

A 6DoF Simulation Laboratory at the University of Naples

Agostino De Marco, University of Naples, "Federico II", Italy

The University of Naples "Federico II" in Italy has recently acquired a 6DOF flight simulation facility. The news is that FlightGear and JSBSim are two important building blocks of this research simulator. Even better, since the earliest design stage, most of the required characteristics to the flight simulation software that would drive the simulator were found to fit perfectly with what FlightGear and JSBSim had to offer: (i) low cost, in our case free, (ii) controllable and easily customizable for special needs, in our case Open Source, and (iii) highly configurable especially in terms of flight model definition, i.e. one of the most valuable features of JSBSim.

The whole system has been designed to be operated both as a driving simulator and as a flight simulator and is going to be managed by two different research teams, one including the author and his colleagues of the Aircraft Design and Aeroflightdynamics Group working at the Department of Aeronautical Design, the other team coming from the Transportation Department of the same University. The simulator includes a full-scale cabin, a motion base, a large projection system, and force feedback modules (fig. 1). The half-body of a real car and the aircraft cockpit mock-up are exchangeable and easily installed on the motion platform.



Figure 1

The author contributed to the specifications, the development and the final acceptance procedure of this facility as for the flight simulation section. The simulator was built by Oktal, France, (www.oktal.fr). The aim of the flight simulator is twofold: serving as a tool for the investigation of flying qualities of light and ultra-light aircraft and offering a training option to the pilots of such airplanes. For these reasons the simulator cockpit has been conceived as a generic cabin of a small aircraft.

There is a strong interconnection between the simulator operating characteristics and the research topics dealt with by the author and his research group colleagues in the fields of flight testing. In particular, they have worked at the design and the whole certification process of some Italian ultra-

light aircraft gaining experience in wind-tunnel testing, in measuring airplane performances, in the analysis of flight test maneuvers, and in the estimation of aircraft aerodynamic and dynamic characteristics from flight data.

Simulator Layout

The building where the simulator is located is divided into three areas: the simulator main room, the supervisor room, the briefing room. The main room is proportionate to the horizontal and vertical motion envelope of the cockpit and to three large

fixed screens located in front of the cabin. Three static DLP projectors (DS30 from Christie Digital; 3000 lumens, 1280×1024, SXVGA) project a composite image of the virtual outside environment reproducing a horizontal field of view of 190 degrees (fig. 2). This particular projection system is preferred in car simulators and proved to be effective in the flight simulator presented here.

During the simulation the pilot is given a motion cue, which is obtained by animating the airplane mock-up with a six-degree-of-freedom motion base Maxcue 610-450-16-12 by cueSim. This motion base has six high efficiency electric actuators arranged in the Stewart platform format. The main performance characteristics of this 1000 kg maximum payload motion platform are summarized

(Continued on page 2)

Inside this issue:

A 6DoF Simulation Lab at the University of Naples	1
Hardware/Software Integration with JSBSim: Manta UAV	5
JSBSim Commander Screenshots	8
News: JSBSim Commander	9
Aerodynamic Modeling	10
Using GnuPlot with JSBSim	12
JSBSim In Use	14
Simulate This!	14

(Continued from page 1)
as follows:

	min/max pos.	V_{peak}	$Accel_{peak}$
Surge	-491/+432mm	718mm/s	$\pm 1.39\text{ g}$
Sway	-425/+425mm	712mm/s	$\pm 1.20\text{ g}$
Heave	-247/+248mm	484mm/s	$\pm 0.59\text{ g}$
Roll	-25/+25deg	50deg/s	575deg/s ²
Pitch	-24/+25deg	48deg/s	595deg/s ²
Yaw	-43/+43deg	82deg/s	1100deg/s ²



Figure 2

Inside the cockpit, a complete and customizable virtual instrument panel is reproduced by means of two tactile LCD screens (fig. 3). One screen is used to display a virtual flight panel. The second screen enables the display of what is needed by the experiments, such as moving maps or flight parameter real-time plots. A space is also reserved for a third screen. The flight controls are made up of a Cirrus II Flight Console from Precision Flight Inc., a yoke, which is included in the original flight console but whose position has been conveniently modified, and a pair of real rudder pedals. The yoke and the rudder pedals are connected to a force feedback system giving to the pilot an additional cue on the piloting effort.



Figure 3a

The simulation of aircraft motion, the cockpit instrument panel and flight controls, the outside scenery are all managed by a number of instances of Flight-Gear, running on dedicated computers, and talking to each other via Flight-Gear's net protocols.

The virtual environment is generated using PC technology, by the graphic board NVIDIA GeForce Quadro FX4000.

Two other software modules also support the simulation. The first is in charge of the motion cue and moves the Stewart platform; the second is designed to manage a control force reproduction system.

Special work on FlightGear and JSBSim code

A number of FlightGear code modifications were necessary to have the simulator facility functioning as expected. Some of them are of minor significance; some others were implied by the chosen version of FlightGear (i.e. v.0.9.8) or by the particular piece of hardware (the Cirrus flight console, for instance). As far as the author knows, this is the first application that makes use of FlightGear together with: a 6DOF motion base, a 3-screen projection system, and flight control force feedback.

As for the outside scenario, an improved management of three different but joint images had to be implemented in FlightGear in order to represent correctly a field of view of 190 degrees. A related modification regarded the connection between the visual and motion cues. The representation of the horizon has been modified and coupled with the motion cue algorithm in order to enhance the perception of acceleration within the limits of the motion base performances. Up to date this approach has proved to be quite safe with respect to the simulation sickness issue.

But there are also a couple of things here that are more relevant for this newsletter audience.

A funny one is that, right in the same period, while the JSBSim on-ground bug had been tackled and worked out in Naples, JSBSim new release 0.9.10 came out at the same time on SourceForge website with the same problem solved. Moreover, quite a bit of work has been done by the author to implement the force feedback software. That covers some to-do requests from JSBSim developer forum and, in author's opinion, his code, being written in C++ and following the JSBSim style is easily adaptable to be incorporated in future versions

(Continued on page 3)

About this newsletter ...

Edited by Jon Berndt

"Back of the Envelope" is a communication tool written generally for a wider audience than core JSBSim developers, including instructors, students, and other users. The articles featured will likely tend to address questions and comments raised in the mailing lists and via email. If you would like to suggest (or even author) an article for a future issue, please email the editor at:

Jon@jsbsim.org

(Continued from page 2)
of JSBSim.

For that reason, and for the purpose of starting an auspicious future discussion on the subject, flight control force computation is outlined here.



Figure 3b

Force Feedback

In order to reproduce realistic piloting efforts, especially for light aircraft with reversible controls, particular care is needed in the implementation of hinge moment equations.

For those who may feel uneasy with terms such as “differential equation” or “equations of motion,” please think of it simply as the equivalent of mathematical model. If this is still too much, just think of it as the one that drives the cockpit controls and gives the pilot the impression that the air (and, we’ll see, something else) tends to move the aerodynamic control surfaces (elevator, ailerons, rudder).

For the sake of example, let’s consider the motion of a conventional elevator. This model is illustrated by fig. 4, above right (“e” stands for elevator).

In a generally maneuvered flight the aerodynamic surfaces are forced to rotate around their hinge axes by three causes: the aerodynamic pressure, the action of the pilot, and surface inertia. The latter is as intuitive as the first two if simply thought of as the moving surface angular acceleration multiplied by its moment of inertia around the hinge line (i.e. the first typical term in the model equation). In the general case, one has to add to the typical inertial term two comparable fellows, which are better known as “inertial coupling” and “hinge line eccentricity” effects. These effects are all torques acting on the elevator around the hinge as well. The moment resulting from the inertial coupling actions is due to a time-varying aircraft pitch rate and/or to a combination of non-zero roll and yaw rates about airplane center of gravity. The eccentricity effects come into play when the center

of gravity of the moving part of the tail does not reside exactly along the elevator hinge line but somewhere else behind, or ahead of it. Aircraft acceleration along its own z-axis results then in a torque applied to the elevator. These additional inertial effects are lumped together in the second term at the left hand side of elevator equation of motion.

