinate system: Body ($F_x$, $F_y$, and $F_z$), or Wind ($F_D$, $F_y$, and $F_L$).

The second output consists of the aerodynamic moments at the center of gravity in body coordinates ($M_x$, $M_y$, and $M_z$).

## Assumptions and Limitations

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.

## 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,* John Wiley & Sons, New York, 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,* John Wiley & Sons, New York, 1992.

## See Also

Aerodynamic Forces and Moments

# Direction Cosine Matrix Body to Wind

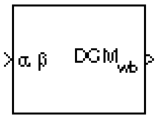
## Purpose

Convert angle of attack and sideslip angle to direction cosine matrix

## Library

Utilities/Axes Transformations

## Description




The Direction Cosine Matrix Body to Wind block converts angle of attack and sideslip angle into a 3-by-3 direction cosine matrix (DCM). 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 $(ox_2, oy_2, oz_2)$

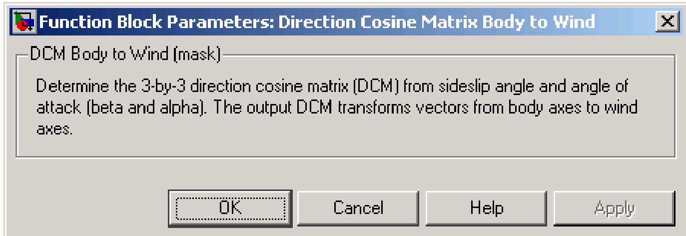
$$\begin{bmatrix} ox_2 \\ oy_2 \\ oz_2 \end{bmatrix} = DCM_{wb} \begin{bmatrix} ox_0 \\ oy_0 \\ oz_0 \end{bmatrix}$$

$$\begin{bmatrix} ox_2 \\ oy_2 \\ oz_2 \end{bmatrix} = \begin{bmatrix} \cos\beta & \sin\beta & 0 \\ -\sin\beta & \cos\beta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\alpha & 0 & \sin\alpha \\ 0 & 1 & 0 \\ -\sin\alpha & 0 & \cos\alpha \end{bmatrix} \begin{bmatrix} ox_0 \\ oy_0 \\ oz_0 \end{bmatrix}$$

Combining the two axis transformation matrices defines the following DCM.

$$DCM_{wb} = \begin{bmatrix} \cos\alpha\cos\beta & \sin\beta & \sin\alpha\cos\beta \\ -\cos\alpha\sin\beta & \cos\beta & -\sin\alpha\sin\beta \\ -\sin\alpha & 0 & \cos\alpha \end{bmatrix}$$

## Dialog Box




## Inputs and Outputs

The input is a 2-by-1 vector containing angle of attack and sideslip angle, in radians.

The output is a 3-by-3 direction cosine matrix which transforms body-fixed vectors to wind-fixed vectors.

## Reference

Stevens, B. L., and F. L. Lewis, *Aircraft Control and Simulation*, John Wiley & Sons, New York, 1992.

## See Also

Direction Cosine Matrix Body to Wind to Alpha and Beta

Direction Cosine Matrix to Euler Angles

Direction Cosine Matrix to Wind Angles

Euler Angles to Direction Cosine Matrix

Wind Angles to Direction Cosine Matrix

# Direction Cosine Matrix Body to Wind to Alpha and Beta

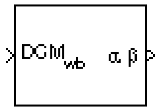
## Purpose

Convert direction cosine matrix to angle of attack and sideslip angle

## Library

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) into angle of attack and sideslip angle. 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.

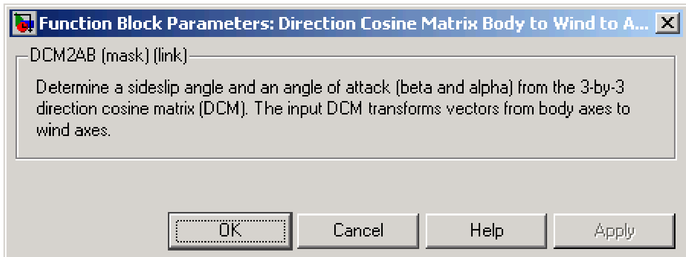
$$DCM_{wb} = \begin{bmatrix} \cos\alpha\cos\beta & \sin\beta & \sin\alpha\cos\beta \\ -\cos\alpha\sin\beta & \cos\beta & -\sin\alpha\sin\beta \\ -\sin\alpha & 0 & \cos\alpha \end{bmatrix}$$

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

$$\alpha = \mathrm{asin}(-DCM(3,1))$$

$$\beta = \mathrm{asin}(DCM(1,2))$$

## Dialog Box




## Inputs and Outputs

The input is a 3-by-3 direction cosine matrix which transforms body-fixed vectors to wind-fixed vectors.

The output is a 2-by-1 vector containing angle of attack and sideslip angle, in radians.

## Assumptions and Limitations

This implementation generates angles that lie between $\pm 90$ degrees.

## Reference

Stevens, B. L., and F. L. Lewis, *Aircraft Control and Simulation*, John Wiley & Sons, New York, 1992.

## See Also

Direction Cosine Matrix Body to Wind

Direction Cosine Matrix to Euler Angles

Direction Cosine Matrix to Wind Angles

Euler Angles to Direction Cosine Matrix

Wind Angles to Direction Cosine Matrix

# Direction Cosine Matrix ECEF to NED

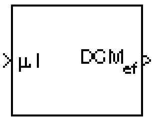
## Purpose

Convert geodetic latitude and longitude to direction cosine matrix

## Library

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 $(ox_0, oy_0, oz_0)$ into a vector in north-east-down (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 $(ox_2, oy_2, oz_2)$

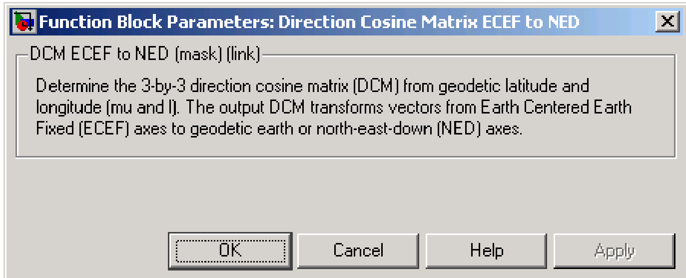
$$\begin{bmatrix} ox_2 \\ oy_2 \\ oz_2 \end{bmatrix} = DCM_{ef} \begin{bmatrix} ox_0 \\ oy_0 \\ oz_0 \end{bmatrix}$$

$$\begin{bmatrix} ox_2 \\ oy_2 \\ oz_2 \end{bmatrix} = \begin{bmatrix} -\sin\mu & 0 & \cos\mu \\ 0 & 1 & 0 \\ -\cos\mu & 0 & -\sin\mu \end{bmatrix} \begin{bmatrix} \cos\iota & \sin\iota & 0 \\ -\sin\iota & \cos\iota & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} ox_0 \\ oy_0 \\ oz_0 \end{bmatrix}$$

Combining the two axis transformation matrices defines the following DCM.

$$DCM_{ef} = \begin{bmatrix} -\sin\mu\cos\iota & -\sin\mu\sin\iota & \cos\mu \\ -\sin\iota & \cos\iota & 0 \\ -\cos\mu\cos\iota & -\cos\mu\sin\iota & -\sin\mu \end{bmatrix}$$

## Dialog Box




## Inputs and Outputs

The input is a 2-by-1 vector containing geodetic latitude and longitude, in degrees.

The output is a 3-by-3 direction cosine matrix which transforms ECEF vectors to NED vectors.

## Assumptions

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.

## References

Stevens, B. L., and F. L. Lewis, *Aircraft Control and Simulation*, John Wiley & Sons, New York, 1992.

Zipfel, P. H., *Modeling and Simulation of Aerospace Vehicle Dynamics*, AIAA Education Series, Reston, Virginia, 2000.

“Atmospheric and Space Flight Vehicle Coordinate Systems,” ANSI/AIAA R-004-1992.

## See Also

Direction Cosine Matrix ECEF to NED to Latitude and Longitude

Direction Cosine Matrix to Euler Angles

Direction Cosine Matrix to Wind Angles

ECEF Position to LLA

Euler Angles to Direction Cosine Matrix

LLA to ECEF Position

Wind Angles to Direction Cosine Matrix

# Direction Cosine Matrix ECEF to NED to Latitude and Longitude

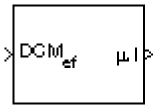
## Purpose

Convert direction cosine matrix to geodetic latitude and longitude

## Library

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 a vector in north-east-down (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.

$$DCM_{ef} = \begin{bmatrix} -\sin\mu\cos\iota & -\sin\mu\sin\iota & \cos\mu \\ -\sin\iota & \cos\iota & 0 \\ -\cos\mu\cos\iota & -\cos\mu\sin\iota & -\sin\mu \end{bmatrix}$$

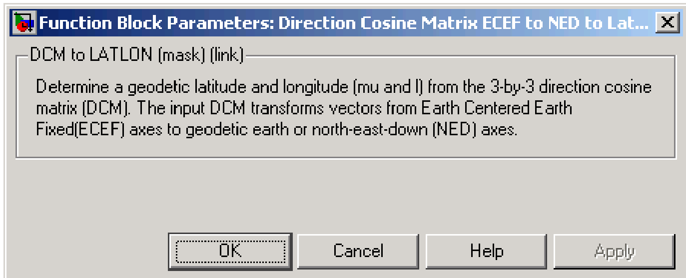
To determine geodetic latitude and longitude from the DCM, the following equations are used:

# Direction Cosine Matrix ECEF to NED to Latitude and Longitude

$$\mu = \operatorname{asin}(-DCM(3,3))$$

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

## Dialog Box




## Inputs and Outputs

The input is a 3-by-3 direction cosine matrix which transforms ECEF vectors to NED vectors.

The output is a 2-by-1 vector containing geodetic latitude and longitude, in degrees.

## Assumptions and Limitations

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.

## References


Stevens, B. L., and F. L. Lewis, *Aircraft Control and Simulation*, John Wiley & Sons, New York, 1992.

Zipfel, P. H., *Modeling and Simulation of Aerospace Vehicle Dynamics*, AIAA Education Series, Reston, Virginia, 2000.

"Atmospheric and Space Flight Vehicle Coordinate Systems," ANSI/AIAA R-004-1992.

## See Also

Direction Cosine Matrix ECEF to NED

Direction Cosine Matrix to Euler Angles

Direction Cosine Matrix to Wind Angles

ECEF Position to LLA

Euler Angles to Direction Cosine Matrix

LLA to ECEF Position

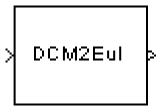
Wind Angles to Direction Cosine Matrix

# Direction Cosine Matrix to Euler Angles

**Purpose** Convert direction cosine matrix to Euler angles

**Library** Utilities/Axes Transformations

**Description**




The Direction Cosine Matrix to Euler Angles block converts a 3-by-3 direction cosine matrix (DCM) into three Euler rotation angles. The DCM matrix performs the coordinate transformation of a vector in inertial axes $(ox_0, oy_0, oz_0)$ into a vector in body axes $(ox_3, oy_3, oz_3)$. The order of the axis rotations required to bring this about is:

1. A rotation about $oz_0$ through the yaw angle $(\psi)$ to axes $(ox_1, oy_1, oz_1)$
2. A rotation about $oy_1$ through the pitch angle $(\theta)$ to axes $(ox_2, oy_2, oz_2)$
3. A rotation about $ox_2$ through the roll angle $(\phi)$ to axes $(ox_3, oy_3, oz_3)$

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

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

Combining the three axis transformation matrices defines the following DCM.

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

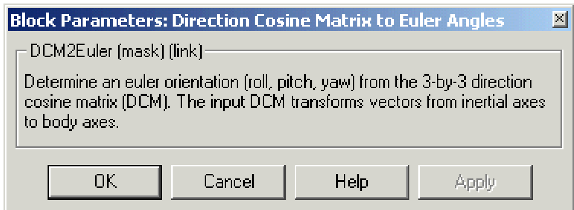
To determine Euler angles from the DCM, the following equations are used:

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

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

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

## Dialog Box




## Inputs and Outputs

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

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

## Assumptions and Limitations

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

## See Also

Direction Cosine Matrix to Quaternions

Euler Angles to Direction Cosine Matrix

Euler Angles to Quaternions

Quaternions to Direction Cosine Matrix

Quaternions to Euler Angles

# Direction Cosine Matrix to Quaternions

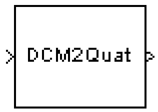
## Purpose

Convert direction cosine matrix to quaternion vector

## Library

Utilities/Axes Transformations

## Description



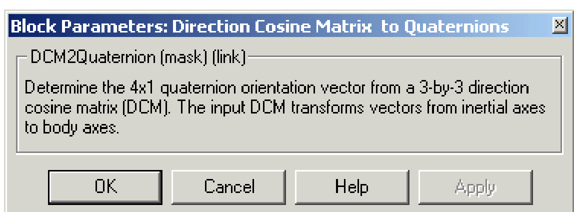

The Direction Cosine Matrix to Quaternions block transforms a 3-by-3 direction cosine matrix (DCM) into a four-element unit quaternion vector ($q_0$,$q_1$,$q_2$,$q_3$). The DCM performs the coordinate transformation of a vector in inertial axes to a vector in body axes.

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

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

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

## Dialog Box



## Inputs and Outputs

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

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

## See Also

Direction Cosine Matrix to Euler Angles

Euler Angles to Direction Cosine Matrix

Euler Angles to Quaternions

Quaternions to Direction Cosine Matrix

Quaternions to Euler Angles

# Direction Cosine Matrix to Wind Angles

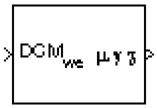
## Purpose

Convert direction cosine matrix to wind angles

## Library

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 $(ox_3, oy_3, oz_3)$. 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.

$$DCM_{we} = \begin{bmatrix} \cos\gamma\cos\chi & \cos\gamma\sin\chi & -\sin\gamma \\ (\sin\mu\sin\gamma\cos\chi - \cos\mu\sin\chi) & (\sin\mu\sin\gamma\sin\chi + \cos\mu\cos\chi) & \sin\mu\cos\gamma \\ (\cos\mu\sin\gamma\cos\chi + \sin\mu\sin\chi) & (\cos\mu\sin\gamma\sin\chi - \sin\mu\cos\chi) & \cos\mu\cos\gamma \end{bmatrix}$$

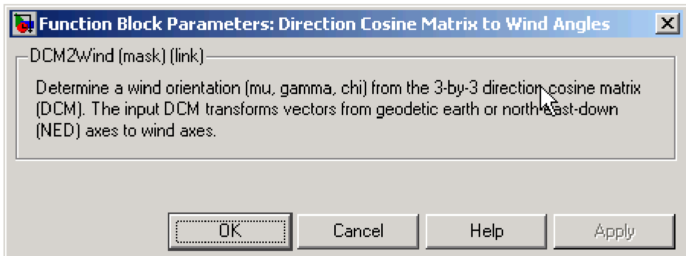
To determine wind angles from the DCM, the following equations are used:

$$\mu = \operatorname{atan}\left(\frac{DCM(2,3)}{DCM(3,3)}\right)$$

$$\gamma = \operatorname{asin}(-DCM(1,3))$$

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

## Dialog Box




## Inputs and Outputs

The input is a 3-by-3 direction cosine matrix which transforms earth vectors to wind vectors.

The output is a 3-by-1 vector of wind angles, in radians.

## Assumptions and Limitations

This implementation generates a flight path angle that lies between $\pm 90$ degrees, and bank and heading angles that lie between $\pm 180$ degrees.

## See Also

Direction Cosine Matrix Body to Wind

Direction Cosine Matrix Body to Wind to Alpha and Beta

Direction Cosine Matrix to Euler Angles

Euler Angles to Direction Cosine Matrix

Wind Angles to Direction Cosine Matrix

# Discrete Wind Gust Model

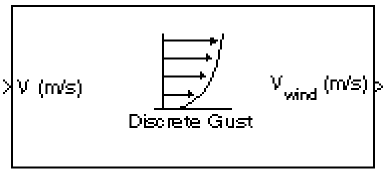
## Purpose

Generate discrete wind gust

## Library

Environment/Wind

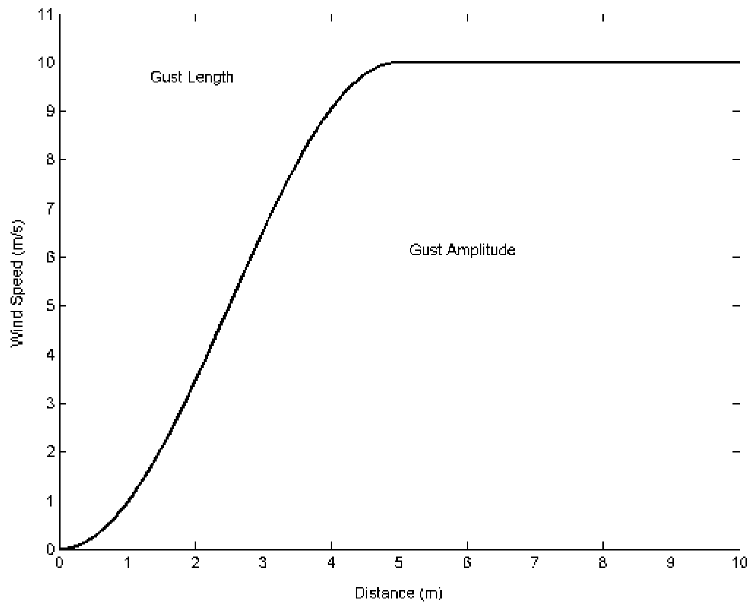
## Description




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

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

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



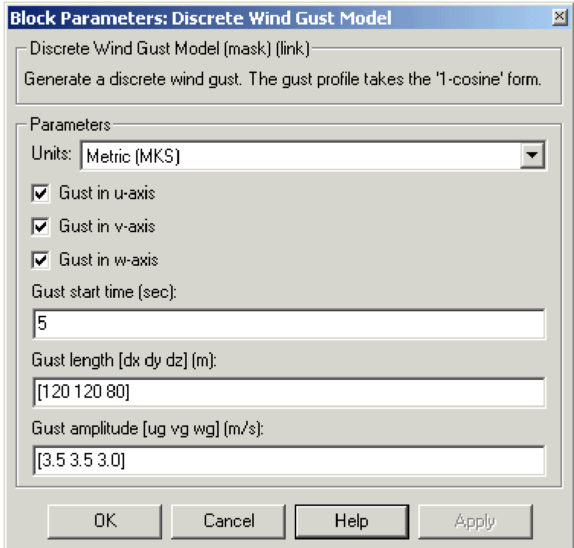

The discrete gust can be used singly or in multiples to assess airplane response to large wind disturbances.

The mathematical representation of the discrete gust is

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

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

## Dialog Box




**Units**

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

The input is airspeed in units selected.

The output is wind speed in units selected.

## Examples

See Airframe in the `aeroblk_HL20` demo 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

# Dryden Wind Turbulence Model (Continuous)

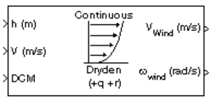
## Purpose

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 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 Ω radians per meter, the circular frequency ω is calculated by multiplying V by Ω. The following table displays the component spectra functions:

| | MIL-F-8785C | MIL-HDBK-1797 |
|---|---|---|
| **Longitudinal** | | |
| $\Phi_u(\omega)$ | $\frac{2\sigma_u^2 L_u}{\pi V}\cdot\frac{1}{1+(L_u\frac{\omega}{V})^2}$ | $\frac{2\sigma_u^2 L_u}{\pi V}\cdot\frac{1}{1+(L_u\frac{\omega}{V})^2}$ |
| $\Phi_{p_g}(\omega)$ | $\frac{\sigma_w^2}{VL_w}\cdot\frac{0.8\left(\frac{\pi L_w}{4b}\right)^{\frac{1}{3}}}{1+\left(\frac{4b\omega}{\pi V}\right)^2}$ | $\frac{\sigma_w^2}{2VL_w}\cdot\frac{0.8\left(\frac{2\pi L_w}{4b}\right)^{\frac{1}{3}}}{1+\left(\frac{4b\omega}{\pi V}\right)^2}$ |
| **Lateral** | | |
| $\Phi_v(\omega)$ | $\frac{\sigma_v^2 L_v}{\pi V}\cdot\frac{1+3(L_v\frac{\omega}{V})^2}{[1+(L_v\frac{\omega}{V})^2]^2}$ | $\frac{2\sigma_v^2 L_v}{\pi V}\cdot\frac{1+12(L_v\frac{\omega}{V})^2}{[1+4(L_v\frac{\omega}{V})^2]^2}$ |

| | MIL-F-8785C | MIL-HDBK-1797 |
|---|---|---|
| $\Phi_r(\omega)$ | $\dfrac{\mp\left(\frac{\omega}{V}\right)^2}{1+\left(\frac{3b\,\omega}{\pi V}\right)^2}\cdot\Phi_v(\omega)$ | $\dfrac{\mp\left(\frac{\omega}{V}\right)^2}{1+\left(\frac{3b\,\omega}{\pi V}\right)^2}\cdot\Phi_v(\omega)$ |
| **Vertical** | | |
| $\Phi_w(\omega)$ | $\dfrac{\sigma_w^{\;2}L_w}{\pi V}\cdot\dfrac{1+3(L_w\frac{\omega}{V})^2}{[1+(L_w\frac{\omega}{V})^2]^2}$ | $\dfrac{2\sigma_w^{\;2}L_w}{\pi V}\cdot\dfrac{1+12(L_w\frac{\omega}{V})^2}{[1+4(L_w\frac{\omega}{V})^2]^2}$ |
| $\Phi_q(\omega)$ | $\dfrac{\pm\left(\frac{\omega}{V}\right)^2}{1+\left(\frac{4b\,\omega}{\pi V}\right)^2}\cdot\Phi_w(\omega)$ | $\dfrac{\pm\left(\frac{\omega}{V}\right)^2}{1+\left(\frac{4b\,\omega}{\pi V}\right)^2}\cdot\Phi_w(\omega)$ |

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 specifications as three variations, which are displayed in the following table:

$$p_g = \frac{\partial w_g}{\partial y} \qquad q_g = \frac{\partial w_g}{\partial x} \qquad r_g = -\frac{\partial v_g}{\partial x}$$

$$p_g = \frac{\partial w_g}{\partial y} \qquad q_g = \frac{\partial w_g}{\partial x} \qquad r_g = \frac{\partial v_g}{\partial x}$$

$$p_g = -\frac{\partial w_g}{\partial y} \qquad q_g = -\frac{\partial w_g}{\partial x} \qquad r_g = \frac{\partial v_g}{\partial x}$$

The variations affect only the vertical ($q_g$) and lateral ($r_q$) turbulence angular rates.

Keep in mind that 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. 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 derived from the spectral square roots of the spectrum equations.

The following table displays the transfer functions:

| | MIL-F-8785C | MIL-HDBK-1797 |
|---|---|---|
| **Longitudinal** | | |
| $H_u(s)$ | $\sigma_u\sqrt{\frac{2L_u}{\pi V}}\frac{1}{1+\frac{L_u}{V}s}$ | $\sigma_u\sqrt{\frac{2L_u}{\pi V}}\frac{1}{1+\frac{L_u}{V}s}$ |
| $H_p(s)$ | $\sigma_w\sqrt{\frac{0.8}{V}}\frac{\left(\frac{\pi}{(4b)}\right)^{1/6}}{L_w^{1/3}\left(1+\left(\frac{4b}{\pi V}\right)s\right)}$ | $\sigma_w\sqrt{\frac{0.8}{V}}\frac{\left(\frac{\pi}{(4b)}\right)^{1/6}}{(2L_w)^{1/3}\left(1+\left(\frac{4b}{\pi V}\right)s\right)}$ |
| **Lateral** | | |
| $H_v(s)$ | $\sigma_v\sqrt{\frac{L_v}{\pi V}}\frac{1+\frac{\sqrt{3}L_v}{V}s}{\left(1+\frac{L_v}{V}s\right)^2}$ | $\sigma_v\sqrt{\frac{2L_v}{\pi V}}\frac{1+\frac{2\sqrt{3}L_v}{V}s}{\left(1+\frac{2L_v}{V}s\right)^2}$ |
| $H_r(s)$ | $\frac{\mp\frac{s}{V}}{\left(1+\left(\frac{3b}{\pi V}\right)s\right)}\cdot H_v(s)$ | $\frac{\mp\frac{s}{V}}{\left(1+\left(\frac{3b}{\pi V}\right)s\right)}\cdot H_v(s)$ |
| **Vertical** | | |
| $H_w(s)$ | $\sigma_w\sqrt{\frac{L_w}{\pi V}}\frac{1+\frac{\sqrt{3}L_w}{V}s}{\left(1+\frac{L_w}{V}s\right)^2}$ | $\sigma_w\sqrt{\frac{2L_w}{\pi V}}\frac{1+\frac{2\sqrt{3}L_w}{V}s}{\left(1+\frac{2L_w}{V}s\right)^2}$ |
| $H_q(s)$ | $\frac{\pm\frac{s}{V}}{\left(1+\left(\frac{4b}{\pi V}\right)s\right)}\cdot H_w(s)$ | $\frac{\pm\frac{s}{V}}{\left(1+\left(\frac{4b}{\pi V}\right)s\right)}\cdot H_w(s)$ |

Divided into two distinct regions, the turbulence scale lengths and intensities are functions of altitude.

---

**Note** The 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 < 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 |
|---|---|
| $L_w = h$ <br> $L_u = L_v = \dfrac{h}{(0.177 + 0.000823h)^{1.2}}$ | $2L_w = h$ <br> $L_u = 2L_v = \dfrac{h}{(0.177 + 0.000823h)^{1.2}}$ |

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.

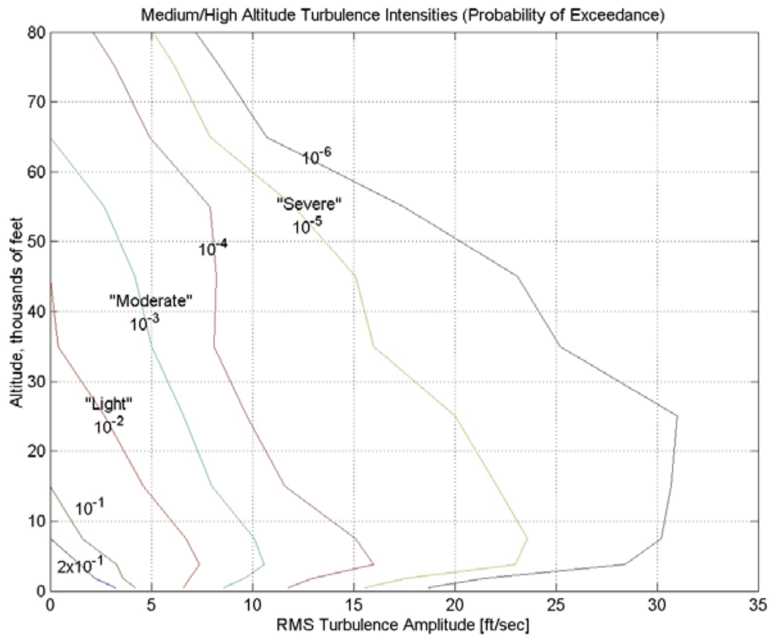
### Medium/High Altitudes (Altitude > 2000 feet)

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 |
|---|---|
| $L_u = L_v = L_w = 1750\ ft$ | $L_u = 2L_v = 2L_w = 1750\ ft$ |

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: $\sigma_u = \sigma_v = \sigma_w$.

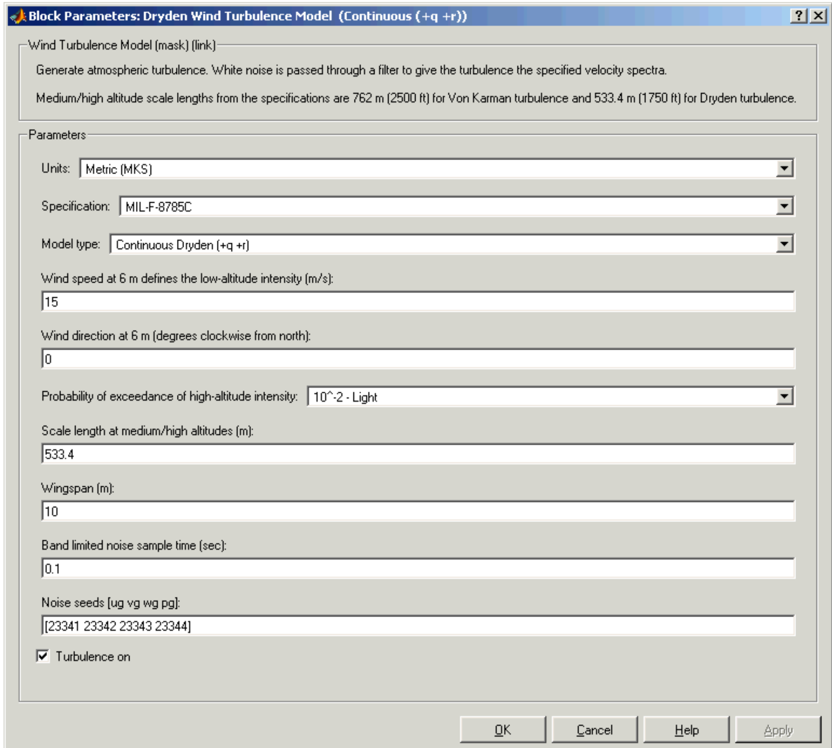
The turbulence axes orientation in this region is defined as being aligned with the body coordinates.




### 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.

## Dialog Box




**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**

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 (m)**

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 (sec)**
The sample time at which the unit variance white noise signal is generated.

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

**Turbulence on**
Selecting the check box generates the turbulence signals.

## Inputs and Outputs

The first input is altitude, in units selected.

The second input is aircraft speed, in units selected.

The third input is a direction cosine matrix.

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

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

## 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 factions (except altitude)

## Examples

See the Airframe subsystem in the `aeroblk_HL20` demo for an example of this block.

## 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.

Hoblit, 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 (Discrete)

Discrete Wind Gust Model

Wind Shear Model

Von Karman Wind Turbulence Model (Continuous)

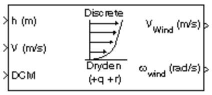
## Purpose

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 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 Ω radians per meter, the circular frequency ω is calculated by multiplying V by Ω. The following table displays the component spectra functions:

| | MIL-F-8785C | MIL-HDBK-1797 |
|---|---|---|
| **Longitudinal** | | |
| $\Phi_u(\omega)$ | $\frac{2\sigma_u^2 L_u}{\pi V}\cdot\frac{1}{1+\left(L_u\frac{\omega}{V}\right)^2}$ | $\frac{2\sigma_u^2 L_u}{\pi V}\cdot\frac{1}{1+\left(L_u\frac{\omega}{V}\right)^2}$ |
| $\Phi_p(\omega)$ | $\frac{\sigma_w^2}{VL_w}\cdot\frac{0.8\left(\frac{\pi L_w}{4b}\right)^{\frac{1}{3}}}{1+\left(\frac{4b\omega}{\pi V}\right)^2}$ | $\frac{\sigma_w^2}{2VL_w}\cdot\frac{0.8\left(\frac{2\pi L_w}{4b}\right)^{\frac{1}{3}}}{1+\left(\frac{4b\omega}{\pi V}\right)^2}$ |
| **Lateral** | | |
| $\Phi_v(\omega)$ | $\frac{\sigma_v^2 L_v}{\pi V}\cdot\frac{1+3\left(L_v\frac{\omega}{V}\right)^2}{\left[1+\left(L_v\frac{\omega}{V}\right)^2\right]^2}$ | $\frac{2\sigma_v^2 L_v}{\pi V}\cdot\frac{1+12\left(L_v\frac{\omega}{V}\right)^2}{\left[1+4\left(L_v\frac{\omega}{V}\right)^2\right]^2}$ |

| | MIL-F-8785C | MIL-HDBK-1797 |
|---|---|---|
| $\Phi_r(\omega)$ | $\dfrac{\mp\left(\frac{\omega}{V}\right)^2}{1+\left(\frac{3b\omega}{\pi V}\right)^2}\cdot\Phi_v(\omega)$ | $\dfrac{\mp\left(\frac{\omega}{V}\right)^2}{1+\left(\frac{3b\omega}{\pi V}\right)^2}\cdot\Phi_v(\omega)$ |
| **Vertical** | | |
| $\Phi_w(\omega)$ | $\dfrac{\sigma_w^2 L_w}{\pi V}\cdot\dfrac{1+3(L_w\frac{\omega}{V})^2}{[1+(L_w\frac{\omega}{V})^2]^2}$ | $\dfrac{2\sigma_w^2 L_w}{\pi V}\cdot\dfrac{1+12(L_w\frac{\omega}{V})^2}{[1+4(L_w\frac{\omega}{V})^2]^2}$ |
| $\Phi_q(\omega)$ | $\dfrac{\pm\left(\frac{\omega}{V}\right)^2}{1+\left(\frac{4b\omega}{\pi V}\right)^2}\cdot\Phi_w(\omega)$ | $\dfrac{\pm\left(\frac{\omega}{V}\right)^2}{1+\left(\frac{4b\omega}{\pi V}\right)^2}\cdot\Phi_w(\omega)$ |

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:

| | | |
|---|---|---|
| $p_g = \dfrac{\partial w_g}{\partial y}$ | $q_g = \dfrac{\partial w_g}{\partial x}$ | $r_g = -\dfrac{\partial v_g}{\partial x}$ |
| $p_g = \dfrac{\partial w_g}{\partial y}$ | $q_g = \dfrac{\partial w_g}{\partial x}$ | $r_g = \dfrac{\partial v_g}{\partial x}$ |
| $p_g = -\dfrac{\partial w_g}{\partial y}$ | $q_g = -\dfrac{\partial w_g}{\partial x}$ | $r_g = \dfrac{\partial v_g}{\partial x}$ |

The variations affect only the vertical ($q_g$) and lateral ($r_q$) 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 |
|---|---|---|
| **Longitudinal** | | |
| $u_g$ | $\left(1-\frac{V}{L_u}T\right)u_g + \sqrt{2\frac{V}{L_u}T}\frac{\sigma_u}{\sigma_\eta}\eta_1$ | $\left(1-\frac{V}{L_u}T\right)u_g + \sqrt{2\frac{V}{L_u}T}\frac{\sigma_u}{\sigma_\eta}\eta_1$ |

| | MIL-F-8785C | MIL-HDBK-1797 |
|---|---|---|
| $p_g$ | $\left(1-\frac{2.6}{\sqrt{L_w b}}T\right)p_g + \sqrt{2\frac{2.6}{\sqrt{L_w b}}T}\,\frac{\frac{0.95}{\sqrt[3]{2L_w b^2}}\sigma_w}{\sigma_\eta}\eta_4$ | $\left(1-\frac{2.6}{\sqrt{2L_w b}}T\right)p_g + \sqrt{2\frac{2.6}{\sqrt{2L_w b}}T}\,\frac{\frac{1.9}{\sqrt{2L_w b}}\sigma_w}{\sigma_\eta}\eta_4$ |
| **Lateral** | | |
| $v_g$ | $\left(1-\frac{V}{L_u}T\right)v_g + \sqrt{2\frac{V}{L_u}T}\frac{\sigma_v}{\sigma_\eta}\eta_2$ | $\left(1-\frac{V}{L_u}T\right)v_g + \sqrt{2\frac{V}{L_u}T}\frac{\sigma_v}{\sigma_\eta}\eta_2$ |
| $r_g$ | $\left(1-\frac{\pi V}{3b}T\right)r_g \mp \frac{\pi}{3b}(v_g - v_{g_{past}})$ | $\left(1-\frac{\pi V}{3b}T\right)r_g \mp \frac{\pi}{3b}(v_g - v_{g_{past}})$ |
| **Vertical** | | |
| $w_g$ | $\left(1-\frac{V}{L_u}T\right)w_g + \sqrt{2\frac{V}{L_u}T}\frac{\sigma_w}{\sigma_\eta}\eta_3$ | $\left(1-\frac{V}{L_u}T\right)w_g + \sqrt{2\frac{V}{L_u}T}\frac{\sigma_w}{\sigma_\eta}\eta_3$ |
| $q_g$ | $\left(1-\frac{\pi V}{4b}T\right)q_g \pm \frac{\pi}{4b}(w_g - w_{g_{past}})$ | $\left(1-\frac{\pi V}{4b}T\right)q_g \pm \frac{\pi}{4b}(w_g - w_{g_{past}})$ |

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 |
|---|---|
| $L_w = h$ <br> $L_u = L_v = \frac{h}{(0.177 + 0.000823h)^{1.2}}$ | $2L_w = h$ <br> $L_u = 2L_v = \frac{h}{(0.177 + 0.000823h)^{1.2}}$ |

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.

