Multi-Sim

Multi-Purpose, Engineering Flight Simulation

User's Manual

For Multi-Sim Demo Version v1.0D

Radical Novelties, Colorado, USA

Table of Contents

Introduction

The Multi-Sim flight simulation was created as an engineering tool. Its purpose is to provide a flexible, easy to use surrogate for a real aerospace vehicle, like an airplane, missile, rocket or even a blimp or dirigible. The simulation is designed to be controlled over a network interface, simply by sending it formatted, binary commands. The network interface allows any other program to control both the simulation and the aircraft that it models. The standardized interface makes the simulation available for a multitude of uses, from simple flight dynamics studies to the core of a manned flight simulator. It's just a matter of what kind of program you write to connect to it. The first version of the simulation was written in 1989 to support the development of an embedded, autonomous control system for UAVs.

The simulation is called Multi-Sim because it is capable of simulating any aerospace vehicle, provided that the vehicle's aerodynamic and mass-properties data are known. The simulation allows the user to select and load different vehicle models. Radical Novelties can provide some models for the user or the user can provide his own. (The demo version of the system is hard-wired to model only a single vehicle, the Cessna 172.) It's also called Multi-Sim because the system was written to operate *extremely* fast. On a 2.13 GHz laptop with a Pentium P6200 processor (running on a single core) it runs at almost 2,500 times real-time. 5 minutes of flight can be simulated in only *one-eighth of a second*. This means that a user can run many copies of the simulation simultaneously on a single computer. If you are simulating multiple aircraft—simulating real, congested airspace or testing the control of swarms of vehicles, for example—you won't need extra hardware. (The demo version only runs as a single instance.) Multiple vehicles, multiple copies, a multitude of uses.

The simulation's speed is also central to keeping it out of your way. If you're using the simulation to develop a control system, for example, the simulation's fast response and low resource consumption allows *your* system the time it needs to do *its* processing. Multi-Sim runs so fast, in fact, that there is no problem with running your control system on the same computer. On the other hand, because Multi-Sim is designed to operate over a network interface, you can also run the simulation on one computer and your control system on another. There is no difference. It also makes no difference whether your program runs on a desktop or embedded system. The simulation's original purpose was to be connected to an embedded control system.

The simulation's software was written in such a manner that the authors can easily add new dynamic component models. Along with modeling the bare-airframe dynamics of the flight vehicle, it can model any sensor or actuator on the vehicle, including their representative errors. (The demo version simulates only bare-airframe responses. It contains no sensor or actuator models.) It can also model the motions and effects of auxiliary systems, like gimbals, landing gear, speed brakes, etc. The simulation's software was designed to be modular. And so adding or deleting models, or even changing the physics of the models, is straight-forward. Custom versions of the simulation are our specialty. If an off-the-shelf version isn't sufficient for your work, please call us and we can arrange to make a version that matches your vehicle in every detail.

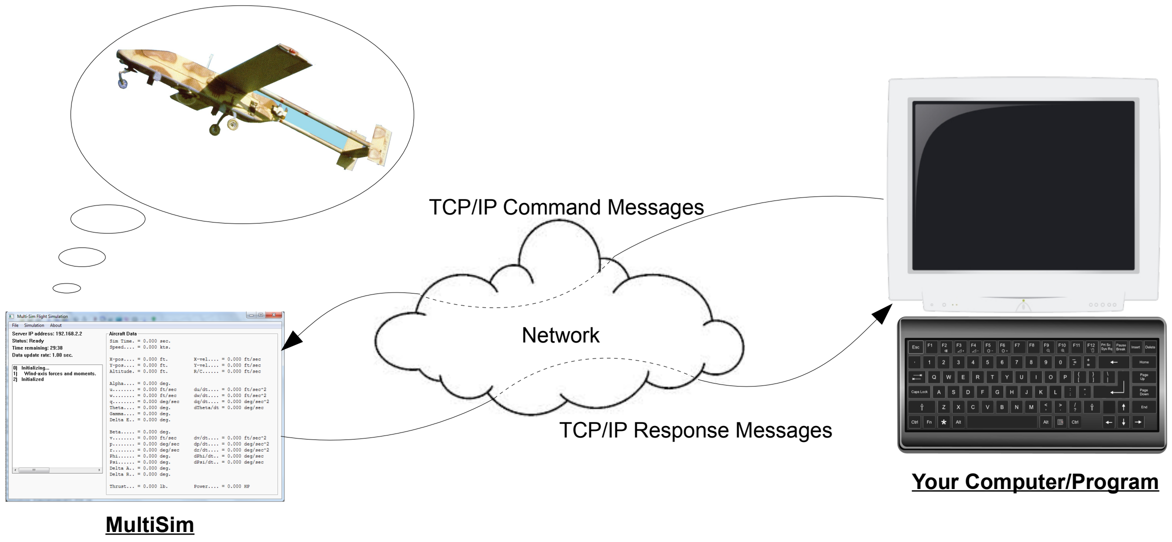
We find the system to be very useful and, when there is time to play, even fun. We hope you will too.

System Overview

While much of the general, high-level information provided in the following parts of this manual applies to all versions of the simulation, the details are specific to the demo version. If you are using a non-demo version of the system, please refer to its specific manual.

The simulation does its work by first calculating the forces and moments on the vehicle. Next, it applies those forces and moments to the six-degree-of-freedom, rigid-body equations of motion. Those equations yield accelerations of the body in space. The simulation integrates those accelerations into velocities and the velocities into displacements using a very accurate integrator. Once these steps are accomplished, the motion of the vehicle has been calculated--advanced--a short time into the future. The simulation can then repeat the process, calculating updated forces and moments with the vehicle in its new orientation, location and speed, and then following through by calculating the vehicle's updated motion. It does this over and over, time-marching the motion of the aircraft as far into the future as we like.

The cyclic nature of the calculations provides a mechanism for another program to control the simulated vehicle. It simply needs to tell the simulation how to move the control surfaces and throttle of the vehicle at the beginning of each calculation cycle. The simulation provides for this by operating as an Internet server. Any program or device can connect to the simulation as a client, as shown in Figure 1.



```
Server IP address: 192.168.2.2
Status: Ready
Time remaining: 29:38
Data update rate: 1.00 sec.

0]  Initializing...
1]    Wind-axis forces and moments.
2]  Initialized

Aircraft Data
Sim Time. = 0.000 sec.
Speed.... = 0.000 kts.

X-pos.... = 0.000 ft.         X-vel.... = 0.000 ft/sec
Y-pos.... = 0.000 ft.         Y-vel.... = 0.000 ft/sec
Altitude. = 0.000 ft.         R/C...... = 0.000 ft/sec

Alpha.... = 0.000 deg.
u........ = 0.000 ft/sec      du/dt.... = 0.000 ft/sec^2
w........ = 0.000 ft/sec      dw/dt.... = 0.000 ft/sec^2
q........ = 0.000 deg/sec     dq/dt.... = 0.000 deg/sec^2
Theta.... = 0.000 deg.        dTheta/dt = 0.000 deg/sec
Gamma.... = 0.000 deg.
Delta E.. = 0.000 deg.

Beta..... = 0.000 deg.
v........ = 0.000 ft/sec      dv/dt.... = 0.000 ft/sec^2
p........ = 0.000 deg/sec     dp/dt.... = 0.000 deg/sec^2
r........ = 0.000 deg/sec     dr/dt.... = 0.000 deg/sec^2
Phi...... = 0.000 deg.        dPhi/dt.. = 0.000 deg/sec
Psi...... = 0.000 deg.        dPsi/dt.. = 0.000 deg/sec
Delta A.. = 0.000 deg.
Delta R.. = 0.000 deg.

Thrust... = 0.000 lb.         Power.... = 0.000 HP
```

Figure 1: The simulation client/server paradigm

The simulation has a small set of commands that it can execute. When a client program connects, it can send any of those commands, along with the necessary data, to the simulation in a TCP/IP message. After executing the command, the simulation sends a TCP/IP response message, with the pertinent data, back to the client. In this way, the simulation works in lockstep with your program, advancing the aircraft in time using your current control inputs.