The aerodynamic (A) actions on the elevator are incorporated in the first term in right-hand-side of the model equation and depend on the current aircraft state, from tailplane geometry, and obviously from tailplane aerodynamics (fig. 5). Such a dependency requires the extension of the current JSBSim flight control definition features.

Finally, the last term in right-hand-side of the equation of fig. 4 is the torque applied by the pilot to the control surface by pushing or pulling the yoke (C stands for commanded). The actual force applied by the pilot is reduced to a moment about the elevator hinge after dividing by the dimensional gearing ratio. The gearing needs a special care because it depends from the particular arrangement of the actual aircraft command line. Generally, in a commanded maneuver the pilot control force is non-zero and is treated as an input in the model equation. A particular case handled naturally by the simulator is that of stick-free flight, that is when the applied pilot force is zero. This enables to investigate on flight qualities of a given airplane in stick-free conditions.

When the force-feedback system is matched with JSBSim, the cockpit control loads are computed at each simulated time step. Stick and pedal resulting motion is controlled between two successive JSBSim state updates with a given frequency. This is also referred to as the “inner” integration loop while the JSBSim job is then called the “outer” integration loop.

Fig. 6 illustrates the model of longitudinal control dynamics. The algorithm controlling the force cue to the pilot (i) evaluates the inertial coupling and aerodynamic terms according to the current aircraft state and to the specific control surface data, (ii) performs appropriate measures of the action exerted by the pilot on the yoke, and (iii) calculates the angular acceleration contained in the first term of the model equation, resulting from the

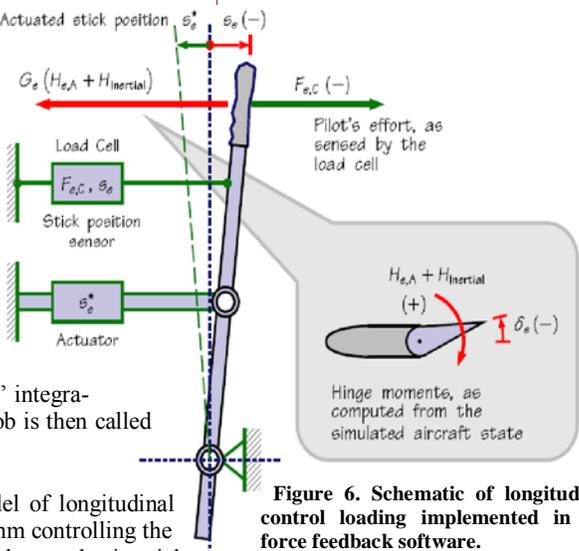


Figure 6. Schematic of longitudinal control loading implemented in the force feedback software.

Elevator Dynamics

- Inertial coupling actions
- Typical inertial term

$$I_e \ddot{\delta}_e + (-H_{e,In.}) = H_{e,A} + \frac{F_{e,C}}{G_e}$$

- Aerodynamic hinge moment
- Pilot's action

Figure 4

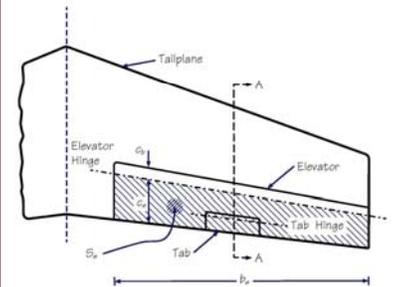


Figure 5

(Continued on page 4)

(Continued from page 3)

difference between the sensed actions and the calculated aerodynamic/inertial actions. Finally, the control loading module orders to the actuator connected to the yoke to move (accelerate) accordingly, giving the desired “feel” to the pilot. If pilot’s action is adequate to react to the feedback and keep the yoke position stationary, the flight conditions remain “stick-fixed,” or nearly so. The stick-fixed, or static equilibrium condition happens when there is a balance between pilot’s action and aerodynamic actions, with a consequent non-varying angular excursion of the cockpit control.

The actuators and the rest of the hardware of the force feedback system are designed to reproduce a realistic amount of effort required to the subject pilot. The following are the main characteristics of the force feedback system:

- Maximum force on yoke ± 400 N (push/pull)
- Maximum torque on yoke ± 40 Nm (turn left/right)
- Maximum force on each pedal 400 N (left or right)

For the sake of uniformity and for possible future integration with newer releases of JSBSim, the control loading code input file follows the JSBSim configuration philosophy. It reads similar XML files containing a description of the following items: i) control surface geometric and mass properties, ii) control system mechanical properties, i.e. various gears, iii) Control surface aerodynamic characteristics, i.e. hinge moment coefficients, iv) control surface auxiliary characteristics, v) data logging directives.

Example of coding

Here is an indicative structure of the class CFFBConfig that handles the control force computation.

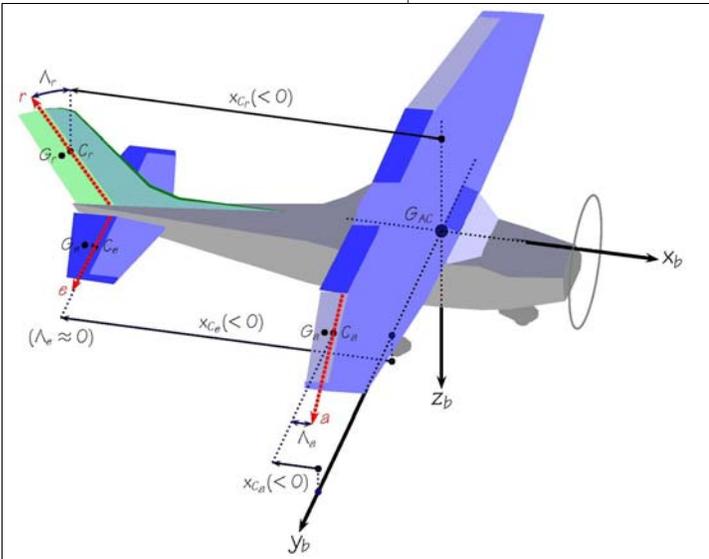


Figure 7. Some geometrical quantities that play a role in the control surface dynamics.



About the author

Agostino De Marco was born in Naples in 1969. He graduated from the University of Naples "Federico II" in 1996, earning a degree in Aerospace Engineering.

In 2001 he received a PhD in Naval Engineering from the same University.

Since 2002 he has worked as a researcher in the Department of Aeronautical Design and teaches *Flight Simulation Techniques*, and *Aircraft Flight Stability And Flight Qualities* as a member of the Engineering faculty.

www.dpa.unina.it/demarco
www.dpa.unina.it/adag

```
class CFFBConfig
{
public:
    CFFBConfig();
    ~CFFBConfig();

    bool LoadFFBFile(const string& strFileName);

    void ComputeFFB(int typeCommand, // AL , EL , RU
double& ffb , // Out
const ForceFeedBackData_t& ffldata ,
double commandPosition,
double commandRate,
double dt ,
double pilotForce);

protected:
    void ResetValues();
    void ExtractAmplification(FGConfigFile *pCfg);
    void ExtractGearing(FGConfigFile *pCfg);

    void ExtractControlMassBalance(FGConfigFile *pCfg);
    void ExtractControlMetrics(FGConfigFile *pCfg);
    void ExtractTable(FGConfigFile *pCfg , CTable2d& tab);
    bool ExtractData(FGConfigFile *pCfg , FFBDData_t& rData);

    amplification_t m_amplification;
    gearing_t m_gearing;
    control_mass_balance_t m_control_mass_balance ;
    control_metrics_t m_control_metrics[3];
    brakes_t m_brakes;
    CTable2d m_Ch[3];
    CTable2d m_ChTrim[3];

    FFBDData_t m_FFBDData[3];
};
```

The aim of this project was to develop a HITL (Hardware In-The-Loop) simulation of an autonomous air vehicle using the FlightGear flight simulator. A standard radio control transmitter would provide manual control of the aircraft and aircraft state data would be returned via a simulated communication downlink. This would enable an external control system to be connected and tested.