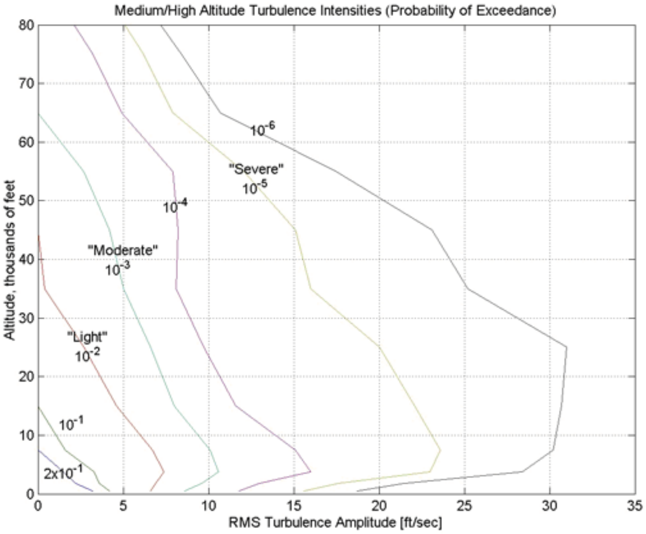
### Medium/High Altitudes (Altitude > 2000 feet)

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 |
|---|---|
| $L_u = L_v = L_w = 1750\ ft$ | $L_u = 2L_v = 2L_w = 1750\ ft$ |

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: $\sigma_u = \sigma_v = \sigma_w$.

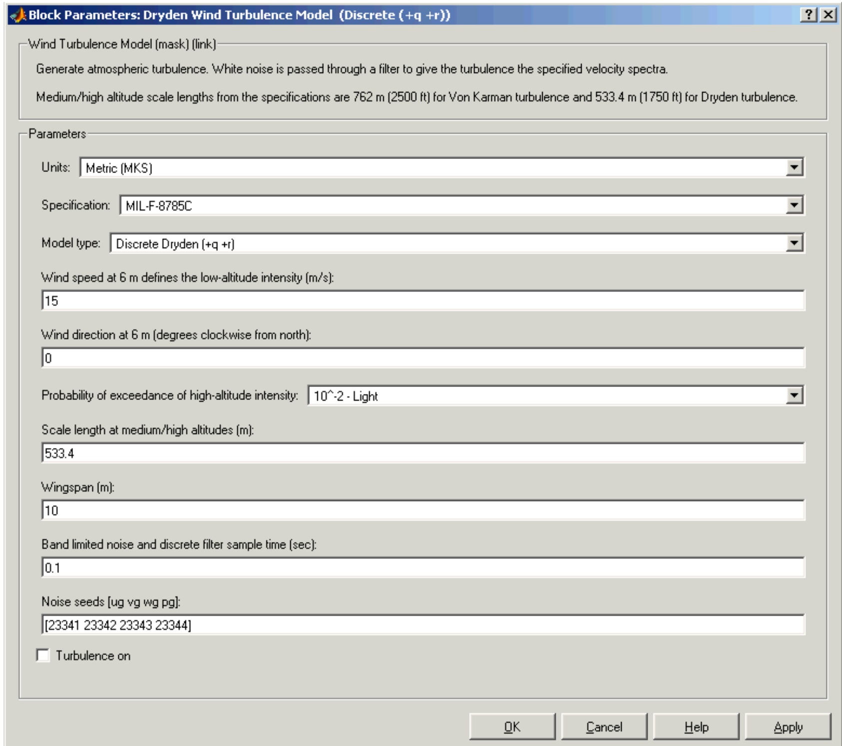
The turbulence axes orientation in this region is defined as being aligned with the body coordinates.




### 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.

## Dialog Box




### Units

Define the units of wind speed due to the turbulence.

| Units | Wind Velocity | Altitude | Airspeed |
|---|---|---|---|
| `Metric (MKS)` | Meters/second | Meters | Meters/second |

| Units | Wind Velocity | Altitude | Airspeed |
|---|---|---|---|
| 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

The first input is altitude, in units selected.

The second input is aircraft speed, in units selected.

The third input is a direction cosine matrix.

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

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

## 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 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.

Hoblit, 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

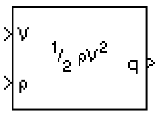
## Purpose

Compute dynamic pressure using velocity and air density

## Library

Flight Parameters

## Description



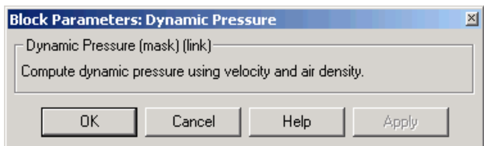

The Dynamic Pressure block computes dynamic pressure.

Dynamic pressure is defined as

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

where $\rho$ is air density and $V$ is velocity.

## Dialog Box




## Inputs and Outputs

The first input is velocity vector.

The second input is air density.

The output of the block is dynamic pressure.

## Examples

See the Airframe subsystem in the `aeroblk_HL20` demo for an example of this block.

## See Also

Aerodynamic Forces and Moments

Mach Number

# ECEF Position to LLA

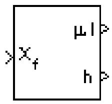
## Purpose

Calculate geodetic latitude, longitude, and altitude above planetary ellipsoid from Earth-centered Earth-fixed (ECEF) position

## Library

Utilities/Axes Transformations

## Description




The ECEF Position to LLA block converts a 3-by-1 vector of ECEF position $(\underline{p})$ into geodetic latitude $(\underline{\mu})$, longitude $(\underline{\iota})$, and altitude $(\underline{h})$ above the planetary ellipsoid.

The ECEF position is defined as

$$\underline{p} = \begin{bmatrix} \underline{p}_x \\ \underline{p}_y \\ \underline{p}_z \end{bmatrix}$$

Longitude is calculated from the ECEF position by

$$\iota = \operatorname{atan}\left(\frac{p_y}{p_x}\right)$$

Geodetic latitude $(\underline{\mu})$ 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 $(\underline{\mu})$ and reduced latitude $(\underline{\beta})$. An initial guess takes the form:

$$\beta = \operatorname{atan}\left(\frac{p_z}{(1-f)s}\right)$$

$$\mu = \operatorname{atan}\left(\frac{p_z + \frac{e^2(1-f)}{(1-e^2)}R(\sin\beta)^3}{s - e^2 R(\cos\beta)^3}\right)$$

where $R$ is the equatorial radius, $f$ 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 $(\beta)$ is recalculated using

$$\beta = \operatorname{atan}\left(\frac{(1-f)\sin\mu}{\cos\mu}\right)$$

and geodetic latitude $(\mu)$ is reevaluated. This last step is repeated until $\mu$ converges.

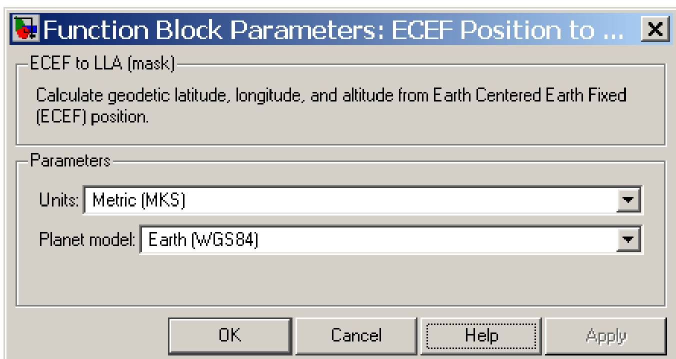
The altitude $(h)$ above the planetary ellipsoid is calculated with

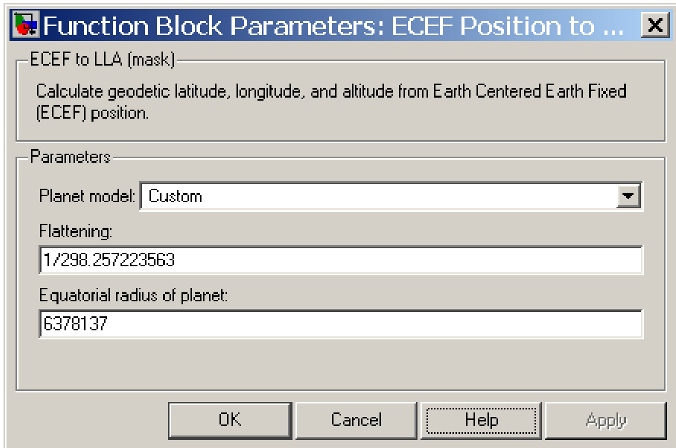
$$h = s\cos\mu + [p_z + e^2 N \sin\mu]\sin\mu - N$$

where the radius of curvature in the vertical prime $(N)$ is given by

$$N = \frac{R}{\sqrt{1 - e^2(\sin\mu)^2}}$$

## Dialog Box






**Units**

Specifies the parameter and output units:

| Units | Position | Equatorial Radius | Altitude |
|---|---|---|---|
| `Metric (MKS)` | Meters | Meters | Meters |
| `English` | Feet | Feet | Feet |

This option is only available when **Planet model** is set to `Earth (WGS84)`.

**Planet model**

Specifies the planet model to use, `Custom` or `Earth (WGS84)`.

**Flattening**

Specifies the flattening of the planet.

This option is available only with **Planet model** set to `Custom`.

**Equatorial radius of planet**

Specifies the radius of the planet at its equator. The equatorial radius units should be the same as the desired units for ECEF position.

This option is available only with **Planet model** set to `Custom`.

## Inputs and Outputs

The input is a 3-by-1 vector containing the position in ECEF frame.

The first output is a 2-by-1 vector containing geodetic latitude and longitude, in degrees.

The second output is a scalar value of altitude above the planetary ellipsoid, in the same units as the ECEF position.

## Assumptions and Limitations

This implementation generates a geodetic latitude that lies between ±90 degrees, and longitude that lies between ±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.

## References

Stevens, B. L., and F. L. Lewis, *Aircraft Control and Simulation*, John Wiley & Sons, New York, 1992.

Zipfel, P. H., *Modeling and Simulation of Aerospace Vehicle Dynamics*, AIAA Education Series, Reston, Virginia, 2000.

"Atmospheric and Space Flight Vehicle Coordinate Systems," ANSI/AIAA R-004-1992.

## See Also

See "About Aerospace Coordinate Systems" on page 2-21.

Direction Cosine Matrix ECEF to NED

Direction Cosine Matrix ECEF to NED to Latitude and Longitude

Geocentric to Geodetic Latitude

LLA to ECEF Position

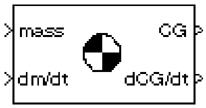
Radius at Geocentric Latitude

**Purpose** Calculate center of gravity location

**Library** Mass Properties

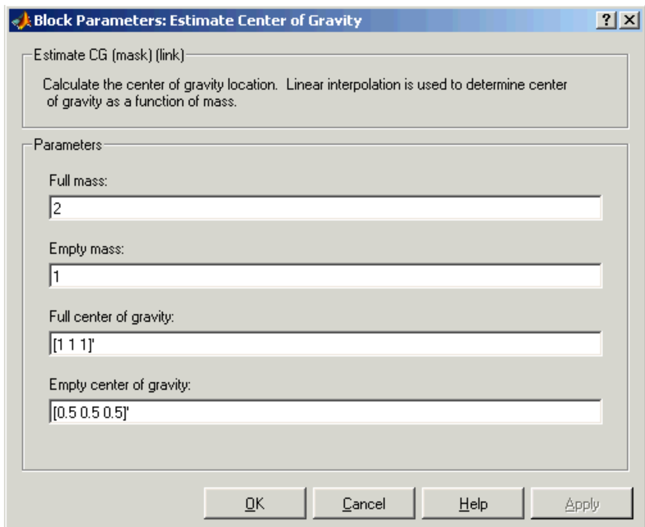
**Description**

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 center of gravity as a function of mass. The rate of change of center of gravity is a linear function of rate of change of mass.

**Dialog Box**




**Full mass**
Specifies the gross mass of the craft.

**Empty mass**
Specifies the empty mass of the craft.

**Full center of gravity**
Specifies the center of gravity at gross mass of the craft.

**Empty center of gravity**
Specifies the center of gravity at empty mass of the craft.

## Inputs and Outputs

The first input is the mass.

The second input is the rate of change of mass.

The first output is the center of gravity location.

The second output is the rate of change of center of gravity location.

## Examples

See the `aeroblk_vmm` demo for an example of this block.

## See Also

Aerodynamic Forces and Moments

Estimate Inertia Tensor

Moments About CG Due to Forces

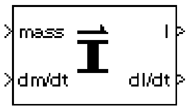
## Purpose

Calculate inertia tensor

## Library

Mass Properties

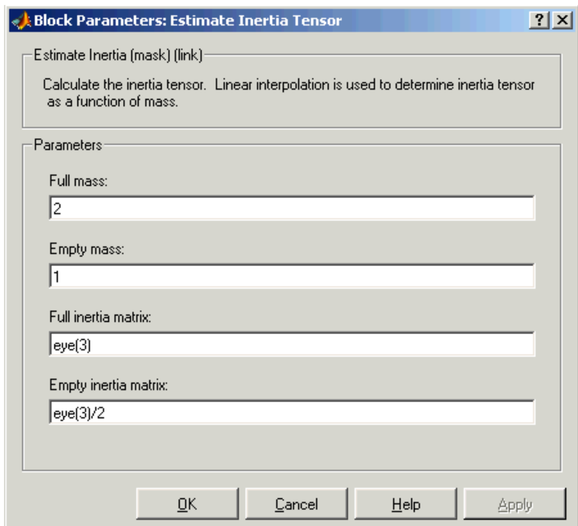
## Description




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.

## Dialog Box




**Full mass**
Specifies the gross mass of the craft.

**Empty mass**
Specifies the empty mass of the craft.

**Full inertia matrix**
Specifies the inertia tensor at gross mass of the craft.

**Empty inertia matrix**
Specifies the inertia tensor at empty mass of the craft.

## Inputs and Outputs

The first input is mass.

The second input is rate of change of mass.

The first output is inertia tensor.

The second output is rate of change of inertia tensor.

## See Also

Estimate Center of Gravity

Symmetric Inertia Tensor

**Purpose** Convert Euler angles to direction cosine matrix

**Library** Utilities/Axes Transformations

**Description**


The Euler Angles to Direction Cosine Matrix block converts the three Euler rotation angles into a 3-by-3 direction cosine matrix (DCM). The DCM matrix performs the coordinate transformation of a vector in inertial axes $(ox_0, oy_0, oz_0)$ into a vector in body axes $(ox_3, oy_3, oz_3)$. The order of the axis rotations required to bring this about is:

1. A rotation about $oz_0$ through the yaw angle $(\psi)$ to axes $(ox_1, oy_1, oz_1)$
2. A rotation about $oy_1$ through the pitch angle $(\theta)$ to axes $(ox_2, oy_2, oz_2)$
3. A rotation about $ox_2$ through the roll angle $(\phi)$ to axes $(ox_3, oy_3, oz_3)$

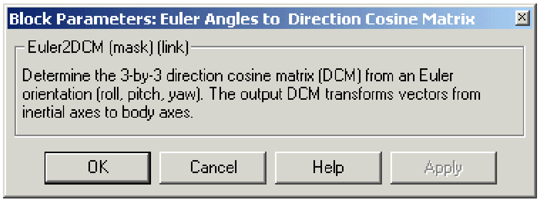
$$\begin{bmatrix} ox_3 \\ oy_3 \\ oz_3 \end{bmatrix} = DCM \begin{bmatrix} ox_0 \\ oy_0 \\ oz_0 \end{bmatrix}$$

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

Combining the three axis transformation matrices defines the following DCM.

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

## Dialog Box




## Inputs and Outputs

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

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

## See Also

Direction Cosine Matrix to Euler Angles

Direction Cosine Matrix to Quaternions

Euler Angles to Quaternions

Quaternions to Direction Cosine Matrix

Quaternions to Euler Angles

# Euler Angles to Quaternions

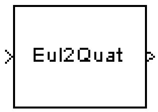
## Purpose

Convert Euler angles to quaternion

## Library

Utilities/Axes Transformations

## Description




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

A quaternion vector represents a rotation about a unit vector ($\mu_x$ $\mu_y$ $\mu_z$) through an 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,

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

where, for the purposes of multiplication,

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

$$\boldsymbol{ij} = -\boldsymbol{ji} = \boldsymbol{k}$$

$$\boldsymbol{jk} = -\boldsymbol{kj} = \boldsymbol{i},$$
$$\boldsymbol{ki} = -\boldsymbol{ik} = \boldsymbol{j}$$

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

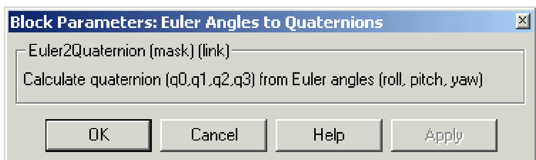
# Euler Angles to Quaternions

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

Expanding the preceding representation gives the four quaternion elements following.

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

## Dialog Box




## Inputs and Outputs

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

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

## See Also

Direction Cosine Matrix to Euler Angles

Direction Cosine Matrix to Quaternions

Euler Angles to Direction Cosine Matrix

Quaternions to Direction Cosine Matrix

Quaternions to Euler Angles

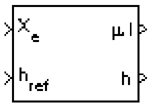
## Purpose

Estimate geodetic latitude, longitude, and altitude from flat Earth position

## Library

Utilities/Axes Transformations

## Description




The Flat Earth to LLA block converts a 3-by-1 vector of Flat Earth position $(\underline{p})$ into geodetic latitude $(\underline{\mu})$, longitude $(\underline{\iota})$, and altitude $(\underline{h})$. 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 $(\underline{\psi})$ 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:

$$R_N = \frac{R}{\sqrt{1-(2f-f^2)\sin^2\mu}}$$

$$R_M = R_N \frac{1-(2f-f^2)}{1-(2f-f^2)\sin^2\mu}$$

where $(R)$ is the equatorial radius of the planet and $(\underline{f})$ 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

$$d\mu = \operatorname{atan}\left(\frac{1}{R_M}\right)dN$$

$$d\iota = \operatorname{atan}\left(\frac{1}{R_N\cos\mu}\right)dE$$

The output latitude and longitude are simply the initial latitude and longitude plus the small changes in latitude and longitude.

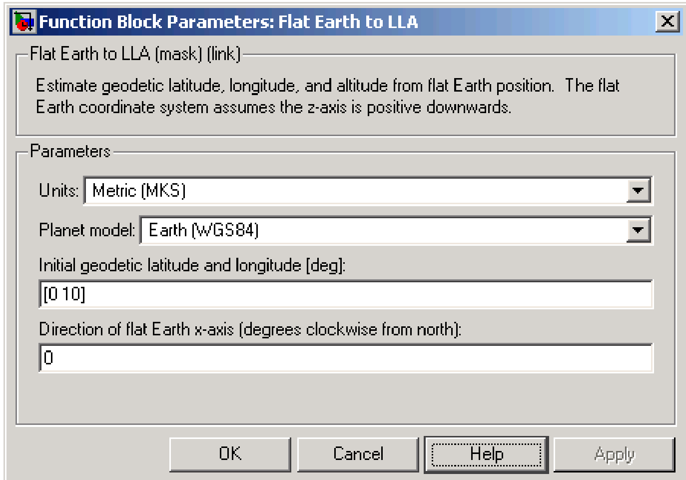
$$\mu = \mu_o + d\mu$$

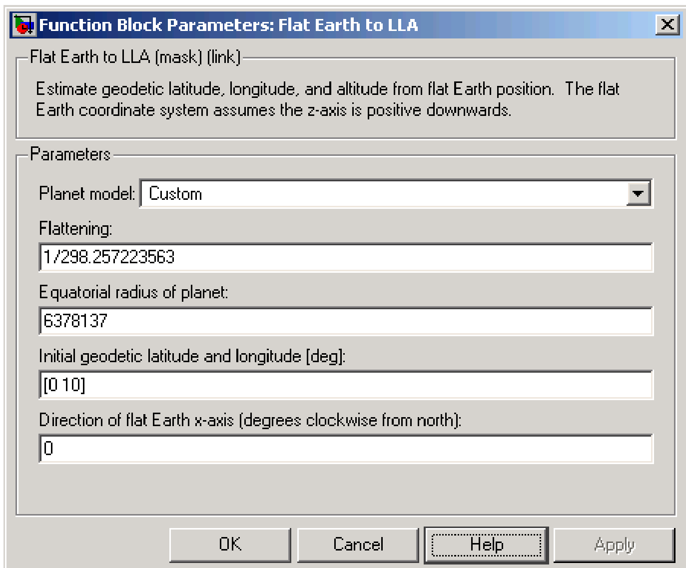
$$\iota = \iota_o + d\iota$$

The altitude is the negative flat Earth $z$-axis value minus the reference height $(h_{ref})$.

$$h = -p_z - h_{ref}$$

## Dialog Box


Function Block Parameters: Flat Earth to LLA
Flat Earth to LLA (mask) (link)
Estimate geodetic latitude, longitude, and altitude from flat Earth position. The flat Earth coordinate system assumes the z-axis is positive downwards.
Parameters
Units: Metric (MKS)
Planet model: Earth (WGS84)
Initial geodetic latitude and longitude [deg]:
[0 10]
Direction of flat Earth x-axis (degrees clockwise from north):
0
OK
Cancel
Help
Apply


Function Block Parameters: Flat Earth to LLA
Flat Earth to LLA (mask) (link)
Estimate geodetic latitude, longitude, and altitude from flat Earth position. The flat Earth coordinate system assumes the z-axis is positive downwards.
Parameters
Planet model: Custom
Flattening:
1/298.257223563
Equatorial radius of planet:
6378137
Initial geodetic latitude and longitude [deg]:
[0 10]
Direction of flat Earth x-axis (degrees clockwise from north):
0
OK
Cancel
Help
Apply

**Units**

Specifies the parameter and output units:

| Units | Position | Equatorial Radius | Altitude |
|---|---|---|---|
| `Metric (MKS)` | Meters | Meters | Meters |
| `English` | Feet | Feet | Feet |

This option is only available when **Planet model** is set to `Earth (WGS84)`.

**Planet model**

Specifies the planet model to use: `Custom` or `Earth (WGS84)`.

**Flattening**

Specifies the flattening of the planet. This option is only available with **Planet model 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 flat Earth position. This option is only available with **Planet model Custom**.

**Initial geodetic latitude and longitude**

Specifies the reference location, in degrees of latitude and longitude, for the origin of the estimation and the origin of the flat Earth coordinate system.

**Direction of flat Earth x-axis (degrees clockwise from north)**

Specifies angle used for converting flat Earth x and y coordinates to North and East coordinates.

## Inputs and Outputs

The first input is a 3-by-1 vector containing the position in flat Earth frame.

The second input is a scalar value of reference altitude in the same units for flat Earth position.

The first output is a 2-by-1 vector containing geodetic latitude and longitude, in degrees.

The second output is a scalar value of altitude above the input reference altitude, in same units as flat Earth position.

## Assumptions and 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 the smaller the variations in latitude. Additionally, longitude is singular at the poles.

## Example

See the `asbhl20` demo for an example of this block.

## References

Etkin, B., *Dynamics of Atmospheric Flight*, John Wiley & Sons, New York, 1972.

Stevens, B. L., and F. L. Lewis, *Aircraft Control and Simulation*, Second Edition, John Wiley & Sons, New York, 2003.

## 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

# FlightGear Preconfigured 6DoF Animation

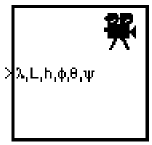
## Purpose

Connect model to FlightGear flight simulator

## Library

Animation/Flight Simulator Interfaces

## Description



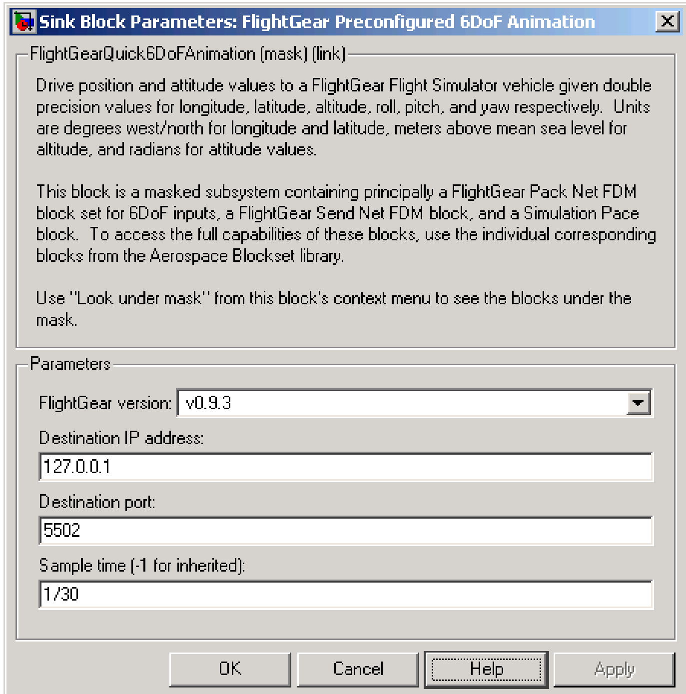

The FlightGear Preconfigured 6DoF Animation block lets you drive position and attitude values to a FlightGear flight simulator vehicle given double precision values for longitude $(\lambda)$, latitude ($L$), altitude ($h$), roll $(\phi)$, pitch $(\theta)$, and yaw $(\psi)$ respectively.

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.

The block is additionally configured as a SimViewingDevice, so that if you generate code for your model using Real-Time Workshop and connect to the running target code using the Real-Time Workshop External Mode available from the model's toolbar, then Simulink can obtain the data from the target on the fly and transmit position and attitude data to FlightGear. The SimViewingDevice facility is described in the Simulink documentation.

This block does not produce deployable code, but can be used with Real-Time Workshop external mode as a SimViewingDevice.

## Dialog Box




**FlightGear version**

Select your FlightGear software version.

Supported versions: `v0.9.3`, `v0.9.8/0.9.8a`, `v0.9.9`, `v0.9.10`.

**Destination IP address**

Specify your destination IP address.

**Destination port**

Specify your destination port.

**Sample time**

Specify the sample time (-1 for inherited).

## Inputs and Outputs

The input is a vector containing longitude, latitude, altitude, roll, pitch, and yaw, in double precision. Units are degrees west/north for longitude and latitude, meters above mean sea level for altitude, and radians for attitude values.

## Reference

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

Send net_fdm Packet to FlightGear

Simulation Pace

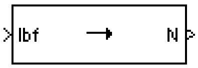
## Purpose

Convert from force units to desired force units

## Library

Utilities/Axes Transformations

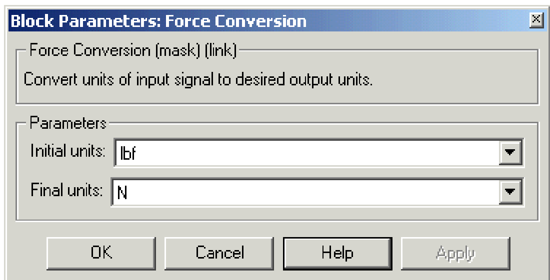
## Description




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

The Force Conversion block icon displays the input and output units selected from the **Initial units** and the **Final units** lists.

## Dialog Box




**Initial units**
: Specifies the input units.

**Final units**
: Specifies the output units.

The following conversion units are available:

| | |
|---|---|
| `lbf` | Pound force |
| `N` | Newtons |

## Inputs and Outputs

The input is force in initial force units.

The output is force in final force units.

## See Also

Acceleration Conversion

Angle Conversion

Angular Acceleration Conversion

Angular Velocity Conversion

Density Conversion

Length Conversion

Mass Conversion

Pressure Conversion

Temperature Conversion

Velocity Conversion

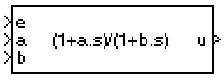
## Purpose

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

## Library

GNC/Controls

## Description



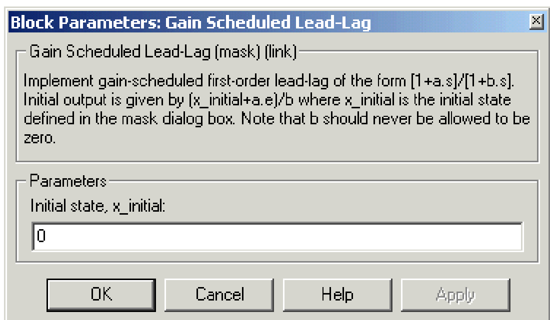

The Gain Scheduled Lead-Lag block implements a first-order lag of the form

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

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

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

## Dialog Box




**Initial state, x_initial**

The initial internal state for the filter `x_initial`. Given this initial state, the initial output is given by

$$u\big|_{t=0} = \frac{xinitial + ae}{b}$$

### Inputs and Outputs

The first input is the filter input.

The second input is the numerator coefficient.

The third input is the denominator coefficient.

The output is the filter output.

# Generate Run Script

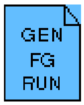
## Purpose

Generate FlightGear run script on current computer

## Library

Animation/Flight Simulator Interfaces

## Description




The Generate Run Script block generates a customized FlightGear run script on the current platform.

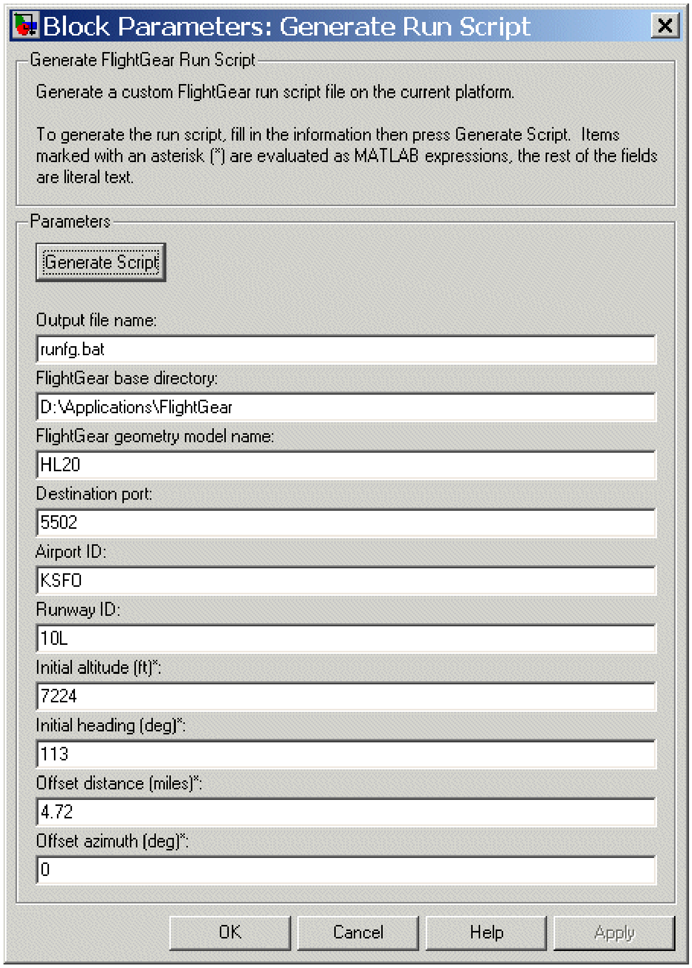
To generate the run script, fill the required information into the dialog's fields, then click **Generate Script**.

Fields in the dialog marked with an asterisk (*) are evaluated as MATLAB expressions. The other fields are treated as literal text.

### For More Information About FlightGear

See "Creating a FlightGear Run Script" on page 2-46 for more about FlightGear.

## Dialog Box




**Generate Script**

Click to generate a run script for FlightGear. Do not click this button until you have entered the correct information in the dialog fields.

**Output file name**
: Specify the name of the output file. The file name is the name of the command you will use to start FlightGear with these initial parameters. The file must have the `.bat` extension.

**FlightGear base directory**
: Specify the name of your FlightGear installation directory.

**FlightGear geometry model name**
: Specify the name of the folder containing the desired model geometry in the *`FlightGear`*`\data\Aircraft` directory.

**Destination port**
: Specify your network flight dynamics model (fdm) port. For more information, see the Send net_fdm Packet to FlightGear block reference.

**Airport ID**
: Specify the airport ID. The list of supported airports is available in the FlightGear interface, under **Location**.

**Runway ID**
: Specify the runway ID.

**Initial altitude**
: Specify the initial altitude of the aircraft, in feet.

**Initial heading**
: Specify the initial heading of the aircraft, in degrees.

**Offset distance**
: Specify the offset distance of the aircraft from the airport, in miles.

**Offset azimuth**
: Specify the offset azimuth of the aircraft, in degrees.

## Examples

See the `asbhl20` demo for an example of this block.

## See Also

FlightGear Preconfigured 6DoF Animation

Pack net_fdm Packet for FlightGear

Send net_fdm Packet to FlightGear

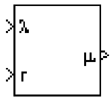
## Purpose

Convert geocentric latitude to geodetic latitude

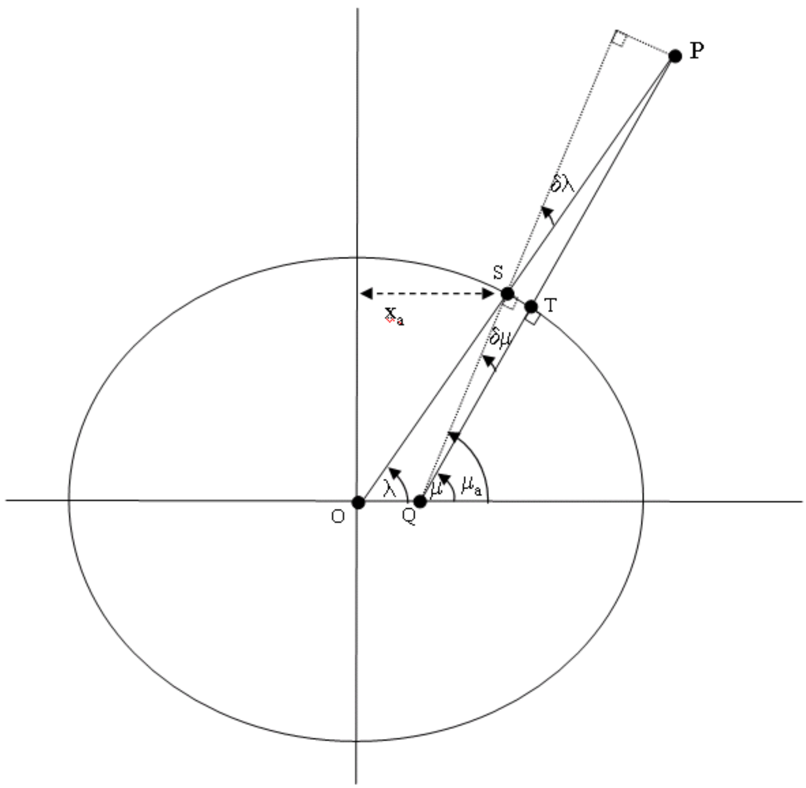
## Library

Utilities/Axes Transformations

## Description




The Geocentric to Geodetic Latitude block converts a geocentric latitude $(\lambda)$ into geodetic latitude $(\mu)$. There are a number of geometric relationships that are used to calculate the geodetic latitude in this noniterative method. A number of angles and points are involved in the calculation, which are shown in following figure.




.