The simulation is able to advance a specified amount of time into the future. And so your program can command the simulation to advance small increments in time when fine input or output resolution is needed, or it can command it to advance large increments in time when no control input is needed.

In many flight simulation scenarios, it's often convenient to begin simulating an aircraft in the air, already flying, rather than having it take off from the ground and then fly to a prescribed point in space. To make this possible, the simulation provides a “trim” command. An aircraft flying straight and level does so with a unique combination of angle of attack, airspeed, control surface deflections and thrust or power. The trim command takes angle of attack, airspeed or elevator deflection and finds the other trim settings required to ensure that the airplane will be flying straight and level when the simulation is started. The demo version of the simulation has no ground reaction model, and so each flight must begin in the air.

The demo version of the simulation executes three commands. The first, the “hello” command, is a good test to see whether a client is able to communicate with the simulation. Upon receipt of the hello command, the simulation responds by sending a message containing the time and date of the computer on which it is running. The second command, “trim”, tells the simulation to trim the aircraft by an angle of attack, airspeed or elevator deflection strategy. The simulation runs its trim routines, then sends a message containing all of the trim settings back to the client. The last command, “advance”, sends the current control settings to the simulation, along with the amount of time to march into the future. The simulation advances the aircraft model the prescribed amount, then returns a full set of aircraft data back to the client.

We refer to Multi-Sim client programs as “cabs”. Like the cab of a truck or a train, they are where the controls reside.

Demo Version Features

- 6 degree-of-freedom, non-linear dynamic model
- Simulates the Cessna 172
- “Flat Earth” spatial model
- International Standard Atmosphere model
- No ground reaction model
- No wind model
- No sensor or actuator models
- Single program instance
- 30 minute time limit

Axis Systems and Physical Values

The simulation uses a collection of axis systems internally to do its work. It's critically important that a user understand these axis systems before writing software to control the simulated aircraft. Here, we'll discuss these axis systems, as well as the physical values that the simulation returns to describe the aircraft's current state.

Multi-Sim calculates the aerodynamic and body-forces (like thrust, gravity, buoyancy, ground loads, etc.) on the vehicle using a set of "body-fixed" axes. These axes have their origin at the vehicle's center of mass and their orientation is fixed to the geometry of the body. (See Figure 2.) The x-axis points forward, out the nose of the aircraft, the y-axis points out the right wing, and the z-axis points down, out of the aircraft's belly. These axes remain fixed to the aircraft, rotating and translating with it as it moves through space. Note that the labels of these axes are lower-case. The aircraft is acted on by many forces and moments, both aerodynamic and inertial. These forces and moments rarely align with the body-fixed axes. So the simulation sums the individual forces and moments and then finds the *components* of those values down each of the body-fixed axes. Once it has completed that process, it can calculate the linear and angular accelerations of the vehicle. When a cab commands the simulation to calculate forward in time using the "advance" command, the simulation returns a full set of vehicle data. Some of the values are measured along and about the body-fixed axes:

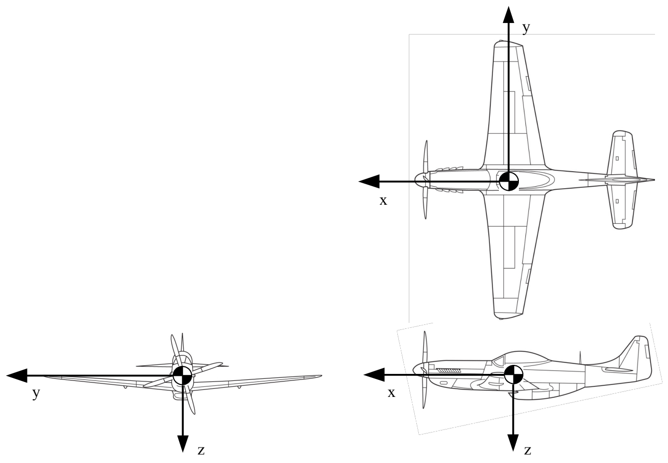


Figure 2: The Body-Fixed Axis System

- $\frac{\delta u}{\delta t}$ The inertial acceleration of the vehicle, along its body-fixed x-axis $\left(\frac{ft}{sec^2}\right)$.
- $\frac{\delta v}{\delta t}$ The inertial acceleration of the vehicle, along its body-fixed y-axis $\left(\frac{ft}{sec^2}\right)$.
- $\frac{\delta w}{\delta t}$ The inertial acceleration of the vehicle, along its body-fixed z-axis $\left(\frac{ft}{sec^2}\right)$.
- u The inertial velocity of the vehicle, along its body-fixed x-axis $\left(\frac{ft}{sec}\right)$.
- v The inertial velocity of the vehicle, along its body-fixed y-axis $\left(\frac{ft}{sec}\right)$.
- w The inertial velocity of the vehicle, along the body-fixed z-axis $\left(\frac{ft}{sec}\right)$.

The term "inertial" here, as well as the nature of the body-fixed accelerations and velocities, can be a little confusing. The values all act along the body-fixed axes and reflect the motion of the vehicle through non-moving, three-dimensional space. (They are *not* relative to the air.) But at the same time, the body-fixed axes are free to rotate in space. And so, for example, while the value of u can be constant, if the body is rotating, then the velocity of the vehicle *isn't* constant, and the path of the vehicle through space isn't a straight line. u, v and w are essentially instantaneous velocities.

- $\frac{\delta p}{\delta t}$ The angular acceleration of the vehicle about its body-fixed x-axis $\left(\frac{rad}{sec^2}\right)$.
- $\frac{\delta q}{\delta t}$ The angular acceleration of the vehicle about its body-fixed y-axis $\left(\frac{rad}{sec^2}\right)$.
- $\frac{\delta r}{\delta t}$ The angular acceleration of the vehicle about its body-fixed z-axis $\left(\frac{rad}{sec^2}\right)$.
- p The angular velocity of the vehicle about its body-fixed x-axis—roll rate $\left(\frac{rad}{sec}\right)$.
- q The angular velocity of the vehicle about its body-fixed y-axis—pitch rate $\left(\frac{rad}{sec}\right)$.
- r The angular velocity of the vehicle about its body-fixed z-axis—yaw rate $\left(\frac{rad}{sec}\right)$.

Again, these values act about the body-fixed axes, but at the same time, reflect the rotation of the axis system in space. So the angular velocities, for example, *aren't* equal to the rate of change of the orientation of the vehicle, but *do* contribute to it.

The demo version of the simulation assumes a "flat Earth" spatial model. (See Figure 3.) This provides a three-dimensional axis system in space. The X- and Y-axes form a horizontal plane and the X- and Z-axes form a vertical plane. The axes don't rotate in space and each extends to infinity. Essentially, we've created a flat Earth that extends off, in all directions. Note that the Z-axis points vertically down, toward the ground, and altitude has an opposite sense. So an altitude of, say 1,000 feet is equal to a Z value of -1,000. Weight always acts along the Z-axis. The axis system has a definite origin, at (0,0,0). On a spherical Earth, this point has to be established rather arbitrarily. Also note the upper-case axis labels. The flat-Earth model is useful and valid for any aircraft that doesn't fly at high speed or for long distances. The demo version of Multi-Sim doesn't have a ground reaction model. So you can fly the airplane through an altitude of 0, if you want to.