The operation of autonomous UAVs is similar to that of a radio controlled aircraft, and most of the hardware is common. An existing radio controlled aircraft had previously been implemented by the author in FlightGear, and was therefore readily available. This formed the starting point of the project. It was assumed that the aircraft would be equipped with a horizon sensing attitude stabilisation system. This is a highly effective and low cost way of stabilising the aircraft in pitch and roll. An example of such system is the FMA Co-Pilot CPD4, shown in Figure 1.



Figure 1. FMA Co-Pilot CPD4.

The attitude stabilisation system uses horizon detectors, such as infrared or thermopile sensors, to detect the aircraft attitude. The aircraft attitude is then fed into the stabilisation system forming a simple feedback loop. This helps maintain the aircraft level in both axes. The attitude stabilisation system was modeled in the aircraft flight dynamics model (JSBSim), as it constitutes a low-level part of the overall control system.

Through the course of the project, two programs were developed to handle and log the aircraft data, output by the simulator. The aircraft pitch and roll motions were analyzed, and, from the aircraft state data logged during “virtual flight tests” (using FlightGear/JSBSim), an approximation for the longitudinal response was derived. This enabled a Matlab Simulink model to be produced, which was used as a development tool for a low level pitch attitude stabilisation system. A roll attitude stabilisation system was also implemented by testing different values of gain in the feedback loop and choosing the one that gave the best handling qualities.

“Flight Tests” and Modeling

Manual control of an autonomous air vehicle is often provided through a standard radio control

transmitter. An off-the-shelf USB interface cable provided a link between the PC running the flight simulator and a transmitter, for manual control.

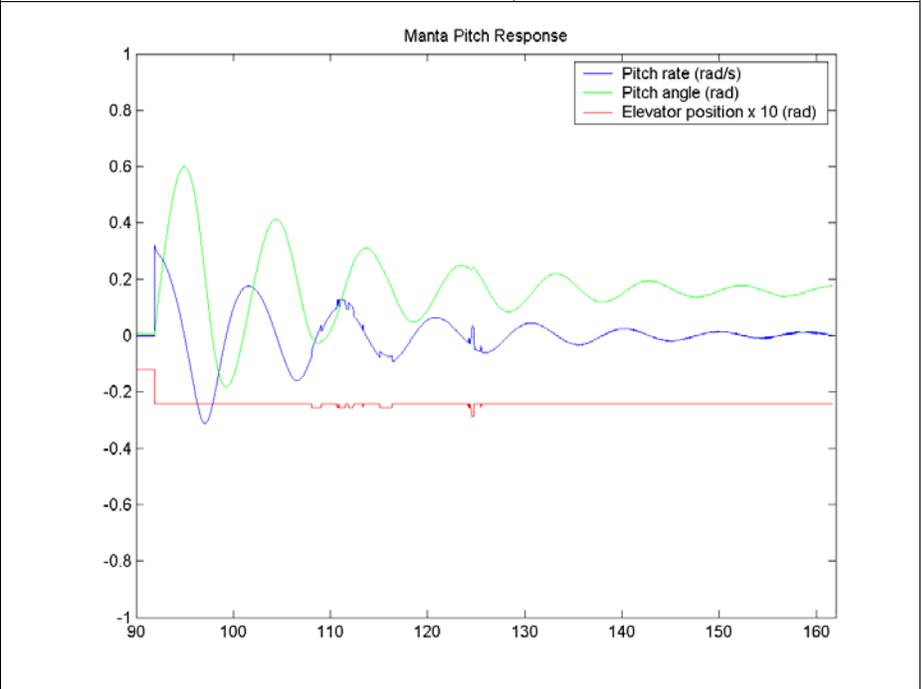


Figure 2. Manta UAV pitch response.

Figure 2 shows a plot of the ‘Manta’ data, logged during a virtual flight test (in FlightGear). It shows the response of the aircraft to a step input, produced by a half deflection of the trim command on the radio control transmitter (0.01 rad deflection). The plot of the elevator position was scaled up by a factor of 10 to make it clearer to see.

The aircraft pitch response is a decaying oscillation with an overshoot of 300% and a natural frequency of approximately 10 seconds. This response is predominantly a phugoid motion, with an almost non-existent short period response. The full longitudinal transfer function for pitch angle response to an elevator input is given in equation (1), this transfer function consists in two modes of response: the short period mode and the phugoid mode. Given the plot of the aircraft pitch angle response, it was apparent that the response could

$$(1) \quad \frac{\theta}{\eta} = A_{\theta} \frac{T_{\theta_1} \cdot s + 1}{[s^2 + 2 \cdot \zeta_{ph} \cdot \omega_{ph} \cdot s + \omega_{ph}^2]} \frac{T_{\theta_2} \cdot s + 1}{[s^2 + 2 \cdot \zeta_{sp} \cdot \omega_{sp} \cdot s + \omega_{sp}^2]}$$

$$(2) \quad \frac{\theta}{\eta} = A_{\theta} \frac{T_{\theta_1} \cdot s + 1}{[s^2 + 2 \cdot \zeta \cdot \omega \cdot s + \omega^2]}$$

be modeled with the phugoid term alone, given in equation (2). This simplification significantly eased the analysis of the aircraft stability.

(Continued on page 6)

(Continued from page 5)

From Figure 2, the frequency, damping, time constant and pitch gain were derived giving the following transfer function:

$$(3) \quad \frac{\theta}{\eta} = -\frac{30s + 5.95}{s^2 + 0.117s + 0.4}$$

A Matlab Simulink model of the aircraft was implemented, as a tool for the development of the stability augmentation system. The model is shown in Figure 3. The graphs produced by the Simulink model, also given in Figure 3, accurately match those obtained from the virtual flight tests on the simulator (FlightGear/JSBSim), thus validating the aircraft model.

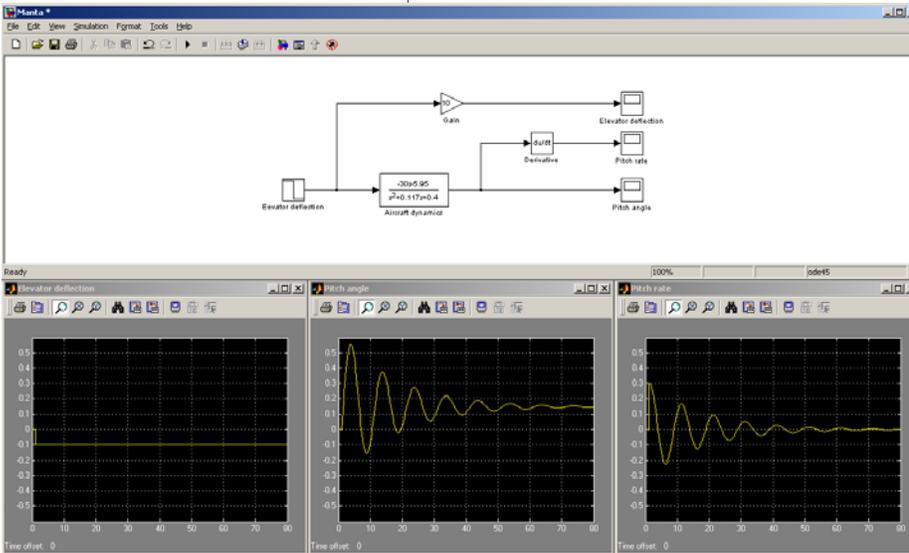


Figure 3. Simulink model of the Manta UAV.

Stability Augmentation

From the virtual flight tests and the plots of the data, it was clear that the ‘Manta’ UAV is an unsteady aircraft that requires constant pilot attention to maintain straight and level flight. This is not acceptable for an autonomous air vehicle. An attitude stabilisation system would help make the aircraft steadier.