Given geocentric latitude $(\lambda)$ and the radius $(r)$ from the center of the planet (O) to the center of gravity (P), this noniterative method starts by computing values for the point of $r$ that intercepts the surface of the planet (S). By rearranging the equation for an ellipse, the horizontal coordinate, $(x_a)$ is determined. When equatorial radius $(R)$, polar radius $((1-f)R)$ and $x_a \tan\lambda$, are substituted for semi-major axis, semi-minor axis and vertical coordinate $(y_a)$, the resulting equation for $x_a$ has the following form:

$$x_a = \frac{(1-f)R}{\sqrt{\tan^2\lambda + (1-f)^2}}$$

To determine the geodetic latitude at S $\mu_a$, the equation for an ellipse with equatorial radius $(R)$, polar radius $((1-f)R)$ is used again. This time it is used to define $y_a$ in terms of $x_a$.

$$y_a = \sqrt{R^2 - x_a^2}(1-f)$$

Additionally, the relationship between geocentric latitude at the planet's surface and geodetic latitude is used.

$$\mu_a = \operatorname{atan}\left(\frac{\tan\lambda}{(1-f)^2}\right)$$

Using the relationship $\tan\lambda = y_a / x_a$ and the two equations above, the resulting equation for $\mu_a$ is obtained.

$$\mu_a = \operatorname{atan}\left(\frac{\sqrt{R^2 - x_a{}^2}}{(1-f)x_a}\right)$$

The correct sign of $\mu_a$ is determined by testing $\lambda$ and if $\lambda$ is less than zero $\mu_a$ changes sign accordingly.

In order to calculate the geodetic latitude of P, a number of geometric relationships are required to be calculated. These calculations follow.

The radius $(r_a)$ from the center of the planet (O) to the surface of the planet (S) is calculated by using trigonometric relationship.

$$r_a = \frac{x_a}{\cos\lambda}$$

The distance from S to P is defined by:

$$l = r - r_a$$

The angular difference between geocentric latitude and geodetic latitude at S $(\delta\lambda)$ is defined by:

$$\delta\lambda = \mu_a - \lambda$$

Using $l$ and $\delta\lambda$, the mean sea-level altitude $(h)$ is estimated.

$$h = l\cos\delta\lambda$$

The equation for the radius of curvature in the Meridian $(\rho_a)$ at $\mu_a$ is

$$\rho_a = \frac{R(1-f)^2}{(1-(2f-f^2)\sin^2\mu_a)^{3/2}}$$

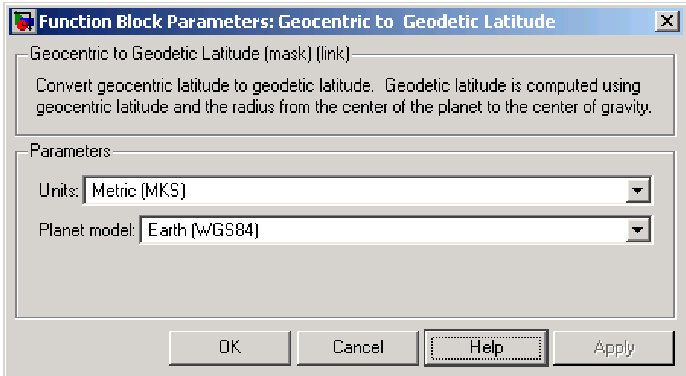
Using $l$, $\delta\lambda$, $h$, and $\rho_a$, the angular difference between geodetic latitude at S $(\mu)$ and geodetic latitude at P $(\mu_a)$ is defined as:

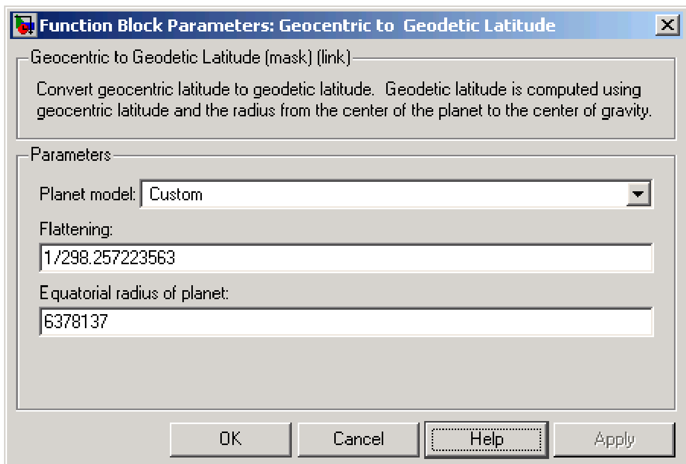
$$\delta\mu = \operatorname{atan}\left(\frac{l\sin\delta\lambda}{\rho_a + h}\right)$$

Subtracting $\delta\mu$ from $\mu_a$ then gives $\mu$.

$$\mu = \mu_a - \delta\mu$$

## Dialog Box




**Units**

Specifies the parameter and output units:

| Units | Radius from CG to Center of Planet | Equatorial Radius |
|---|---|---|
| Metric (MKS) | Meters | Meters |
| English | Feet | Feet |

This option is only available when **Planet model** is set to Earth (WGS84).

**Planet model**
Specifies the planet model to use: Custom or Earth (WGS84).

**Flattening**
Specifies the flattening of the planet. This option is only available with **Planet model** 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 radius. This option is only available with **Planet model** set to Custom.

## Inputs and Outputs

The first input is a scalar value of geocentric latitude, in degrees.

The second input is a scalar value of radius from center of the planet to the center of gravity.

The output is a scalar value of geodetic latitude, in degrees.

## Assumptions and Limitations

This implementation generates a geodetic latitude that lies between $\pm 90$ degrees.

## References

Jackson, E. B., *Manual for a Workstation-based Generic Flight Simulation Program (LaRCsim) Version 1.4*, NASA TM 110164, April, 1995.

Hedgley, D. R., Jr., “An Exact Transformation from Geocentric to Geodetic Coordinates for Nonzero Altitudes,” NASA TR R-458, March, 1976.

Clynch, J. R., “Radius of the Earth - Radii Used in Geodesy,” Naval Postgraduate School, 2002, http://www.oc.nps.navy.mil/oc2902w/geodesy/radiigeo.pdf.

Stevens, B. L., and F. L. Lewis, *Aircraft Control and Simulation*, John Wiley & Sons, New York, 1992.

Edwards, C. H., and D. E. Penny, *Calculus and Analytical Geometry 2nd Edition*, Prentice-Hall, Englewood Cliffs, New Jersey, 1986.

## See Also

ECEF Position to LLA

Flat Earth to LLA

Geodetic to Geocentric Latitude

LLA to ECEF Position

# Geodetic to Geocentric Latitude

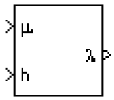
## Purpose

Convert geodetic latitude to geocentric latitude

## Library

Utilities/Axes Transformations

## Description



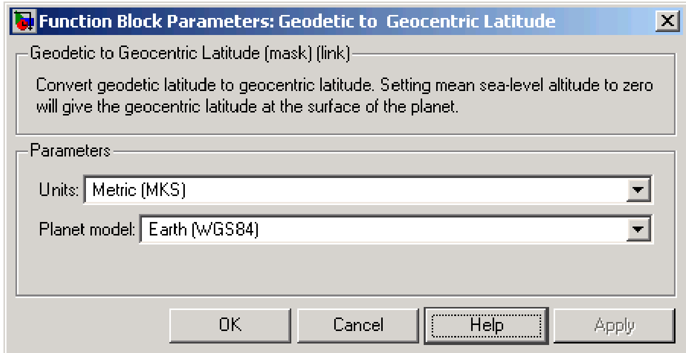

The Geodetic to Geocentric Latitude block converts a geodetic latitude $(\mu)$ into geocentric latitude $(\lambda)$. Geocentric latitude at the planet surface $(\lambda_s)$ is defined by flattening $(f)$, and geodetic latitude in the following relationship.

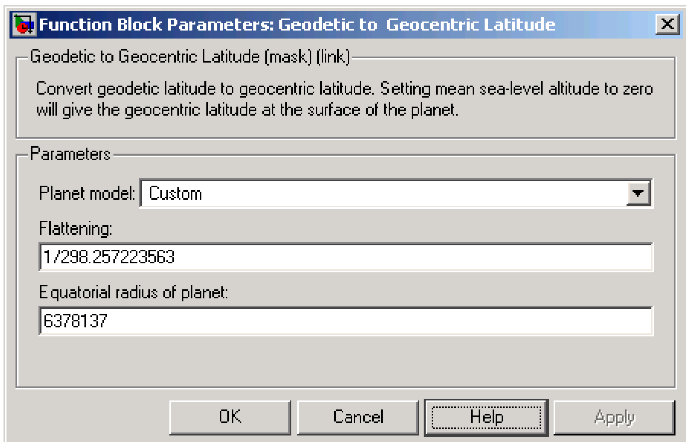
$$\lambda_s = \operatorname{atan}((1-f)^2 \tan \mu)$$

Geocentric latitude is defined by mean sea-level altitude $(h)$, geodetic latitude, radius of the planet $(r_s)$ and geocentric latitude at the planet surface in the following relationship.

$$\lambda = \operatorname{atan}\left(\frac{h \sin \mu + r_s \sin \lambda_s}{h \cos \mu + r_s \cos \lambda_s}\right)$$

## Dialog Box




**Units**

Specifies the parameter and output units:

| Units | Altitude | Equatorial Radius |
|---|---|---|
| `Metric (MKS)` | Meters | Meters |
| `English` | Feet | Feet |

This option is only available when **Planet model** is set to `Earth (WGS84)`.

**Planet model**

Specifies the planet model to use: `Custom` or `Earth (WGS84)`.

**Flattening**

Specifies the flattening of the planet. This option is only available with **Planet model** 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 altitude. This option is only available with **Planet model** set to `Custom`.

## Inputs and Outputs

The first input is a scalar value of geodetic latitude, in degrees.

The second input is a scalar value of mean sea-level altitude (MSL).

The output is a scalar value of geocentric latitude, in degrees.

## Assumptions and Limitations

This implementation generates a geocentric latitude that lies between $\pm 90$ degrees.

## Reference

Stevens, B. L., and F. L. Lewis, *Aircraft Control and Simulation*, John Wiley & Sons, New York, 1992.

## See Also

ECEF Position to LLA

Flat Earth to LLA

Geocentric to Geodetic Latitude

LLA to ECEF Position

Radius at Geocentric Latitude

# Horizontal Wind Model

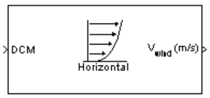
## Purpose

Transform horizontal wind into body-axes coordinates

## Library

Environment/Wind

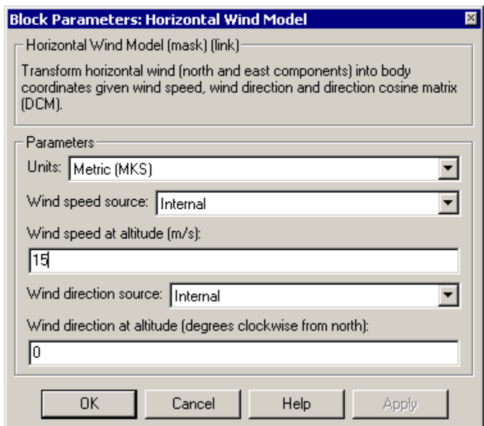
## Description




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.

## Dialog Box




**Units**

Specifies the input and output units:

| Units | Wind Speed | Wind Velocity |
|---|---|---|
| `Metric (MKS)` | Meters per second | Meters per second |

| Units | Wind Speed | Wind Velocity |
|---|---|---|
| `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

The first input is direction cosine matrix.

The second optional input is the wind speed in selected units.

The third optional input is the wind direction in degrees.

The output of the block is 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

Von Karman Wind Turbulence Model (Continuous)

Wind Shear Model

# Ideal Airspeed Correction

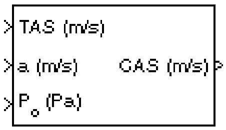
## Purpose

Calculate equivalent airspeed (EAS), calibrated airspeed (CAS), or true airspeed (TAS) from each other

## Library

Flight Parameters

## Description




The Ideal Airspeed Correction block calculates one of the following airspeeds: equivalent airspeed (EAS), calibrated airspeed (CAS), or true airspeed (TAS), from one of the other two airspeeds.

Three equations are used to implement the Ideal Airspeed Correction block. The first equation shows TAS as a function of EAS, relative pressure ratio at altitude (δ), and speed of sound at altitude (a).

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

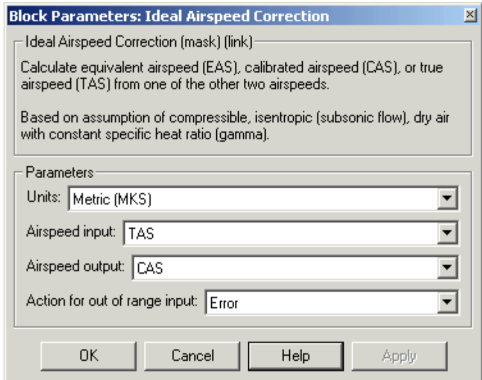
Using the compressible form of Bernoulli's equation and assuming isentropic conditions, the last two equations for EAS and CAS are derived.

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

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

In order to generate a correction table and its approximate inverse, these two equations were solved for dynamic pressure ($q$). Having values of q by a function of *EAS* and ambient pressure at altitude ($P$) or by a function of *CAS*, allows the two equations to be solved using the other's solution for $q$, thus creating a solution for *EAS* that depends on $P$ and *CAS* and a solution for *CAS* that depends on $P$ and *EAS*.

## Dialog Box




**Units**

Specifies the input and output units:

| 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 |

**Airspeed input**

Specify the airspeed input type:

| | |
|---|---|
| TAS | True airspeed |
| EAS | Equivalent airspeed |
| CAS | Calibrated airspeed |

**Airspeed output**

Specify the airspeed output type:

| Velocity Input | Velocity Output |
|---|---|
| TAS | EAS (equivalent airspeed) |
| | CAS (calibrated airspeed) |
| EAS | TAS (true airspeed) |
| | CAS (calibrated airspeed) |
| CAS | TAS (true airspeed) |
| | EAS (equivalent airspeed) |

**Action for out of range input**

Specify if an out-of-range input (supersonic airspeeds) invokes a warning, an error, or no action.

## Inputs and Outputs

The first input is the selected airspeed in the selected units.

The second input is the speed of sound in the selected units.

The third input is the static pressure in the selected units.

The output of the block is the selected airspeed in the selected units.

## Assumptions and Limitations

This block assumes that the air flow is compressible, isentropic (subsonic flow), dry air with constant specific heat ratio, $\gamma$.

## Examples

See the `aeroblk_indicated` model and the `aeroblk_calibrated` model for examples of this block.

## References

Lowry, J. T., *Performance of Light Aircraft*, AIAA Education Series, Washington, DC, 1999.

*Aeronautical Vestpocket Handbook*, United Technologies Pratt & Whitney, August, 1986.

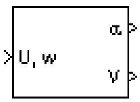
## Purpose

Calculate incidence and airspeed

## Library

Flight Parameters

## Description

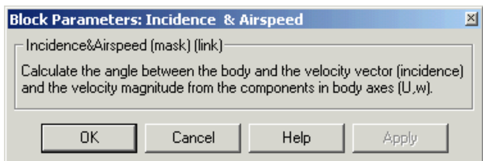



The Incidence & Airspeed block supports the 3DoF equations of motion model by calculating the angle between the velocity vector and the body, and also the total airspeed from the velocity components in the body-fixed coordinate frame.

$$\alpha = \operatorname{atan}\left(\frac{w}{u}\right)$$

$$V = \sqrt{u^2 + w^2}$$

## Dialog Box




## Inputs and Outputs

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

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

The second output is the airspeed of the body.

## Examples

See the `aeroblk_guidance` model and the `aero_guidance_airframe` model for examples of this block.

## See Also

Incidence, Sideslip & Airspeed

# Incidence, Sideslip & Airspeed

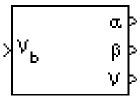
## Purpose

Calculate incidence, sideslip, and airspeed

## Library

Flight Parameters

## Description



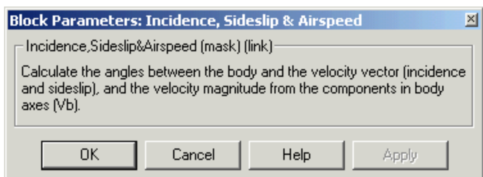

The Incidence, Sideslip & Airspeed block supports the 6DoF (Euler Angles) and 6DoF (Quaternion) models by calculating the angles between the velocity vector and the body, and also the total airspeed from the velocity components in the body-fixed coordinate frame.

$$\alpha = \mathrm{atan}\left(\frac{w}{u}\right)$$

$$\beta = \mathrm{asin}\left(\frac{v}{V}\right)$$

$$V = \sqrt{u^2 + v^2 + w^2}$$

## Dialog Box




## Inputs and Outputs

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

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

The second output of the block is the sideslip angle in radians.

The third output is the airspeed of the body.

## Examples

See Airframe in the `aeroblk_HL20` model for an example of this block.

## See Also

Incidence & Airspeed

## Purpose

Return interpolated matrix for given input

## Library

GNC/Controls

## Description


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

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

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

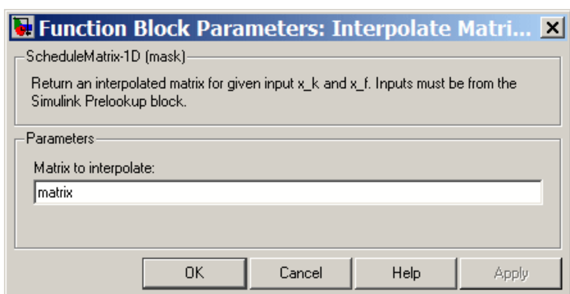
where the interpolation fraction is defined as

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

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

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

## Dialog Box



**Matrix to interpolate**

Matrix to be interpolated, with three indices and the third index labeling the interpolating values of $x$.

## Inputs and Outputs

The first input is the interpolation index $i$.

The second input is the interpolation fraction $\lambda$.

The output is the interpolated matrix.

## Assumptions and Limitations

This block must be driven from the Simulink Prelookup block.

## Examples

See the following block reference pages: 1D Controller [A(v),B(v),C(v),D(v)], 1D Observer Form [A(v),B(v),C(v),F(v),H(v)], and 1D Self-Conditioned [A(v),B(v),C(v),D(v)].

## See Also

Interpolate Matrix(x,y)

Interpolate Matrix(x,y,z)

## Purpose

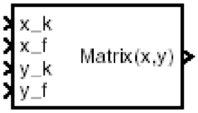
Return interpolated matrix for given inputs

## Library

GNC/Controls

## Description

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




This two-dimensional case assumes the matrix is defined as a function of two independent variables, $\boldsymbol{x} = [\, x_1 x_2\, x_3 \ldots\, x_i\, x_{i+1} \,\ldots\, x_n]$ and $\boldsymbol{y} = [\, y_1\, y_2\, y_3 \ldots\, y_j\, y_{j+1} \,\ldots\, y_m]$. 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

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

In the two-dimensional case, the interpolation is carried out first on $x$ and then $y$.

The matrix to be interpolated should be four dimensional, the first two dimensions corresponding to the matrix at each value of $x$ and $y$. For example, if you have four matrices $A$, $B$, $C$, and $D$ defined at `(x = 0.0,y = 1.0)`, `(x = 0.0,y = 3.0)`, `(x = 1.0,y = 1.0)` and `(x = 1.0,y = 3.0)`, then the input matrix is given by

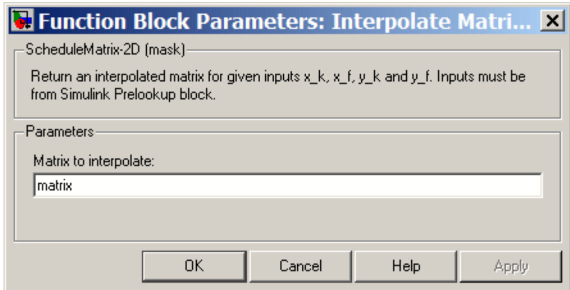
```
matrix(:,:,1,1) = A;

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

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

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

## Dialog Box




**Matrix to interpolate**

Matrix to be interpolated, with four indices and the third and fourth indices labeling the interpolating values of $x$ and $y$.

## Inputs and Outputs

The first input is the first interpolation index $i$.

The second input is the first interpolation fraction $\lambda_x$.

The third input is the second interpolation index $j$.

The fourth input is the second interpolation fraction $\lambda_y$.

The output is the interpolated matrix.

## Assumptions and Limitations

This block must be driven from the Simulink Prelookup block.

## Examples

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

## See Also

Interpolate Matrix(x)

Interpolate Matrix(x,y,z)

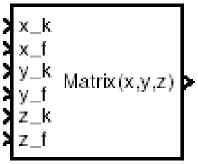
## Purpose

Return interpolated matrix for given inputs

## Library

GNC/Controls

## 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

$$x = [\, x_1\, x_2\, x_3 \,\ldots\, x_i\, x_{i+1} \,\ldots\, x_n \,],\; y = [\, y_1\, y_2\, y_3 \,\ldots\, y_j\, y_{j+1} \,\ldots\, y_m]$$
$$z = [\, z_1\, z_2\, z_3 \,\ldots\, z_k\, z_{k+1} \,\ldots\, z_p \,]$$

For given values of $x$, y, and $z$, eight matrices are interpolated. Then for

$$x_i < x < x_{i+1},\; y_j < y < y_{j+1}$$
$$z_k < z < z_{k+1}$$

the output matrix is given by

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

where the three interpolation fractions are denoted by

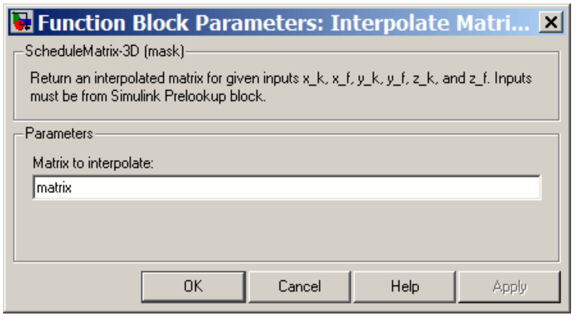
$$\lambda_x = (x - x_i)/(x_{i+1} - x_i)$$
$$\lambda_y = (y - y_j)/(y_{j+1} - y_j)$$
$$\lambda_z = (z - z_k)/(z_{k+1} - z_k)$$

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

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;
```

## Dialog Box




**Matrix to interpolate**

Matrix to be interpolated, with five indices and the third, fourth, and fifth indices labeling the interpolating values of $x$, $y$, and $z$.

## Inputs and Outputs

The first input is the first interpolation index $i$.

The second input is the first interpolation fraction $\lambda_x$.

The third input is the second interpolation index $j$.

The fourth input is the second interpolation fraction $\lambda_y$.

The fifth input is the third interpolation index $k$.

The sixth input is the third interpolation fraction $\lambda_z$.

The output is the interpolated matrix.

## Assumptions and Limitations

This block must be driven from the Simulink Prelookup block.

## Examples

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

## See Also

Interpolate Matrix(x)

Interpolate Matrix(x,y)

# Invert 3x3 Matrix

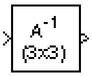
## Purpose

Compute inverse of 3-by-3 matrix using determinant

## Library

Utilities/Math Operations

## Description



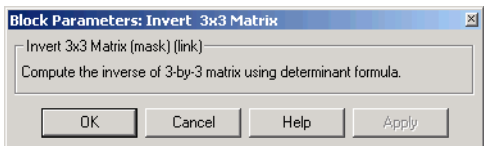

The Invert 3x3 Matrix block computes the inverse of 3-by-3 matrix using determinant formula.

The inverse of the matrix is calculated by

$$inv(A) = \frac{adj(A)}{det(A)}$$

If the $det(A) = 0$, an error is thrown and the simulation will stop.

## Dialog Box




## Inputs and Outputs

The input is a 3-by-3 matrix.

The output of the block is 3-by-3 matrix inverse of input matrix.

## See Also

Adjoint of 3x3 Matrix

Create 3x3 Matrix

Determinant of 3x3 Matrix

# ISA Atmosphere Model

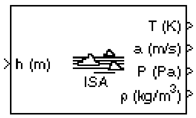
## Purpose

Implement International Standard Atmosphere (ISA)

## Library

Environment/Atmosphere

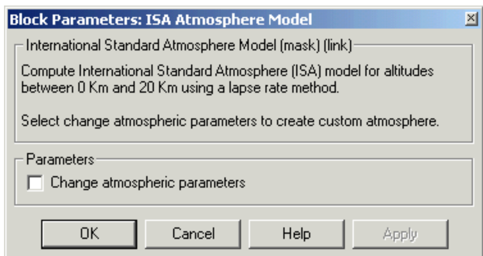
## Description




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 block icon displays the input and output metric units.

## Dialog Box




**Change atmospheric parameters**
Select to customize various atmospheric parameters to be different from the ISA values.

## Inputs and Outputs

The input is geopotential height.

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

## Assumptions and Limitations

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

## Reference

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

## See Also

COESA Atmosphere Model, Lapse Rate Model

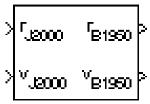
## Purpose

Transform position and velocity components from Standard Julian Epoch (J2000) to discontinued Standard Besselian Epoch (B1950)

## Library

Utilities/Axes Transformations

## Description




The Julian Epoch to Besselian Epoch block transforms two 3-by-1 vectors of Julian Epoch position ($\underline{r}_{J2000}$), and Julian Epoch velocity ($\underline{v}_{J2000}$) into Besselian Epoch position ($\underline{r}_{B1950}$), and Besselian Epoch velocity ($\underline{v}_{B1950}$). The transformation is calculated using:

$$\begin{bmatrix} \underline{r}_{B1950} \\ \underline{v}_{B1950} \end{bmatrix} = \begin{bmatrix} \underline{M}_{rr} & \underline{M}_{vr} \\ \underline{M}_{rv} & \underline{M}_{vv} \end{bmatrix}^T \begin{bmatrix} \underline{r}_{J2000} \\ \underline{v}_{J2000} \end{bmatrix}$$

where

$$(\underline{M}_{rr}, \underline{M}_{vr}, \underline{M}_{rv}, \underline{M}_{vv})$$

are defined as:

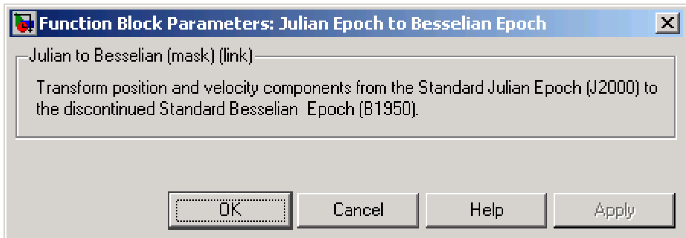
$$\underline{M}_{rr} = \begin{bmatrix} 0.9999256782 & -0.0111820611 & -0.0048579477 \\ 0.0111820610 & 0.9999374784 & -0.0000271765 \\ 0.0048579479 & -0.0000271474 & 0.9999881997 \end{bmatrix}$$

$$\underline{M}_{vr} = \begin{bmatrix} 0.00000242395018 & -0.00000002710663 & -0.00000001177656 \\ 0.00000002710663 & 0.00000242397878 & -0.0000000006587 \\ 0.00000001177656 & -0.0000000006582 & 0.00000242410173 \end{bmatrix}$$

$$\underline{M}_{rv} = \begin{bmatrix} -0.000551 & -0.238565 & 0.435739 \\ 0.238514 & -0.002667 & -0.008541 \\ -0.435623 & 0.012254 & 0.002117 \end{bmatrix}$$

$$\underline{M}_{\upsilon\upsilon} = \begin{bmatrix} 0.99994704 & -0.01118251 & -0.00485767 \\ 0.01118251 & 0.99995883 & -0.00002718 \\ 0.00485767 & -0.00002714 & 1.00000956 \end{bmatrix}$$

## Dialog Box




## Inputs and Outputs

The first input is a 3-by-1 vector containing the position in Standard Julian Epoch (J2000).

The second input is a 3-by-1 vector containing the velocity in Standard Julian Epoch (J2000).

The first output is a 3-by-1 vector containing the position in Standard Besselian Epoch (B1950).

The second output is a 3-by-1 vector containing the velocity in Standard Besselian Epoch (B1950).

## Reference

"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.

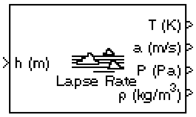
## See Also

Besselian Epoch to Julian Epoch

**Purpose** Implement lapse rate model for atmosphere

**Library** 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, described below, by specifying atmospheric properties in the block dialog.




The following equations define the troposphere

$$T = T_o - Lh$$

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

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

$$a = \sqrt{\gamma R T}$$

The following equations define the tropopause (lower stratosphere)

$$T = T_o - L \cdot hts$$

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

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

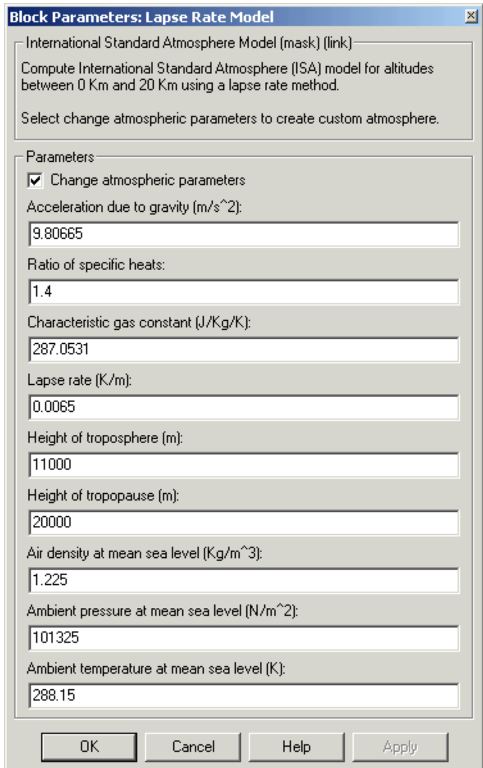
$$a = \sqrt{\gamma R T}$$

where:

| | |
|---|---|
| $T_0$ | Absolute temperature at mean sea level in kelvin (K) |
| $\rho_0$ | Air density at mean sea level in kg/m$^3$ |
| $P_0$ | Static pressure at mean sea level in N/m$^2$ |
| $h$ | Altitude in m |
| $hts$ | Height of the troposphere in m |
| $T$ | Absolute temperature at altitude $h$ in kelvin (K) |
| $\rho$ | Air density at altitude $h$ in kg/m$^3$ |
| $P$ | Static pressure at altitude $h$ in N/m$^2$ |
| $a$ | Speed of sound at altitude $h$ in 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 m/s$^2$ |

The Lapse Rate Model block icon displays the input and output metric units.

## Dialog Box




**Change atmospheric parameters**
: When selected, the following atmospheric parameters can be customized to be different from the ISA values.

**Acceleration due to gravity**
: Specify the acceleration due to gravity (g).

**Ratio of specific heats**
: Specify the ratio of specific heats $\gamma$.

**Characteristic gas constant**
: Specify the characteristic gas constant (R).

**Lapse rate**
Specify the lapse rate of the troposphere (L).

**Height of troposphere**
Specify the upper altitude of the troposphere, a range of decreasing temperature.

**Height of tropopause**
Specify the upper altitude of the tropopause, a range of constant temperature.

**Air density at mean sea level**
Specify the air density at sea level ($\rho_0$).

**Ambient pressure at mean sea level**
Specify the ambient pressure at sea level ($P_0$).

**Ambient temperature at mean sea level**
Specify the ambient temperature at sea level ($T_0$).

## Inputs and Outputs

The input is geopotential height.

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

## Assumptions and Limitations

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

## Reference

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

## See Also

COESA Atmosphere Model

ISA Atmosphere Model

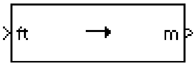
## Purpose

Convert from length units to desired length units

## Library

Utilities/Unit Conversions

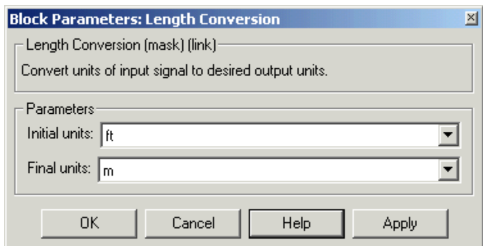
## Description




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 icon displays the input and output units selected from the **Initial units** and the **Final units** lists.

## Dialog Box




**Initial units**
Specifies the input units.

**Final units**
Specifies the output units.

The following conversion units are available:

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

# Length Conversion

## Inputs and Outputs

The input is length in initial length units.

The output is length in final length units.

## See Also

Acceleration Conversion

Angle Conversion

Angular Acceleration Conversion

Angular Velocity Conversion

Density Conversion

Force Conversion

Mass Conversion

Pressure Conversion

Temperature Conversion

Velocity Conversion

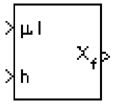
## Purpose

Calculate Earth-centered Earth-fixed (ECEF) position from geodetic latitude, longitude, and altitude above planetary ellipsoid

## Library

Utilities/Axes Transformations

## Description




The LLA to ECEF Position block converts geodetic latitude $(\underline{\mu})$, longitude $(\underline{\iota})$, and altitude $(\underline{h})$ above the planetary ellipsoid into a 3-by-1 vector of ECEF position $(\underline{p})$. The ECEF position is calculated from geocentric latitude at mean sea-level $(\lambda_s)$ and longitude using:

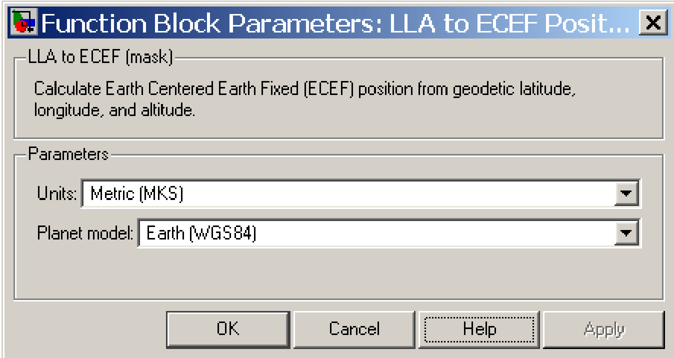
$$\underline{p} = \begin{bmatrix} \underline{p}_x \\ \underline{p}_y \\ \underline{p}_z \end{bmatrix} = \begin{bmatrix} r_s \cos\lambda_s \cos\iota + h\cos\mu\cos\iota \\ r_s \cos\lambda_s \sin\iota + h\cos\mu\sin\iota \\ r_s \sin\lambda_s + h\sin\mu \end{bmatrix}$$

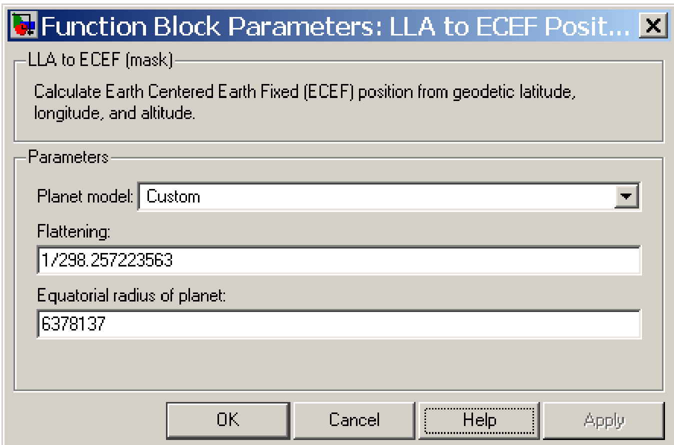
where geocentric latitude at mean sea-level and the radius at a surface point $(r_s)$ are defined by flattening $(\underline{f})$, and equatorial radius $(\underline{R})$ in the following relationships.

$$\lambda_s = \operatorname{atan}((1-f)^2 \tan\mu)$$

$$r_s = \sqrt{\frac{R^2}{1 + [1/(1-f)^2 - 1]\sin^2\lambda_s}}$$

## Dialog Box




**Units**

Specifies the parameter and output units:

| Units | Altitude | Equatorial Radius | Position |
|---|---|---|---|
| `Metric (MKS)` | Meters | Meters | Meters |
| `English` | Feet | Feet | Feet |

This option is only available when **Planet model** is set to `Earth (WGS84)`.