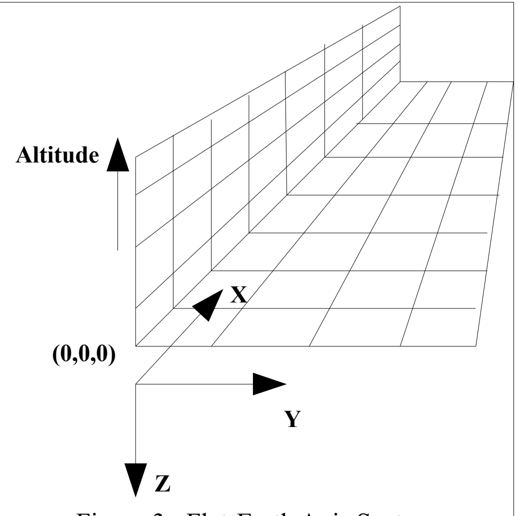


Figure 3: Flat-Earth Axis System

When the simulation returns a full set of data after completing the "advance" command, there is a set of data that references the flat-Earth spatial model:

- $\frac{\delta X}{\delta t}$ The velocity of the vehicle's center of mass, in the spatial X-direction $\left(\frac{ft}{sec}\right)$.
- $\frac{\delta Y}{\delta t}$ The velocity of the vehicle's center of mass, in the spatial Y-direction $\left(\frac{ft}{sec}\right)$.
- $\frac{\delta Z}{\delta t}$ The velocity of the vehicle's center of mass, in the spatial Z-direction $\left(\frac{ft}{sec}\right)$.
- X The position of the vehicle's center of mass, along the spatial X-axis *(ft)*.
- Y The position of the vehicle's center of mass, along the spatial Y-axis *(ft)*.
- Z The position of the vehicle's center of mass, along the spatial Z-*axis (ft)*.

- Altitude Height along the negative Z-axis *(ft)*.

$\frac{\delta X}{\delta t}$, $\frac{\delta Y}{\delta t}$ and $\frac{\delta Z}{\delta t}$ together, form the velocity vector of the vehicle. At any given moment, it has a velocity through space given by these three components. With the velocity and the flat-Earth axis system, we can describe the orientation or attitude of the vehicle. The orientation of the vehicle is described using the Euler Angles, ψ, θ and φ:

- $\frac{\delta\psi}{\delta t}$ The heading angle rate $\left(\frac{rad}{sec}\right)$.
- $\frac{\delta\theta}{\delta t}$ The orientation angle rate $\left(\frac{rad}{sec}\right)$.
- $\frac{\delta\phi}{\delta t}$ The roll rate $\left(\frac{rad}{sec}\right)$.
- ψ The first Euler rotation, about the body-fixed z-axis—heading angle *(rad)*.
- θ The second Euler rotation, about the body-fixed y-axis—longitudinal orientation angle *(rad)*.
- φ The third Euler rotation, about the body-fixed x-axis—roll angle (When the pitch angle is low, bank angle.) *(rad)*.

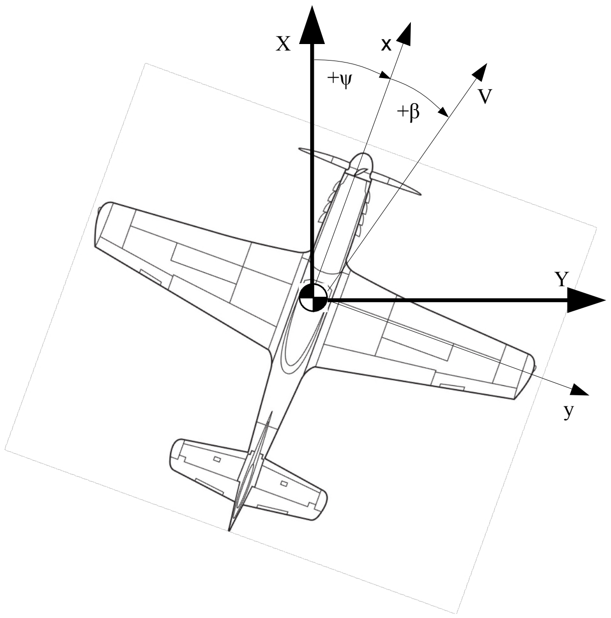


Figure 4: Aircraft Attitude, Top View

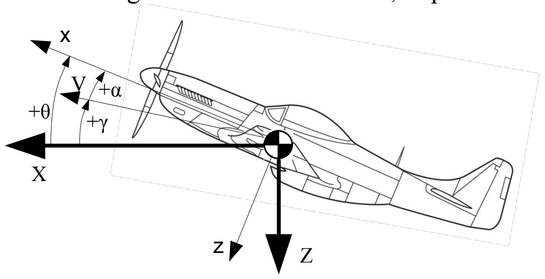


Figure 5: Aircraft Attitude, Side View

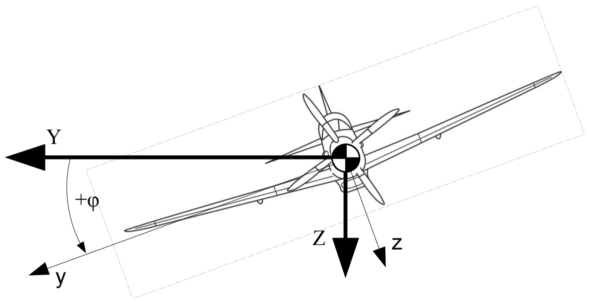


Figure 6: Aircraft Attitude, Front View

The Euler angles are the same angles that are read from an aircraft's artificial horizon or attitude indicator. Each combination of the Euler angles specifies a unique orientation of the vehicle in space. But the three rotations *aren't* commutative. That is, the user must apply them in the correct order or the final orientation will be wrong. To find the vehicle's orientation, first assume that the vehicle is aligned with the flat-Earth axis system—its body-fixed x-axis is aligned with the Earth X-axis, its body-fixed y-axis is aligned with the Earth Y-axis and its body-fixed z-axis is aligned with the Earth Z-axis. Now the vehicle is rotated about its body-fixed z-axis by the value ψ. This establishes the heading angle of the vehicle, the angle between the Earth X-axis and the body-fixed x-axis. Next, the vehicle is rotated about its body-fixed y-axis by θ. This establishes its pitch angle, the angle between the vehicle's longitudinal axis and the horizon. Last, the vehicle is rotated about its body-fixed x-axis by φ. At low values of θ, this establishes the vehicle's “bank angle” or the angle between the vehicle's lateral axis and the horizon. But at higher pitch angles, this becomes less well defined and is simply the roll angle. After the vehicle has been rotated through the three angles in order, it is in the current orientation. The sense of each of these angles is illustrated in Figures 4, 5 and

6.

The demo version of the simulation contains no wind model. This means that the air mass is fixed in space and the velocity of the vehicle through the air is identical to the velocity of the vehicle through space. This also means that the component inertial velocities, u, v and w, reflect the motion of the vehicle through the air. The air flows over the vehicle in the direction *opposite* these velocities. The vehicle rarely points in the same direction as its spatial velocity vector, meaning the v and w velocity components are rarely zero. And this means that the velocity of the wind over the vehicle is at some angle to the body-fixed x-axis. We describe the relationship between the vehicle and its velocity vector with the following angles:

- α The angle of attack of the vehicle *(rad)*.
- γ The flightpath angle of the vehicle *(rad)*.

In the side view of Figure 5, note the angle between the vehicle's body-fixed x-axis and its velocity vector. This is the "angle of attack" of the aircraft. The vehicle's lift, drag and pitching moment are functions of this angle. Also note the angle between the vehicle's velocity vector and the horizon. This is the "flightpath angle" of the vehicle.

- β The sideslip angle *(rad)*.

On the top view of Figure 4, the angle between the vehicle's body-fixed x-axis and its velocity vector is the "sideslip" angle. The sense of the sideslip angle is chosen to be positive when it results in the vehicle experiencing a positive velocity along its y-axis. Thus, a positive angle of β results from the nose of the vehicle swinging *left*. The lateral force, rolling moment and moment about the z-axis are functions of this value.