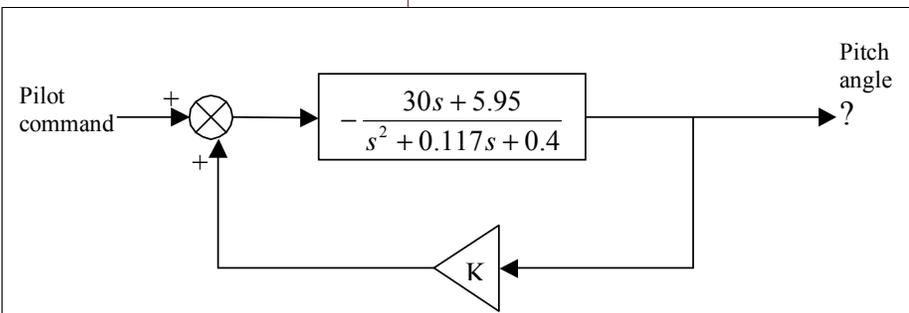


Figure 4. Pitch angle feedback to elevator.

As previously shown, the Manta UAV pitch response is dominantly a phugoid motion and pitch angle feedback is an appropriate way of damping phugoid oscillation. The technique is the same as that used in early autopilots. Since the pitch angle is fed back, the elevator moves to oppose any pitch angle error. As the phugoid mode has fairly large

pitch angle response, this system is well suited for damping phugoid response. Figure 4 illustrates how the system can be implemented.

In order to determine the required gain K, a root locus was plotted in Matlab (Figure 7.2) and, with the help of the ‘rlocfind’ function, the required gain value for a damping of approximately 0.7 was derived:

$$K_{\zeta=0.7} = 0.032$$

Having obtained the amount of gain needed, the pitch angle feedback loop was added to the Simulink model. The results produced are shown in Figure 5 (next page).

The aircraft JSBSim flight control system was then altered to provide the feedback loop. For this two extra blocks ‘Pitch Feedback’ and ‘Elevator Control Aug’ were added. The ‘Pitch Feedback’ block takes the pitch angle and multiplies it with the desired gain. The ‘Elevator Control Aug’ block sums the pitch angle feedback and the open-loop elevator control to give the elevator position. The pitch control section of the aircraft FDM is given below [Ed. I modified Sam’s original specification to reflect the new JSBSim-ML v2.0 format]:

```
<summer name="pitch trim sum">
<input>fcs/elevator-cmd-norm</input>
<input>fcs/pitch-trm-cmd-norm</input>
<clipto>
<min> -1 </min>
<max> 1 </max>
</clipto>
</summer>

<aerosurface_scale name="elevatorctl">
<input> fcs/pitch-trim-sum </input>
<min> -0.5 </min>
<max> 0.5 </max>
</aerosurface_scale>

<pure_gain name="pitch feedback">
<input> attitude/theta-rad </input>
<gain> 0.032 </gain>
</pure_gain>

<summer name="elevator control aug">
<input>fcs/elevatorctl</input>
<input> fcs/pitch-feedback </input>
<clipto>
<min> -0.5 </min>
<max> 0.5 </max>
</clipto>
<output>fcs/elevator-pos-rad</output>
</summer>
```

Additional virtual flight tests were carried out and the necessary flight data recorded. The response observed is plotted in Figure 6 (next page). The plots accurately matched the Simulink model predictions. In terms of handling through the radio control transmitter, the aircraft felt much more stable, as hoped. Sharp inputs were much better damped and the aircraft regained stability faster. Pilot workload was thus greatly reduced.

(Continued on page 7)

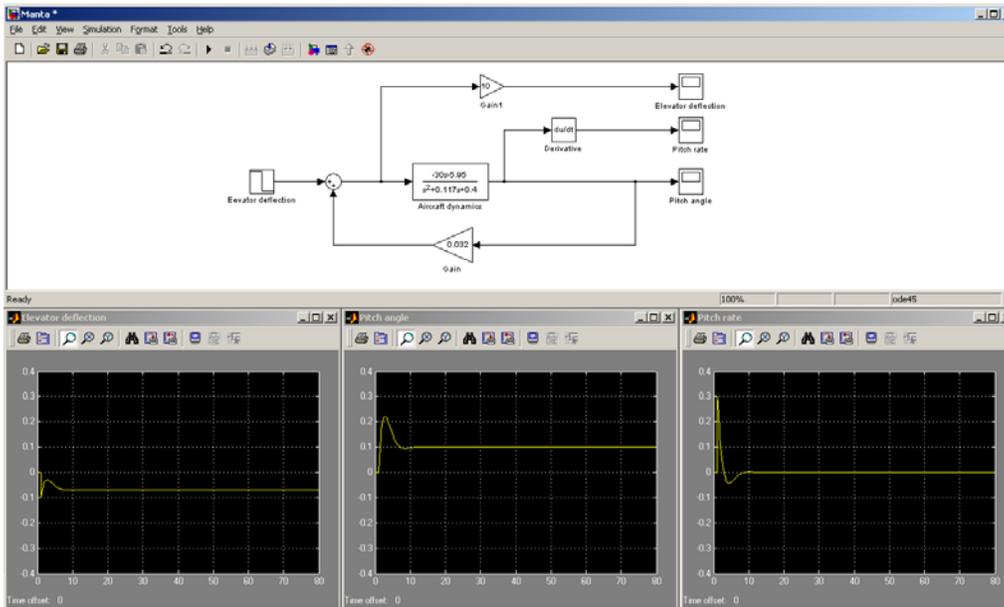


Figure 5: Simulink Model Predictions for Pitch Angle Feedback ($K=0.032$)

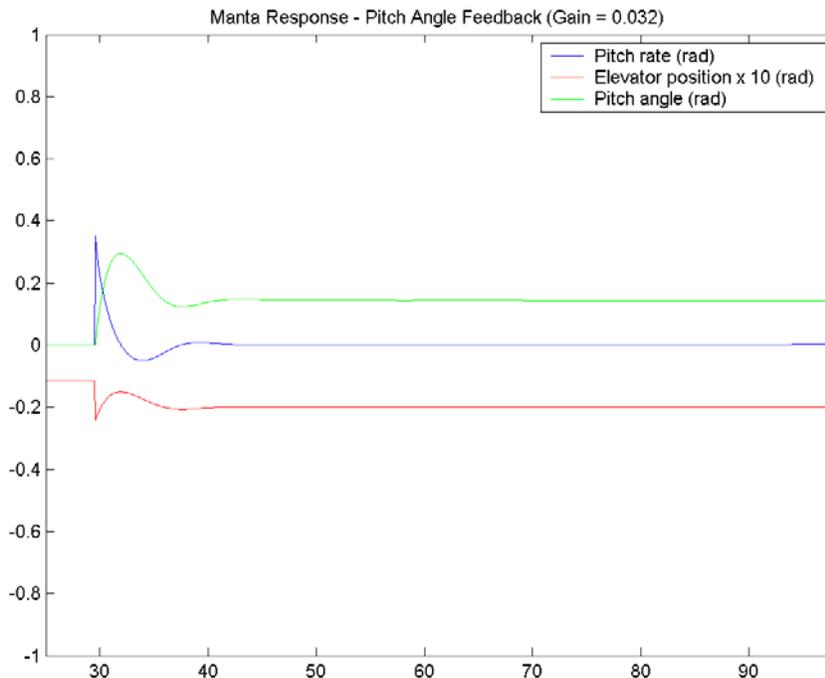


Figure 6. Aircraft Pitch Response With Pitch Angle Feedback

(Continued from page 6)

Results

From the correlation between the Simulink predictions of the pitch stabilisation and the flight test results, it was clear that the aircraft could be modeled using Matlab Simulink and that the model could be used as a development tool for an attitude stabilisation system.

It was also apparent that an attitude stabilisation system, based on attitude feedback, was an

effective way of stabilising the aircraft and that embedding it in the JSBSim flight dynamics model was justifiable.

More information and aircraft model available on the project webpage (<http://seis.bris.ac.uk/~sh1823/Manta.htm>).