**Planet model**
Specifies the planet model to use: `Custom` or `Earth (WGS84)`.

**Flattening**
Specifies the flattening of the planet. This option is only available with **Planet model** 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 altitude. This option is only available with **Planet model** set to `Custom`.

## Inputs and Outputs

The first input is a 2-by-1 vector containing geodetic latitude and longitude, in degrees.

The second input is a scalar value of altitude above the planetary ellipsoid.

The output is a 3-by-1 vector containing the position in ECEF frame, in same units as altitude.

## Assumptions and Limitations

The planet is assumed to be ellipsoidal by setting flattening to 0.0 a spherical planet can be achieved.

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 being the mean spin axis of the planet, 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*, John Wiley & Sons, New York, 1992.

Zipfel, P. H., *Modeling and Simulation of Aerospace Vehicle Dynamics*, AIAA Education Series, Reston, Virginia, 2000.

"Atmospheric and Space Flight Vehicle Coordinate Systems," ANSI/AIAA R-004-1992.

## See Also

See "About Aerospace Coordinate Systems" on page 2-21.

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

## Purpose

Compute Mach number using velocity and speed of sound

## Library

Flight Parameters

## Description

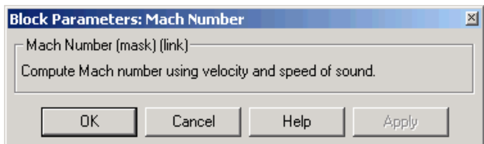

The Mach Number block computes Mach number.

Mach number is defined as

$$Mach = \frac{\sqrt{V \cdot V}}{a}$$

where $a$ is speed of sound and $V$ is velocity vector.

## Dialog Box




## Inputs and Outputs

The first input is the velocity vector.

The second input is the speed of sound.

The output of the block is the Mach number.

## Examples

See Airframe in the `aeroblk_HL20` model for an example of this block.

## See Also

Aerodynamic Forces and Moments

Dynamic Pressure

# Mass Conversion

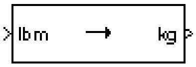
## Purpose

Convert from mass units to desired mass units

## Library

Utilities/Unit Conversions

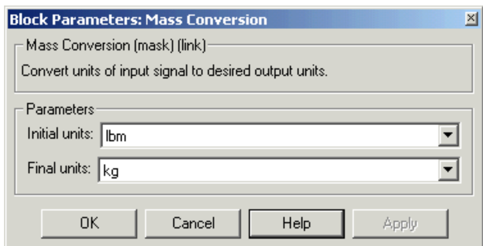
## Description




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 block icon displays the input and output units selected from the **Initial units** and the **Final units** lists.

## Dialog Box




**Initial units**
Specifies the input units.

**Final units**
Specifies the output units.

The following conversion units are available:

| | |
|---|---|
| `lbm` | Pound mass |
| `kg` | Kilograms |
| `slug` | Slugs |

## Inputs and Outputs

The input is the mass in initial mass units.

The output is the mass in final mass units.

## See Also

Acceleration Conversion

Angle Conversion

Angular Acceleration Conversion

Angular Velocity Conversion

Density Conversion

Force Conversion

Length Conversion

Pressure Conversion

Temperature Conversion

Velocity Conversion

# MATLAB Animation

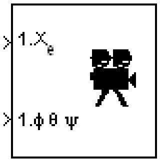
## Purpose

Create six–degrees-of-freedom multibody custom geometry block

## Library

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.

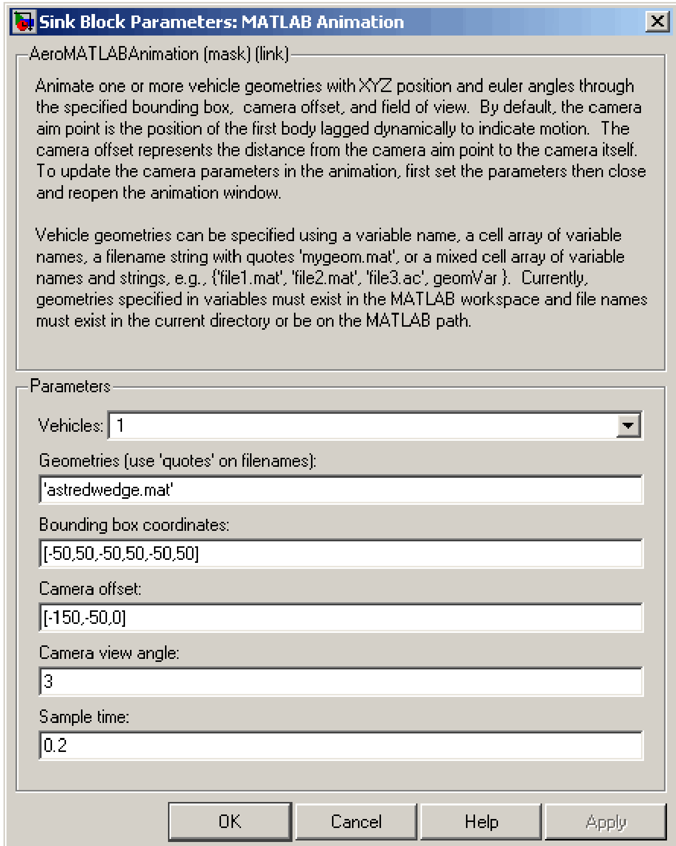
To access the dialog box for this block, right-click the block, then select **Mask Parameters**. Alternatively, double-click the block to display the MATLAB Animation window, then click the **Block Parameters** icon.

---

**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.

---

## Dialog Box




**Vehicles**

Specifies the vehicle to animate. From the list, select from 1 to 10. The block mask inputs change to reflect the number of vehicles you select. Each vehicle has its own set of inputs, denoted by the number at the beginning of the input label.

**Geometries**

Specifies the vehicle geometries. You can specify these geometries using one of the following:

- Variable name, for example `geomVar`
- Cell array of variable names, for example `{geomVar, AltGeomVar}`
- String with single quotes, for example, `'astredwedge.mat'`
- Mixed cell array of variable names and strings, 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 directory or be on the MATLAB path.

---

**Bounding box coordinates**
Specifies the boundary coordinates for the vehicle.

This parameter is not tunable during simulation. A change to this parameter takes effect after simulation stops.

**Camera offset**
Specifies the distance from the camera aim point to the camera itself.

This parameter is not tunable during simulation. A change to this parameter takes effect after simulation stops.

**Camera view angle**
Specifies the camera view angle. 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.

**Sample time**
Specify the sample time (-1 for inherited).

## Inputs and Outputs

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

The second input is a vector containing the Euler angles (roll, pitch, and yaw) of the vehicle.

## See Also

`Aero.Animation` in the Aerospace Toolbox documentation

# Moments About CG Due to Forces

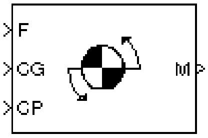
## Purpose

Compute moments about center of gravity due to forces applied at a point, not center of gravity

## Library

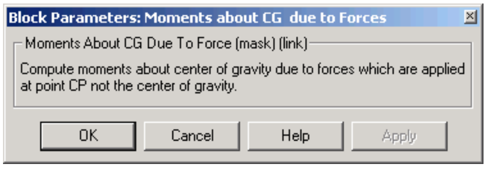
Mass Properties

## 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.

## Dialog Box




## Inputs and Outputs

The first input is the forces applied at point CP.

The second input is the center of gravity.

The third input is the application point of forces.

The output of the block is moments at the center of gravity in $x$-axes, $y$-axes and $z$-axes.

## See Also

Aerodynamic Forces and Moments

Estimate Center of Gravity

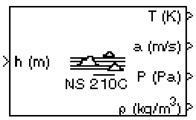
## Purpose

Implement MIL-STD-210C climatic data

## Library

Environment/Atmosphere

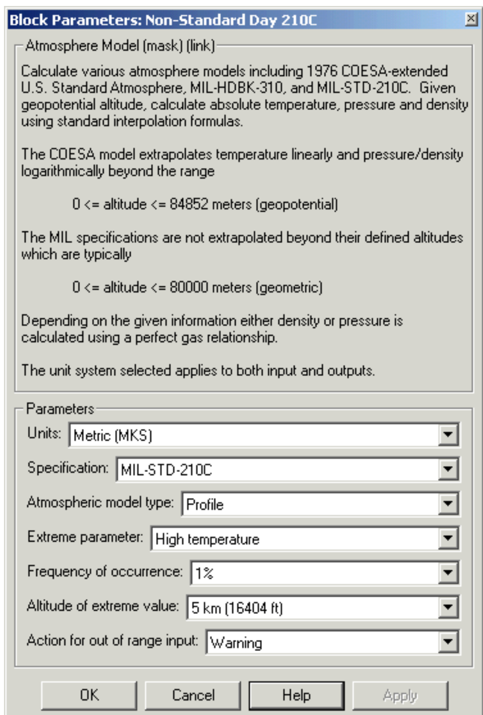
## Description




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 262,000 feet geometric) for absolute temperature, pressure, density, and speed of sound for the input geopotential altitude.

The Non-Standard Day 210C block icon displays the input and output units selected from the **Units** list.

## Dialog Box



**Units**

Specifies the input and output units:

| 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 |

**Specification**

Specify the atmosphere model type from one of the following atmosphere models. The default is `MIL-STD-210C`.

`1976 COESA-extended U.S. Standard Atmosphere`

This selection is linked to the COESA Atmosphere Model block. See the block reference for more information.

`MIL-HDBK-310`

This selection is linked to the Non-Standard Day 310 block. See the block reference for more information.

`MIL-STD-210C`

This selection is linked to the Non-Standard Day 210C block. See the block reference for more information.

**Atmospheric model type**

Select the representation of the atmospheric data.

| | |
|---|---|
| `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. |

**Extreme parameter**

Select the atmospheric parameter that is the extreme value.

| | |
|---|---|
| `High temperature` | Option always available |
| `Low temperature` | Option always available |
| `High density` | Option always available |
| `Low density` | Option always available |
| `High pressure` | This option is available only when `Envelope` is selected for **Atmospheric model type** |
| `Low pressure` | This option is available only when `Envelope` is selected for **Atmospheric model type** |

**Frequency of occurrence**

Select percent of time the values would occur.

| | |
|---|---|
| `Extreme values` | This option is available only when `Envelope` is selected for **Atmospheric model type**. |
| `1%` | Option always available |
| `5%` | This option is available only when `Envelope` is selected for **Atmospheric model type**. |

| | |
|---|---|
| `10%` | Option always available |
| `20%` | This option is available only when `Envelope` is selected for **Atmospheric model type**. |

**Altitude of extreme value**

Select geometric altitude at which the extreme values occur. Applies to the profile atmospheric model only.

`5 km (16404 ft)`

`10 km (32808 ft)`

`20 km (65617 ft)`

`30 km (98425 ft)`

`40 km (131234 ft)`

**Action for out of range input**

Specify if out-of-range input invokes a warning, error, or no action.

## Inputs and Outputs

The input is geopotential height.

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

## Assumptions and Limitations

All values are held below the geometric altitude of 0 m (0 feet) and above the geometric altitude of 80,000 meters (approximately 262,000 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.

## Reference

Global Climatic Data for Developing Military Products (MIL-STD-210C), 9 January 1987, Department of Defense, Washington, D.C.

## See Also

COESA Atmosphere Model

ISA Atmosphere Model

Non-Standard Day 310

# Non-Standard Day 310

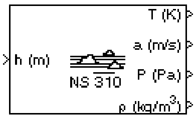
## Purpose

Implement MIL-HDBK-310 climatic data

## Library

Environment/Atmosphere

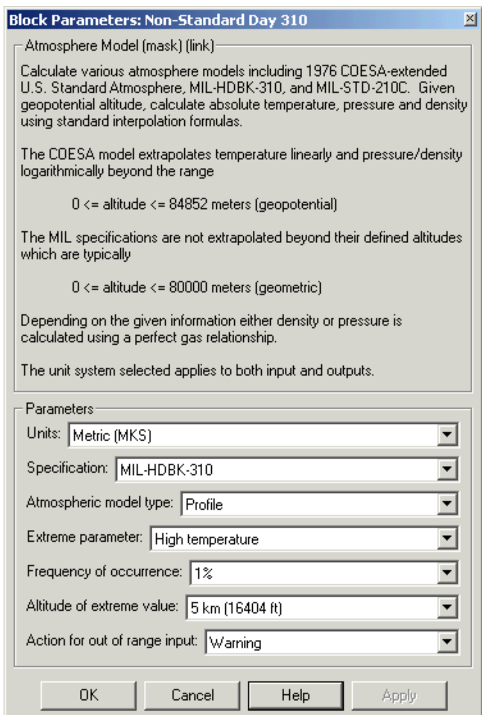
## Description




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 262,000 feet geometric) for absolute temperature, pressure, density, and speed of sound for the input geopotential altitude.

The Non-Standard Day 310 block icon displays the input and output units selected from the **Units** list.

## Dialog Box



**Units**

Specifies the input and output units:

| 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 |

**Specification**

Specify the atmosphere model type from one of the following atmosphere models. The default is `MIL-HDBK-310`.

`1976 COESA-extended U.S. Standard Atmosphere`

This selection is linked to the COESA Atmosphere Model block. See the block reference for more information.

`MIL-HDBK-310`

This selection is linked to the Non-Standard Day 310 block. See the block reference for more information.

`MIL-STD-210C`

This selection is linked to the Non-Standard Day 210C block. See the block reference for more information.

**Atmospheric model type**

Select the representation of the atmospheric data.

| | |
|---|---|
| `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. |

**Extreme parameter**

Select the atmospheric parameter which is the extreme value.

| | |
|---|---|
| `High temperature` | Option always available |
| `Low temperature` | Option always available |
| `High density` | Option always available |
| `Low density` | Option always available |
| `High pressure` | This option is available only when `Envelope` is selected for **Atmospheric model type**. |
| `Low pressure` | This option is available only when `Envelope` is selected for **Atmospheric model type**. |

**Frequency of occurrence**

Select percent of time the values would occur.

| | |
|---|---|
| `Extreme values` | This option is available only when `Envelope` is selected for **Atmospheric model type**. |
| `1%` | Option always available |
| `5%` | This option is available only when `Envelope` is selected for **Atmospheric model type**. |

| | |
|---|---|
| `10%` | Option always available |
| `20%` | This option is available only when `Envelope` is selected for **Atmospheric model type**. |

**Altitude of extreme value**

Select geometric altitude at which the extreme values occur. Applies to the profile atmospheric model only.

`5 km (16404 ft)`

`10 km (32808 ft)`

`20 km (65617 ft)`

`30 km (98425 ft)`

`40 km (131234 ft)`

**Action for out of range input**

Specify if out-of-range input invokes a warning, error, or no action.

## Inputs and Outputs

The input is geopotential height.

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

## Assumptions and Limitations

All values are held below the geometric altitude of 0 m (0 feet) and above the geometric altitude of 80,000 meters (approximately 262,000 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.

## Reference

Global Climatic Data for Developing Military Products (MIL-HDBK-310), 23 June 1997, Department of Defense, Washington, D.C.

**See Also**

COESA Atmosphere Model

ISA Atmosphere Model

Non-Standard Day 210C

# Pack net_fdm Packet for FlightGear

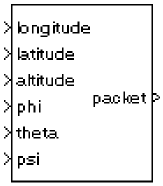
## Purpose

Generate `net_fdm` packet for FlightGear

## Library

Animation/Flight Simulator Interfaces

## Description

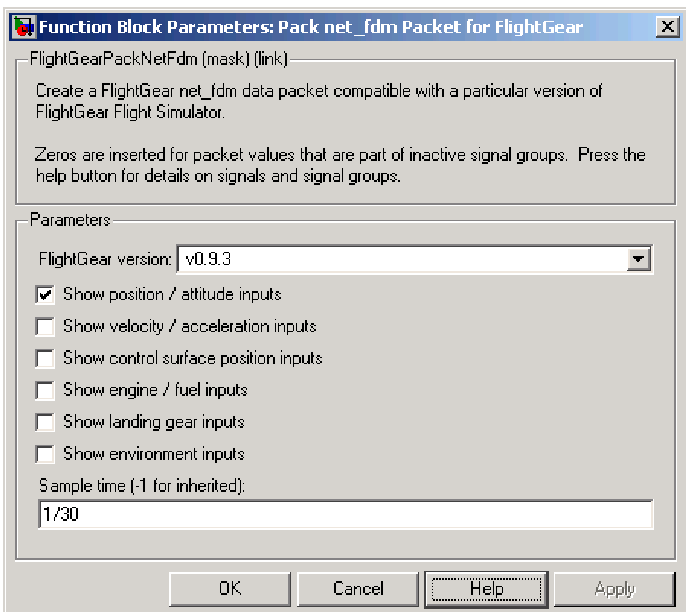



The Pack net_fdm Packet for FlightGear block creates, from separate inputs, a FlightGear `net_fdm` data packet compatible with a particular version of FlightGear flight simulator. All the signals supported by the FlightGear `net_fdm` data packet are supported by this block. The signals are arranged into six groups. Any group can be turned on or off. Zeros are inserted for packet values that are part of inactive signal groups.

See "Inputs and Outputs" on page 5-323 for details on signals and signal groups.

Supported FlightGear versions: v0.9.3, v0.9.8/0.9.8a, v0.9.9, v0.9.10

## Dialog Box



**FlightGear version**
Select your FlightGear software version.

Supported FlightGear versions: `v0.9.3`, `v0.9.8/0.9.8a`, `v0.9.9`, `v0.9.10`.

**Show position/altitude inputs**
Select this check box to include the position and altitude inputs (signal group 1) into the FlightGear `net_fdm` data packet.

**Show velocity/acceleration inputs**
Select this check box to include the velocity and acceleration inputs (signal group 2) into the FlightGear `net_fdm` data packet.

**Show control surface position inputs**
Select this check box to include the control surface position inputs (signal group 3) into the FlightGear `net_fdm` data packet.

**Show engine/fuel inputs**
Select this check box to include the engine and fuel inputs (signal group 4) into the FlightGear `net_fdm` data packet.

**Show landing gear inputs**
Select this check box to include the landing gear inputs (signal group 5) into the FlightGear `net_fdm` data packet.

**Show environment inputs**
Select this check box to include the environment inputs (signal group 6) into the FlightGear `net_fdm` data packet.

**Sample time**
Specify the sample time (-1 for inherited).

## Inputs and Outputs

### Input Signals Supported for FlightGear 0.9.3

This table lists all the input signals supported for Version 0.9.3:

# Pack net_fdm Packet for FlightGear

| Name | Units | Type | Width | Description |
|---|---|---|---|---|
| **Signal Group 1: ShowPositionAttitudeInputs** | | | | |
| longitude | rad | double | 1 | Geodetic longitude |
| latitude | rad | double | 1 | Geodetic altitude |
| altitude | m | double | 1 | Altitude above sea level |
| phi | rad | single | 1 | Roll |
| theta | rad | single | 1 | Pitch |
| psi | rad | single | 1 | Yaw or true heading |
| **Signal Group 2: ShowVelocityAccelerationInputs** | | | | |
| phidot | rad/sec | single | 1 | Roll rate |
| thetadot | rad/sec | single | 1 | Pitch rate |
| psidot | rad/sec | single | 1 | Yaw rate |
| vcas | kts | single | 1 | Calibrated airspeed |
| climb_rate | ft/sec | single | 1 | Climb rate |
| v_north | ft/sec | single | 1 | North velocity in local/body frame |
| v_east | ft/sec | single | 1 | East velocity in local/body frame |
| v_down | ft/sec | single | 1 | Down/vertical velocity in local/body frame |
| v_wind_body_north | ft/sec | single | 1 | Body north velocity relative to local airmass |

| Name | Units | Type | Width | Description |
|---|---|---|---|---|
| v_wind_body_east | ft/sec | single | 1 | Body east velocity relative to local airmass |
| v_wind_body_down | ft/sec | single | 1 | Body down/vertical velocity relative to local airmass |
| stall_warning | — | single | 1 | 0.0–1.0, indicating the amount of stall |
| A_X_pilot | ft/sec$^2$ | single | 1 | X acceleration in body frame |
| A_Y_pilot | ft/sec$^2$ | single | 1 | Y acceleration in body frame |
| A_Z_pilot | ft/sec$^2$ | single | 1 | Z acceleration in body frame |
| **Signal Group 3: ShowControlSurfacePositionInputs** | | | | |
| elevator | geometry-specific units | single | 1 | Elevator position |
| flaps | geometry-specific units | single | 1 | Flaps position |
| left_aileron | geometry-specific units | single | 1 | Left aileron position |
| right_aileron | geometry-specific units | single | 1 | Right aileron position |
| rudder | geometry-specific units | single | 1 | Rudder position |
| speedbrake | geometry-specific units | single | 1 | Speed brake position |

| Name | Units | Type | Width | Description |
|---|---|---|---|---|
| spoilers | geometry-specific units | single | 1 | Spoilers position |
| **Signal Group 4: ShowEngineFuelInputs** | | | | |
| num_engines | — | int32 | 1 | Number of valid engines |
| eng_state | enum | int32 | 4 | Engine state (0=off, 1=cranking, 2=running) |
| rpm | rev/min | single | 4 | Engine RPM |
| fuel_flow | gal/hr | single | 4 | Fuel flow |
| EGT | ºF | single | 4 | Exhaust gas temp |
| oil_temp | ºF | single | 4 | Oil temp |
| oil_px | lbf/in$^2$ | single | 4 | Oil pressure |
| num_tanks | — | int32 | 1 | Max number of fuel tanks |
| fuel_quantity | — | single | 4 | Amount of fuel in tanks (0–1 fraction) |
| **Signal Group 5: ShowLandingGearInputs** | | | | |
| num_wheels | — | int32 | 1 | Maximum number of wheels |
| wow | — | boolean | 3 | Weight on wheels signal (1=wheel is on ground) |
| gear_pos | — | single | 3 | Landing gear position (0-1, indicating amount deployed) |

| Name | Units | Type | Width | Description |
|---|---|---|---|---|
| gear_steer | — | single | 3 | Landing gear steering angle |
| gear_compression | — | single | 3 | Landing gear compression |
| **Signal Group 6: ShowEnvironmentInputs** | | | | |
| agl | m | single | 1 | Above ground level |
| cur_time | sec | int32 | 1 | Current UNIX time |
| warp | sec | int32 | 1 | Offset in seconds to UNIX time |
| visibility | m | single | 1 | Visibility in meters (for visual effects) |

### Input Signals Supported for FlightGear 0.9.8/0.9.8a

This table lists all the input signals supported for Versions 0.9.8/0.9.8a:

| Name | Units | Type | Width | Description |
|---|---|---|---|---|
| **Signal Group 1: ShowPositionAttitudeInputs** | | | | |
| longitude | rad | double | 1 | Geodetic longitude |
| latitude | rad | double | 1 | Geodetic altitude |
| altitude | m | double | 1 | Altitude above sea level |
| phi | rad | single | 1 | Roll |
| theta | rad | single | 1 | Pitch |
| psi | rad | single | 1 | Yaw or true heading |
| **Signal Group 2: ShowVelocityAccelerationInputs** | | | | |

# Pack net_fdm Packet for FlightGear

| Name | Units | Type | Width | Description |
|---|---|---|---|---|
| alpha | rad | single | 1 | Angle of attack |
| beta | rad | single | 1 | sideslip angle |
| phidot | rad/sec | single | 1 | Roll rate |
| thetadot | rad/sec | single | 1 | Pitch rate |
| psidot | rad/sec | single | 1 | Yaw rate |
| vcas | kts | single | 1 | Calibrated airspeed |
| climb_rate | ft/sec | single | 1 | Climb rate |
| v_north | ft/sec | single | 1 | North velocity in local/body frame |
| v_east | ft/sec | single | 1 | East velocity in local/body frame |
| v_down | ft/sec | single | 1 | Down/vertical velocity in local/body frame |
| v_wind_body_north | ft/sec | single | 1 | Body north velocity relative to local airmass |
| v_wind_body_east | ft/sec | single | 1 | Body east velocity relative to local airmass |
| v_wind_body_down | ft/sec | single | 1 | Body down/vertical velocity relative to local airmass |
| A_X_pilot | $ft/sec^2$ | single | 1 | X acceleration in body frame |
| A_Y_pilot | $ft/sec^2$ | single | 1 | Y acceleration in body frame |

| Name | Units | Type | Width | Description |
|---|---|---|---|---|
| A_Z_pilot | ft/sec$^2$ | single | 1 | Z acceleration in body frame |
| stall_warning | — | single | 1 | 0.0–1.0, indicating the amount of stall |
| slip_deg | deg | single | 1 | Slip ball deflection |
| **Signal Group 3: ShowControlSurfacePositionInputs** | | | | |
| elevator | geometry-specific units | single | 1 | Elevator position |
| elevator_trim_tab | geometry-specific units | single | 1 | Elevator trim position |
| left_flap | geometry-specific units | single | 1 | Left flap position |
| right_flap | geometry-specific units | single | 1 | Right flap position |
| left_aileron | geometry-specific units | single | 1 | Left aileron position |
| right_aileron | geometry-specific units | single | 1 | Right aileron position |
| rudder | geometry-specific units | single | 1 | Rudder position |
| nose_wheel | geometry-specific units | single | 1 | Nose wheel position |

| Name | Units | Type | Width | Description |
|---|---|---|---|---|
| speedbrake | geometry-specific units | single | 1 | Speed brake position |
| spoilers | geometry-specific units | single | 1 | Spoilers position |
| **Signal Group 4: ShowEngineFuelInputs** | | | | |
| num_engines | — | int32 | 1 | Number of valid engines |
| eng_state | enum | int32 | 4 | Engine state (0=off, 1=cranking, 2=running) |
| rpm | rev/min | single | 4 | Engine RPM |
| fuel_flow | gal/hr | single | 4 | Fuel flow |
| EGT | ºF | single | 4 | Exhaust gas temp |
| cht | ºF | single | 4 | Cylinder head temperature |
| mp_osi | psi | single | 4 | Manifold pressure |
| tit | ºF | single | 4 | Turbine inlet temperature |
| oil_temp | ºF | single | 4 | Oil temp |
| oil_px | lbf/in$^2$ | single | 4 | Oil pressure |
| num_tanks | — | int32 | 1 | Max number of fuel tanks |
| fuel_quantity | — | single | 4 | Amount of fuel in tanks (0–1 fraction) |
| **Signal Group 5: ShowLandingGearInputs** | | | | |

| Name | Units | Type | Width | Description |
|---|---|---|---|---|
| num_wheels | — | int32 | 1 | Maximum number of wheels |
| wow | — | boolean | 3 | Weight on wheels signal (1=wheel is on ground) |
| gear_pos | — | single | 3 | Landing gear position (0–1, indicating amount deployed) |
| gear_steer | — | single | 3 | Landing gear steering angle |
| gear_compression | — | single | 3 | Landing gear compression |
| **Signal Group 6: ShowEnvironmentInputs** | | | | |
| agl | m | single | 1 | Above ground level |
| cur_time | sec | int32 | 1 | Current UNIX time |
| warp | sec | int32 | 1 | Offset in seconds to UNIX time |
| visibility | m | single | 1 | Visibility in meters (for visual effects) |

### Input Signals Supported for FlightGear 0.9.9

This table lists all the input signals supported for Version 0.9.9:

| Name | Units | Type | Width | Description |
|---|---|---|---|---|
| **Signal Group 1: ShowPositionAttitudeInputs** | | | | |
| longitude | rad | double | 1 | Geodetic longitude |
| latitude | rad | double | 1 | Geodetic latitude |

| Name | Units | Type | Width | Description |
|---|---|---|---|---|
| altitude | m | double | 1 | Altitude above sea level |
| phi | rad | single | 1 | Roll |
| theta | rad | single | 1 | Pitch |
| psi | rad | single | 1 | Yaw or true heading |
| **Signal Group 2: ShowVelocityAccelerationInputs** | | | | |
| alpha | rad | single | 1 | Angle of attack |
| beta | rad | single | 1 | sideslip angle |
| phidot | rad/sec | single | 1 | Roll rate |
| thetadot | rad/sec | single | 1 | Pitch rate |
| psidot | rad/sec | single | 1 | Yaw rate |
| vcas | kts | single | 1 | Calibrated airspeed |
| climb_rate | ft/sec | single | 1 | Climb rate |
| v_north | ft/sec | single | 1 | North velocity in local/body frame |
| v_east | ft/sec | single | 1 | East velocity in local/body frame |
| v_down | ft/sec | single | 1 | Down/vertical velocity in local/body frame |
| v_wind_body_north | ft/sec | single | 1 | Body north velocity relative to local airmass |
| v_wind_body_east | ft/sec | single | 1 | Body east velocity relative to local airmass |

| Name | Units | Type | Width | Description |
|---|---|---|---|---|
| v_wind_body_down | ft/sec | single | 1 | Body down/vertical velocity relative to local airmass |
| A_X_pilot | ft/sec$^2$ | single | 1 | X acceleration in body frame |
| A_Y_pilot | ft/sec$^2$ | single | 1 | Y acceleration in body frame |
| A_Z_pilot | ft/sec$^2$ | single | 1 | Z acceleration in body frame |
| stall_warning | — | single | 1 | 0.0–1.0, indicating the amount of stall |
| slip_deg | deg | single | 1 | Slip ball deflection |
| **Signal Group 3: ShowControlSurfacePositionInputs** | | | | |
| elevator | geometry-specific units | single | 1 | Elevator position |
| elevator_trim_tab | geometry-specific units | single | 1 | Elevator trim position |
| left_flap | geometry-specific units | single | 1 | Left flap position |
| right_flap | geometry-specific units | single | 1 | Right flap position |
| left_aileron | geometry-specific units | single | 1 | Left aileron position |

| Name | Units | Type | Width | Description |
|---|---|---|---|---|
| right_aileron | geometry-specific units | single | 1 | Right aileron position |
| rudder | geometry-specific units | single | 1 | Rudder position |
| nose_wheel | geometry-specific units | single | 1 | Nose wheel position |
| speedbrake | geometry-specific units | single | 1 | Speed brake position |
| spoilers | geometry-specific units | single | 1 | Spoilers position |
| **Signal Group 4: ShowEngineFuelInputs** | | | | |
| num_engines | — | uint32 | 1 | Number of valid engines |
| eng_state | enum | uint32 | 4 | Engine state (0=off, 1=cranking, 2=running) |
| rpm | rev/min | single | 4 | Engine RPM |
| fuel_flow | gal/hr | single | 4 | Fuel flow |
| EGT | °F | single | 4 | Exhaust gas temp |
| cht | °F | single | 4 | Cylinder head temperature |
| mp_osi | psi | single | 4 | Manifold pressure |
| tit | °F | single | 4 | Turbine inlet temperature |

| Name | Units | Type | Width | Description |
|---|---|---|---|---|
| oil_temp | °F | single | 4 | Oil temp |
| oil_px | lbf/in$^2$ | single | 4 | Oil pressure |
| num_tanks | — | uint32 | 1 | Max number of fuel tanks |
| fuel_quantity | — | single | 4 | Amount of fuel in tanks (0–1 fraction) |
| **Signal Group 5: ShowLandingGearInputs** | | | | |
| num_wheels | — | uint32 | 1 | Maximum number of wheels |
| wow | — | uint32 | 3 | Weight on wheels signal (1=wheel is on ground) |
| gear_pos | — | single | 3 | Landing gear position (0-1, indicating amount deployed) |
| gear_steer | — | single | 3 | Landing gear steering angle |
| gear_compression | — | single | 3 | Landing gear compression |
| **Signal Group 6: ShowEnvironmentInputs** | | | | |
| agl | m | single | 1 | Above ground level |
| cur_time | sec | uint32 | 1 | Current UNIX time |
| warp | sec | int32 | 1 | Offset in seconds to UNIX time |
| visibility | m | single | 1 | Visibility in meters (for visual effects) |

### Input Signals Supported for FlightGear 0.9.10

This table lists all the input signals supported for Version 0.9.10:

| Name | Units | Type | Width | Description |
|---|---|---|---|---|
| **Signal Group 1: ShowPositionAttitudeInputs** | | | | |
| longitude | rad | double | 1 | Geodetic longitude |
| latitude | rad | double | 1 | Geodetic latitude |
| altitude | m | double | 1 | Altitude above sea level |
| phi | rad | single | 1 | Roll |
| theta | rad | single | 1 | Pitch |
| psi | rad | single | 1 | Yaw or true heading |
| **Signal Group 2: ShowVelocityAccelerationInputs** | | | | |
| alpha | rad | single | 1 | Angle of attack |
| beta | rad | single | 1 | sideslip angle |
| phidot | rad/sec | single | 1 | Roll rate |
| thetadot | rad/sec | single | 1 | Pitch rate |
| psidot | rad/sec | single | 1 | Yaw rate |
| vcas | kts | single | 1 | Calibrated airspeed |
| climb_rate | ft/sec | single | 1 | Climb rate |
| v_north | ft/sec | single | 1 | North velocity in local/body frame |
| v_east | ft/sec | single | 1 | East velocity in local/body frame |
| v_down | ft/sec | single | 1 | Down/vertical velocity in local/body frame |

| Name | Units | Type | Width | Description |
|---|---|---|---|---|
| v_wind_body_north | ft/sec | single | 1 | Body north velocity relative to local airmass |
| v_wind_body_east | ft/sec | single | 1 | Body east velocity relative to local airmass |
| v_wind_body_down | ft/sec | single | 1 | Body down/vertical velocity relative to local airmass |
| A_X_pilot | $ft/sec^2$ | single | 1 | X acceleration in body frame |
| A_Y_pilot | $ft/sec^2$ | single | 1 | Y acceleration in body frame |
| A_Z_pilot | $ft/sec^2$ | single | 1 | Z acceleration in body frame |
| stall_warning | — | single | 1 | 0.0–1.0, indicating the amount of stall |
| slip_deg | deg | single | 1 | Slip ball deflection |
| **Signal Group 3: ShowControlSurfacePositionInputs** | | | | |
| elevator | geometry-specific units | single | 1 | Elevator position |
| elevator_trim_tab | geometry-specific units | single | 1 | Elevator trim position |
| left_flap | geometry-specific units | single | 1 | Left flap position |

| Name | Units | Type | Width | Description |
|---|---|---|---|---|
| right_flap | geometry-specific units | single | 1 | Right flap position |
| left_aileron | geometry-specific units | single | 1 | Left aileron position |
| right_aileron | geometry-specific units | single | 1 | Right aileron position |
| rudder | geometry-specific units | single | 1 | Rudder position |
| nose_wheel | geometry-specific units | single | 1 | Nose wheel position |
| speedbrake | geometry-specific units | single | 1 | Speed brake position |
| spoilers | geometry-specific units | single | 1 | Spoilers position |
| **Signal Group 4: ShowEngineFuelInputs** | | | | |
| num_engines | — | uint32 | 1 | Number of valid engines |
| eng_state | enum | uint32 | 4 | Engine state (0=off, 1=cranking, 2=running) |
| rpm | rev/min | single | 4 | Engine RPM |
| fuel_flow | gal/hr | single | 4 | Fuel flow |
| fuel_px | psi | single | 4 | Fuel pressure |

| Name | Units | Type | Width | Description |
|---|---|---|---|---|
| EGT | °F | single | 4 | Exhaust gas temp |
| cht | °F | single | 4 | Cylinder head temperature |
| mp_osi | psi | single | 4 | Manifold pressure |
| tit | °F | single | 4 | Turbine inlet temperature |
| oil_temp | °F | single | 4 | Oil temp |
| oil_px | $lbf/in^2$ | single | 4 | Oil pressure |
| num_tanks | — | uint32 | 1 | Max number of fuel tanks |
| fuel_quantity | — | single | 4 | Amount of fuel in tanks (0–1 fraction) |
| **Signal Group 5: ShowLandingGearInputs** | | | | |
| num_wheels | — | uint32 | 1 | Maximum number of wheels |
| wow | — | uint32 | 3 | Weight on wheels signal (1=wheel is on ground) |
| gear_pos | — | single | 3 | Landing gear position (0–1, indicating amount deployed) |
| gear_steer | — | single | 3 | Landing gear steering angle |
| gear_compression | — | single | 3 | Landing gear compression |
| **Signal Group 6: ShowEnvironmentInputs** | | | | |
| agl | m | single | 1 | Above ground level |
| cur_time | sec | uint32 | 1 | Current UNIX time |

| Name | Units | Type | Width | Description |
|---|---|---|---|---|
| warp | sec | int32 | 1 | Offset in seconds to UNIX time |
| visibility | m | single | 1 | Visibility in meters (for visual effects) |

### Output Signal

The output signal is the FlightGear `net_fdm` data packet.