Trimming the Aircraft

An airplane can fly straight and level over a range of airspeeds, from stall to its maximum speed. At any given speed, the airplane will be flying with a unique combination of angle of attack, elevator deflection and thrust (or power). These are the "trim" settings for that airspeed. To initiate a simulation run with the airplane flying straight and level, at least one of these settings must be specified, and then the rest can be calculated. Multi-Sim has a built-in set of routines that provide this functionality.

There are three strategies for trimming the airplane, based on which parameter is specified. The user can specify the trim airspeed, and then Multi-Sim will find the angle of attack, elevator deflection and thrust that will result in the airplane being trimmed at that airspeed. The user can also specify the angle of attack, and Multi-Sim will find the trim airspeed, elevator deflection and thrust. Last, the user can specify elevator deflection and Multi-Sim will find the airspeed, angle of attack and thrust:

Trim Strategy	Airspeed	Angle of Attack	Elevator Deflection	Thrust
By Airspeed	Specified	Found	Found	Found
By Angle of Attack	Found	Specified	Found	Found
By Elevator Angle	Found	Found	Specified	Found

Note that the airspeed we talk about here is "true" airspeed, or the geometric speed with which the airplane is moving through the air. "Indicated" airspeed is this value adjusted for air density; the lower the density, the lower the indicated airspeed. So at an altitude of, say, 5,000 ft., the indicated airspeed will be *lower* than the true airspeed.

If the airplane being simulated is asymmetrical across its x-z plane, or if the airplane has a thrust line that doesn't pass through its center of mass, Multi-Sim will also find rudder and aileron deflection settings to ensure that it is flying straight and level. They are returned with the rest of the data.

Multi-Sim's trim command allows a user to trim the airplane to his wishes. It takes the strategy to be used, along with the pertinent data, and then trims the airplane. It returns all of the trim settings, along with the thrust, power and throttle position required to fly at those settings. The user must take note of the throttle setting returned. It can be greater than 100%. Also note the importance of transferring the trim settings from your program's trim response data structure to its data structure for the "advance" command. This is necessary to ensure that the simulated airplane begins flying with the correct control settings.

Last, note that there are settings for which there is no trim solution. If, for example, you ask for too low an angle of attack, there may be no airspeed at which the airplane will be able to fly. The same is true for too nose-down a value for elevator deflection. The trim response contains a success field that indicates whether the simulation was able to find a valid trim solution.

The Simulated Vehicle

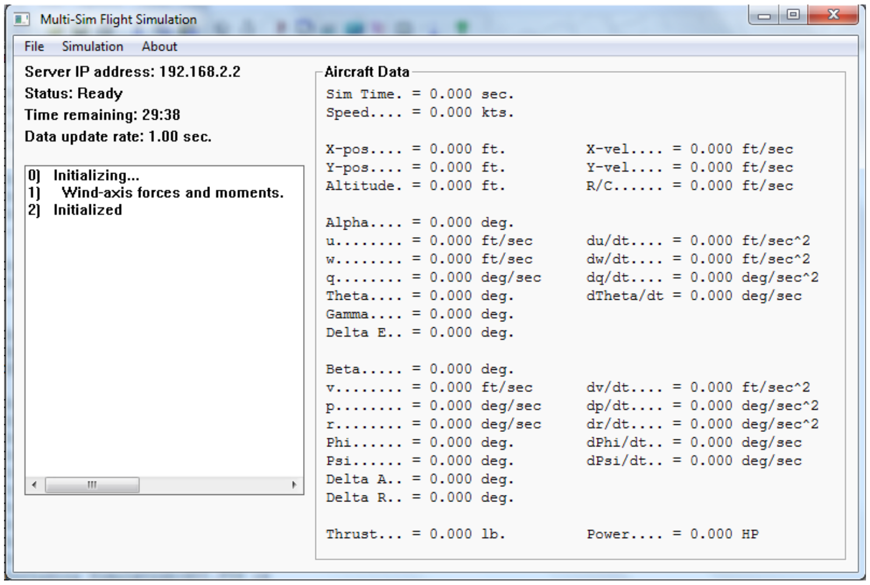
In this demo version of the program, the vehicle is hard-wired to be the Cessna 172. (See Figure 7.) The 172 is a single-engine, four-seat airplane created by Cessna Aircraft Corp. and is commonly used for both training and transportation. The data for the model used in the simulation are listed below. The aerodynamic and mass-properties data were estimated by Jan Roskam in his book **Airplane Flight Dynamics and Automatic Flight Controls, Part I,** DAR Corporation, Lawrence, KS, 1995.

Figure 7: Cessna 172

Wingspan:	35.8 ft.
Length:	26.9 ft.
Weight:	2,300 lbs.
Engine:	Lycoming O-320, horizontally-opposed four-cylinder
Power:	160 hp. at sea-level
Stall speed:	47 kts.
Cruise speed:	122 kts. (75% power)
C.G. Loc.:	3.5 ft. from the datum (center of range at 2,300 lbs.)

- Aileron deflection, δA, is positive for a positive moment about the x-axis, that is, a roll to the right. Maximum deflection is +/- 20 degrees.
- Elevator deflection, δE, is negative for a positive moment about the y-axis; negative for the trailing edge of the elevator to move up. Maximum deflection is -28 to 23 degrees.
- Rudder deflection, δR, is positive for a positive moment about the z-axis, that is, yawing the nose to the right. Maximum deflection is +/- 16 degrees.

Running Multi-Sim

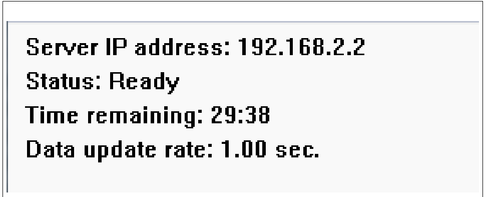


```
Server IP address: 192.168.2.2
Status: Ready
Time remaining: 29:38
Data update rate: 1.00 sec.

0)  Initializing...
1)    Wind-axis forces and moments.
2)  Initialized
```

```
Aircraft Data
Sim Time. = 0.000 sec.
Speed.... = 0.000 kts.

X-pos.... = 0.000 ft.          X-vel.... = 0.000 ft/sec
Y-pos.... = 0.000 ft.          Y-vel.... = 0.000 ft/sec
Altitude. = 0.000 ft.          R/C...... = 0.000 ft/sec

Alpha.... = 0.000 deg.
u........ = 0.000 ft/sec       du/dt.... = 0.000 ft/sec^2
w........ = 0.000 ft/sec       dw/dt.... = 0.000 ft/sec^2
q........ = 0.000 deg/sec      dq/dt.... = 0.000 deg/sec^2
Theta.... = 0.000 deg.         dTheta/dt = 0.000 deg/sec
Gamma.... = 0.000 deg.
Delta E.. = 0.000 deg.

Beta..... = 0.000 deg.
v........ = 0.000 ft/sec       dv/dt.... = 0.000 ft/sec^2
p........ = 0.000 deg/sec      dp/dt.... = 0.000 deg/sec^2
r........ = 0.000 deg/sec      dr/dt.... = 0.000 deg/sec^2
Phi...... = 0.000 deg.         dPhi/dt.. = 0.000 deg/sec
Psi...... = 0.000 deg.         dPsi/dt.. = 0.000 deg/sec
Delta A.. = 0.000 deg.
Delta R.. = 0.000 deg.

Thrust... = 0.000 lb.          Power.... = 0.000 HP
```

Figure 8: The Multi-Sim Window

Multi-Sim is a very easy program to use. The executable file name is `Multi-Sim.exe`. In Windows Explorer, go to the folder where you saved Multi-Sim and double-click on `Multi-Sim.exe`. The program should come up very quickly, and you will see the Multi-Sim window, Figure 8. The first time you start Multi-Sim, Windows will inform you that the program is trying to access the network, and will ask whether you want to allow it. If you are certain that you will only connect to the simulation from a program on the same computer, you can allow Windows to block it. Otherwise, select the boxes so that Windows will allow Multi-Sim to have access to your local network and/or the internet.