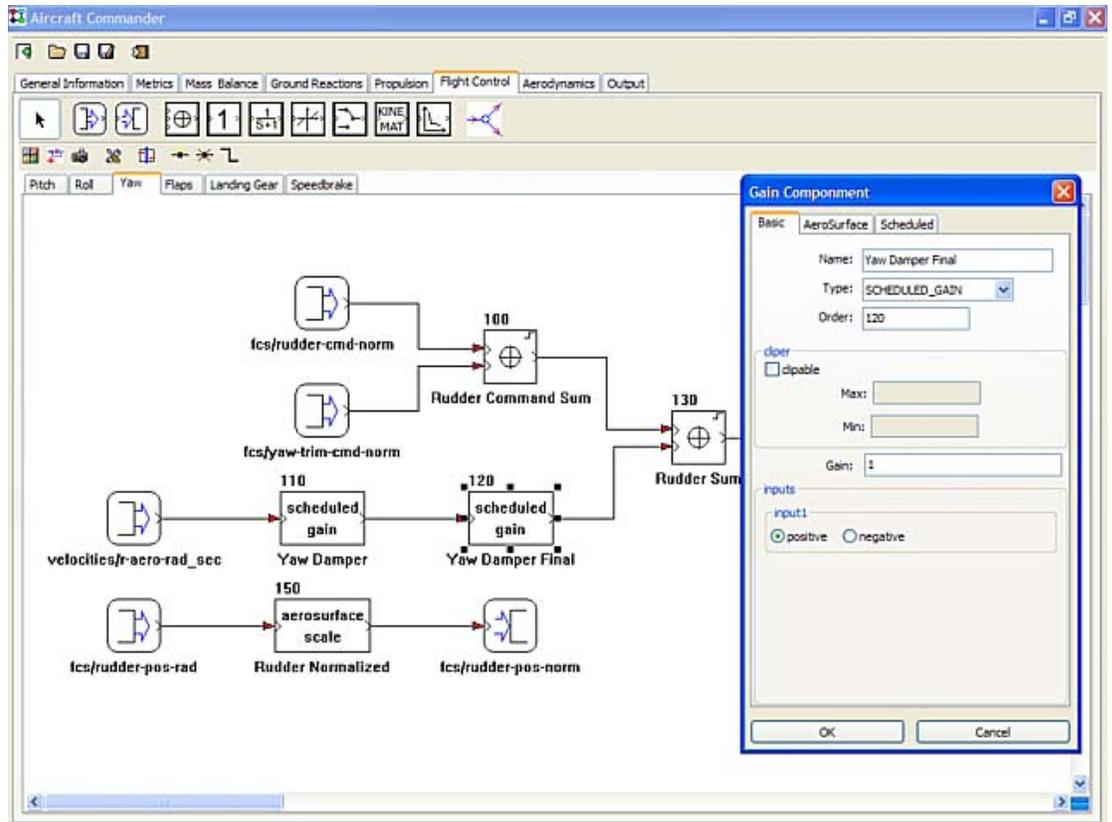


About the author

Sam Heyman is in his final year of 'Avionic Systems Engineering' at the University of Bristol in England. He spent his third year abroad, at Linköping University in Sweden, where he took part in the 'Manta' UAV group project. He was responsible for modeling the aircraft in FlightGear in order to help predict the flying characteristics. He has recently completed his research project, in which he used the FlightGear model of the 'Manta' to set-up a hardware-in-the-loop simulation of an autonomous air vehicle. The main focus of the project was to develop and embed a stability augmentation system in the flight dynamics model.

In his spare time Sam is a keen skydiver and micro-light pilot. He also enjoys flying RC aircraft.

JSBSim Commander Screenshots



Above: Flight control systems for JSBSim aircraft can be created and modified in the flight control system editor.

Below: The propulsion system editor allows selection of engines, placement and orientation.

The screenshot shows the 'Propulsion' editor in JSBSim Commander. The main window contains a table of 'Subscribed Engine(s)':

Engine	Thruster
CFM56	direct
CFM56	direct

 An 'Engine/Thruster Setup' dialog box is open, showing configuration for a new engine and thruster:

- Engine Name:** CFM56
- Location:** x = 540, y = -193, z = -40
- Orient:** roll = 0, pitch = 0, yaw = 0 (DEG)
- Feed:** 0.2
- Thruster Name:** direct
- Thruster Location:** x = 540, y = -193, z = -40
- Thruster Orient:** roll = 0, pitch = 0, yaw = 0 (DEG)

 The background window also shows 'Available Engines', 'Available Thrusters', and 'Tanks' sections.

Tank	Weight	Location
FUEL	10000 LBS	[520, -80, -18] in
FUEL	10000 LBS	[520, 80, -18] in
FUEL	4000 LBS	[480, 0, -18] in

JSBSim Commander Source Opened

Last year Back of the Envelope featured a story about an application called, JSBSim Commander. The application was meant to provide a GUI editor for JSBSim aircraft and engines, and also to act as an interface to running JSBSim and analyzing results. JSBSim Commander was inspired by an original application created by Matthew Gong. Matthew has taken his version JSBSim Commander quite far—to the point of actually working, and his version is written using wxWidgets, a cross-platform GUI toolkit. The project is in the process of becoming an open source project hosted on SourceForge. The web site is at www.sf.net/projects/jsbsimcommander. The source code has now been uploaded to the web site, and screen shots (see page 8) have been uploaded.

Aerocross Systems and JSBSim

Aerocross Systems, Inc. is utilizing JSBSim as the 6DOF simulation core in their real-time, flight hardware-in-the-loop test platform. This platform is used for flight hardware/software development, integration, and testing in support of several Unmanned Aerial System (UAS) programs. Aerocross Systems has also successfully integrated JSBSim with commercial off-the-shelf and open source Control System Computer Aided Design (CSCAD) packages to support flight control law development and analysis.

New JSBSim Version Released with New Flight-Gear Version

The new version of FlightGear incorporates some significant modifications and enhancements with the newest revision of JSBSim, v0.9.10. This release of JSBSim features a revamped specification language that is more well-formed XML. The new version of the JSBSim config file format is called JSBSim-ML v2.0 (JSBSim Markup Language).

Among the new capabilities afforded by JSBSim-ML v2.0 are:

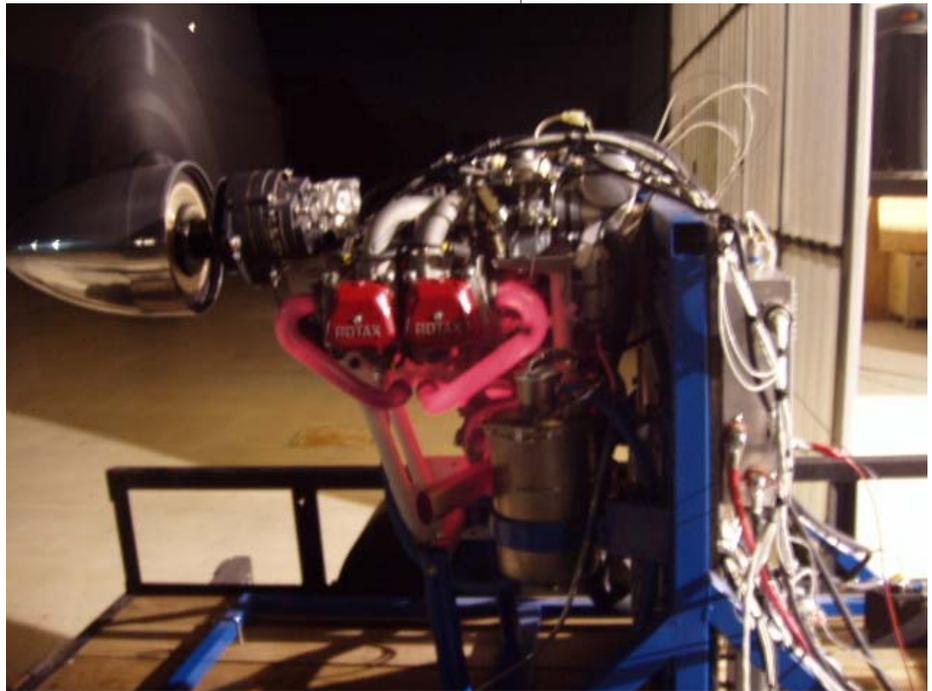
- There is an early beta version of an XML schema (JSBSim.xsd) that can be used to validate any new aircraft config file.
- There is an early beta version of an XML stylesheet (JSBSim.xsl) that can be used to format an aircraft config file for presentation in a web page.
- Units can now be specified for some aircraft parameters.
- Functions can be specified in an aerodynamic "coefficient" definition. Etc.