## Examples

See the `asbhl20` demo for an example of this block.

## See Also

FlightGear Preconfigured 6DoF Animation

Generate Run Script

Send net_fdm Packet to FlightGear

## Purpose

Provide joystick interface on Windows platform

## Library

Animation/Animation Support Utilities

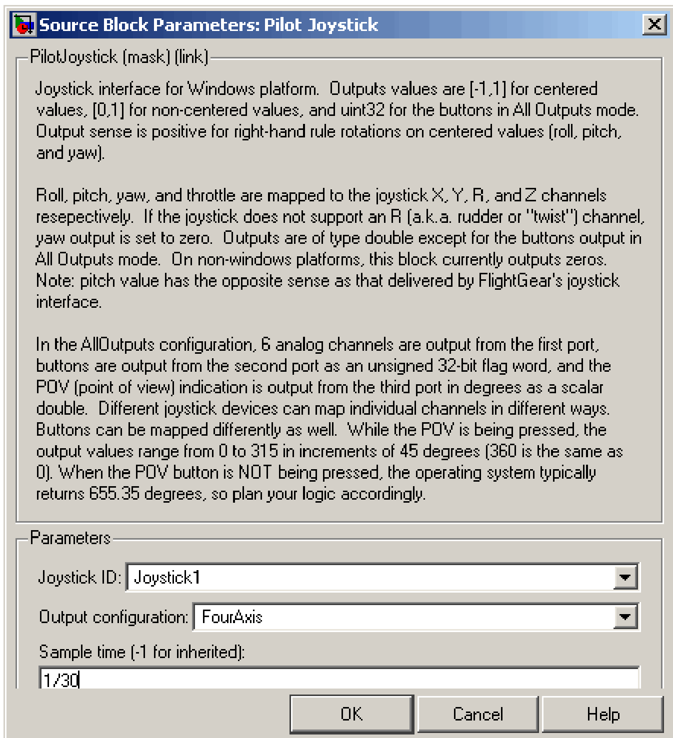
## Description


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`.

This block does not produce deployable code.

## Dialog Box



**Joystick ID**
Specify the joystick ID: `Joystick 1`, `Joystick 2`, or `None`.

**Output configuration**
Specify the output configuration: `FourAxis` or `AllOutputs` (see Pilot Joystick All). `FourAxis` is the default.

**Sample time**
Specify the sample time (-1 for inherited).

## Inputs and Outputs

The block has the following outputs.

### Four Axis Mode (All Double Precision Values)

| Port Number | Output Range | Joystick | Description |
|---|---|---|---|
| 1 | [-1, 1] | [left, right] | Roll command |
| 2 | [-1, 1] | [forward/down, back/up] | Pitch command |
| 3 | [-1, 1] | [left, right] | Yaw command |
| 4 | [ 0, 1] | [min, max] | Throttle command |

### All Outputs Mode (All Values Double Precision, Except for Buttons)

| Port Number | Array Number | Channel | Output Range | Joystick | Description |
|---|---|---|---|---|---|
| 1 | 1 | X | [-1, 1] | [left, right] | Roll command |
| 1 | 2 | Y | [-1, 1] | [forward/down, back/up] | Pitch command |
| 1 | 3 | Z | [ 0, 1] | [min, max] | Throttle command |
| 1 | 4 | R | [-1, 1] | [left, right] | Yaw command |

| Port Number | Array Number | Channel | Output Range | Joystick | Description |
|---|---|---|---|---|---|
| 1 | 5 | U | [ 0, 1] | [min, max] | U channel value |
| 1 | 6 | V | [ 0, 1] | [min, max] | V channel value |
| 2 | | buttons | | | uint32 flagword containing up to 32 button states. Bit 0 is button 1, etc. |
| 3 | | POV | | | Point-of-view hat value in degrees as a double. Zero degrees is straight ahead, 90 is to the left, etc. |

Output values are [-1,1] for centered values, [0,1] for noncentered values, and uint32 for the buttons in All Outputs mode. Output sense is positive for right-hand rule rotations on centered values (roll, pitch, and yaw).

## Assumptions and 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 for the buttons output in `AllOutputs` mode, which is a uint32 flagword of bits. On non-Windows platforms, this block currently outputs zeros.

---

**Note** Pitch value has the opposite sense as that delivered by FlightGear’s joystick interface.

---

## See Also

Pilot Joystick All, Simulation Pace

# Pilot Joystick All

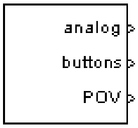
## Purpose

Provide joystick interface on Windows platform

## Library

Animation/Animation Support Utilities

## Description

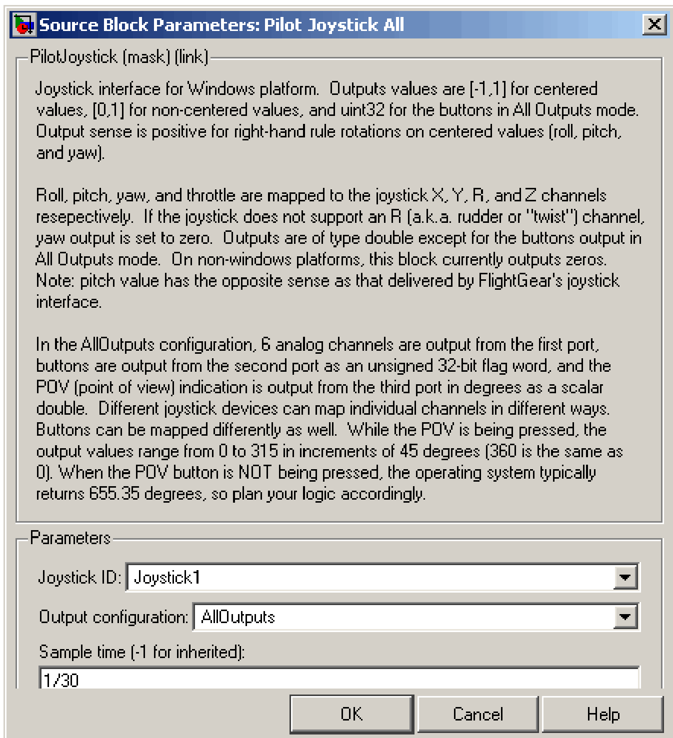



The Pilot Joystick block provides a pilot joystick interface for a Windows platform. Roll, pitch, yaw, and throttle are mapped to the joystick X, Y, R, and Z channels respectively.

You can also configure the block to output four axes by setting the **Output configuration** parameter to `FourAxis`.

This block does not produce deployable code.

## Dialog Box



**Joystick ID**
Specify the joystick ID: `Joystick 1`, `Joystick 2`, or `None`.

**Output configuration**
Specify the output configuration: `FourAxis` (see Pilot Joystick) or `AllOutputs`. `AllOutputs` is the default.

**Sample time**
Specify the sample time (-1 for inherited).

## Inputs and Outputs

The block has the following outputs.

### Four Axis Mode (All Double Precision Values)

| Port Number | Output Range | Joystick | Description |
|---|---|---|---|
| 1 | [-1, 1] | [left, right] | Roll command |
| 2 | [-1, 1] | [forward/down, back/up] | Pitch command |
| 3 | [-1, 1] | [left, right] | Yaw command |
| 4 | [ 0, 1] | [min, max] | Throttle command |

### All Outputs Mode (All Values Double Precision, Except for Buttons)

| Port Number | Array Number | Channel | Output Range | Joystick | Description |
|---|---|---|---|---|---|
| 1 | 1 | X | [-1, 1] | [left, right] | Roll command |
| 1 | 2 | Y | [-1, 1] | [forward/down, back/up] | Pitch command |
| 1 | 3 | Z | [ 0, 1] | [min, max] | Throttle command |
| 1 | 4 | R | [-1, 1] | [left, right] | Yaw command |

| Port Number | Array Number | Channel | Output Range | Joystick | Description |
|---|---|---|---|---|---|
| 1 | 5 | U | [ 0, 1] | [min, max] | U channel value |
| 1 | 6 | V | [ 0, 1] | [min, max] | V channel value |
| 2 | | buttons | | | uint32 flagword containing up to 32 button states. Bit 0 is button 1, etc. |
| 3 | | POV | | | Point-of-view hat value in degrees as a double. Zero degrees is straight ahead, 90 is to the left, etc. |

Output values are [-1,1] for centered values, [0,1] for noncentered values, and uint32 for the buttons in All Outputs mode. Output sense is positive for right-hand rule rotations on centered values (roll, pitch, and yaw).

## Assumptions and 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 for the buttons output in `AllOutputs` mode, which is a uint32 flagword of bits. On non-Windows platforms, this block currently outputs zeros.

---

**Note** Pitch value has the opposite sense as that delivered by FlightGear's joystick interface.

---

## See Also

Pilot Joystick, Simulation Pace

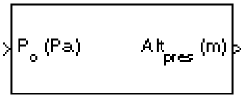
## Purpose

Calculate pressure altitude based on ambient pressure

## Library

Environment/Atmosphere

## Description

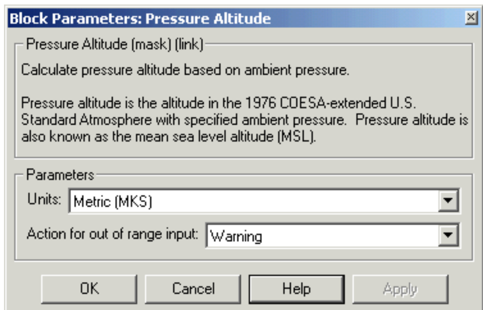



The Pressure Altitude block computes the pressure altitude based on ambient pressure. Pressure altitude is the altitude in the 1976 Committee on the Extension of the Standard Atmosphere (COESA) United States with specified ambient pressure.

Pressure altitude is also known as the mean sea level (MSL) altitude.

The Pressure Altitude block icon displays the input and output units selected from the **Units** list.

## Dialog Box




**Units**

Specifies the input units:

| Units | Pstatic | Alt_p |
|---|---|---|
| `Metric (MKS)` | Pascal | Meters |
| `English` | Pound force per square inch | Feet |

**Action for out of range input**

Specify if out-of-range input invokes a warning, error, or no action.

### Inputs and Outputs

The input is the static pressure.

The output is the pressure altitude.

### Assumptions and 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.

### Reference

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

### See Also

COESA Atmosphere Model

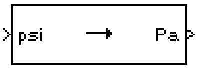
## Purpose

Convert from pressure units to desired pressure units

## Library

Utilities/Unit Conversions

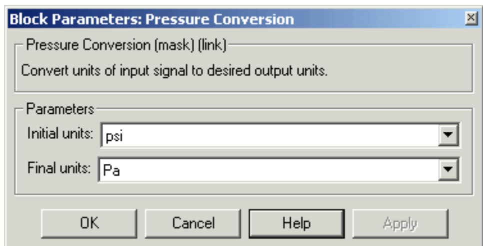
## Description




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 icon displays the input and output units selected from the **Initial units** and the **Final units** lists.

## Dialog Box




**Initial units**
Specifies the input units.

**Final units**
Specifies the output units.

The following conversion units are available:

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

## Inputs and Outputs

The input is the pressure in initial pressure units.

The output is the pressure in final pressure units.

## See Also

Acceleration Conversion

Angle Conversion

Angular Acceleration Conversion

Angular Velocity Conversion

Density Conversion

Force Conversion

Length Conversion

Mass Conversion

Temperature Conversion

Velocity Conversion

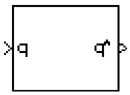
## Purpose

Calculate conjugate of quaternion

## Library

Utilities/MathOperations

## Description



The Quaternion Conjugate block calculates the conjugate for a given quaternion.

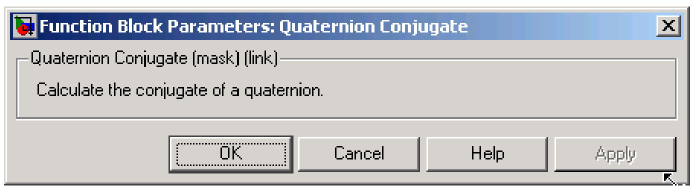
The quaternion has the form of

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

The quaternion conjugate has the form of

$$q' = q_0 - \boldsymbol{i}q_1 - \boldsymbol{j}q_2 - \boldsymbol{k}q_3$$

## Dialog Box




## Inputs and Outputs

The input is a quaternion or vector of quaternions in the form of [$q_0$, $r_0$, ..., $q_1$, $r_1$, ... , $q_2$, $r_2$, ... , $q_3$, $r_3$, ...].

The output is a quaternion conjugate or vector of quaternion conjugates in the form of [$q_0$', $r_0$, ... , $q_1$', $r_1$', ... , $q_2$', $r_2$', ... , $q_3$', $r_3$', ...].

## See Also

Quaternion Division

Quaternion Inverse

Quaternion Modulus

Quaternion Multiplication

Quaternion Norm

Quaternion Normalize

Quaternion Rotation

# Quaternion Division

**Purpose** Divide quaternion by another quaternion

**Library** Utilities/Math Operations

**Description** The Quaternion Division block divides a given quaternion by another.


The quaternions have the form of

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

and

$$r = r_0 + \boldsymbol{i}r_1 + \boldsymbol{j}r_2 + \boldsymbol{k}r_3$$

The resulting quaternion from the division has the form of

$$t = \frac{q}{r} = t_0 + \boldsymbol{i}t_1 + \boldsymbol{j}t_2 + \boldsymbol{k}t_3$$

where

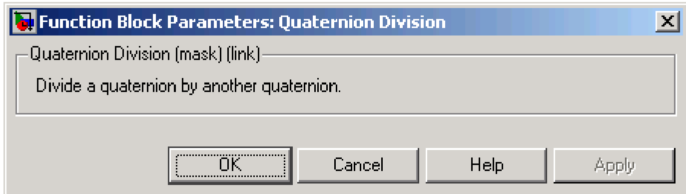
$$t_0 = \frac{(r_0q_0 + r_1q_1 + r_2q_2 + r_3q_3)}{r_0^2 + r_1^2 + r_2^2 + r_3^2}$$

$$t_1 = \frac{(r_0q_1 - r_1q_0 - r_2q_3 + r_3q_2)}{r_0^2 + r_1^2 + r_2^2 + r_3^2}$$

$$t_2 = \frac{(r_0q_2 + r_1q_3 - r_2q_0 - r_3q_1)}{r_0^2 + r_1^2 + r_2^2 + r_3^2}$$

$$t_3 = \frac{(r_0q_3 - r_1q_2 + r_2q_1 - r_3q_0)}{r_0^2 + r_1^2 + r_2^2 + r_3^2}$$

## Dialog Box




## Inputs and Outputs

The first input is a quaternion or vector of quaternions in the form of $[q_0, p_0, \ldots, q_1, p_1, \ldots , q_2, p_2, \ldots , q_3, p_3, \ldots]$.

The second input is a quaternion or vector of quaternions in the form of $[s_0, r_0, \ldots, s_1, r_1, \ldots , s_2, r_2, \ldots , s_3, r_3, \ldots]$.

The output is the resulting quaternion from the division or vector of resulting quaternions from division.

## See Also

Quaternion Conjugate

Quaternion Inverse

Quaternion Modulus

Quaternion Multiplication

Quaternion Norm

Quaternion Normalize

Quaternion Rotation

# Quaternion Inverse

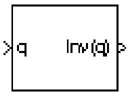
## Purpose

Calculate inverse of quaternion

## Library

Utilities/Math Operations

## Description




The Quaternion Inverse block calculates the inverse for a given quaternion.

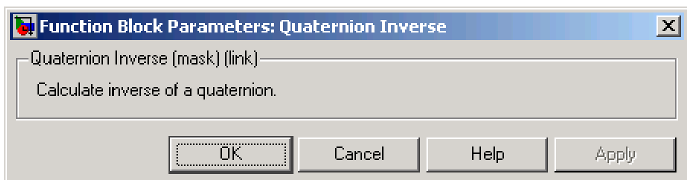
The quaternion has the form of

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

The quaternion inverse has the form of

$$q^{-1} = \frac{q_0 - \boldsymbol{i}q_1 - \boldsymbol{j}q_2 - \boldsymbol{k}q_3}{q_0^2 + q_1^2 + q_2^2 + q_3^2}$$

## Dialog Box




## Inputs and Outputs

The input is a quaternion or vector of quaternions in the form of [$q_0$, $r_0$, ..., $q_1$, $r_1$, ... , $q_2$, $r_2$, ... , $q_3$, $r_3$, ...].

The output is a quaternion inverse or vector of quaternion inverses.

## See Also

Quaternion Conjugate

Quaternion Division

Quaternion Modulus

Quaternion Multiplication

Quaternion Norm

Quaternion Normalize

Quaternion Rotation

# Quaternion Modulus

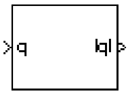
## Purpose

Calculate modulus of quaternion

## Library

Utilities/Math Operations

## Description




The Quaternion Modulus block calculates the magnitude for a given quaternion.

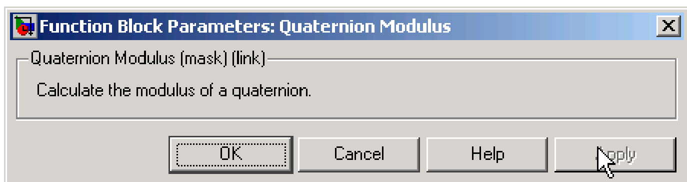
The quaternion has the form of

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

The quaternion modulus has the form of

$$|q| = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2}$$

## Dialog Box




## Inputs and Outputs

The input is a quaternion or vector of quaternions in the form of [$q_0$, $r_0$, ..., $q_1$, $r_1$, ... , $q_2$, $r_2$, ... , $q_3$, $r_3$, ...].

The output is a quaternion modulus or vector of quaternion modulus in the form of [|q|, |r|, ...].

## See Also

Quaternion Conjugate

Quaternion Division

Quaternion Inverse

Quaternion Multiplication

Quaternion Norm

Quaternion Normalize

Quaternion Rotation

# Quaternion Multiplication

## Purpose

Calculate product of two quaternions

## Library

Utilities/Math Operations

## Description


The Quaternion Multiplication block calculates the product for two given quaternions.

The quaternions have the form of

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

and

$$r = r_0 + \boldsymbol{i}r_1 + \boldsymbol{j}r_2 + \boldsymbol{k}r_3$$

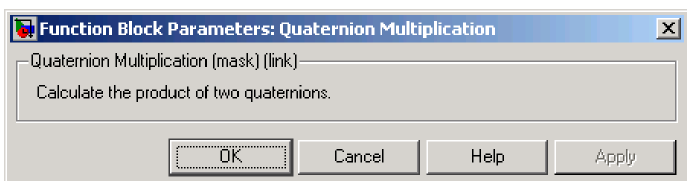
The quaternion product has the form of

$$t = q \times r = t_0 + \boldsymbol{i}t_1 + \boldsymbol{j}t_2 + \boldsymbol{k}t_3$$

where

$$t_0 = (r_0q_0 - r_1q_1 - r_2q_2 - r_3q_3)$$
$$t_1 = (r_0q_1 + r_1q_0 - r_2q_3 + r_3q_2)$$
$$t_2 = (r_0q_2 + r_1q_3 + r_2q_0 - r_3q_1)$$
$$t_3 = (r_0q_3 - r_1q_2 + r_2q_1 + r_3q_0)$$

## Dialog Box




## Inputs and Outputs

The first input is a quaternion or vector of quaternions in the form of $[q_0, p_0, ..., q_1, p_1, ... , q_2, p_2, ... , q_3, p_3, ...]$.

The second input is a quaternion or vector of quaternions in the form of $[s_0, r_0, ..., s_1, r_1, ... , s_2, r_2, ... , s_3, r_3, ...]$.

The output is a quaternion product or vector of quaternion products.

## See Also

Quaternion Conjugate

Quaternion Division

Quaternion Inverse

Quaternion Modulus

Quaternion Norm

Quaternion Normalize

Quaternion Rotation

# Quaternion Norm

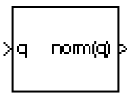
## Purpose

Calculate norm of quaternion

## Library

Utilities/Math Operations

## Description




The Quaternion Norm block calculates the norm for a given quaternion.

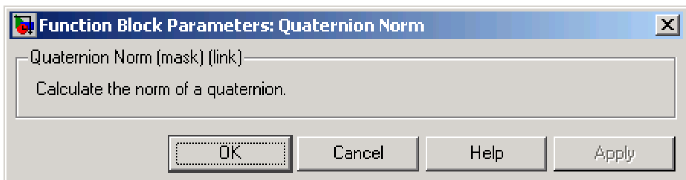
The quaternion has the form of

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

The quaternion norm has the form of

$$norm(q) = q_0^2 + q_1^2 + q_2^2 + q_3^2$$

## Dialog Box




## Inputs and Outputs

The input is a quaternion or vector of quaternions in the form of [$q_0$, $r_0$, ..., $q_1$, $r_1$, ... , $q_2$, $r_2$, ... , $q_3$, $r_3$, ...].

The output is a quaternion norm or vector of quaternion norms in the form of [*norm*(q), *norm*(r), ...].

## See Also

Quaternion Conjugate

Quaternion Division

Quaternion Inverse

Quaternion Modulus

Quaternion Multiplication

Quaternion Normalize

Quaternion Rotation

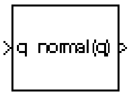
## Purpose

Normalize quaternion

## Library

Utilities/Math Operations

## Description




The Quaternion Normalize block calculates a normalized quaternion for a given quaternion.

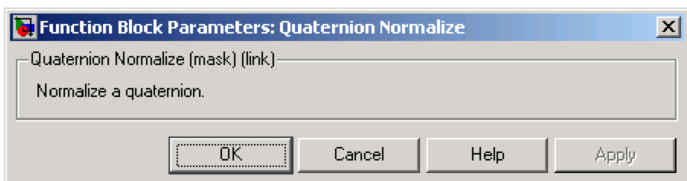
The quaternion has the form of

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

The normalized quaternion has the form of

$$normal(q) = \frac{q_0 + \boldsymbol{i}q_1 + \boldsymbol{j}q_2 + \boldsymbol{k}q_3}{\sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2}}$$

## Dialog Box




## Inputs and Outputs

The input is a quaternion or vector of quaternions in the form of [$q_0$, $r_0$, ..., $q_1$, $r_1$, ... , $q_2$, $r_2$, ... , $q_3$, $r_3$, ...].

The output is a normalized quaternion or vector of normalized quaternions.

## See Also

Quaternion Conjugate

Quaternion Division

Quaternion Inverse

Quaternion Modulus

Quaternion Multiplication

Quaternion Norm

Quaternion Rotation

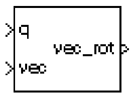
## Purpose

Rotate vector by quaternion

## Library

Utilities/Math Operations

## Description




The Quaternion Rotation block rotates a vector by a quaternion.

The quaternion has the form of

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

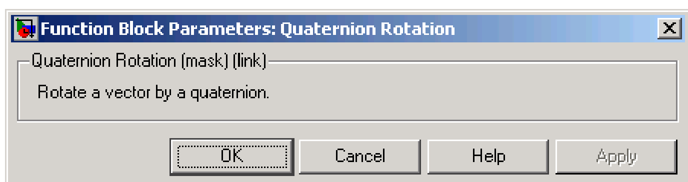
The vector has the form of

$$v = \boldsymbol{i}v_1 + \boldsymbol{j}v_2 + \boldsymbol{k}v_3$$

The rotated vector has the form of

$$v' = \begin{bmatrix} v_1' \\ v_2' \\ v_3' \end{bmatrix} = \begin{bmatrix} (1 - 2q_2^2 - 2q_3^2) & 2(q_1q_2 + q_0q_3) & 2(q_1q_3 - q_0q_2) \\ 2(q_1q_2 - q_0q_3) & (1 - 2q_1^2 - 2q_3^2) & 2(q_2q_3 + q_0q_1) \\ 2(q_1q_3 + q_0q_2) & 2(q_2q_3 - q_0q_1) & (1 - 2q_1^2 - 2q_2^2) \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}$$

## Dialog Box




## Inputs and Outputs

The first input is a quaternion or vector of quaternions in the form of $[q_0, r_0, \ldots, q_1, r_1, \ldots , q_2, r_2, \ldots , q_3, r_3, \ldots]$.

The second input is a vector or vector of vectors in the form of $[v_1, u_1, \ldots , v_2, u_2, \ldots , v_3, u_3, \ldots]$.

The output is a rotated vector or vector of rotated vectors.

## See Also

Quaternion Conjugate

Quaternion Division

Quaternion Inverse

Quaternion Modulus

Quaternion Multiplication

Quaternion Norm

Quaternion Normalize

# Quaternions to Direction Cosine Matrix

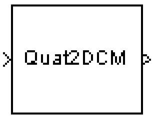
## Purpose

Convert quaternion vector to direction cosine matrix

## Library

Utilities/Axes Transformations

## Description




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

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

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

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

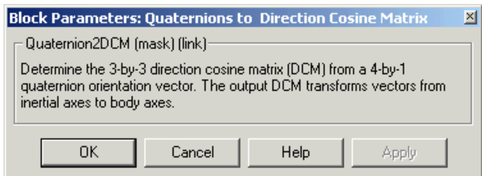
$$P' = \begin{bmatrix} 0 \\ x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} 0 \\ (q_0^2 + q_1^2 - q_2^2 - q_3^2)x + 2(q_1q_2 - q_0q_3)y + 2(q_1q_3 + q_0q_2)z \\ 2(q_0q_3 + q_1q_2)x + (q_0^2 - q_1^2 + q_2^2 - q_3^2)y + 2(q_2q_3 - q_0q_1)z \\ 2(q_1q_3 - q_0q_2)x + 2(q_0q_1 + q_2q_3)y + (q_0^2 - q_1^2 -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

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.

# Quaternions to Direction Cosine Matrix

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

## Dialog Box




## Inputs and Outputs

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

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

## See Also

Direction Cosine Matrix to Euler Angles

Direction Cosine Matrix to Quaternions

Euler Angles to Direction Cosine Matrix

Euler Angles to Quaternions

Quaternions to Euler Angles

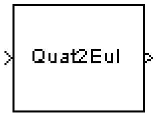
## Purpose

Convert quaternion vector to Euler angles

## Library

Utilities/Axes Transformations

## Description




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

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

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

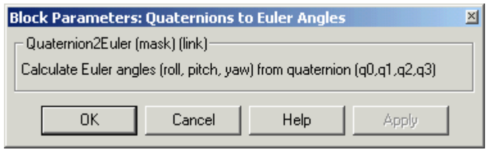
$$DCM = \begin{bmatrix} (q_0^2 + q_1^2 - q_2^2 - q_3^2) & 2(q_1q_2 + q_0q_3) & 2(q_1q_3 - q_0q_2) \\ 2(q_1q_2 - q_0q_3) & (q_0^2 - q_1^2 + q_2^2 - q_3^2) & 2(q_2q_3 + q_0q_1) \\ 2(q_1q_3 + q_0q_2) & 2(q_2q_3 - q_0q_1) & (q_0^2 - q_1^2 - q_2^2 + q_3^2) \end{bmatrix}$$

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

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

# Quaternions to Euler Angles

## Dialog Box




## Inputs and Outputs

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

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

## Assumptions and Limitations

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

The Euler angle solution is singular when the pitch angle $\theta$ is equal to $\pm 90$ degrees.

## Examples

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

## See Also

Direction Cosine Matrix to Euler Angles

Direction Cosine Matrix to Quaternions

Euler Angles to Direction Cosine Matrix

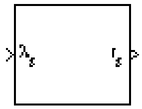
Euler Angles to Quaternions

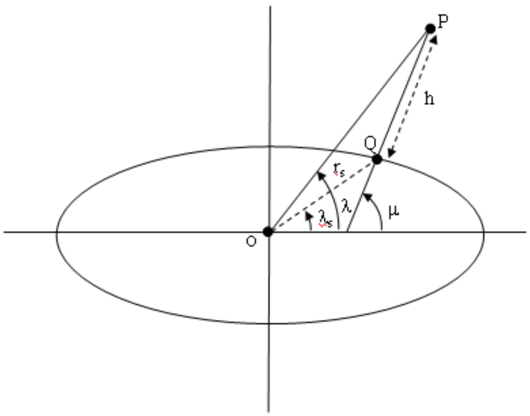
Quaternions to Direction Cosine Matrix

**Purpose** Estimate radius of ellipsoid planet at geocentric latitude

**Library** Flight Parameters

**Description** The Radius at Geocentric Latitude block estimates the radius $(r_s)$ of an ellipsoid planet at a particular geocentric latitude $(\lambda_s)$.

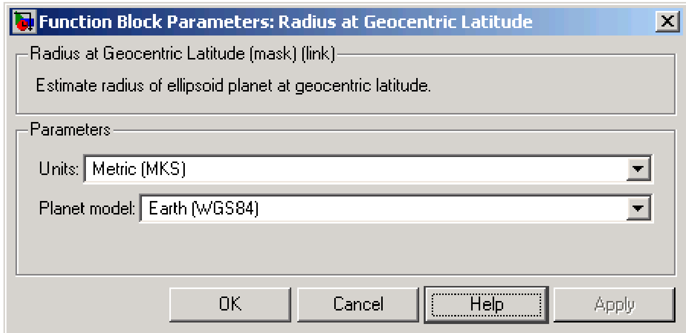


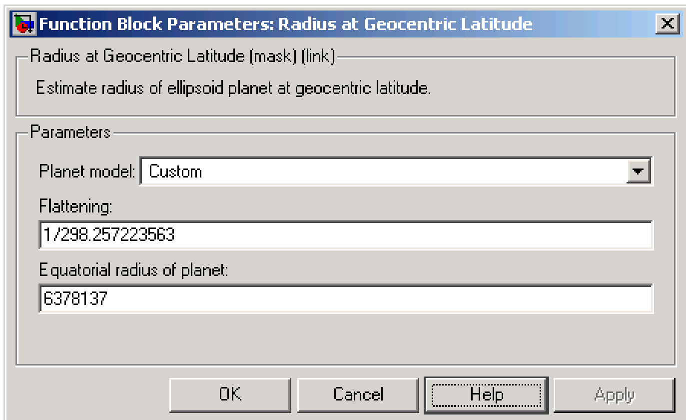




The following equation estimates the ellipsoid radius $(r_s)$ using flattening $(f)$, geocentric latitude $(\lambda_s)$, and equatorial radius $(R)$.

$$r_s = \sqrt{\frac{R^2}{1 + [1/(1-f)^2 - 1]\sin^2\lambda_s}}$$

## Dialog Box




**Units**

Specifies the parameter and output units:

| Units | Equatorial Radius | Radius at Geocentric Latitude |
|---|---|---|
| `Metric (MKS)` | Meters | Meters |
| `English` | Feet | Feet |

This option is only available when **Planet model** is set to `Earth (WGS84)`.

**Planet model**
Specifies the planet model to use:

`Custom`

`Earth (WGS84)`

**Flattening**
Specifies the flattening of the planet. This option is only available with **Planet model** set to `Custom`.

**Equatorial radius of planet**
Specifies the radius of the planet at its equator. This option is only available with **Planet model** set to `Custom`.

## Inputs and Outputs

The input is geocentric latitude, in degrees.

The output is radius of planet at geocentric latitude, in the same as the units as flattening.

## References

Stevens, B. L., and F. L. Lewis, *Aircraft Control and Simulation*, John Wiley & Sons, New York, 1992.

Zipfel, P. H., *Modeling and Simulation of Aerospace Vehicle Dynamics*, AIAA Education Series, Reston, Virginia, 2000.

## See Also

ECEF Position to LLA

Direction Cosine Matrix ECEF to NED

Direction Cosine Matrix ECEF to NED to Latitude and Longitude

Geocentric to Geodetic Latitude

Geodetic to Geocentric Latitude

LLA to ECEF Position

# Relative Ratio

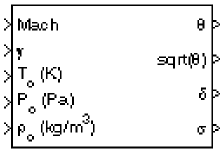
## Purpose

Calculate relative atmospheric ratios

## Library

Flight Parameters

## Description




The Relative Ratio block computes the relative atmospheric ratios, including relative temperature ratio (θ), $\sqrt{\theta}$, relative pressure ratio (δ), and relative density ratio (σ).

θ represents the ratio of the air stream temperature at a chosen reference station relative to sea level standard atmospheric conditions.

$$\theta = \frac{T}{T_o}$$

δ represents the ratio of the air stream pressure at a chosen reference station relative to sea level standard atmospheric conditions.

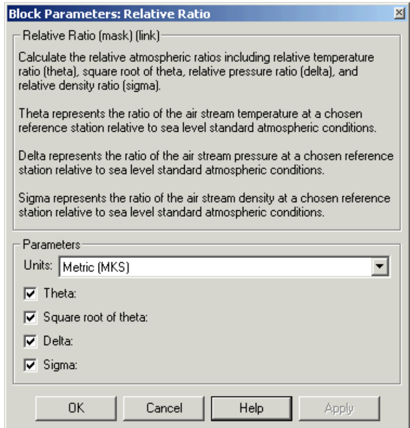
$$\delta = \frac{P}{P_o}$$

σ represents the ratio of the air stream density at a chosen reference station relative to sea level standard atmospheric conditions.

$$\sigma = \frac{\rho}{\rho_o}$$

The Relative Ratio block icon displays the input units selected from the **Units** list.

## Dialog Box




**Units**

Specifies the input units:

| 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 |

**Theta**

When selected, the $\theta$ is calculated and static temperature is a required input.

**Square root of theta**

When selected, the $\sqrt{\theta}$ is calculated and static temperature is a required input.

**Delta**

When selected, the $\delta$ is calculated and static pressure is a required input.

**Sigma**

When selected, the $\sigma$ is calculated and static density is a required input.

## Inputs and Outputs

The four possible inputs are Mach number, static temperature, static pressure, and static density.

The four possible outputs are $\theta$, $\sqrt{\theta}$, $\delta$, and $\sigma$.

## Assumptions

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.

## Reference

*Aeronautical Vestpocket Handbook*, United Technologies Pratt & Whitney, August, 1986.

## Purpose

Implement second-order linear actuator

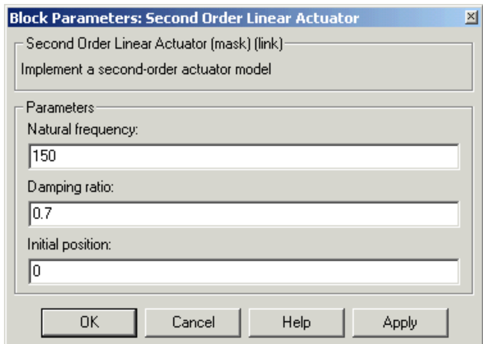
## Library

Actuators

## Description


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

## Dialog Box




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

**Damping ratio**
The damping ratio of the actuator. A dimensionless parameter.

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

## Inputs and Outputs

The input is the demanded actuator position.

The output is the actual actuator position.

## See Also

Second Order Nonlinear Actuator

# Second Order Nonlinear Actuator

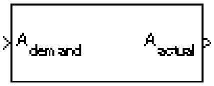
## Purpose

Implement second-order actuator with rate and deflection limits

## Library

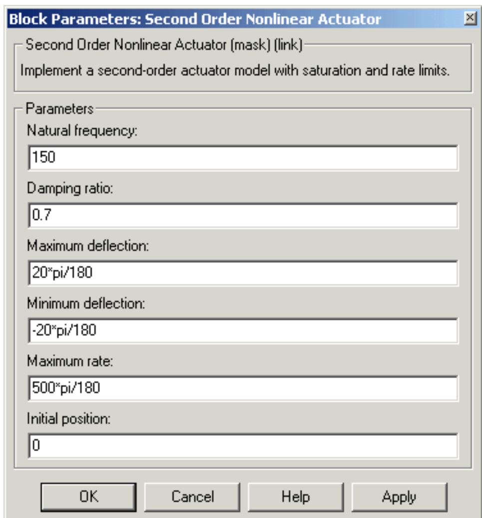
Actuators

## Description




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

## Dialog Box




**Natural frequency**

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

**Damping ratio**

The damping ratio of the actuator. A dimensionless parameter.