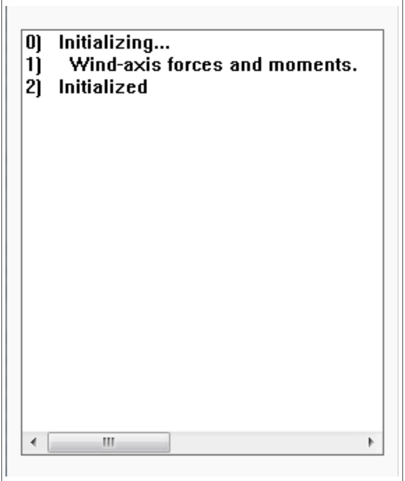


```
Server IP address: 192.168.2.2
Status: Ready
Time remaining: 29:38
Data update rate: 1.00 sec.
```

Figure 9: Multi-Sim Status Fields

The Multi-Sim window is broken up into three main areas. At the upper-left, you'll see the status fields (Figure 9). These provide information about the status of Multi-Sim itself. The first line reports the IP address of the computer. This is useful if you are running Multi-Sim on one computer and a cab on another. The cab will have to know the IP address of the computer running Multi-Sim. The next line shows the simulation's status. When it reads "Ready", Multi-Sim is idle and waiting for a cab to connect over TCP/IP. When a cab connects, it will say "Connected". The demo version of Multi-Sim will run for 30 minutes at a time. The third line shows the amount of time left before the user must close the program and restart it. The last line reports the rate at which the simulation is updating its data fields. Displaying the aircraft's data in the data fields takes time and computing power, slowing the simulation's computations and response times. The update rate is set to a default value of 1 second when you start the program; the simulation will update the data fields every simulated second. If you want the simulation to run faster, you can set this value higher, 60 seconds, for example. Then, the data will be updated every simulated minute. The "Sim Time" value is updated every simulated second regardless of the update setting. If you want to see the data updated at very small intervals, you can set this value to fractions of a second. But it will slow the simulation down dramatically.

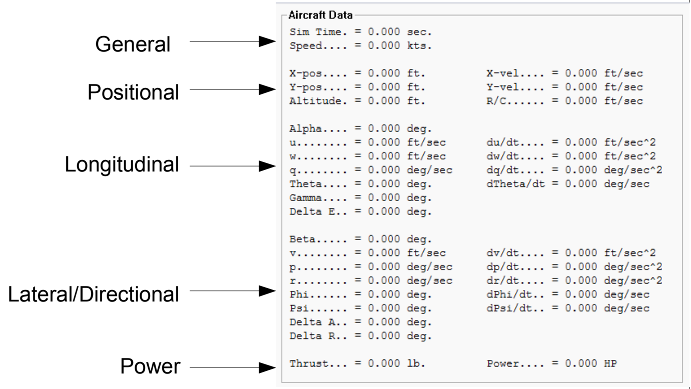
Below the status fields is the simulation's message window (Figure 10). Here, you will see notices posted by the simulation as it does its work. When the simulation is first started, it automatically initializes all of its internal variables. The simulation also does this when a cab commands the simulation to trim the simulated aircraft. It initializes all of the vehicle values and resets the simulated time to zero. You'll see a message posted when this happens. In Figure 10, you'll also note the message, “Wind-axis forces and moments”. This indicates that the aircraft data file provides aerodynamic values that were measured in the “wind axes”, and the simulation is making its aerodynamic calculations based on that measurement scheme. The standard versions of Multi-Sim allow a user to select and load vehicle data files. This message gives the user confidence that the simulation is applying the data correctly. When a cab connects, the system announces, “Beginning session with xxx.xxx.xxx.xxx,” reporting the IP address of the connecting cab. When the cab disconnects, the system announces, “Closing session.” When a cab sends the “advance” command, the system doesn't post any kind of notice. That would slow down computation and response. Instead, it simply advances the simulation ahead in time and then updates the data fields.



```
0)  Initializing...
1)    Wind-axis forces and moments.
2)  Initialized
```

Figure 10: The Multi-Sim Message Window

The simulation's data fields display the current state of the vehicle. The values displayed here are described in the “Axis Systems and Physical Values” section. The values are collected into five groups. The first group provides general information, the simulation time and the vehicle's scalar speed. The second group provides positional data. The third group provides what is commonly considered “longitudinal” data. The fourth group provides “lateral/directional” data. And the fifth group provides power-related data. The left-hand columns of each group contain the pertinent values. The right-hand columns contain the *rates*, the first derivatives with respect to time, of some of the values



```
Sim Time. = 0.000 sec.
Speed.... = 0.000 kts.

X-pos.... = 0.000 ft.          X-vel.... = 0.000 ft/sec
Y-pos.... = 0.000 ft.          Y-vel.... = 0.000 ft/sec
Altitude. = 0.000 ft.          R/C...... = 0.000 ft/sec

Alpha.... = 0.000 deg.
u........ = 0.000 ft/sec       du/dt.... = 0.000 ft/sec^2
w........ = 0.000 ft/sec       dw/dt.... = 0.000 ft/sec^2
q........ = 0.000 deg/sec      dq/dt.... = 0.000 deg/sec^2
Theta.... = 0.000 deg.         dTheta/dt = 0.000 deg/sec
Gamma.... = 0.000 deg.
Delta E.. = 0.000 deg.

Beta..... = 0.000 deg.
v........ = 0.000 ft/sec       dv/dt.... = 0.000 ft/sec^2
p........ = 0.000 deg/sec      dp/dt.... = 0.000 deg/sec^2
r........ = 0.000 deg/sec      dr/dt.... = 0.000 deg/sec^2
Phi...... = 0.000 deg.         dPhi/dt.. = 0.000 deg/sec
Psi...... = 0.000 deg.         dPsi/dt.. = 0.000 deg/sec
Delta A.. = 0.000 deg.
Delta R.. = 0.000 deg.

Thrust... = 0.000 lb.          Power.... = 0.000 HP
```

Figure 11: Multi-Sim Data Fields

Multi-Sim's menu contains only a few items. Selecting “File” allows a user to select and load a vehicle data file, containing aerodynamic and mass-properties values. (This is disabled in the demo version.) Selecting “Simulation” then “Set data update rate” brings up a dialog box allowing the user to change Multi-Sim's data update rate from the default value of 1 simulated second. Enter any value greater than zero. Finally, the about box gives details on the system's origin and copyright information.

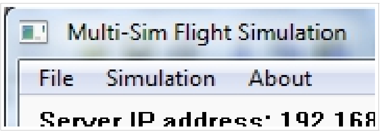


Figure 12: Multi-Sim's Menu

Utilizing Multi-Sim: Commands and Message Formats

This is what Multi-Sim is all about: allowing you to control the simulated vehicle. Multi-Sim operates as a TCP/IP server. To communicate with the simulation, a client application must connect using a TCP connection. The demo version of the simulation operates on:

TCP Port Number: 20248 or 0x4F18

This port number seems to be available on most Windows 7 systems.

After establishing a TCP connection with the simulation, a client is free to send command messages and receive response messages. The messages and their formats follow. Note that all values in these messages are sent *little endian*. That is, each value in the message is stored into its field *least significant byte first*. Integer values are all 32 bits (8 bytes). Floating point values are all 64-bit (8 byte) "doubles". The simulation always copies the command value and includes it in its response as the first field. This allows a client to determine which response message it is receiving, if necessary. In each of the following message diagrams, memory addresses increase *down* the page.