Several new capabilities have been added in code, too:

- An experimental ground reactions enhancement reduced ground jitter at low or zero velocity for most aircraft. This is still a work in progress.
- More error handling, more robust handling of input files.
- A new turboprop engine model has been added.
- The propeller model has been enhanced. Sensors can be modeled in the flight control definition for an aircraft.
- Functions can be modeled as part of a flight control definition. Etc.
- Gimbaling can be done for rockets, etc.

As part of this significant upgrade, be advised that not all aircraft have been updated to the new JSBSim-ML v2.0 format. Some aircraft have temporarily lost aerosurface animations. It is not difficult to add these features back. Instructions and tools for converting from the old format to the new are available at the JSBSim web site: www.jsbsim.org. Any problems with a JSBSim aircraft model should be promptly reported to either this list or the JSBSim mailing list.

For more information on the new JSBSim capabilities and on JSBSim-ML, see the project web site at www.jsbsim.org.



Aerocross Systems, Inc. is testing a Rotax 914 engine in this image, exhaust pipes glowing with gases passing through at >1650°F! More information on the Rotax 914 can be found here: http://www.rotax-aircraft-engines.com/aircraft/aircraft.nsf/eng_914F?OpenPage

Aerodynamic Modeling in JSBSim

Jon S. Berndt

The thought of creating an aircraft flight model in JSBSim can seem daunting. One must gather a considerable amount of information about an aircraft before it can be accurately modeled. To help guide flight modelers in the task of creating a flight model, a document is currently being written that serves as a journal of this author's efforts towards creating an accurate flight model of the Beech A99A Airliner. What follows here are excerpts from that document. It is hoped that the full document can be finished and posted on the JSBSim web site in the near future (before the next newsletter comes out in July).

As an example, let's examine the lift force due to angle of attack (α). We know that increasing the angle of attack increases lift – up to a point. Lift force is the product of dynamic pressure (q_{bar}), wing area (S_w), and lift coefficient (C_L). In this case, the lift coefficient is determined via a lookup table, using α as a lookup index into the table (Listing 1). What the function results in is for lift due to α to be calculated each pass through the aerodynamics calculations in JSBSim. The value of the function is the product of q_{bar} , wing area, and lift coefficient as determined by the lookup table. For instance, if α (in radians) is 0 degrees, the lift coefficient is 0.240.

```

Aerodynamic forces and moments [in JSBSim] are modeled using the classic coefficient build up method. The lift, drag, and side forces, and the pitch, roll, and yaw moments need to be calculated to determine the flight path that an aircraft will take. Many factors contribute to each of the forces and moments acting on an aircraft. For instance, lift is a function of the inherent lift of the aircraft at zero degrees angle of attack, the lift due to a change in angle of attack from zero degrees, the lift due to a deflection of the horizontal stabilizer (pitch command), the lift due to pitch rate, the lift due to flap deflection, etc. Likewise, for the other five axes, the total force or moment is the sum of the individual contributions.

The aerodynamic definition for a JSBSim aircraft flight model is comprised of six sections corresponding to the forces and moments about three axes each (for a total of six degrees of freedom). In each section, any number of functions are present that define individual contributions to a force or moment. Additionally, "global" functions may be defined at the beginning of the aerodynamics section, outside of any axis section, for later use by another function.

A function in JSBSim may be composed of common arithmetic and/or trigonometric operations on one or more operands. MathML, the XML markup language for mathematical constructs, inspires the format of the JSBSim function but the JSBSim function is much simpler. A cue is also taken from AeroML, the proposed standard under consideration by AIAA for exchange of simulation models.

```

```

<function name="aero/coefficient/CLalpha">
  <description>Lift_due_to_alpha</description>
  <product>
    <property>aero/qbar-psf</property>
    <property>metrics/Sw-sqft</property>
    <table name="CL">
      <independentVar lookup="row">aero/alpha-rad</independentVar>
      <tableData>
        -0.20 -0.720
        0.00 0.240
        0.22 1.300
        0.60 0.664
      </tableData>
    </table>
  </product>
</function>

```

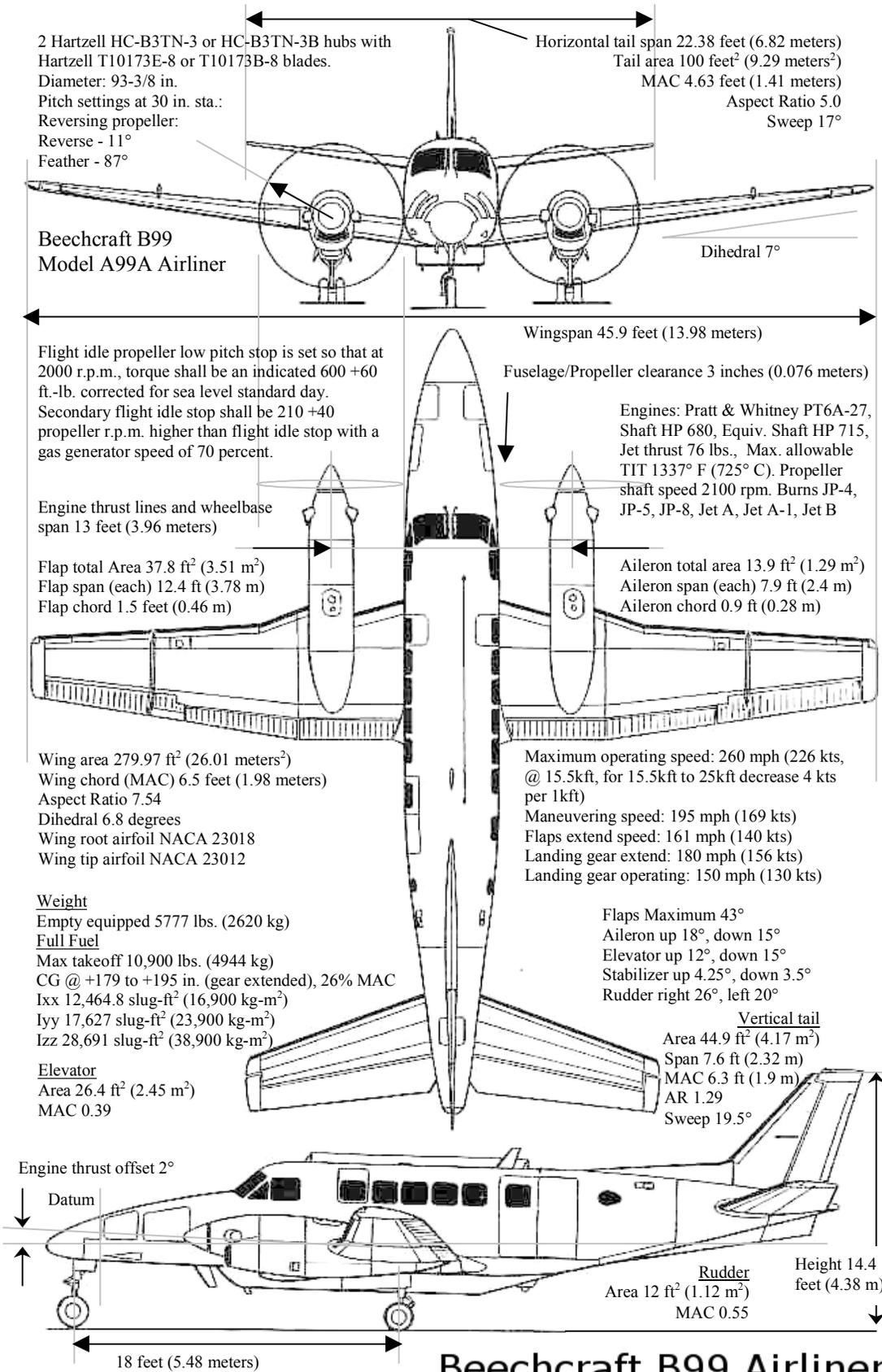
Listing 1. Lift coefficient as a function of α .