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

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

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

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

## Inputs and Outputs

The input is the demanded actuator position.

The output is the actual actuator position.

## Examples

See the `aero_guidance` model and the Actuators subsystem in the `aeroblk_HL20` model for an example of this block.

## See Also

Second Order Linear Actuator

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

## Purpose

Implement state-space controller in self-conditioned form

## Library

GNC/Controls

## Description


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

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

in the self-conditioned form

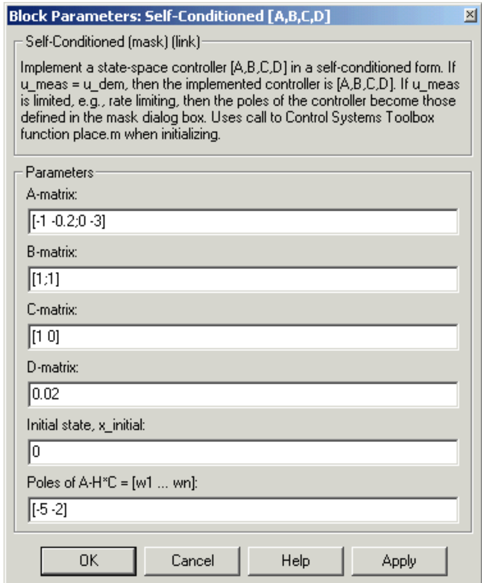
$$\dot{z} = (A - HC)z + (B - HD)e + Hu_{meas}$$

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

The input $u_{\mathrm{meas}}$ is a vector of the achieved actuator positions, and the output $u_{\mathrm{dem}}$ is the vector of controller actuator demands. In the case that the actuators are not limited, then $u_{\mathrm{meas}} = udem$ 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 $\mathrm{u}_{\mathrm{meas}}$ but at the same time not so fast that noise on e is propagated to $u_{\mathrm{dem}}$. The matrix H is designed by a callback to the Control System Toolbox command `place` to place the poles at defined locations.

## Dialog Box




**A-matrix**
: A-matrix of the state-space implementation.

**B-matrix**
: B-matrix of the state-space implementation.

**C-matrix**
: C-matrix of the state-space implementation.

**D-matrix**
: D-matrix of the state-space implementation.

**Initial state, x_initial**
: This is a vector of initial states for the controller, i.e., initial values for the state vector, $z$. It should have length equal to the size of the first dimension of A.

**Poles of A-H*C**

This is a vector of the desired poles of A-H*C. Hence the number of pole locations defined should be equal to the dimension of the *A*-matrix.

## Inputs and Outputs

The first input is the control error.

The second input is the measured actuator position.

The output is the actuator demands.

## Assumptions and Limitations

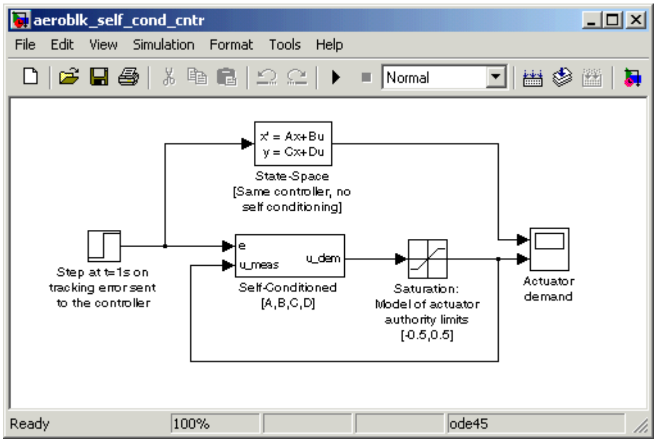
---

**Note** This block requires Control System Toolbox.

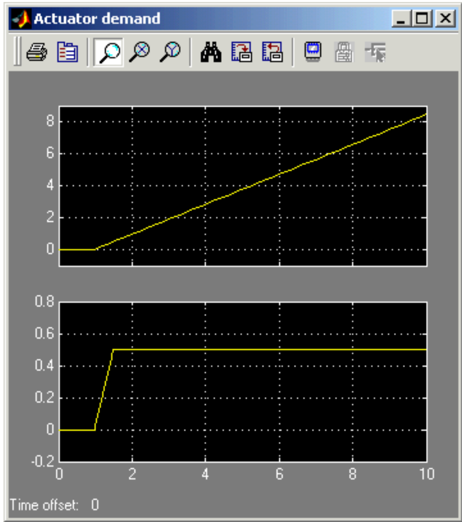
---

## Examples

This Simulink model shows a state-space controller implemented in both self-conditioned and standard state-space forms. The actuator authority limits of +/- 0.5 units are modeled by the saturation block.




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.

## Reference

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

## See Also

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

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

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

# Send net_fdm Packet to FlightGear

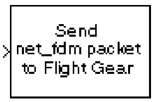
## Purpose

Transmit `net_fdm` packet to destination IP address and port for FlightGear session

## Library

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
```

The second port in the list, 5502, is the network flight dynamics model (fdm) port.

This block does not product deployable code.

### Determining the Destination IP Address

You can use one of several techniques to determine the destination IP address, such as:

- Use 127.0.0.1 for “this” 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`*:

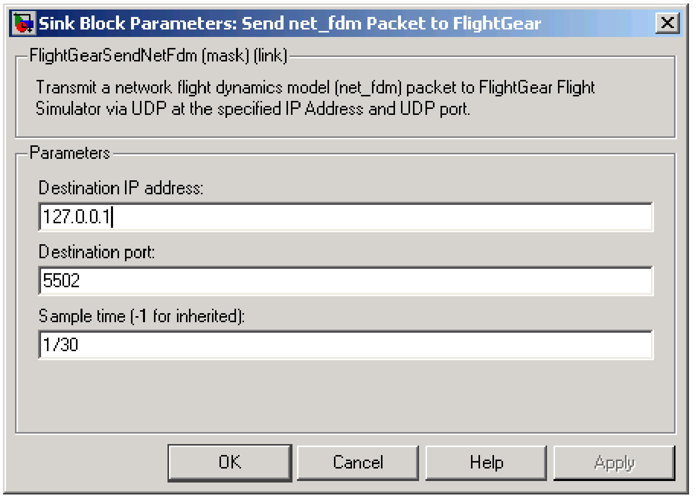
```
H:\>ipconfig