Hello Command

The hello command is a simple command that aids in the development and debugging of client programs. The command itself consists of a single value, a 32-bit zero.

Field	Value	Size	Address
Command	0x00000000	32-bit integer	Base + 0

Hello Response

The simulation responds by sending the time and date reported by the host computer.

Field	Value	Size	Address
Command	0x00000000	32-bit integer	Base + 0
Seconds	–	32-bit integer	+4
Minutes	–	32-bit integer	+8
Hours	–	32-bit integer	+12
Day	–	32-bit integer	+16
Month	–	32-bit integer	+20
Year	–	32-bit integer	+24

Trim Command

The trim command sends the trim data to the simulation and commands it to find the angle of attack, speed, control surface deflections and throttle setting needed to make the airplane fly straight and level.

Trim strategy values, placed in the "Trim Strategy" field, are:

- 0: By angle of attack
- 1: By airspeed
- 2: By elevator deflection angle

When the simulation receives the trim command, it initializes itself, resetting the simulated time to zero and placing the airplane at its initial position in space, in its initial direction of flight. In the command message, only the relevant trim command field needs to be filled in. If the trim strategy is angle of attack, for example, then the "Angle of Attack" field must contain a valid value. Both "Airspeed" and "Elevator Angle" are ignored. X, Y and Z are the spatial locations where the airplane will be placed at time zero. Note that Z = -altitude. So if you desire a starting altitude of 5,000 ft, Z must be set to -5000. Heading is the compass direction that you'd like the airplane to be pointed at the start of flight. A value of 0 will align the airplane's body-fixed x-axis with the spatial X-axis. A value of $\frac{\pi}{2}$ (90°) will align the body-fixed x-axis with the spatial Y-axis.

Field	Value	Size	Address
Command	0x00000002	32-bit integer	Base + 0
Trim Strategy	0x0,1,2	32-bit integer	+4
Angle of Attack	radians	64-bit float	+8
Airspeed	ft/sec	64-bit float	+16
Elevator Angle	radians	64-bit float	+24
X	ft	64-bit float	+32
Y	ft	64-bit float	+40
Z	ft	64-bit float	+48
Heading	radians	64-bit float	+56

Trim Response

After finding the trim settings for the airplane, initializing the simulation to zero time and placing the airplane in its initial position and heading, the simulation sends the trim response back to the client. If the simulation was able to find a trim solution, the "Trim Result" field will contain a TRUE (-1) value. Again note that there are some trim settings that aren't valid. The simulation won't find a solution in those cases and will return a FALSE (0) value. While finding the trim settings, the simulation will also find the throttle position required for straight and level flight. It's possible to enter valid trim settings that require more power than is available. In those cases, the throttle setting will be greater than 100%. The throttle setting is scaled linearly with power. The simulation includes an engine model, and maximum power decreases with air density—altitude. So a throttle setting of 75% at 5,000 feet will yield less power than the same setting at sea-level. The sideslip angle will be zero for any airplane that is symmetrical across its x-z plane. But it will be a finite value for an airplane that isn't so.

Field	Value	Size	Address
Command	0x00000002	32-bit integer	Base + 0
Trim Result	-1 (0xFFFFFFFF) or 0	32-bit integer	+4
Angle of Attack (α)	radians	64-bit float	+8
Angle of Sideslip (β)	radians	64-bit float	+16
Airspeed	ft/sec	64-bit float	+24
Aileron Angle (δA)	radians	64-bit float	+32
Elevator Angle (δE)	radians	64-bit float	+40
Rudder Angle (δR)	radians	64-bit float	+48
Thrust	lbs	64-bit float	+56
Power	ft-lb/sec	64-bit float	+64
Throttle Setting	Fraction (0.0 – 1.0)	64-bit float	+72

Advance Command

The advance command is used to fly the simulated airplane. It gives the simulation a full set of control inputs, along with a time to calculate into the future. The simulation takes the control inputs, applies them to the airplane, then marches the airplane's response forward in time the specified amount. The advance time can be any floating point value except for zero. If you need to see the airplane's attitude and apply control inputs at fine intervals, then use small values, like 0.1 seconds. If you are leaving the controls fixed for a long period of time, say 60 seconds, and also don't wish to see the attitude values until then, you can use any larger value.

Field	Value	Size	Address
Command	0x00000001	32-bit integer	Base + 0
Advance Time	seconds	64-bit float	+4
Aileron Angle (δA)	radians	64-bit float	+12
Elevator Angle (δE)	radians	64-bit float	+20
Rudder Angle (δR)	radians	64-bit float	+28
Throttle Setting	Fraction (0.0 – 1.0)	64-bit float	+36

Advance Response

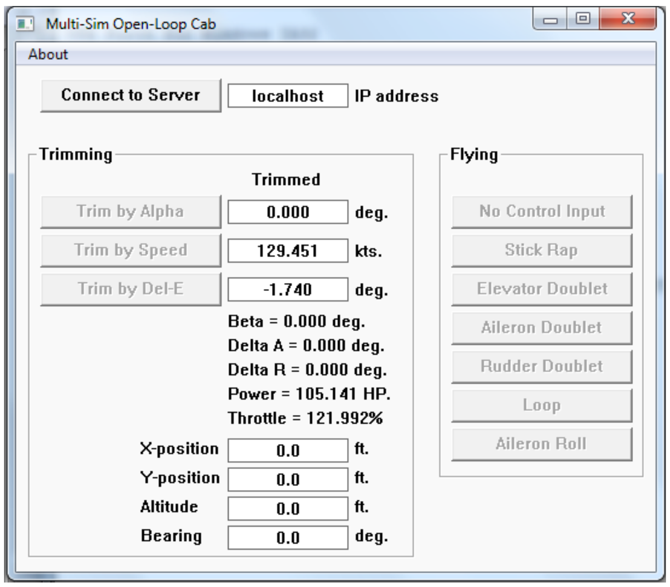
After receiving the advance command, the simulation marches the airplane's responses ahead in time the specified amount. Then it sends back a complete compliment of aircraft data. You'll note that the control inputs are returned with the rest of the data. In the standard versions of Multi-Sim, the control actuators can be simulated. This means that the controls' actual positions may not match the control input values. In the demo version of the program, the controls move instantaneously, so these values will always match the commanded values.

Field	Value	Size	Address
Command	0x00000001	32-bit integer	Base + 0
Simulation Time	sec	64-bit float	+4
u Velocity	ft/sec	64-bit float	+12
v Velocity	ft/sec	64-bit float	+20
w Velocity	ft/sec	64-bit float	+28
p Angular Velocity	rad/sec	64-bit float	+36
q Angular Velocity	rad/sec	64-bit float	+44
r Angular Velocity	rad/sec	64-bit float	+52
ψ Euler Angle	radians	64-bit float	+60
θ Euler Angle	radians	64-bit float	+68
φ Euler Angle	radians	64-bit float	+76
X Spatial Position	ft	64-bit float	+84
Y Spatial Position	ft	64-bit float	+92
Altitude	ft	64-bit float	+100
Angle of Attack (α)	radians	64-bit float	+108
Angle of Sideslip (β)	radians	64-bit float	+116
Flightpath Angle (γ)	radians	64-bit float	+124
x-Axis Acceleration $\frac{\delta u}{\delta t}$	ft/sec^2	64-bit float	+132
y-Axis Acceleration $\frac{\delta v}{\delta t}$	ft/sec^2	64-bit float	+140
z-Axis Acceleration $\frac{\delta w}{\delta t}$	ft/sec^2	64-bit float	+148
x-Angular Accel. $\frac{\delta p}{\delta t}$	rad/sec^2	64-bit float	+156
y-Angular Accel. $\frac{\delta q}{\delta t}$	rad/sec^2	64-bit float	+164
z-Angular Accel. $\frac{\delta r}{\delta t}$	rad/sec^2	64-bit float	+172
Heading Rate $\frac{\delta \psi}{\delta t}$	rad/sec	64-bit float	+180
Pitch Rate $\frac{\delta \theta}{\delta t}$	rad/sec	64-bit float	+188
Roll Rate $\frac{\delta \phi}{\delta t}$	rad/sec	64-bit float	+196