So, where do we get this information about the Beech 99? Aeromatic has created for us a boilerplate configuration file populated with plausible values for many aerodynamic terms. However, there are a lot of refinements that can – and *should* – be made. First of all, the document mentioned at the beginning of this article contains specific information about the aerodynamic qualities of the aircraft. Let's look at how to *tweak* a JSBSim aerodynamics specification for a specific aircraft.

Let's consider $C_{L\alpha}$ first. The symbol $C_{L\alpha}$ is shorthand for $\delta C_L / \delta \alpha$, or the *change in lift coefficient with a change in alpha*. This is just the slope of the lift coefficient curve (see Fig. 5 for an example). The lift curve slope for an ideal 2D airfoil section is 2π (per radian). For a cambered airfoil, the lift curve does *not* pass through zero – that is, there is an inherent lift to the airfoil even at zero angle of attack. Furthermore, a real wing (i.e. *not* a 2D airfoil section) is not as efficient as an airfoil, and the slope of the lift curve will be somewhat less than 2π . Additionally, a real wing will eventually stall at a particular angle of attack, perhaps around 12-16 degrees. The information given in the Beech 99 report is valuable, however, some adjustments will have to be made to account for stalling. The value of $C_{L\alpha}$ defines a slope – a line that goes on forever! [to be continued later ...]

The aerodynamic definition for a JSBSim aircraft flight model is comprised of six sections corresponding to the forces and moments about three axes each (for a total of six degrees of freedom). In each section, any number of functions are present that define individual contributions to a force or moment. Additionally, "global" functions may be defined at the beginning of the aerodynamics section, outside of any axis section, for later use by another function.

A function in JSBSim may be composed of common arithmetic and/or trigonometric operations on one or more operands. MathML, the XML markup language for mathematical constructs, inspires the format of the JSBSim function but the JSBSim function is much simpler. A cue is also taken from AeroML, the proposed standard under consideration by AIAA for exchange of simulation models.



When crafting a flight model for any specific aircraft, collecting as much data as possible about that aircraft is essential to the process. The diagram at left was used by the author to collect and record much relevant information about the chosen aircraft, the Beech A99A Airliner.

This diagram is taken from the paper "A Journal for the Creation and Refinement of a JSBSim Aircraft Flight Model", to be released in the coming months.

Beechcraft B99 Airliner

Using gnuplot with JSBSim

Jon Berndt

According to the project web site:

Gnuplot is a portable command-line driven interactive data and function plotting utility for UNIX, IBM OS/2, MS Windows, DOS, Macintosh, VMS, Atari and many other platforms. The software is copyrighted but freely distributed (i.e., you don't have to pay for it). It was originally intended as to allow scientists and students to visualize mathematical functions and data. It does this job pretty well, but has grown to support many non-interactive uses, including web scripting and integration as a plotting engine for third-party applications like Octave. Gnuplot has been supported and under development since 1986.

```
{with <style>}
{, {definitions,} <function> ...}
```

A relatively easy way to learn gnuplot is to start it up and play with it. Typing “gnuplot” will start the application and give the prompt. For your first command, type “plot 1”. The result is a chart with a horizontal line at $y=1$. Let’s leave the simple stuff behind and go straight to a JSBSim data log file in comma separated value (csv) format.

When working with csv files, the first thing that needs to be done is to tell gnuplot that it should use a comma “,” as a data delimiter:

```
set datafile separator ","
```

The next thing is to tell gnuplot which data to plot. In my setup, I simply type this command:

```
plot "c172outE0.csv" using 1:37
```

which results in the plot at left.

The plot command is specifies that column 1 data should be used for the X axis, and column 37 data should be plotted on the Y axis. In JSBSim, column 1 (there is no column 0) is always Time. In the particular data log (“c172outE0.csv”), column 37 is Distance AGL.

Unfortunately, it can be kind of a pain to find which terms you want to plot. One convenient script of shell commands can be used to get an alphabetically sorted list of terms and the column number for that term in the data log:

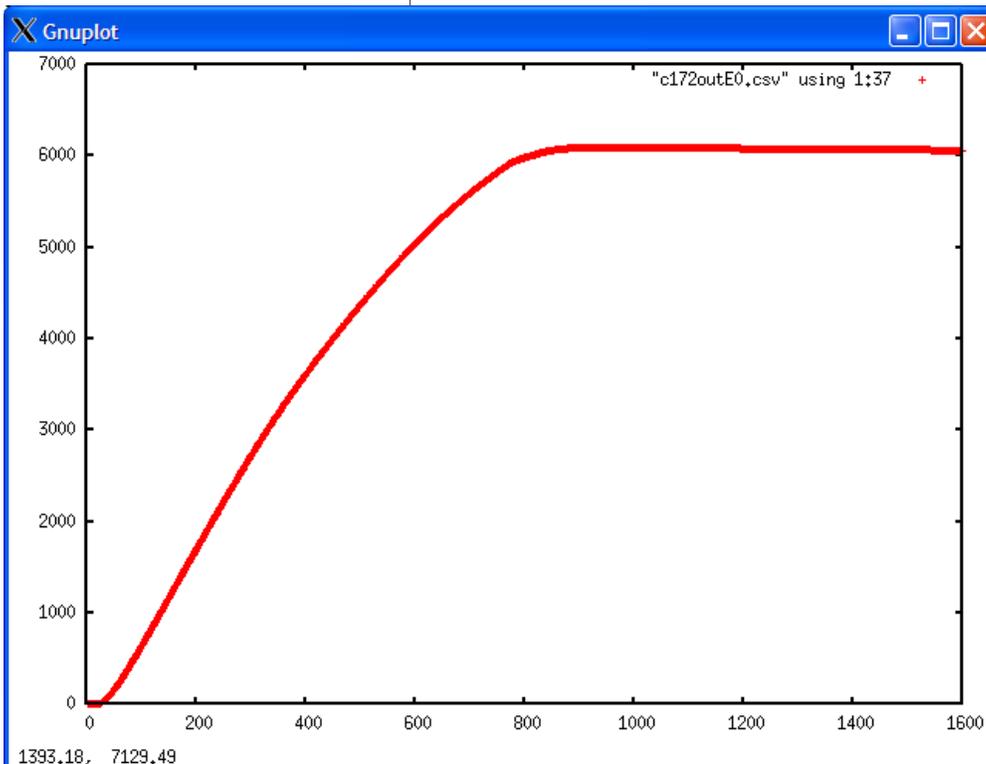
```
head -1 $1 | sed "s/, /\\n/g" |
cat -b | sort -fdk2
```

I have a script called index.sh that contains the above commands. For that script to work, the filename containing the data must be given on the command line as an argument:

```
./index.sh c172outE0.csv
```

The output from the above command is, **column variable**:

```
33 Alpha
66 Alt Error Lag
29 Altitude
65 Altitude Error
69 AP Alt Hold Switch
34 Beta
73 Control Summer
37 Distance AGL
19 Drag
```



Gnuplot is a great complement when used with JSBSim data logging capabilities. It’s not a typical plotting tool, in that directives to gnuplot that describe how a plot is to be constructed and how it should look are given on the command line or through a script file. It takes a little getting used to, but once you become familiar with it, it is found to be easy to use and powerful.