Windows 2000 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
```

## Dialog Box




**Destination IP address**

Specify your destination IP address.

**Destination port**
Specify your destination port.

**Sample time**
Specify the sample time (-1 for inherited).

## Inputs and Outputs

The input signal is the FlightGear `net_fdm` data packet.

## Examples

See the `asbhl20` for an example of this block.

## See Also

FlightGear Preconfigured 6DoF Animation

Generate Run Script

Pack net_fdm Packet for FlightGear

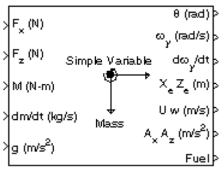
## Purpose

Implement three-degrees-of-freedom equations of motion of simple variable mass with respect to body axes

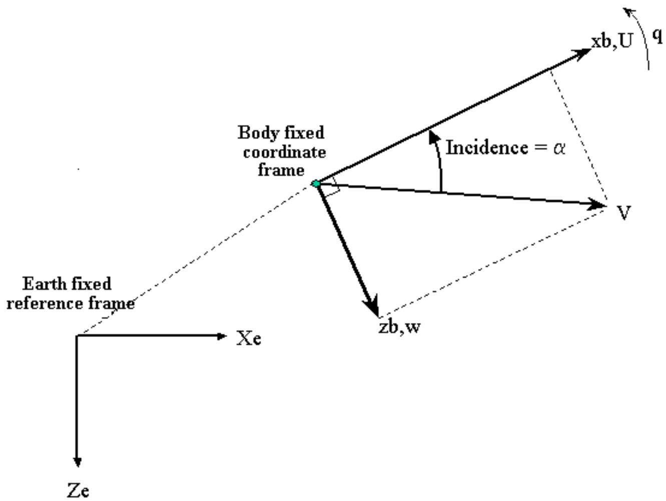
## Library

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 an Earth-fixed reference frame.




The equations of motion are

$$\dot{u} = \frac{F_x}{m} - \frac{\dot{m}U}{m} - q w - g\sin\theta$$

$$\dot{w} = \frac{F_z}{m} - \frac{\dot{m}w}{m} + q u + g\cos\theta$$

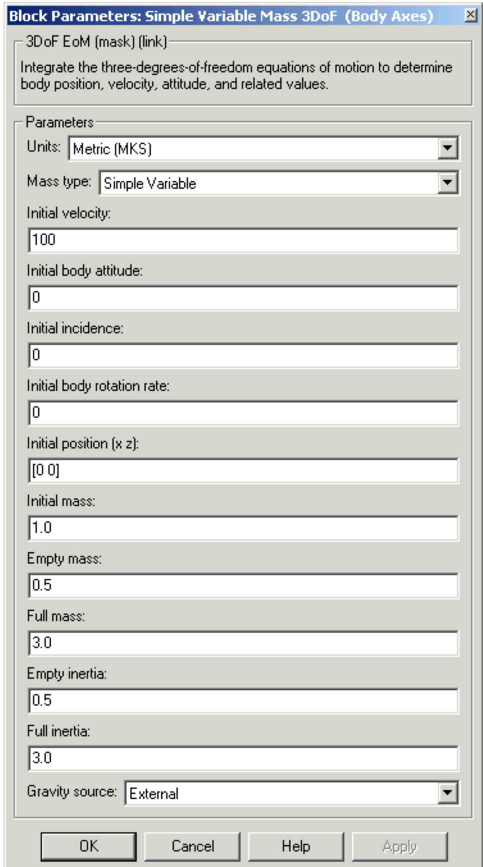
$$\dot{q} = \frac{M - \dot{I}_{yy}q}{I_{yy}}$$

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

$$\dot{I}_{yy} = \frac{I_{yyfull} - I_{yy\,empty}}{m_{full} - m_{empty}}\dot{m}$$

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

## Dialog Box




**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 Earth-fixed reference frame.

**Initial mass**
A scalar value for the initial mass 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 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.

## Inputs and Outputs

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

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

The third input to the block is the applied pitch moment, ($M$).

The fourth input to the block is the rate of change of mass, $(\dot{m})$.

The fifth optional input to the block is gravity in the selected units.

The first output from the block is the pitch attitude, in radians $(\theta)$.

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

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

The fourth output is a two-element vector containing the location of the body, in the Earth-fixed reference frame, (*Xe,Ze*).

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

The sixth output is a two-element vector containing the acceleration of the body resolved into the body-fixed coordinate frame, (*Ax,Az*).

The seventh output is a scalar element containing a flag for fuel tank status, (*Fuel*):

- 1 indicates that the tank is full.
- 0 indicates that the integral is neither full nor empty.
- -1 indicates that the tank is empty.

## 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)

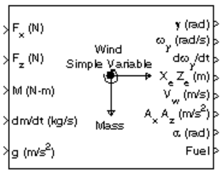
## Purpose

Implement three-degrees-of-freedom equations of motion of simple variable mass with respect to wind axes

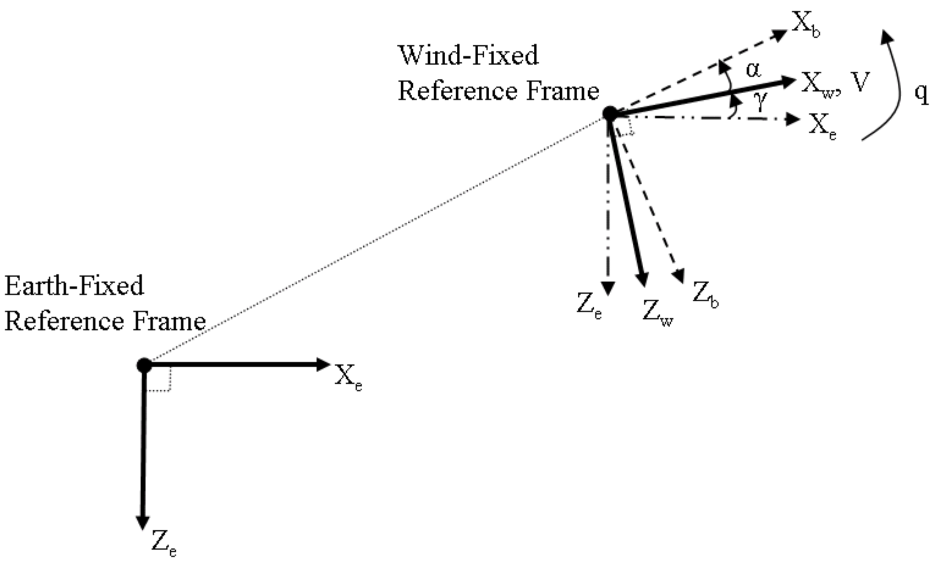
## Library

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 an Earth-fixed reference frame.




The equations of motion are

$$\dot{V} = \frac{F_{x_{wind}}}{m} - \frac{\dot{m}V}{m} - g\sin\gamma$$

$$\dot{\alpha} = \frac{F_{z_{wind}}}{mV} + q + \frac{g}{V}\cos\gamma$$

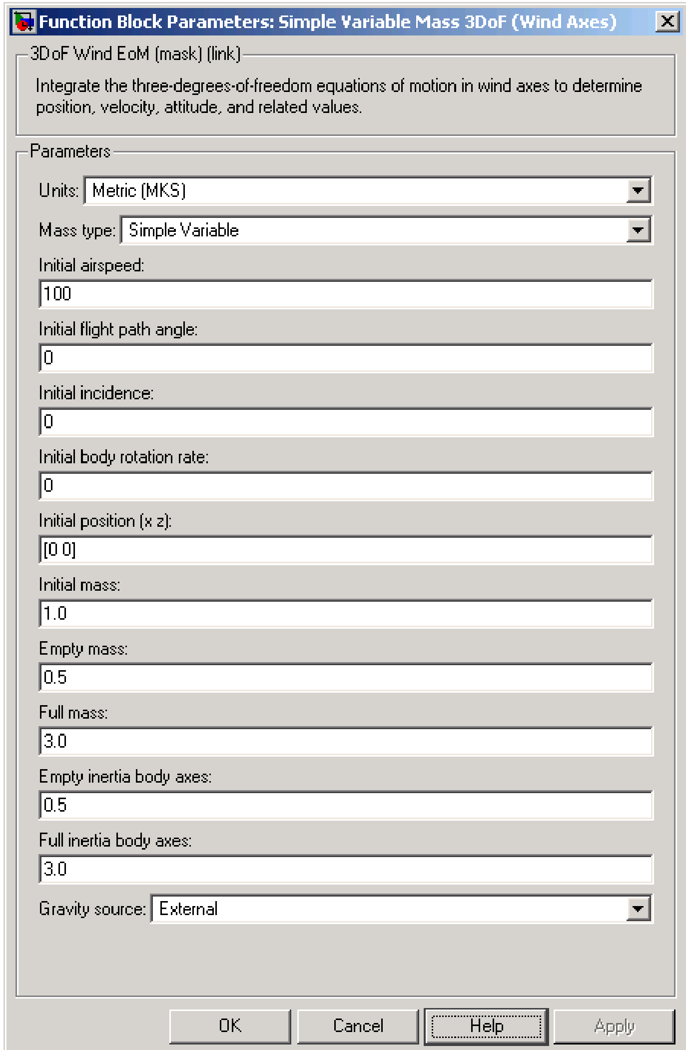
$$\dot{q} = \dot{\theta} = \frac{M_{y_{body}} - \dot{I}_{yy}q}{I_{yy}}$$

$$\dot{\gamma} = q - \dot{\alpha}$$

$$\dot{I}_{yy} = \frac{I_{yyfull} - I_{yy\,empty}}{m_{full} - m_{empty}}\dot{m}$$

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

## Dialog Box


Function Block Parameters: Simple Variable Mass 3DoF (Wind Axes)
3DoF Wind EoM (mask) (link)
Integrate the three-degrees-of-freedom equations of motion in wind axes to determine position, velocity, attitude, and related values.
Parameters
Units: Metric (MKS)
Mass type: Simple Variable
Initial airspeed:
100
Initial flight path angle:
0
Initial incidence:
0
Initial body rotation rate:
0
Initial position (x z):
[0 0]
Initial mass:
1.0
Empty mass:
0.5
Full mass:
3.0
Empty inertia body axes:
0.5
Full inertia body axes:
3.0
Gravity source: External
OK
Cancel
Help
Apply

**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 Earth-fixed reference frame.

**Initial mass**

A scalar value for the initial mass 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 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.

## Inputs and Outputs

The first input to the block is the force acting along the wind *x*-axis, $(F_x)$.

The second input to the block is the force acting along the wind *z*-axis, $(F_z)$.

The third input to the block is the applied pitch moment in body axes, $(M)$.

The fourth input to the block is the rate of change of mass, $(\dot{m})$.

The fifth optional input to the block is gravity in the selected units.

The first output from the block is the flight path angle, in radians $(\gamma)$.

The second output is the pitch angular rate, in radians per second $(\omega_y)$.

The third output is the pitch angular acceleration, in radians per second squared $(d\omega_y/dt)$.

The fourth output is a two-element vector containing the location of the body, in the Earth-fixed reference frame, (*Xe,Ze*).

The fifth output is a two-element vector containing the velocity of the body resolved into the wind-fixed coordinate frame, (*V,0*).

The sixth output is a two-element vector containing the acceleration of the body resolved into the body-fixed coordinate frame, (*Ax,Az*).

The seventh output is a scalar containing the angle of attack, $(\alpha)$.

The eighth output is a scalar element containing a flag for fuel tank status, (*Fuel*):

- 1 indicates that the tank is full.
- 0 indicates that the integral is neither full nor empty.
- -1 indicates that the tank is empty.

## 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)

# Simple Variable Mass 6DoF (Euler Angles)

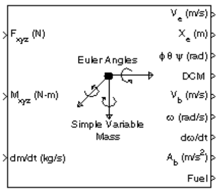
## Purpose

Implement Euler angle representation of six-degrees-of-freedom equations of motion of simple variable mass

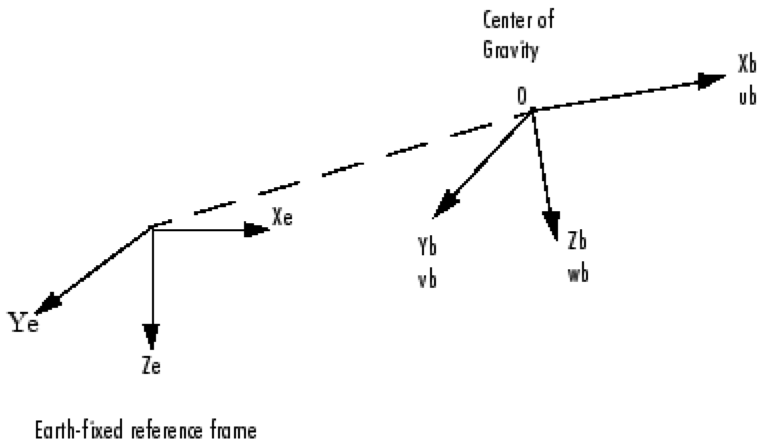
## Library

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 an Earth-fixed reference frame $(X_e, Y_e, Z_e)$. The origin of the body-fixed coordinate frame is the center of gravity of the body, and the body is assumed to be rigid, an assumption that eliminates the need to consider the forces acting between individual elements of mass. The Earth-fixed reference frame is considered inertial, 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 body-fixed coordinate frame is given below, where the applied forces $[F_x\ F_y\ F_z]^T$ are in the body-fixed frame.

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

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

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

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

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

The 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.

$$\dot{I} = \frac{I_{full} - I_{empty}}{m_{full} - m_{empty}}\dot{m}$$

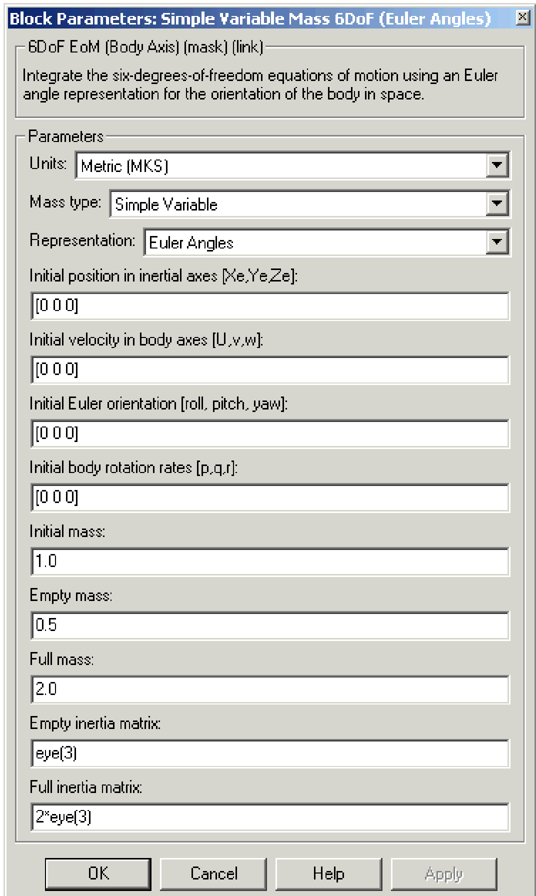
The relationship between the body-fixed angular velocity vector, $[p\ q\ r]^T$, and the rate of change of the Euler angles, $[\dot{\phi}\ \dot{\theta}\ \dot{\psi}]^T$, can be determined by resolving the Euler rates into the body-fixed coordinate frame.

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

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

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

## Dialog Box



**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 Earth-fixed 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.

## Inputs and Outputs

The first input to the block is a vector containing the three applied forces.

The second input is a vector containing the three applied moments.

The third input is a scalar containing the rate of change of mass.

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

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

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

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

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

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

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

The eighth output is a three-element vector containing the accelerations in body-fixed axes.

The ninth output is a scalar element containing a flag for fuel tank status:

- 1 indicates that the tank is full.
- 0 indicates that the integral is neither full nor empty.
- -1 indicates that the tank is empty.

## Assumptions and Limitations

The block assumes that the applied forces are acting at the center of gravity of the body.

## Reference

Mangiacasale, L., *Flight Mechanics of a μ-Airplane with a MATLAB Simulink Helper,* Edizioni Libreria CLUP, Milan, 1998.

## 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)

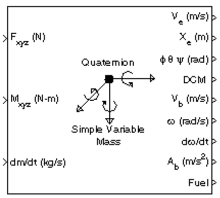
## Purpose

Implement quaternion representation of six-degrees-of-freedom equations of motion of simple variable mass with respect to body axes

## Library

Equations of Motion/6DoF

## Description



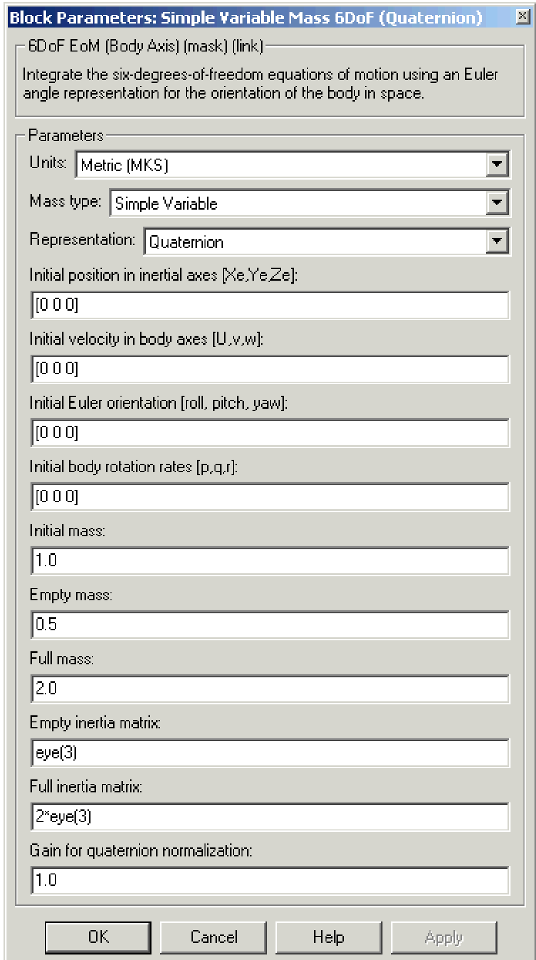

For a description of the coordinate system employed and the translational dynamics, see the block description for the Simple Variable Mass 6DoF (Euler Angles) block.

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 $\epsilon$ 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.

$$\begin{bmatrix} \dot{q}_0 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} = \frac{1}{2}\begin{bmatrix} 0 & -p & -q & -r \\ p & 0 & r & -q \\ q & -r & 0 & p \\ r & q & -p & 0 \end{bmatrix}\begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} + K\varepsilon\begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix}$$

$$\varepsilon = 1 - (q_0^2 + q_1^2 + q_3^2 + q_4^2)$$

## Dialog Box




**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 `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 Earth-fixed 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.

**Gain for quaternion normalization**

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

## Inputs and Outputs

The first input to the block is a vector containing the three applied forces.

The second input is a vector containing the three applied moments.

The third input is a scalar containing the rate of change of mass.

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

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

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

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

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

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

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

The eighth output is a three-element vector containing the accelerations in body-fixed axes.

The ninth output is a scalar element containing a flag for fuel tank status:

- 1 indicates that the tank is full.
- 0 indicates that the integral is neither full nor empty.
- -1 indicates that the tank is empty.

## Assumptions and Limitations

The block assumes that the applied forces are acting at the center of gravity of the body.

## Reference

Mangiacasale, L., *Flight Mechanics of a* μ*-Airplane with a MATLAB Simulink Helper,* Edizioni Libreria CLUP, Milan, 1998.

## 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 ECEF (Quaternion)

Simple Variable Mass 6DoF Wind (Quaternion)

Simple Variable Mass 6DoF Wind (Wind Angles)

# Simple Variable Mass 6DoF ECEF (Quaternion)

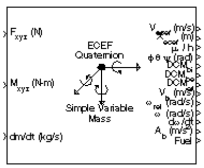
## Purpose

Implement quaternion representation of six-degrees-of-freedom equations of motion of simple variable mass in Earth-centered Earth-fixed (ECEF) coordinates

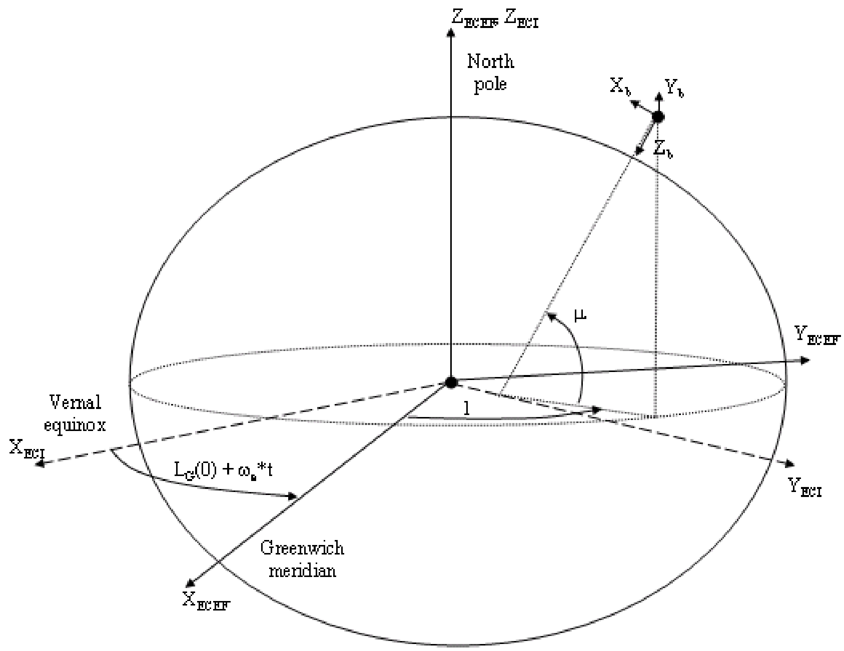
## Library

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.

$$\bar{F}_b = \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} = m(\dot{\bar{V}}_b + \bar{\omega}_b \times \bar{V}_b + DCM_{bf}\bar{\omega}_e \times \bar{V}_b) + DCM_{bf}(\bar{\omega}_e \times (\bar{\omega}_e \times \bar{X}_f)) + \dot{m}(\bar{V}_b + DCM_{bf}(\bar{\omega}_e \times \bar{X}_f))$$

where the change of position in ECEF $\dot{\underline{x}}_f$ $(\dot{\underline{x}}_i)$ 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 $(\underline{V}_b)$, angular rates of the body with respect to ECI frame, expressed in body frame $(\underline{\omega}_b)$. Earth rotation rate $(\underline{\omega}_e)$, and relative angular rates of the body with respect to north-east-down (NED) frame, expressed in body frame $(\underline{\omega}_{rel})$ are defined as

$$\bar{V}_b = \begin{bmatrix} u \\ v \\ \omega \end{bmatrix}, \bar{\omega}_{rel} = \begin{bmatrix} p \\ q \\ r \end{bmatrix}, \bar{\omega}_e = \begin{bmatrix} 0 \\ 0 \\ \omega_e \end{bmatrix}, \bar{\omega}_b = \bar{\omega}_{rel} + DCM_{bf}\bar{\omega}_e + DCM_{be}\bar{\omega}_{ned}$$

$$\bar{\omega}_{ned} = \begin{bmatrix} \dot{l}\cos\mu \\ -\dot{\mu} \\ -\dot{l}\sin\mu \end{bmatrix} = \begin{bmatrix} V_E/(N+h) \\ -V_N/(M+h) \\ V_E \bullet \tan\mu/(N+h) \end{bmatrix}$$

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.

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

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

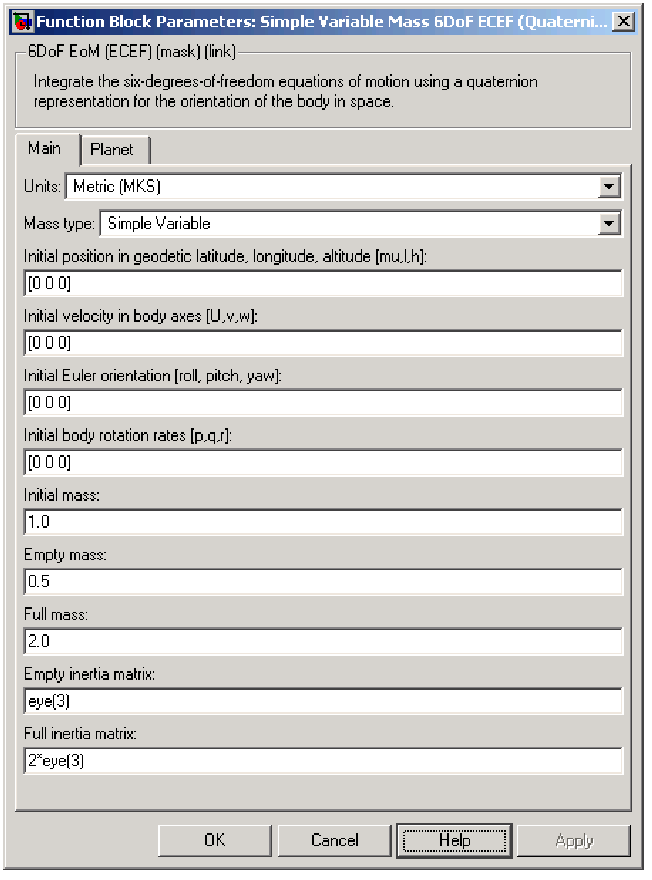
The inertia tensor is determined using a table lookup which linearly interpolates between $I_{full}$ and $I_{empty}$ based on mass (m). The rate of change of the inertia tensor is estimated by the following equation.

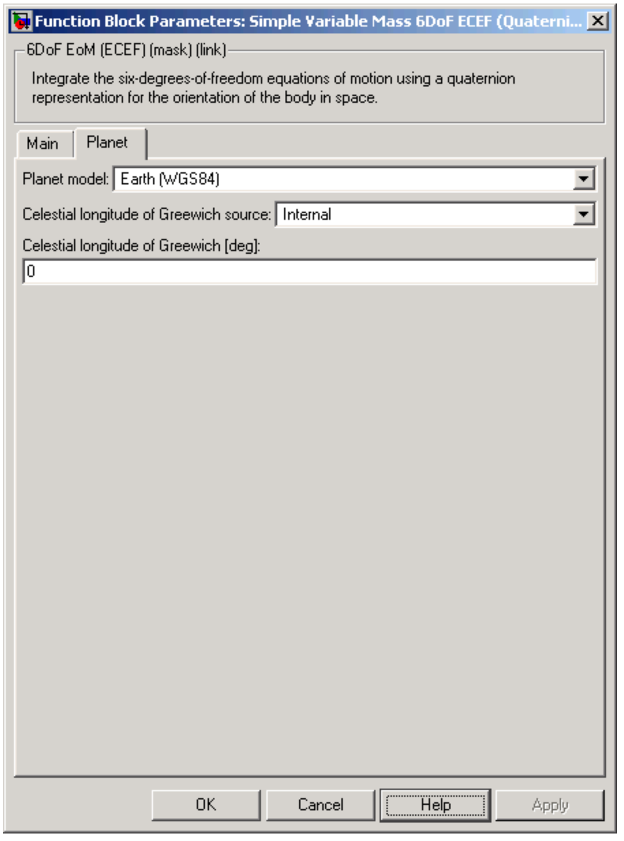
$$\dot{I} = \frac{I_{full} - I_{empty}}{m_{full} - m_{empty}}\dot{m}$$

The integration of the rate of change of the quaternion vector is given below.

$$\begin{bmatrix} \dot{q}_0 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} = -\tfrac{1}{2} \begin{bmatrix} 0 & \omega_b(1) & \omega_b(2) & \omega_b(3) \\ -\omega_b(1) & 0 & -\omega_b(3) & \omega_b(2) \\ -\omega_b(2) & \omega_b(3) & 0 & -\omega_b(1) \\ -\omega_b(3) & -\omega_b(2) & \omega_b(1) & 0 \end{bmatrix} \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix}$$

## Dialog Box






**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 (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.

**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.

**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.

**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.

## Inputs and Outputs

| Input | Dimension Type | Description |
|---|---|---|
| First input | Vector | Contains the three applied forces in body-fixed axes. |
| Second input | Vector | Contains the three applied moments in body-fixed axes. |
| Third input | Scalar | Contains the rate of change of mass. |

| Output | Dimension Type | Description |
|---|---|---|
| First output | Three-element vector | Contains the velocity of body respect to ECEF frame, expressed in ECEF frame. |
| Second output | Three-element vector | Contains the position in the ECEF reference frame. |

| Output | Dimension Type | Description |
|---|---|---|
| Third output | Three-element vector | Contains the position in geodetic latitude, longitude and altitude, in degrees, degrees and selected units of length respectively. |
| Fourth output | 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 output | 3-by-3 matrix | Applies to the coordinate transformation from ECI axes to body-fixed axes. |
| Sixth output | 3-by-3 matrix | Applies to the coordinate transformation from geodetic axes to body-fixed axes. |
| Seventh output | 3-by-3 matrix | Applies to the coordinate transformation from ECEF axes to geodetic axes. |
| Eighth output | Three-element vector | Contains the velocity of body with respect to ECEF frame, expressed in body frame. |
| Ninth output | Three-element vector | Contains the relative angular rates of body with respect to NED frame, expressed in body frame, in radians per second. |
| Tenth output | Three-element vector | Contains the angular rates of the body with respect to ECI frame, expressed in body frame, in radians per second. |

| Output | Dimension Type | Description |
|---|---|---|
| Eleventh output | Three-element vector | Contains the angular accelerations of the body with respect to ECI frame, expressed in body frame, in radians per second. |
| Twelfth output | Three-element vector | Contains the accelerations in body-fixed axes. |
| Thirteenth output | Scalar | Is an element containing a flag for fuel tank status:<br>• 1 indicates that the tank is full.<br>• 0 indicates that the integral is neither full nor empty.<br>• -1 indicates that the tank is empty. |

## 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 through the center of the Sun 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)

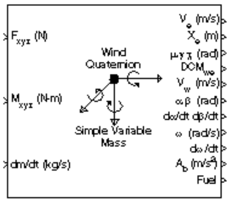
## Purpose

Implement quaternion representation of six-degrees-of-freedom equations of motion of simple variable mass with respect to wind axes

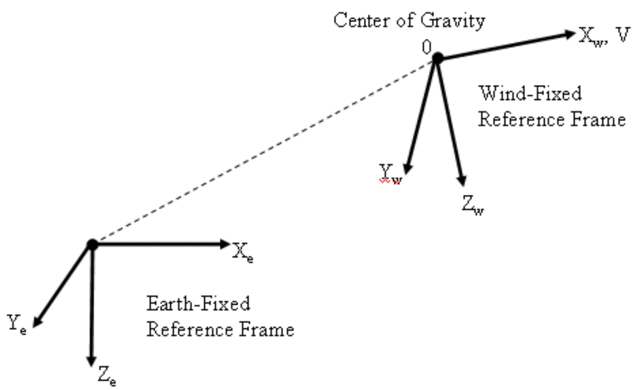
## Library

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 Earth-fixed 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 Earth-fixed 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.

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

$$\underline{V}_w = \begin{bmatrix} V \\ 0 \\ 0 \end{bmatrix}, \underline{\omega}_w = \begin{bmatrix} p_w \\ q_w \\ r_w \end{bmatrix} = DMC_{wb} \begin{bmatrix} p_b - \dot{\beta}\sin\alpha \\ q_b - \dot{\alpha} \\ r_b + \dot{\beta}\cos\alpha \end{bmatrix}, \underline{\omega}_b = \begin{bmatrix} p_b \\ q_b \\ r_b \end{bmatrix}$$

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

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

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

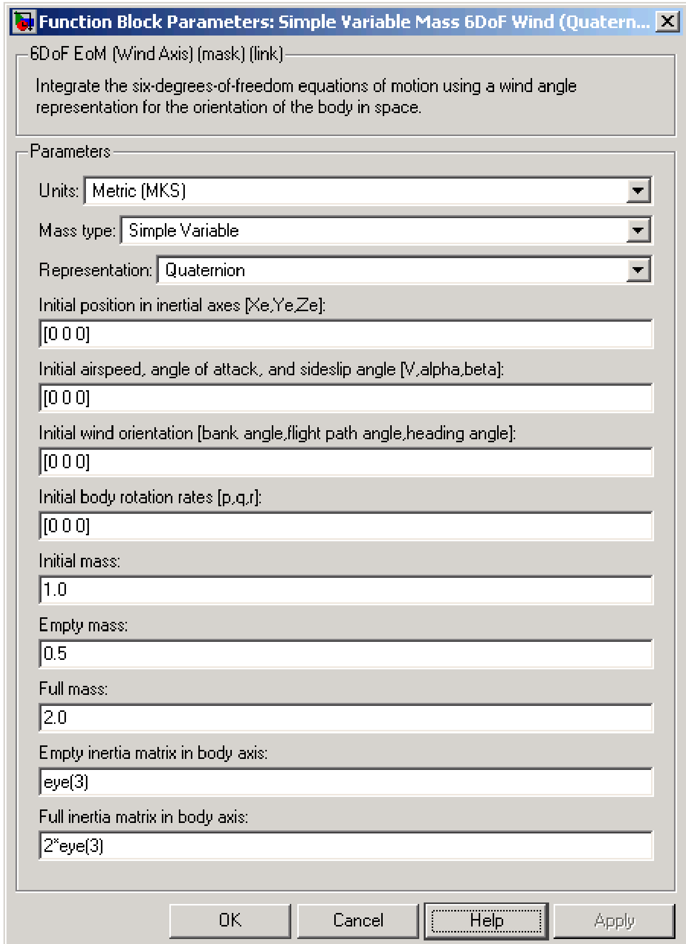
The inertia tensor is determined using a table lookup which linearly interpolates between $I_{full}$ and $I_{empty}$ based on mass (m). While the rate of change of the inertia tensor is estimated by the following equation.

$$\dot{I} = \frac{I_{full} - I_{empty}}{m_{full} - m_{empty}}\dot{m}$$

The integration of the rate of change of the quaternion vector is given below.

$$\begin{bmatrix} \dot{q}_0 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} = -\frac{1}{2}\begin{bmatrix} 0 & p & q & r \\ -p & 0 & -r & q \\ -q & r & 0 & -p \\ -r & -q & p & 0 \end{bmatrix}\begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix}$$

## Dialog Box




**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 `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 Earth-fixed 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.

## Inputs and Outputs

The first input to the block is a vector containing the three applied forces in wind-fixed axes.

The second input is a vector containing the three applied moments in body-fixed axes.

The third input is a scalar containing the rate of change of mass.

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

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

The third output is a three-element vector containing the wind rotation angles [bank, flight path, heading], in radians.

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

The fifth output is a three-element vector containing the velocity in the wind-fixed frame.

The sixth output is a two-element vector containing the angle of attack and sideslip angle, in radians.

The seventh output is a two-element vector containing the rate of change of angle of attack and rate of change of sideslip angle, in radians per second.

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

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

The tenth output is a three-element vector containing the accelerations in body-fixed axes.

The eleventh output is a scalar element containing a flag for fuel tank status:

- 1 indicates that the tank is full.
- 0 indicates that the integral is neither full nor empty.
- -1 indicates that the tank is empty.

## Assumptions and Limitations

The block assumes that the applied forces are acting at the center of gravity of the body.

## References

Mangiacasale, L., *Flight Mechanics of a μ-Airplane with a MATLAB Simulink Helper,* Edizioni Libreria CLUP, Milan, 1998.

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)

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)

# Simple Variable Mass 6DoF Wind (Wind Angles)

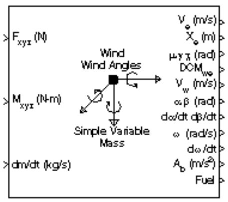
## Purpose

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.

$$\begin{bmatrix} \dot{\mu} \\ \dot{\gamma} \\ \dot{\chi} \end{bmatrix} = J \begin{bmatrix} p_w \\ q_w \\ r_w \end{bmatrix} = \begin{bmatrix} 1 & (\sin\mu\tan\gamma) & (\cos\mu\tan\gamma) \\ 0 & \cos\mu & -\sin\mu \\ 0 & \dfrac{\sin\mu}{\cos\gamma} & \dfrac{\cos\mu}{\cos\gamma} \end{bmatrix} \begin{bmatrix} p_w \\ q_w \\ r_w \end{bmatrix}$$

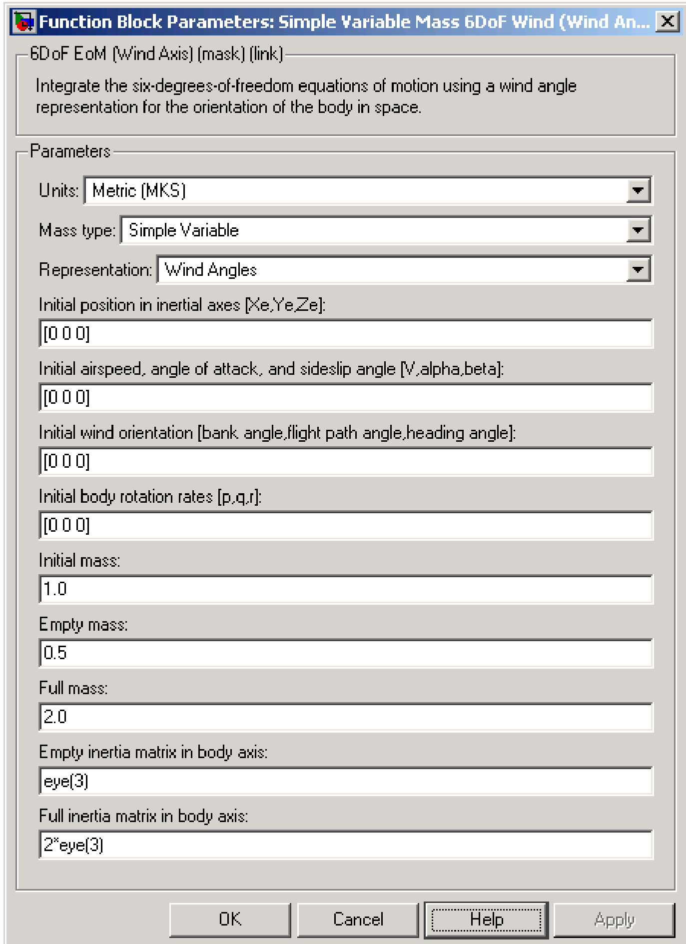
The body-fixed angular rates are related to the wind-fixed angular rate by the following equation.

$$\begin{bmatrix} p_w \\ q_w \\ r_w \end{bmatrix} = DMC_{wb} \begin{bmatrix} p_b - \dot{\beta}\sin\alpha \\ q_b - \dot{\alpha} \\ r_b + \dot{\beta}\cos\alpha \end{bmatrix}$$

Using this relationship in the wind rate vector equations, gives the relationship between the wind rate vector and the body-fixed angular rates.

$$\begin{bmatrix}\dot{\mu}\\ \dot{\gamma}\\ \dot{\chi}\end{bmatrix} = J\begin{bmatrix}p_w\\ q_w\\ r_w\end{bmatrix} = \begin{bmatrix}1 & (\sin\mu\tan\gamma) & (\cos\mu\tan\gamma)\\ 0 & \cos\mu & -\sin\mu\\ 0 & \frac{\sin\mu}{\cos\gamma} & \frac{\cos\mu}{\cos\gamma}\end{bmatrix} DMC_{wb}\begin{bmatrix}p_b - \dot{\beta}\sin\alpha\\ q_b - \dot{\alpha}\\ r_b + \dot{\beta}\cos\alpha\end{bmatrix}$$

## Dialog Box




### 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 Earth-fixed 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.

## Inputs and Outputs

The first input to the block is a vector containing the three applied forces in wind-fixed axes.

The second input is a vector containing the three applied moments in body-fixed axes.

The third input is a scalar containing the rate of change of mass.

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

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

The third output is a three-element vector containing the wind rotation angles [bank, flight path, heading], in radians.

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

The fifth output is a three-element vector containing the velocity in the wind-fixed frame.

The sixth output is a two-element vector containing the angle of attack and sideslip angle, in radians.

The seventh output is a two-element vector containing the rate of change of angle of attack and rate of change of sideslip angle, in radians per second.

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

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

The tenth output is a three-element vector containing the accelerations in body-fixed axes.

The eleventh output is a scalar element containing a flag for fuel tank status:

- 1 indicates that the tank is full.
- 0 indicates that the integral is neither full nor empty.
- -1 indicates that the tank is empty.

## Assumptions and Limitations

The block assumes that the applied forces are acting at the center of gravity of the body.

## References

Mangiacasale, L., *Flight Mechanics of a μ-Airplane with a MATLAB Simulink Helper,* Edizioni Libreria CLUP, Milan, 1998.

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)

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)

## Purpose

Set simulation rate for FlightGear flight simulator

## Library

Animation/Animation Support Utilities

## Description


The Simulation Pace block lets you run the simulation at the specified pace so that connected animations appear aesthetically pleasing.

This block does not product deployable code.

Use the **Sample time** parameter to set how often Simulink synchronizes with the wall clock.

The sample time of this block should be considered for human interaction with visualizations. The default is 1/30th of a second, chosen to correspond to a 30 frames-per-second visualization rate (typical for desktop computers).

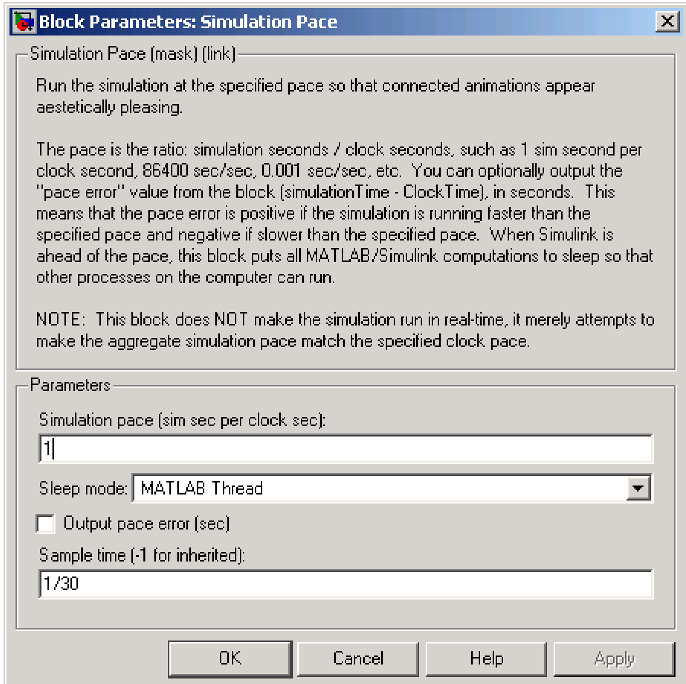
---

**Caution**

Choose as slow of a sample time as needed for smooth animation, since oversampling has little benefit and undersampling can cause animation “jumpiness” and potentially block the MATLAB main thread on your computer.

---

## Dialog Box




**Simulation pace**

Specifies the ratio of simulation time to clock time. The default is 1 second of simulation time per second of clock time.

**Sleep mode**

Setting the **Sleep mode** parameter to `off` lets you disable the pace functionality and run as fast as possible.

**Output pace error**

If you select this check box, the block 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.

**Sample time**

Specify the sample time (-1 for inherited). Larger sample times result in more efficient simulations, but less smooth in output pace when there are multiple Simulink time steps between pacer block samples. If the **Sample time** is too large, MATLAB may become less responsive as MATLAB and Simulink calculations are blocked from running when the block puts MATLAB to sleep.

## Inputs and Outputs

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 make decisions about the simulation if the simulation is too slow to keep up with the specified pace.

## Assumptions and Limitations

The simulation pace is implemented by putting the entire MATLAB thread to sleep until it needs to run again to keep up the pace. Simulink is single threaded and runs on the one MATLAB thread, so only one Simulation Pace block can be active at a time.

## Examples

See the `asbhl20` demo for an example of this block.

## See Also

Pilot Joystick

# SinCos

## Purpose

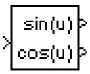
Compute sine and cosine of angle

## Library

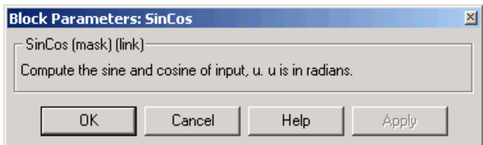
Utilities/Math Operations

## Description

The SinCos block computes the sine and cosine of the input angle, theta.




## Dialog Box




## Inputs and Outputs

The first input is an angle, in radians.

The first output is the sine of the input angle.

The second output is the cosine of the input angle.

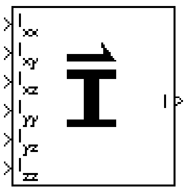
## Purpose

Create inertia tensor from moments and products of inertia

## Library

Mass Properties

## Description

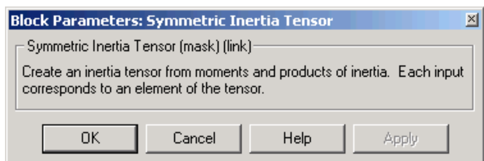


The Symmetric Inertia Tensor block creates an inertia tensor from moments and products of inertia. Each input corresponds to an element of the tensor.

The inertia tensor has the form of

$$Inertia = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{yz} \\ -I_{xy} & I_{yy} & -I_{xz} \\ -I_{yz} & -I_{xz} & I_{zz} \end{bmatrix}$$

## Dialog Box




## Inputs and Outputs

The first input is the moment of inertia about the *x*-axis.

The second input is the product of inertia in the *xy* plane.

The third input is the product of inertia in the *xz* plane.

The fourth input is the moment of inertia about the *y*-axis.

The fifth input is the product of inertia in the *yz* plane.

The sixth input is the moment of inertia about the *z*-axis.

The output of the block is a symmetric 3-by-3 inertia tensor.

## See Also

Create 3x3 Matrix

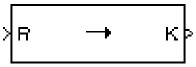
## Purpose

Convert from temperature units to desired temperature units

## Library

Utilities/Unit Conversions

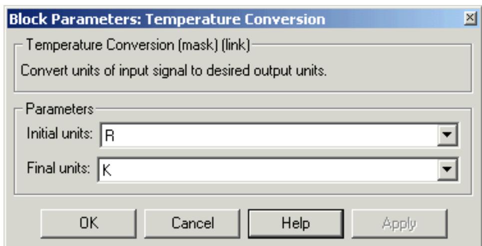
## Description




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 icon displays the input and output units selected from the **Initial units** and the **Final units** lists.

## Dialog Box




**Initial units**
Specifies the input units.

**Final units**
Specifies the output units.

The following conversion units are available:

| | |
|---|---|
| K | Kelvin |
| F | Degrees Fahrenheit |
| C | Degrees Celsius |
| R | Degrees Rankine |

## Inputs and Outputs

The input is the temperature in initial temperature units.

The output is the temperature in final temperature units.

**See Also**

Acceleration Conversion

Angle Conversion

Angular Acceleration Conversion

Angular Velocity Conversion

Density Conversion

Force Conversion

Length Conversion

Mass Conversion

Pressure Conversion

Velocity Conversion

# Three-Axis Accelerometer

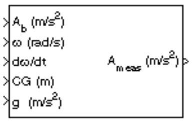
## Purpose

Implement three-axis accelerometer

## Library

GNC/Navigation

## Description




The Three-Axis Accelerometer block implements an accelerometer on each of the three axes. The ideal measured accelerations $(\underline{A}_{imeas})$ include the acceleration in body axes at the center of gravity $(\underline{A}_b)$, lever arm effects due to the accelerometer not being at the center of gravity, and, optionally, gravity in body axes can be removed.

$$\underline{A}_{imeas} = \underline{A}_b + \underline{\omega}_b \times (\underline{\omega}_b \times \underline{d}) + \dot{\underline{\omega}}_b \times \underline{d} - \underline{g}$$

where $\underline{\omega}_b$ are body-fixed angular rates, $\dot{\underline{\omega}}_b$ are body-fixed angular accelerations and $\underline{d}$ is the lever arm. The lever arm $(\underline{d})$ is defined as the distances that the accelerometer group is forward, right and below the center of gravity.

$$\underline{d} = \begin{bmatrix} d_x \\ d_y \\ d_z \end{bmatrix} = \begin{bmatrix} -(x_{acc} - x_{CG}) \\ y_{acc} - y_{CG} \\ -(z_{acc} - z_{CG}) \end{bmatrix}$$

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 this measurement axes are opposite the body-fixed axes producing the negative signs in the lever arms for $x$-axis and $z$-axis.

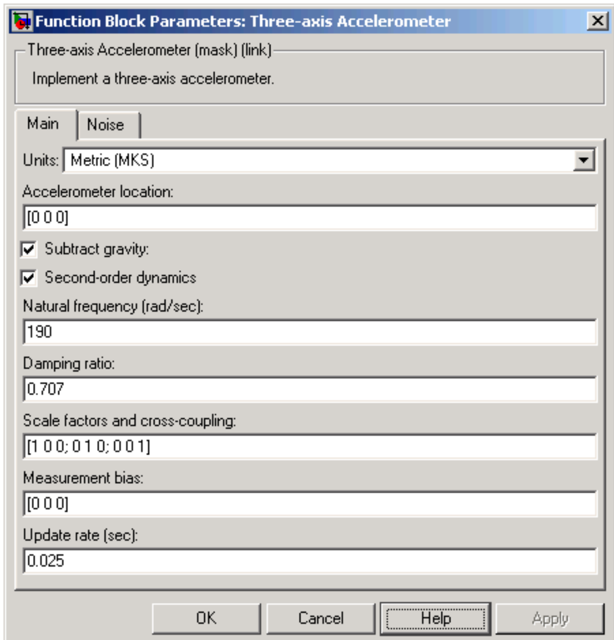
Measured accelerations $(\underline{A}_{meas})$ output by this block contain error sources and are defined as

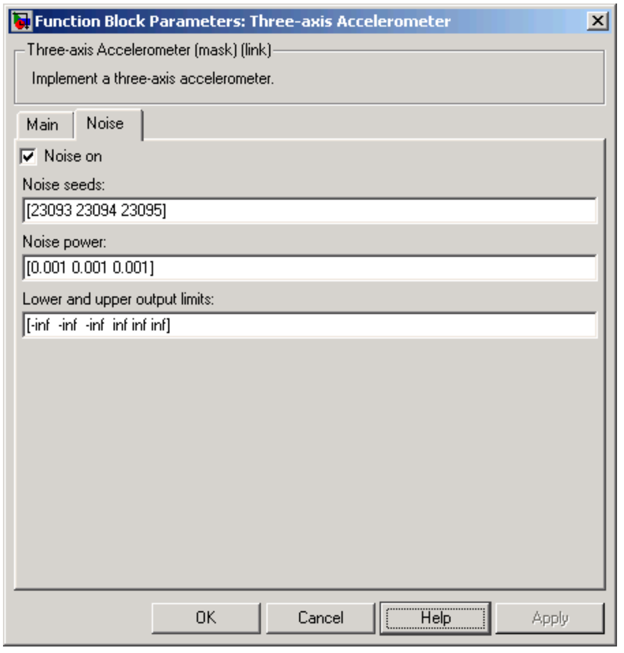
$$\underline{A}_{meas} = \underline{A}_{imeas} \cdot \underline{A}_{SFCC} + \underline{A}_{bias} + noise$$

where $\underline{A}_{SFCC}$ is a 3-by-3 matrix of scaling factors on the diagonal and misalignment terms in the nondiagonal, and $\underline{A}_{bias}$ are the biases.

Optionally discretizations can be applied to the block inputs and dynamics along with nonlinearizations of the measured accelerations via a Saturation block.

## Dialog Box






**Units**

Specifies the input and output units:

| Units | Acceleration | Length |
|---|---|---|
| `Metric (MKS)` | Meters per second squared | Meters |
| `English` | Feet per second squared | Feet |

**Accelerometer location**

The location of the accelerometer group 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 selected length units.

**Subtract gravity**
Select to subtract gravity from acceleration readings.

**Second order dynamics**
Select to apply second-order dynamics to acceleration readings.

**Natural frequency (rad/sec)**
The natural frequency of the accelerometer. The units of natural frequency are radians per second.

**Damping ratio**
The damping ratio of the accelerometer. A dimensionless parameter.

**Scale factors and cross-coupling**
The 3-by-3 matrix used to skew the accelerometer from body axes and to scale accelerations along body axes.

**Measurement bias**
The three-element vector containing long-term biases along the accelerometer axes. The units are in selected acceleration units.

**Update rate (sec)**
Specify the update rate of the accelerometer. An update rate of 0 will create a continuous accelerometer. If noise is selected and the update rate is 0, then the noise will be updated at the rate of 0.1. The units of update rate are seconds.

**Noise on**
Select to apply white noise to acceleration readings.

**Noise seeds**
The scalar seeds for the Gaussian noise generator for each axis of the accelerometer.

**Noise power**
The height of the PSD of the white noise for each axis of the accelerometer.

**Lower and upper output limits**
The six-element vector containing three minimum values and three maximum values of acceleration in each of the accelerometer axes. The units are in selected acceleration units.

## Inputs and Outputs

The first input is a three-element vector containing the actual accelerations in body-fixed axes, in selected units.

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

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

The fourth input is a three-element vector containing the location of the center of gravity, in selected units.

The optional fifth input is a three-element vector containing the gravity, in selected units.

The output is a three-element vector containing the measured accelerations from the accelerometer, in selected units.

## Assumptions and Limitations

Vibropendulous error and hysteresis effects are not accounted for in this block. Additionally, this block is not intended to model the internal dynamics of different forms of the instrument.

---

**Note** This block requires Control System Toolbox for discrete operation (nonzero sample time).

---

## Reference

Rogers, R. M., *Applied Mathematics in Integrated Navigation Systems*, AIAA Education Series, 2000.

## See Also

Three-Axis Gyroscope

Three-Axis Inertial Measurement Unit

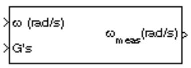
## Purpose

Implement three-axis gyroscope

## Library

GNC/Navigation

## Description



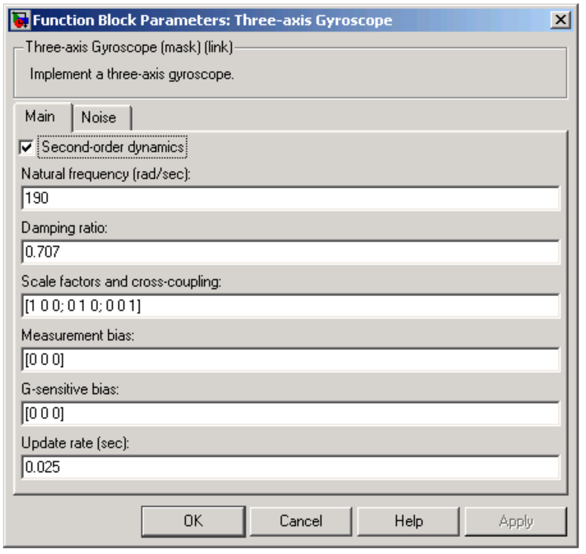

The Three-Axis Gyroscope block implements a gyroscope on each of the three axes. The measured body angular rates $(\underline{\omega}_{meas})$ include the body angular rates $(\underline{\omega}_b)$, errors, and optionally discretizations and nonlinearizations of the signals.

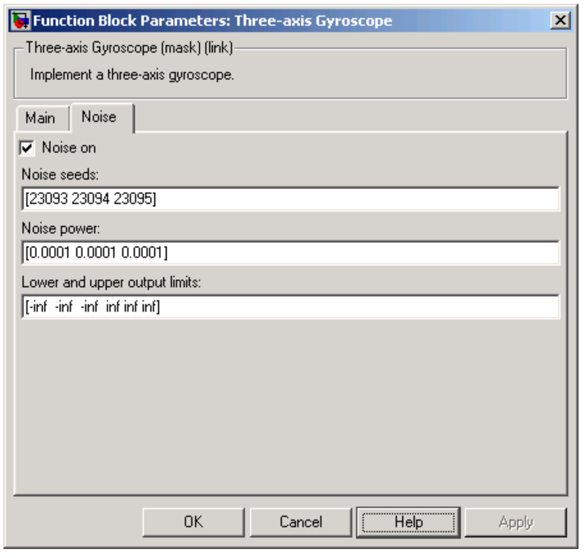
$$\underline{\omega}_{meas} = \underline{\omega}_b \cdot \underline{\omega}_{SFCC} + \underline{\omega}_{bias} + Gs \cdot \underline{\omega}_{gsens} + noise$$

where $\underline{\omega}_{SFCC}$ is a 3-by-3 matrix of scaling factors on the diagonal and misalignment terms in the nondiagonal, $\underline{\omega}_{bias}$ are the biases, $(Gs)$ are the Gs on the gyroscope, and $\underline{\omega}_{gsens}$ are the g-sensitive biases.

Optionally discretizations can be applied to the block inputs and dynamics along with nonlinearizations of the measured body angular rates via a Saturation block.

## Dialog Box


Function Block Parameters: Three-axis Gyroscope
Three-axis Gyroscope (mask) (link)
Implement a three-axis gyroscope.
Main
Noise
Second-order dynamics
Natural frequency (rad/sec):
190
Damping ratio:
0.707
Scale factors and cross-coupling:
[1 0 0; 0 1 0; 0 0 1]
Measurement bias:
[0 0 0]
G-sensitive bias:
[0 0 0]
Update rate (sec):
0.025
OK
Cancel
Help
Apply


Function Block Parameters: Three-axis Gyroscope
Three-axis Gyroscope (mask) (link)
Implement a three-axis gyroscope.
Main
Noise
Noise on
Noise seeds:
[23093 23094 23095]
Noise power:
[0.0001 0.0001 0.0001]
Lower and upper output limits:
[-inf -inf -inf inf inf inf]
OK
Cancel
Help
Apply

**Second order dynamics**
Select to apply second-order dynamics to gyroscope readings.

**Natural frequency (rad/sec)**
The natural frequency of the gyroscope. The units of natural frequency are radians per second.

**Damping ratio**
The damping ratio of the gyroscope. A dimensionless parameter.

**Scale factors and cross-coupling**
The 3-by-3 matrix used to skew the gyroscope from body axes and to scale angular rates along body axes.

**Measurement bias**
The three-element vector containing long-term biases along the gyroscope axes. The units are in radians per second.

**G-sensitive bias**
The three-element vector contains the maximum change in rates due to linear acceleration. The units are in radians per second per g-unit.

**Update rate (sec)**
Specify the update rate of the gyroscope. An update rate of 0 will create a continuous gyroscope. If noise is selected and the update rate is 0, then the noise will be updated at the rate of 0.1. The units of update rate are seconds.

**Noise on**
Select to apply white noise to gyroscope readings.

**Noise seeds**
The scalar seeds for the Gaussian noise generator for each axis of the gyroscope.

**Noise power**
The height of the PSD of the white noise for each axis of the gyroscope.

**Lower and upper output limits**
The six-element vector containing three minimum values and three maximum values of angular rates in each of the gyroscope axes. The units are in radians per second.

## Inputs and Outputs

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

The second input is a three-element vector containing the accelerations in body-fixed axes, in Gs.

The output is a three-element vector containing the measured angular rates from the gyroscope, in radians per second.

## Assumptions and Limitations

Anisoelastic bias and anisoinertial bias effects are not accounted for in this block. Additionally, this block is not intended to model the internal dynamics of different forms of the instrument.

---

**Note** This block requires Control System Toolbox for discrete operation (nonzero sample time).

---

## Reference

Rogers, R. M., *Applied Mathematics in Integrated Navigation Systems*, AIAA Education Series, 2000.

## See Also

Three-Axis Accelerometer

Three-Axis Inertial Measurement Unit

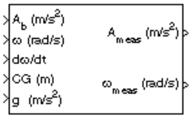
## Purpose

Implement three-axis inertial measurement unit (IMU)

## Library

GNC/Navigation

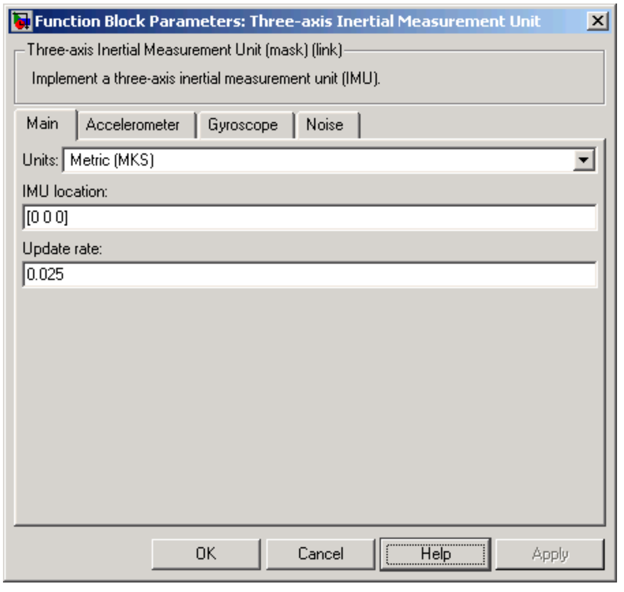
## Description

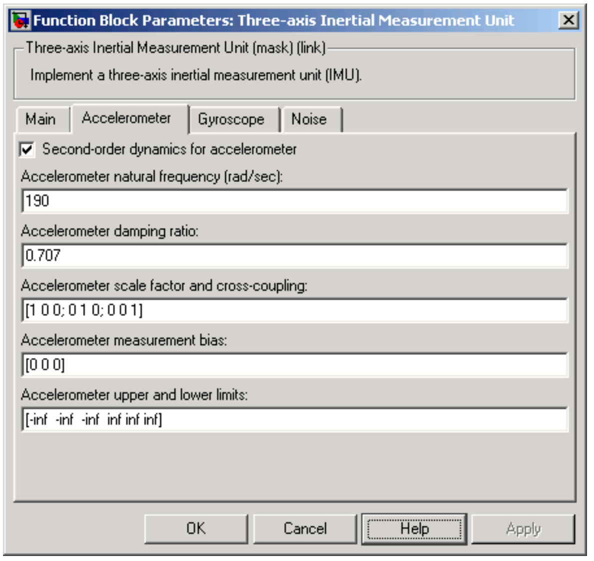


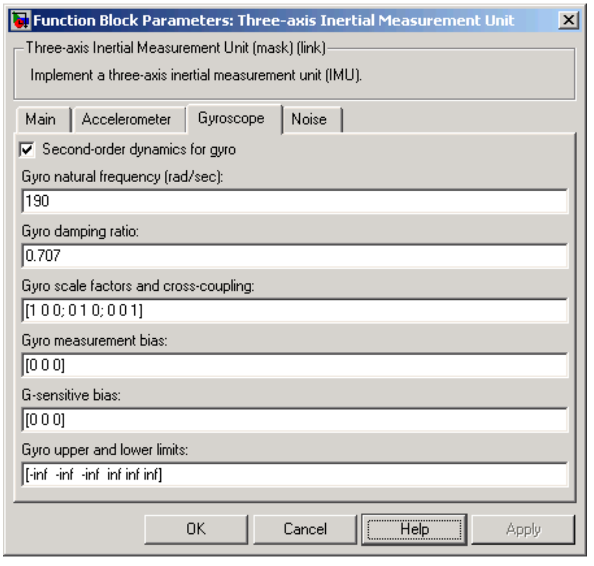

The Three-Axis Inertial Measurement Unit block implements an inertial measurement unit (IMU) containing a three-axis accelerometer and a three-axis gyroscope.

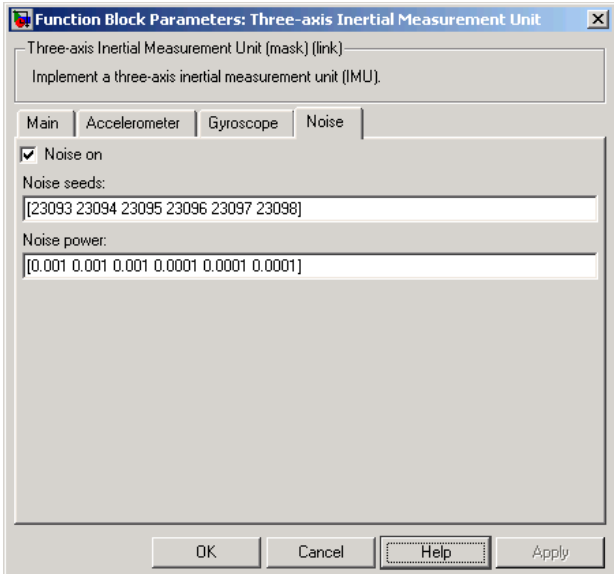
For a description of the equations and application of errors, see the Three-Axis Accelerometer block and the Three-Axis Gyroscope block reference pages.

## Dialog Box




Function Block Parameters: Three-axis Inertial Measurement Unit
Three-axis Inertial Measurement Unit (mask) (link)
Implement a three-axis inertial measurement unit (IMU).
Main
Accelerometer
Gyroscope
Noise
Second-order dynamics for accelerometer
Accelerometer natural frequency (rad/sec):
190
Accelerometer damping ratio:
0.707
Accelerometer scale factor and cross-coupling:
[1 0 0; 0 1 0; 0 0 1]
Accelerometer measurement bias:
[0 0 0]
Accelerometer upper and lower limits:
[-inf -inf -inf inf inf inf]
OK
Cancel
Help
Apply


Function Block Parameters: Three-axis Inertial Measurement Unit
Three-axis Inertial Measurement Unit (mask) (link)
Implement a three-axis inertial measurement unit (IMU).
Main
Accelerometer
Gyroscope
Noise
Second-order dynamics for gyro
Gyro natural frequency (rad/sec):
190
Gyro damping ratio:
0.707
Gyro scale factors and cross-coupling:
[1 0 0; 0 1 0; 0 0 1]
Gyro measurement bias:
[0 0 0]
G-sensitive bias:
[0 0 0]
Gyro upper and lower limits:
[-inf -inf -inf inf inf inf]
OK
Cancel
Help
Apply




**Units**

Specifies the input and output units:

| **Units** | **Acceleration** | **Length** |
|---|---|---|
| `Metric (MKS)` | Meters per second squared | Meters |
| `English` | Feet per second squared | Feet |

**IMU location**

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 selected length units.

**Update rate (sec)**

Specify the update rate of the accelerometer and gyroscope. An update rate of 0 will create a continuous accelerometer and continuous gyroscope. If noise is selected and the update rate is 0, then the noise will be updated at the rate of 0.1. The units of update rate are seconds.

**Second order dynamics for accelerometer**

Select to apply second-order dynamics to acceleration readings.

**Accelerometer natural frequency (rad/sec)**

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

**Accelerometer damping ratio**

The damping ratio of the accelerometer. A dimensionless parameter.

**Accelerometer scale factors and cross-coupling**

The 3-by-3 matrix used to skew the accelerometer from body-axis and to scale accelerations along body-axis.

**Accelerometer measurement bias**

The three-element vector containing long-term biases along the accelerometer axes. The units are in selected acceleration units.

**Accelerometer lower and upper output limits**

The six-element vector containing three minimum values and three maximum values of acceleration in each of the accelerometer axes. The units are in selected acceleration units.

**Gyro second order dynamics**

Select to apply second-order dynamics to gyroscope readings.

**Gyro natural frequency (rad/sec)**

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

**Gyro damping ratio**

The damping ratio of the gyroscope. A dimensionless parameter.

**Gyro scale factors and cross-coupling**
The 3-by-3 matrix used to skew the gyroscope from body axes and to scale angular rates along body axes.

**Gyro measurement bias**
The three-element vector containing long-term biases along the gyroscope axes. The units are in radians per second.

**G-sensitive bias**
The three-element vector contains the maximum change in rates due to linear acceleration. The units are in radians per second per g-unit.

**Gyro lower and upper output limits**
The six-element vector containing three minimum values and three maximum values of angular rates in each of the gyroscope axes. The units are in radians per second.

**Noise on**
Select to apply white noise to acceleration and gyroscope readings.

**Noise seeds**
The scalar seeds for the Gaussian noise generator for each axis of the accelerometer and gyroscope.

**Noise power**
The height of the PSD of the white noise for each axis of the accelerometer and gyroscope.

## Inputs and Outputs

The first input is a three-element vector containing the actual accelerations in body-fixed axes, in selected units.

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

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

The fourth input is a three-element vector containing the location of the center of gravity, in selected units.

The fifth input is a three-element vector containing the gravity, in selected units.

The first output is a three-element vector containing the measured accelerations from the accelerometer, in selected units.

The second output is a three-element vector containing the measured angular rates from the gyroscope, in radians per second.

## Assumptions and Limitations

Vibropendulous error, hysteresis affects, anisoelastic bias and anisoinertial bias are not accounted for in this block. Additionally, this block is not intended to model the internal dynamics of different forms of the instrument.

---

**Note** This block requires Control System Toolbox for discrete operation (nonzero sample time).

---

## Examples

See the `asbhl20` demo for an example of this block.

## Reference

Rogers, R. M., *Applied Mathematics in Integrated Navigation Systems*, AIAA Education Series, 2000.

## See Also

Three-Axis Accelerometer

Three-Axis Gyroscope

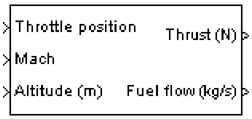
## Purpose

Implement first-order representation of turbofan engine with controller

## Library

Propulsion

## Description

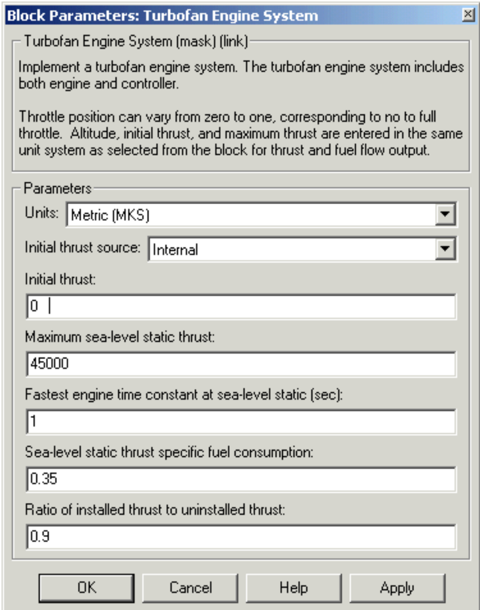



The Turbofan Engine System block computes the thrust and the weight of fuel flow of a turbofan engine and controller at a specific throttle position, Mach number, and altitude.

This system is represented by a first-order system with unitless heuristic lookup tables for thrust, thrust specific fuel consumption (TSFC), and engine time constant. For the lookup table data, thrust is a function of throttle position and Mach number, TSFC is a function of thrust and Mach number, and engine time constant is a function of thrust. The unitless lookup table outputs are corrected for altitude using the relative pressure ratio $\delta$ and relative temperature ratio $\theta$, and scaled by maximum sea level static thrust, fastest engine time constant at sea level static, sea level static thrust specific fuel consumption, and ratio of installed thrust to uninstalled thrust.

The Turbofan Engine System block icon displays the input and output units selected from the **Units** list.

## Dialog Box




**Units**

Specifies the input and output units:

| Units | Altitude | Thrust | Fuel Flow |
|---|---|---|---|
| `Metric (MKS)` | Meters | Newtons | Kilograms per second |
| `English` | Feet | Pound force | Pound mass per second |

**Initial thrust source**

Specifies the source of initial thrust:

| | |
|---|---|
| `Internal` | Use initial thrust value from mask dialog. |
| `External` | Use external input for initial thrust value. |

**Initial thrust**
Initial value for thrust.

**Maximum sea-level static thrust**
Maximum thrust at sea-level and at Mach = 0.

**Fastest engine time constant at sea-level static**
Fastest engine time at sea level.

**Sea-level static thrust specific fuel consumption**
Thrust specific fuel consumption at sea level, at Mach = 0, and at maximum thrust, in specified mass units per hour per specified thrust units.

**Ratio of installed thrust to uninstalled thrust**
Coefficient representing the loss in thrust due to engine installation.

## Inputs and Outputs

The first input is the throttle position. Throttle position can vary from zero to one, corresponding to no to full throttle.

The second input is the Mach number.

The third input is the altitude in specified length units.

The first output is the thrust in specified force units.

The second output is the fuel flow in specified mass units per second.

## Assumptions and 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.

## References

*Aeronautical Vestpocket Handbook*, United Technologies Pratt & Whitney, August, 1986.

Raymer, D. P., *Aircraft Design: A Conceptual Approach*, AIAA Education Series, Washington, DC, 1989.

Hill, P. G., and C. R. Peterson, *Mechanics and Thermodynamics of Propulsion,* Addison-Wesley Publishing Company, Reading, Massachusetts, 1970.

## Purpose

Convert from velocity units to desired velocity units

## Library

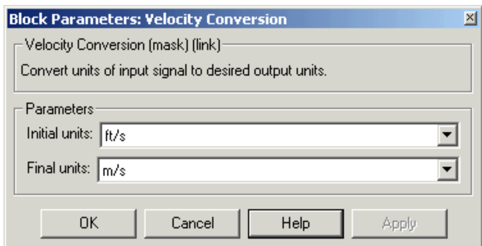
Utilities/Unit Conversions

## 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 icon displays the input and output units selected from the **Initial units** and the **Final units** lists.

## Dialog Box




**Initial units**

Specifies the input units.

**Final units**

Specifies the output units.

The following conversion units are available:

| | |
|---|---|
| `m/s` | Meters per second |
| `ft/s` | Feet per second |
| `km/s` | Kilometers per second |
| `in/s` | Inches per second |
| `km/h` | Kilometers per hour |
| `mph` | Miles per hour |

| | |
|---|---|
| `kts` | Nautical miles per hour |
| `ft/min` | Feet per minute |

**Inputs and Outputs**

The input is the velocity in initial velocity units.

The output is the velocity in final velocity units.

**See Also**

Acceleration Conversion

Angle Conversion

Angular Acceleration Conversion

Angular Velocity Conversion

Density Conversion

Force Conversion

Length Conversion

Mass Conversion

Pressure Conversion

Temperature Conversion

# Von Karman Wind Turbulence Model (Continuous)

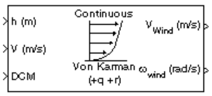
## Purpose

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 Ω radians per meter, the circular frequency ω is calculated by multiplying V by Ω. The following table displays the component spectra functions:

| | MIL-F-8785C | MIL-HDBK-1797 |
|---|---|---|
| **Longitudinal** | | |
| $\Phi_u(\omega)$ | $\frac{2\sigma_u^2 L_u}{\pi V} \cdot \frac{1}{[1+(1.339 L_u \frac{\omega}{V})^2]^{5/6}}$ | $\frac{2\sigma_u^2 L_u}{\pi V} \cdot \frac{1}{[1+(1.339 L_u \frac{\omega}{V})^2]^{5/6}}$ |
| $\Phi_p(\omega)$ | $\frac{\sigma_w^2}{V L_w} \cdot \frac{0.8\left(\frac{\pi L_w}{4b}\right)^{\frac{1}{3}}}{1+\left(\frac{4b\,\omega}{\pi V}\right)^2}$ | $\frac{\sigma_w^2}{2 V L_w} \cdot \frac{0.8\left(\frac{2\pi L_w}{4b}\right)^{\frac{1}{3}}}{1+\left(\frac{4b\,w}{\pi V}\right)^2}$ |
| **Lateral** | | |
| $\Phi_v(\omega)$ | $\frac{\sigma_v^2 L_v}{\pi V} \cdot \frac{1+\frac{8}{3}(1.339 L_v \frac{\omega}{V})^2}{[1+(1.339 L_v \frac{\omega}{V})^2]^{11/6}}$ | $\frac{2\sigma_v^2 L_v}{\pi V} \cdot \frac{1+\frac{8}{3}(2.678 L_v \frac{\omega}{V})^2}{[1+(2.678 L_v \frac{\omega}{V})^2]^{11/6}}$ |

| | MIL-F-8785C | MIL-HDBK-1797 |
|---|---|---|
| $\Phi_r(\omega)$ | $\dfrac{\mp\left(\frac{\omega}{V}\right)^2}{1+\left(\frac{3b\omega}{\pi V}\right)^2}\cdot\Phi_v(\omega)$ | $\dfrac{\mp\left(\frac{\omega}{V}\right)^2}{1+\left(\frac{3b\omega}{\pi V}\right)^2}\cdot\Phi_v(\omega)$ |
| **Vertical** | | |
| $\Phi_w(\omega)$ | $\dfrac{\sigma_w^2 L_w}{\pi V}\cdot\dfrac{1+\frac{8}{3}(1.339L_w\frac{\omega}{V})^2}{[1+(1.339L_w\frac{\omega}{V})^2]^{11/6}}$ | $\dfrac{2\sigma_w^2 L_w}{\pi V}\cdot\dfrac{1+\frac{8}{3}(2.678L_w\frac{\omega}{V})^2}{[1+(2.678L_w\frac{\omega}{V})^2]^{11/6}}$ |
| $\Phi_q(\omega)$ | $\dfrac{\pm\left(\frac{\omega}{V}\right)^2}{1+\left(\frac{4b\omega}{\pi V}\right)^2}\cdot\Phi_w(\omega)$ | $\dfrac{\pm\left(\frac{\omega}{V}\right)^2}{1+\left(\frac{4b\omega}{\pi V}\right)^2}\cdot\Phi_w(\omega)$ |

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:

| | | |
|---|---|---|
| $p_g = \dfrac{\partial w_g}{\partial y}$ | $q_g = \dfrac{\partial w_g}{\partial x}$ | $r_g = -\dfrac{\partial v_g}{\partial x}$ |
| $p_g = \dfrac{\partial w_g}{\partial y}$ | $q_g = \dfrac{\partial w_g}{\partial x}$ | $r_g = \dfrac{\partial v_g}{\partial x}$ |
| $p_g = -\dfrac{\partial w_g}{\partial y}$ | $q_g = -\dfrac{\partial w_g}{\partial x}$ | $r_g = \dfrac{\partial v_g}{\partial x}$ |

The variations affect only the vertical ($q_g$) and lateral ($r_q$) 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 |
|---|---|
| **Longitudinal** | |
| $H_u(s)$ | $\dfrac{\sigma_u\sqrt{\frac{2}{\pi}\cdot\frac{L_u}{V}}\left(1+0.25\frac{L_u}{V}s\right)}{1+1.357\frac{L_u}{V}s+0.1987\left(\frac{L_u}{V}\right)^2 s^2}$ |

| | MIL-F-8785C |
|---|---|
| $H_p(s)$ | $\sigma_w \sqrt{\frac{0.8}{V}} \frac{\left(\frac{\pi}{(4b)}\right)^{1/6}}{L_w^{1/3}\left(1+\left(\frac{4b}{\pi V}\right)s\right)}$ |
| **Lateral** | |
| $H_v(s)$ | $\frac{\sigma_v \sqrt{\frac{1}{\pi} \cdot \frac{L_v}{V}}\left(1+2.7478\frac{L_v}{V}s+0.3398\left(\frac{L_v}{V}\right)^2 s^2\right)}{1+2.9958\frac{L_v}{V}s+1.9754\left(\frac{L_v}{V}\right)^2 s^2+0.1539\left(\frac{L_v}{V}\right)^3 s^3}$ |
| $H_r(s)$ | $\frac{\mp\frac{s}{V}}{\left(1+\left(\frac{3b}{\pi V}\right)s\right)} \cdot H_v(s)$ |
| **Vertical** | |
| $H_w(s)$ | $\frac{\sigma_w \sqrt{\frac{1}{\pi} \cdot \frac{L_w}{V}}\left(1+2.7478\frac{L_w}{V}s+0.3398\left(\frac{L_w}{V}\right)^2 s^2\right)}{1+2.9958\frac{L_w}{V}s+1.9754\left(\frac{L_w}{V}\right)^2 s^2+0.1539\left(\frac{L_w}{V}\right)^3 s^3}$ |
| $H_q(s)$ | $\frac{\pm\frac{s}{V}}{\left(1+\left(\frac{4b}{\pi V}\right)s\right)} \cdot H_w(s)$ |

| | MIL-HDBK-1797 |
|---|---|
| **Longitudinal** | |
| $H_u(s)$ | $\frac{\sigma_u \sqrt{\frac{2}{\pi} \cdot \frac{L_u}{V}}\left(1+0.25\frac{L_u}{V}s\right)}{1+1.357\frac{L_u}{V}s+0.1987\left(\frac{L_u}{V}\right)^2 s^2}$ |

| | MIL-HDBK-1797 |
|---|---|
| $H_p(s)$ | $\sigma_w\sqrt{\frac{0.8}{V}}\frac{\left(\frac{\pi}{(4b)}\right)^{1/6}}{(2L_w)^{1/3}\left(1+\left(\frac{4b}{\pi V}\right)s\right)}$ |
| **Lateral** | |
| $H_v(s)$ | $\frac{\sigma_v\sqrt{\frac{1}{\pi}\cdot\frac{2L_v}{V}}\left(1+2.7478\frac{2L_v}{V}s+0.3398\left(\frac{2L_v}{V}\right)^2s^2\right)}{1+2.9958\frac{2L_v}{V}s+1.9754\left(\frac{2L_v}{V}\right)^2s^2+0.1539\left(\frac{2L_v}{V}\right)^3s^3}$ |
| $H_r(s)$ | $\frac{\mp\frac{s}{V}}{\left(1+\left(\frac{3b}{\pi V}\right)s\right)}\cdot H_v(s)$ |
| **Vertical** | |
| $H_w(s)$ | $\frac{\sigma_w\sqrt{\frac{1}{\pi}\cdot\frac{2L_w}{V}}\left(1+2.7478\frac{2L_w}{V}s+0.3398\left(\frac{2L_w}{V}\right)^2s^2\right)}{1+2.9958\frac{2L_w}{V}s+1.9754\left(\frac{2L_w}{V}\right)^2s^2+0.1539\left(\frac{2L_w}{V}\right)^3s^3}$ |
| $H_q(s)$ | $\frac{\pm\frac{s}{V}}{\left(1+\left(\frac{4b}{\pi V}\right)s\right)}\cdot H_w(s)$ |

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:

| MIL-F-8785C | MIL-HDBK-1797 |
|---|---|
| $L_w = h$<br>$L_u = L_v = \frac{h}{(0.177 + 0.000823h)^{1.2}}$ | $2L_w = h$<br>$L_u = 2L_v = \frac{h}{(0.177 + 0.000823h)^{1.2}}$ |

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.

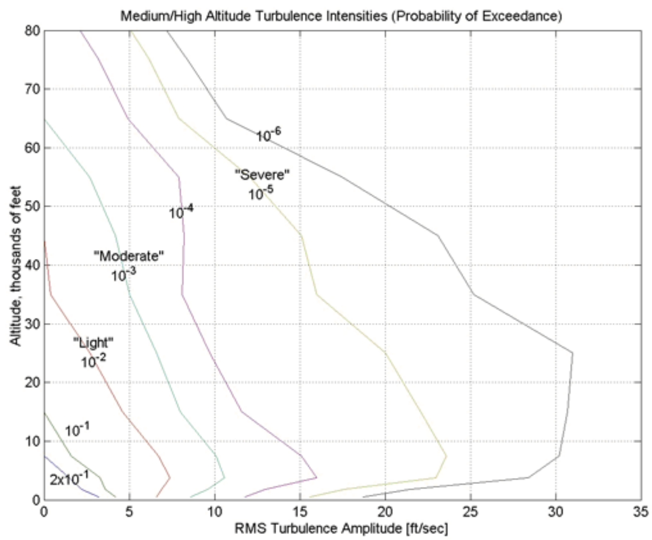
### Medium/High Altitudes (Altitude > 2000 feet)

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 |
|---|---|
| $L_u = L_v = L_w = 2500\ ft$ | $L_u = 2L_v = 2L_w = 2500\ ft$ |

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: $\sigma_u = \sigma_v = \sigma_w$.

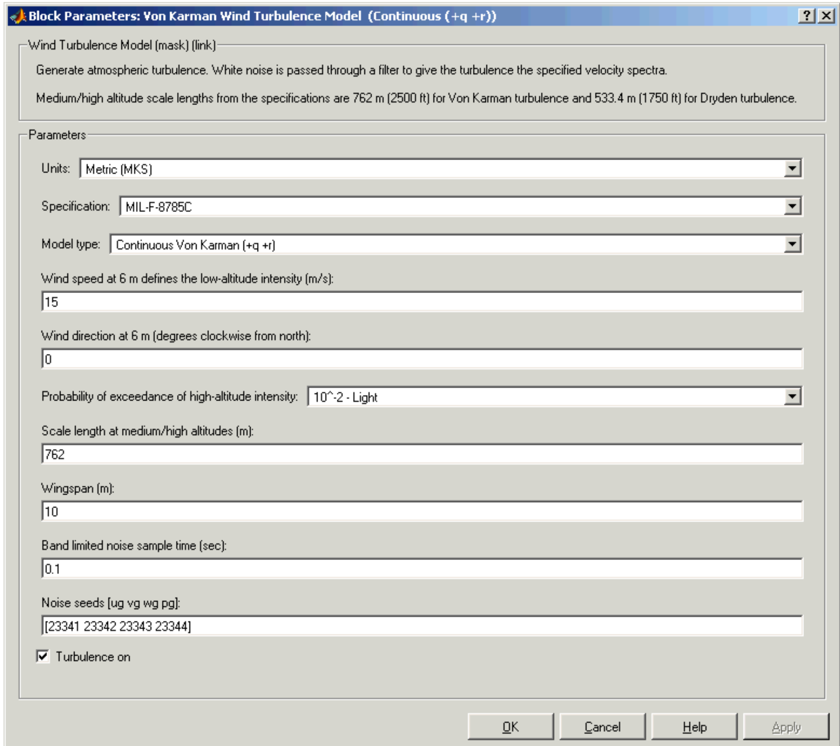
The turbulence axes orientation in this region is defined as being aligned with the body coordinates:




### 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.

## Dialog Box




**Units**

Define the units of wind speed due to the turbulence.

| Units | Wind Velocity | Altitude | Air Speed |
|---|---|---|---|
| `Metric (MKS)` | Meters/second | Meters | Meters/second |

| Units | Wind Velocity | Altitude | Air Speed |
|---|---|---|---|
| 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

The first input is the altitude in units selected.

The second input is the aircraft speed in units selected.

The third input is a direction cosine matrix.

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

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

## 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.

Hoblit, 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

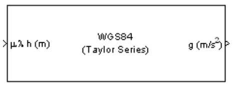
## Purpose

Implement 1984 World Geodetic System (WGS84) representation of Earth's gravity

## Library

Environment/Gravity

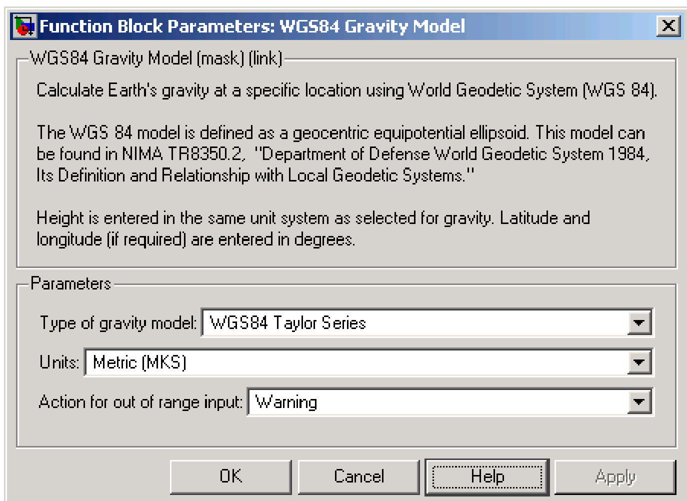
## Description




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.

## Dialog Box




**Type of gravity model**

Specifies the method to calculate gravity:

- `WGS84 Taylor Series`
- `WGS84 Close Approximation`

- `WGS84 Exact`

**Units**

Specifies the input and output units:

| 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**. 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 **Precessing 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 **Precessing 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 **Precessing 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 input | Three-element vector | Contains the position in geodetic latitude, longitude and altitude, with units in degrees, degrees, and selected units of length respectively. |
| Second input (Optional) | Scalar | Contains the user-specified Julian centuries. |

| Output | Dimension Type | Description |
|---|---|---|
| Output | Vector | Applies to gravity in the north-east-down (NED) coordinate system. The Exact method should output 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.

Use of the WGS84 Close Approximation model should be limited to a geodetic height of 20,000.0 m (approximately 65,620.0 feet). Below this height, it gives results with submicrogal precision.

## Examples

See the Airframe subsystem in 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."

# Wind Angles to Direction Cosine Matrix

## Purpose

Convert wind angles to direction cosine matrix

## Library

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 $(ox_3, oy_3, oz_3)$. 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 $(ox_3, oy_3, oz_3)$

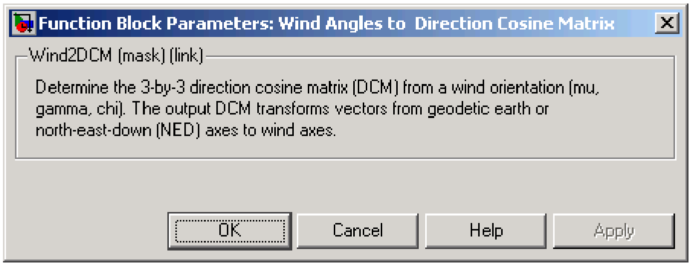
$$\begin{bmatrix} ox_3 \\ oy_3 \\ oz_3 \end{bmatrix} = DCM_{we} \begin{bmatrix} ox_0 \\ oy_0 \\ oz_0 \end{bmatrix}$$

$$\begin{bmatrix} ox_3 \\ oy_3 \\ oz_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\mu & \sin\mu \\ 0 & -\sin\mu & \cos\mu \end{bmatrix} \begin{bmatrix} \cos\gamma & 0 & -\sin\gamma \\ 0 & 1 & 0 \\ \sin\gamma & 0 & \cos\gamma \end{bmatrix} \begin{bmatrix} \cos\chi & \sin\chi & 0 \\ -\sin\chi & \cos\chi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} ox_0 \\ oy_0 \\ oz_0 \end{bmatrix}$$

Combining the three axis transformation matrices defines the following DCM.

$$DCM_{we} = \begin{bmatrix} \cos\gamma\cos\chi & \cos\gamma\sin\chi & -\sin\gamma \\ (\sin\mu\sin\gamma\cos\chi - \cos\mu\sin\chi) & (\sin\mu\sin\gamma\sin\chi + \cos\mu\cos\chi) & \sin\mu\cos\gamma \\ (\cos\mu\sin\gamma\cos\chi + \sin\mu\sin\chi) & (\cos\mu\sin\gamma\sin\chi - \sin\mu\cos\chi) & \cos\mu\cos\gamma \end{bmatrix}$$

## Dialog Box




## Inputs and Outputs

The input is a 3-by-1 vector of wind angles, in radians.

The output is a 3-by-3 direction cosine matrix which transforms earth vectors to wind vectors.

## Assumptions and Limitations

This implementation generates a flight path angle that lies between $\pm 90$ degrees, and bank and heading angles that lie between $\pm 180$ degrees.

## See Also

Direction Cosine Matrix Body to Wind

Direction Cosine Matrix to Euler Angles

Direction Cosine Matrix to Wind Angles

Euler Angles to Direction Cosine Matrix

# Wind Angular Rates

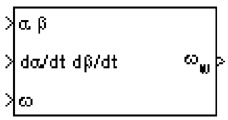
## Purpose

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

Flight Parameters

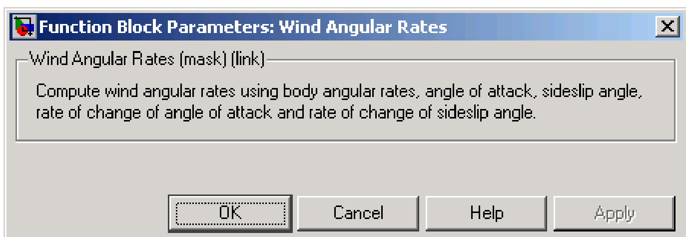
## Description




The Wind Angular Rates block supports the equations of motion in wind-fixed frame models by calculating the wind-fixed angular rates $(p_w, q_w, r_w)$. 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 rate by the following equation.

$$\begin{bmatrix} p_w \\ q_w \\ r_w \end{bmatrix} = \begin{bmatrix} \cos\alpha\cos\beta & \sin\beta & \sin\alpha\cos\beta \\ -\cos\alpha\sin\beta & \cos\beta & -\sin\alpha\sin\beta \\ -\sin\alpha & 0 & \cos\alpha \end{bmatrix} \begin{bmatrix} p_b - \dot{\beta}\sin\alpha \\ q_b - \dot{\alpha} \\ r_b + \dot{\beta}\cos\alpha \end{bmatrix}$$

## Dialog Box




## Inputs and Outputs

The first input is the 2-by-1 vector containing angle of attack and sideslip, in radians.

The second input is the 2-by-1 vector containing rate of change of angle of attack and rate of change of sideslip, in radians per second.

The third input is the body angular rates, in radians per second.

The output is the wind angular rates, in radians per second.

## 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)

# Wind Shear Model

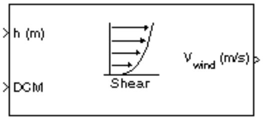
## Purpose

Calculate wind shear conditions

## Library

Environment/Wind

## Description



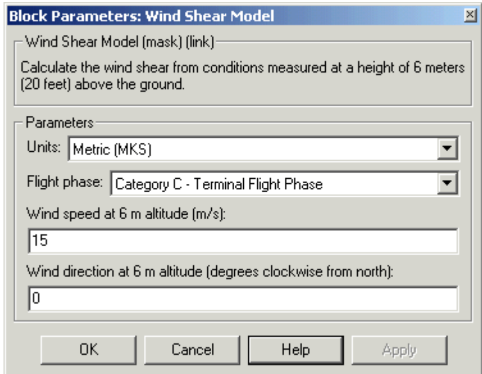

The Wind Shear Model block adds wind shear to the aerospace model. This implementation is based on the mathematical representation in the Military Specification MIL-F-8785C [1]. The magnitude of the wind shear is given by the following equation for the mean wind profile as a function of altitude and the measured wind speed at 20 feet (6 m) above the ground.

$$u_w = W_{20} \frac{\ln\left(\frac{h}{z_0}\right)}{\ln\left(\frac{20}{z_0}\right)}, \quad 3ft < h < 1000ft$$

where $u_w$ is the mean wind speed, $W_{20}$ is the measured wind speed at an altitude of 20 feet, $h$ is the altitude, and $z_0$ is a constant equal to 0.15 feet for Category C flight phases and 2.0 feet for all other flight phases. Category C flight phases are defined in reference [1] to be terminal flight phases, which include takeoff, approach, and landing.

The resultant mean wind speed in the Earth-fixed axis frame is changed to body-fixed axis coordinates by multiplying by the direction cosine matrix (DCM) input to the block. The block output is the mean wind speed in the body-fixed axis.

## Dialog Box




**Units**

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

The first input is the altitude in units selected.

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

The output is a 3-by-1 vector of the mean wind speed in the body axes frame, in the selected units.

## Examples

See the Airframe subsystem in the `aeroblk_HL20` model 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)

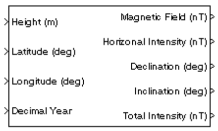
## Purpose

Calculate Earth's magnetic field at specific location and time using World Magnetic Model 2000 (WMM2000)

## Library

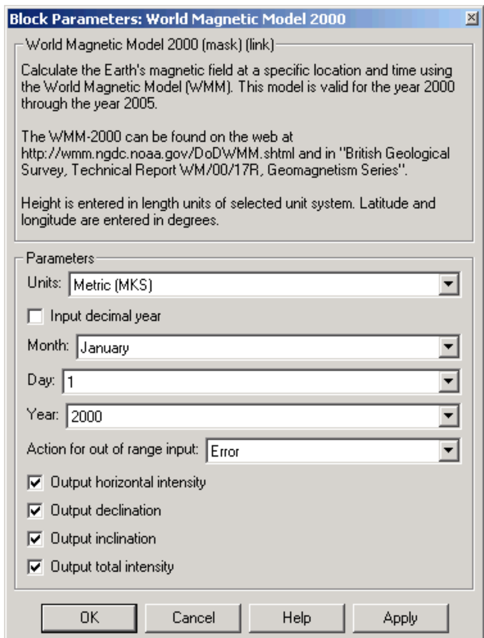
Environment/Gravity

## 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.

## Dialog Box




### 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

The first input is the height, in selected units.

The second input is the latitude in degrees.

The third input is the longitude in degrees.

The fourth optional input is 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, 2005:

```
%%%BEGIN CODE%%%
year = '2005';
year_selected = str2num(year);
month = 'March';
day = '21';

if (mod(year_selected,400)&&~mod(year_selected,100))
% leapyear = false;
ndays = 365;
elseif ~mod(year_selected,4)
% leapyear = true;
ndays = 366;
else
% leapyear = false;
ndays = 365;
end

day_of_year = datenum([day '-' month '-'
year])-datenum(['1-january-' year]);
dyear = year_selected + day_of_year/ndays;
%%%END CODE%%%
```

The first output is the magnetic field vector in selected units.

The second optional output is the horizontal intensity in selected units.

The third optional output is the declination in degrees.

The fourth optional output is the inclination in degrees.

The fifth optional output is 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

Macmillian, S. and J. M. Quinn, 2000. "The Derivation of the World Magnetic Model 2000," *British Geological Survey Technical Report* WM/00/17R.

`http://www.ngdc.noaa.gov/seg/WMM/DoDWMM.shtml`

## See Also

World Magnetic Model 2005

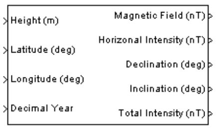
## Purpose

Calculate Earth's magnetic field at specific location and time using World Magnetic Model 2005 (WMM2005)

## Library

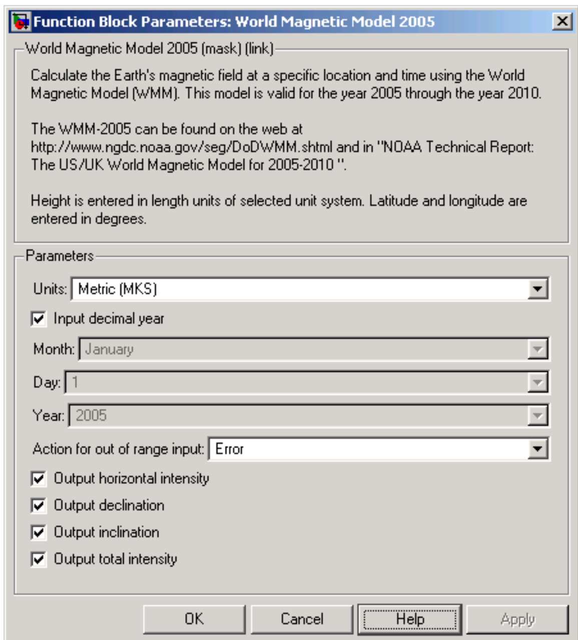
Environment/Gravity

## 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.

## Dialog Box



**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

The first input is the height, in selected units.

The second input is the latitude in degrees.

The third input is the longitude in degrees.

The fourth optional input is 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, 2005:

```
%%%BEGIN CODE%%%
year = '2005';
year_selected = str2num(year);
month = 'March';
day = '21';

if (mod(year_selected,400)&&~mod(year_selected,100))
% leapyear = false;
ndays = 365;
elseif ~mod(year_selected,4)
% leapyear = true;
ndays = 366;
else
% leapyear = false;
ndays = 365;
end

day_of_year = datenum([day '-' month '-'
year])-datenum(['1-january-' year]);
dyear = year_selected + day_of_year/ndays;
%%%END CODE%%%
```

The first output is the magnetic field vector in selected units.

The second optional output is the horizontal intensity in selected units.

The third optional output is the declination in degrees.

The fourth optional output is the inclination in degrees.

The fifth optional output is 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

`http://www.ngdc.noaa.gov/seg/WMM/DoDWMM.shtml`

## See Also

World Magnetic Model 2000

# Aerospace Units

The main blocks of Aerospace Blockset support standard measurement systems. The Unit Conversion blocks support all units listed in this table.

| Quantity | Metric (MKS) | English |
|---|---|---|
| Acceleration | meters/second$^2$ (m/s$^2$), kilometers/second$^2$ (km/s$^2$), (kilometers/hour)/second (km/h-s), g-unit (*g*) | inches/second$^2$ (in/s$^2$), feet/second$^2$ (ft/s$^2$), (miles/hour)/second (mph/s), g-unit (*g*) |
| Angle | radian (rad), degree (deg), revolution | radian (rad), degree (deg), revolution |
| Angular acceleration | radians/second$^2$ (rad/s$^2$), degrees/second$^2$ (deg/s$^2$), revolutions/minute (rpm), revolutions/second (rps) | radians/second$^2$ (rad/s$^2$), degrees/second$^2$ (deg/s$^2$), 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$^3$ (kg/m$^3$) | pound mass/foot$^3$ (lbm/ft$^3$), slug/foot$^3$ (slug/ft$^3$), pound mass/inch$^3$ (lbm/in$^3$) |
| Force | newton (N) | pound (lb) |
| Inertia | kilogram-meter$^2$ (kg-m$^2$) | slug-foot$^2$ (slug-ft$^2$) |
| Length | meter (m) | inch (in), foot (ft), mile (mi), nautical mile (nm) |
| Mass | kilogram (kg) | slug (slug), pound mass (lbm) |

| Quantity | Metric (MKS) | English |
|---|---|---|
| Pressure | Pascal (Pa) | pound/inch$^2$ (psi), pound/foot$^2$ (psf), atmosphere (atm) |
| Temperature | kelvin (K), degrees Celsius (ºC) | degrees Fahrenheit (ºF), degrees Rankine (ºR) |
| 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 |

# Index

## D



## E



## F



## G