Spatial Velocity $\frac{\delta X}{\delta t}$	ft/sec	64-bit float	+204
Spatial Velocity $\frac{\delta Y}{\delta t}$	ft/sec	64-bit float	+212
Spatial Velocity $\frac{\delta Z}{\delta t}$	ft/sec	64-bit float	+220
Aileron Angle δA	radians	64-bit float	+228
Elevator Angle δE	radians	64-bit float	+236
Rudder Angle δR	radians	64-bit float	+244
Thrust	lb	64-bit float	+252
Power	ft-lb/sec	64-bit float	+260
Throttle Setting	fraction (0.0 – 1.0)	64-bit float	+268

Hitting the Ground Running: The Open-Loop Cab

Multi-Sim comes with an example client program, the Open-Loop Cab. This simple cab program allows a user to experience what it's like to use Multi-Sim, trimming and then flying a simulated airplane. The Open-Loop cab provides a set of controls you can use to first trim the simulated airplane (See Figure 13), then, using a second set of controls, you can fly the simulated airplane through a collection of open-loop maneuvers. As the airplane is being flown through the maneuvers, the cab saves the aircraft data to a file, so you can view and graph it using any spreadsheet program.



localhost

Trimmed

0.000 deg.

129.451 kts.

-1.740 deg.

Beta = 0.000 deg.
Delta A = 0.000 deg.
Delta R = 0.000 deg.
Power = 105.141 HP.
Throttle = 121.992%

X-position 0.0 ft.
Y-position 0.0 ft.
Altitude 0.0 ft.
Bearing 0.0 deg.

Figure 13: The Open-Loop Cab

The Open-Loop Cab provides three services. First, it allows you to become familiar with the way Multi-Sim works, giving you a tool that you can use to exercise the simulation. Second, for the interested individual, it provides a very good tool to use for learning the bare-airframe flight dynamics of an airplane, and how an airplane responds to control inputs. Last, and most importantly, the cab provides both an example of how to write a cab for Multi-Sim, *and*, because it comes with source code, provides a set of basic files that you can build upon when writing your own cab, whatever its functionality might be.

Running the Cab

You will find two copies of the cab in the Multi-Sim distribution package. In Windows Explorer, go to the folder where you saved the Multi-Sim files. Note the two sub-folders, `Forth` and `C`. You'll find a copy of the cab, named `Open-Loop-Cab.exe`, in each folder. The copy in the `Forth` folder is written in the Forth programming language, and the copy in the `C` folder is written in the C programming language. They are functionally identical. Pick one and start it by double-clicking on it. Next, go back to the main folder and start Multi-Sim by double-click on `Multi-Sim.exe`. Position the two windows on your monitor so that you can easily see both.

You'll see that most of the buttons on the cab are deactivated. Click on the “Connect to Server” button. This will start a TCP/IP session with the simulation. When the simulation accepts the connection, the trim keys will be activated and the button you just clicked will say, “Disconnect.” The word “localhost” in the IP Address field indicates that the cab will try to connect to the server on the same computer. If you want, you can run the simulation on another computer. Then, just enter the IP address displayed in the status field of the simulation in the address field on the cab. Click connect, and it should be able to connect to the simulation across the network.

Once the cab has connected, you can trim the airplane and ready it for flight. Enter a value between 0 and 10 degrees in the “Trim by Alpha” field. Now click on “Trim by Alpha”. The cab will send the trim command to the simulation, telling it to trim the airplane to the value you entered. The cab should now tell you that the airplane is trimmed. Note how the speed, Del-E, power and throttle fields have been filled in. If the airplane was asymmetrical across its x-z plane, or if its engine was off center, the Beta, Delta A and Delta R fields would also have finite values in them. What you see in the fields is the unique set of trim settings for the airplane at the angle of attack that you entered. Now try entering a value of -5 degrees in the Alpha field and again click “Trim by Alpha”. The cab should tell you that the airplane is “>UNTRIMMED<”. This is because the angle of attack is so low that the airplane can't support its own weight at any airspeed. Note the corresponding message in the message window of the simulation. Re-enter the first value you used and trim again. If you want, you can enter values into the position and heading fields *first*, enter a value into one of the “Trim by...” fields, and then click on that button. The simulation will trim the airplane and put it at the position you requested. Note how the simulation informs you that it is initializing and trimming the airplane, and how the values in the simulation's data field match the values in the cab's fields. The airplane is now flying and the simulation is ready to advance its responses forward in time.

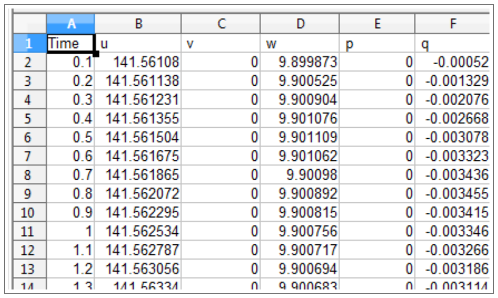
When the airplane is trimmed, the “Flying” buttons are activated. Now you can click on one of them and the cab will fly the airplane. Each button flies the airplane through a different maneuver:

- “**No Control Input**” flies the airplane straight and level, a good test to see if the simulation really found accurate trim settings for the airplane.
- “**Stick Rap**” flies the airplane straight and level for 3 seconds. Next, the pilot pushes the stick forward, deflecting the elevator 8 degrees down, holding it there for one second, then returns it back to its original position. Then the airplane flies for an additional 116 seconds with the controls held in place.
- “**Elevator Doublet**” flies the airplane straight and level for three seconds. Next the pilot pulls the stick back, deflecting the elevator 4 degrees up and holds it for one second. Next, he pushes the stick forward, deflecting the elevator 4 degrees down (from its trim position) and holds again for one second. Then he returns the stick to its trim position and the airplane flies for an additional 115 seconds.
- “**Aileron Doublet**” flies the airplane straight and level for three seconds. Next the pilot pushes the stick to the left, deflecting the ailerons 4 degrees, and holds it for one second. Next, he pushes the stick to the right, deflecting the ailerons 4 degrees the other direction, and holds again for one second. Then he returns the stick to its trim position and the airplane flies for an additional 115 seconds.
- “**Rudder Doublet**” flies the airplane straight and level for three seconds. Next the pilot steps on the right rudder pedal, deflecting the rudder 4 degrees to the right, and holds it for one second. Next, he steps on the left rudder pedal, deflecting the rudder 4 degrees to the left, and holds again for one second. Then he centers the rudder and the airplane flies for an additional 115 seconds.
- “**Loop**” is actually a semi-open-loop maneuver. The pilot flies the airplane straight and level for three seconds. Next, he pushes the throttle full-forward and pulls back on the stick, deflecting the elevator up 10 degrees. He holds the stick until the airplane completes a single loop, then returns the throttle and stick to their trim positions, flying the airplane for 10 more

seconds.

- "**Aileron Roll**" is also a semi-open-loop maneuver. The pilot flies the airplane straight and level for 3 seconds. Next, the pilot pushes the stick to the right, deflecting the ailerons 10 degrees. He holds the stick there until the airplane has rolled 360 degrees, then he centers the stick and flies the airplane for another 10 seconds.