You can download gnuplot from www.gnuplot.info. For those with cygwin or Linux, it may already be on your system. Typing “gnuplot” at the command line will show that. One of the most powerful commands to use with gnuplot is the plot command. It has a lot of options:

```
plot {<ranges>}
{<function> |
{<datafile>" {datafile-modifiers}}
{axes <axes>} {<title-spec>}
```

(Continued on page 13)

(Continued from page 12)

```
74 Elevator  
... ..  
etc.
```

Labels, titles, a key, line style can be added. Labels, the title, and margins are set before the call to plot:

```
set title "Altitude"  
set ylabel "Altitude (ft) "  
set xlabel "Time (sec) "  
set key on left top box  
set lmargin 12  
set rmargin 5  
set tmargin 5  
set bmargin 5
```

Several traces can be made on one plot simply by specifying several items to plot:

```
plot "c172outE0.csv" using 1:37 title "E0" with lines lw 1,\  
"c172outE1.csv" using 1:37 title "E1" with lines lw 1,\  
"c172outE2.csv" using 1:37 title "E2" with lines lw 1,\  
"c172outE3.csv" using 1:37 title "E3" with lines lw 1,\  
"c172outE4.csv" using 1:37 title "E4" with lines lw 1,\  
"c172outE5.csv" using 1:37 title "E5" with lines lw 1
```

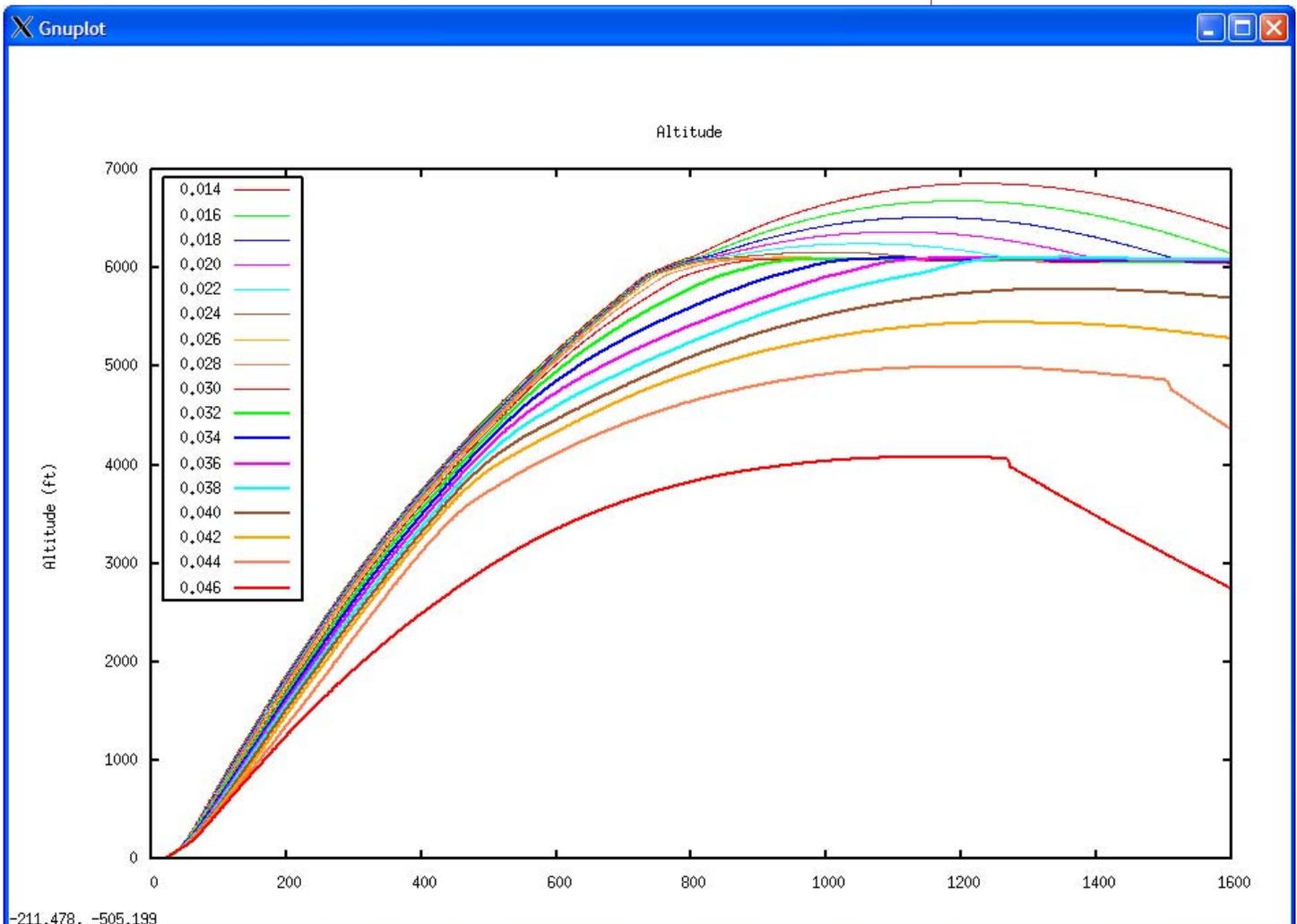
In the above example, the "with lines lw 1"

options tell gnuplot to draw the traces with a solid line of width one (see figure below).

The plot below was created in a study to evaluate a series of gains to use in a proportional controller for an altitude hold autopilot. A script file was created that used sed to change the gain, and run JSBSim (specifying an output file to store the data log in – this is a new capability with stand-alone JSBSim). When all the runs were completed, gnuplot was used to plot the altitude traces in each of the data log files.

Gnuplot commands can be stored in a script that creates a series of plots at once. The output type (a "terminal" type) can be specified, as well, so that output can be created in several popular formats, including Adobe PDF, png, gif, SVG, X11 screen display, etc.

Gnuplot is a very capable application with a lot of options. However, only a few are really needed to get a lot of use out of it, and creative use of gnuplot can be a valuable aid in designing control systems and debugging.





Highlighted References

Online:

"Measurement of the Effect of Accelerations on the Longitudinal and Lateral Motion of an Airship Model"
<http://naca.central.cranfield.ac.uk/reports/arc/rm/613.pdf>

JSBSim in Use ...

In a recent article for a bi-monthly publication, these uses for JSBSim were listed:

- Teaching modeling and simulation concepts as part of the Modeling and Simulation Familiarization Tool developed at the Air Force Research Laboratory (Wright-Patterson AFB).
- Providing flight modeling capabilities for a battlefield simulation framework developed by the Man-System Interaction department at the Swedish Defence Research Agency.
- Serving as the flight model for a dynamic soaring study at Sandia Laboratories.
- Serving as the flight model for an Ares Mars Airplane educational demonstrator (see www.redcanyonsoftware.com).
- Serving as the 6DOF simulation core in a real-time, flight-hardware-in-the-loop test platform that is used for flight hardware/software development, integration, and testing in support of several Unmanned Aerial System (UAS) programs.
- JSBSim has also been successfully integrated with COTS and open source Control System Computer Aided Design (CSCAD) packages to support flight control law development and analysis.

Simulate This!

Worldwide Aeros Corporation has conceived a multi-use, heavier-than-air dirigible. From their web site: "*The word Aeroscraft describes a flying craft that derives its lift partially from lifting gas (helium) and partially from the traditional dynamic lift created by the shape of the body. Aeros has designed a craft that takes advantage of both methods of lift. This design approach has resulted in the evolution of a craft that can fly further, operate more economically, and lift more than any other craft in the skies. The Aeroscraft has been designed to fill the very widest range of missions and conditions.*"

Modeling an airship adds a whole new dimension, because not only can one add buoyancy to the list of items that can cause forces and moments on an aircraft, but also the effects of "virtual mass" must be accounted for. When an aircraft (or submarine, fish, etc.) is moving through a medium of about the same (or greater) density, when accelerating the aircraft itself, some of the medium (e.g. air in this case) around it must also be accelerated. The same is true when decelerating.

There has been interest in adding this capability to JSBSim, and there is at least one effort aimed at doing so.



Visit us on the web at:
www.jsbsim.org