When you click on one of the buttons, the cab will begin flying the airplane. Note how the data values begin changing on the simulation. The button you clicked will say "working..." until the maneuver is complete. When the flight is done, the values on the simulation will stop changing. Most of the maneuvers fly for 2 simulated minutes. Note how they are completed in a second or two of real time. While flying the maneuver, the cab saves all of the aircraft data values out to a comma-separated-value file named `FlightData.csv`. (In the same folder as the executable file.) You can load this data into any spreadsheet program and look at it. (See Figure 14.) Try graphing some of the values with respect to time using a "scatter" plot. (See Figure 15.) In graphic form, you see the response of the airplane to control inputs, and the "bare airframe" dynamics of the airpalane after it has been displaced from its trimmed state. Even for the experienced pilot or Aerospace Engineer, the plots can be very illuminating. Each time you trim the airplane, the simulated time is set to zero and any residual motion from a previous flight is erased. You can string multiple maneuvers together by clicking on the flying buttons without retrimming. But keep in mind that the maneuvers are open-loop. Nobody is on duty, keeping the airplane from entering a very bad attitude.



	A	B	C	D	E	F
1	Time	u	v	w	p	q
2	0.1	141.56108	0	9.899873	0	-0.00052
3	0.2	141.561138	0	9.900525	0	-0.001329
4	0.3	141.561231	0	9.900904	0	-0.002076
5	0.4	141.561355	0	9.901076	0	-0.002668
6	0.5	141.561504	0	9.901109	0	-0.003078
7	0.6	141.561675	0	9.901062	0	-0.003323
8	0.7	141.561865	0	9.90098	0	-0.003436
9	0.8	141.562072	0	9.900892	0	-0.003455
10	0.9	141.562295	0	9.900815	0	-0.003415
11	1	141.562534	0	9.900756	0	-0.003346
12	1.1	141.562787	0	9.900717	0	-0.003266
13	1.2	141.563056	0	9.900694	0	-0.003186
14	1.3	141.56334	0	9.900683	0	-0.003114

Figure 14: CSV Data in a Spreadsheet Program

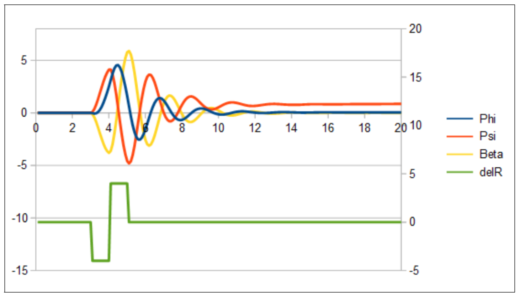


Figure 15: Lateral/Directional Responses from a Rudder Doublet

That said, try playing. You can't break the simulation or the cab. Try trimming the airplane to different speeds and then running it through the maneuvers. You'll find that the airplane can react differently to the same maneuver when it enters the maneuver at different speeds.

Getting You Started

The main reason we wrote the Open-Loop Cab is to provide you with a set of source-code files that will get you started writing your own cabs to fly the simulated airplane. There are two sets of source-code files to work with. They are located in the `Forth\Source Code` and `C\Source Code` folders in the Multi-Sim distribution. The Forth and C cabs are written in a parallel fashion, using files of the same name and, within them, functions with the same name and functionality.

C Version

On the C side, the primary files are:

- DataValues.h
- NetworkSupport.h
- Commands.h

To provide the Open-Loop Cab program, they are augmented by:

- ProgrammerAids.h
- OpenLoop.h
- main.cpp

The C version of the Open-Loop Cab was written and compiled using Code::Blocks and its gcc compiler, and was written with a Windows GUI. As the files exist in the directory, if you have Code::Blocks loaded on your computer, you should be able to double-click on `Flight Simulation Client.cbp` and the project will open. It should compile without any issues. Some of the command functions display information or report error conditions using Windows message boxes. These require the handle of a parent window, which you'll see passed. If you aren't compiling for Windows, simply remove the MessageBox calls and remove the window handle parameter from the function declarations.

`main.cpp` is the top-level file and is also used as a load file. You only need to look at it to see what libraries are being loaded and to get the correct order of compilation for the files listed above. The code within the file is all related to the Windows GUI for the cab, and is only relevant if you want to see how that was written.

`DataValues.h` is *very* important. This is where the data structures are created that form the command and response messages to and from the simulation. Read the notes at the top of the file and you should be able to understand what is going on below. Note how all of the fields are defined for each message. The Windows TCP/IP `send` and `recv` functions expect to be passed a character array filled with data to be sent, or an empty array with space to place a received message. We need to send and receive integer and floating point data, however. We accomplish this by creating character arrays of appropriate length for the messages and then assigning pointers into those arrays, marking the field for each integer or floating point value.

NetworkSupport.h provides a set of functions that send messages to the simulation and receive responses.

Commands.h is also an important file. It provides a set of high-level functions that first form command messages, send the command to the simulation and then receive the response. **These are the functions that you can use to write your own cab.** They are simple to use, and there are only a few of them.

DataValues.h, NetworkSupport.h and Commands.h are the only files you need to write your own cab. Simply load them before your own code and call the functions within Commands.h.

OpenLoop.h is included as an example. It provides the routines that fly the airplane in the Open-Loop Cab. There, you will see good examples of how to use the command functions in Commands.FTH to trim and fly the simulated airplane.

ProgrammerAids.h. is a support file for the GUI. Here, you'll find a small set of functions that are largely used for unit conversions.

And finally, main.cpp embodies the GUI that allows you to run the routines in OpenLoop.h from a window (actually, a dialog box).

Forth Version

On the Forth side, the primary files are:

- Network-Support.FTH
- Data-Values.FTH
- Commands.FTH
- FlightSim.FTH

To provide the Open-Loop Cab program, they are augmented by:

- Windows-Globals.FTH
- ASCII-Float.FTH
- Open-Loop.FTH
- Client-GUI.FTH

The Forth version of the cab was compiled under VFX Forth, from MicroProcesor Engineering, Ltd., in Southampton, UK. http://www.mpeforth.com/. MPE offers a trial version of VFX for free. Go to http://www.mpeforth.com/arena.htm#trial and scroll down to "VFX Forth for Windows." The source code for the open-loop cab is all written to be ANS Forth-compliant, so you should be able to use it under your favorite Forth system without any changes.

FligthSim.FTH is the load file for the program. There, you will see two VFX library files being loaded, NDP387.FTH, which supplies floating point words, and SOCKETIO.FTH, which supplies

TCP/IP networking words. You'll need to find and load the same library words on your own Forth system.

`Network-Support.FTH` provides a small set of words that are used to connect to the simulation, send messages to the simulation and receive responses.

`Data-Values.FTH` is a *very* important file. It creates a set of structures that form the messages for the simulation commands and responses. Buffers of the appropriate size are created for each message. Then the structures are used to address each individual field within the buffers. The buffers are loaded with data and passed to the VFX TCP/IP `writesock` word to send a command. Empty buffers are passed to `readsock` to receive the response from the simulation.

`Commands.FTH` is also an important file. It provides a set of high-level words that form command messages, send them to the simulation, then receive response messages. **These are the words that you can use to write your own cab.** They are simple to use, and there are only a few of them.

`NetworkSupport.FTH`, `DataValues.FTH` and `Commands.FTH` are the only files you need to write your own cab. Simply load them before your own code and call the words within `Commands.FTH`.

`Open-Loop.FTH` is included as an example. It provides the routines that fly the airplane in the Open-Loop Cab. There, you will see good examples of how to use the command words in `Commands.FTH` to trim and fly the simulated airplane.

`Windows-Globals.FTH` loads any Windows API functions that aren't included in the VFX Forth kernel by default.

Finally, `Client-GUI.FTH` comprises the Windows GUI for the Open-Loop Cab. It's provided in case you're interested in writing your own GUI. The Forth version of the cab employs a main window for its GUI. In the C version of the cab, you will find a GUI with identical functionality, but written using a dialog box and no main window. The dialog box technique requires a great deal less code.

System Requirements

- Windows XP, Vista, 7 and 8
- Intel 80x86 processor with numeric coprocessor