The quarterly newsletter for JSBSim, an open source flight dynamics model in C++

Volume 1, Issue 3

15 October 2004

Back of the Envelope

Inside this issue:

JSBSim Presented at the 2004 AIAA MST Conference

Report on the AIAA Modeling and Simulation Conference in Providence

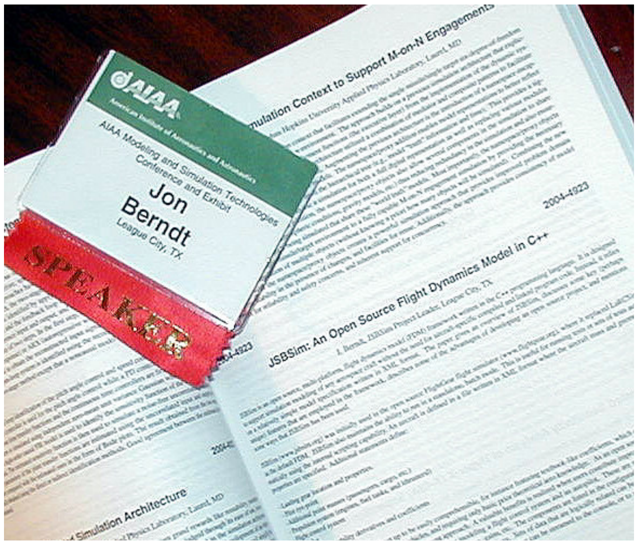
From August 15th through the 18th I attended the 2004 AIAA Modeling and Simulation Technology Conference in Providence, Rhode Island. It was well worth the expense for me. If my memory serves me well, there were over 800 attendees at the co-located conferences that were held together. Several hundred papers were fielded by authors from the U.S., Australia, China, Germany, Canada, France, Great Britain, and Italy, and elsewhere.

I had been worried in the days prior to the conference that my airline flight would be canceled – or worse – *not* canceled and I'd fly smack into really bad weather as the remnants of Hurricane Charley arrived in Providence at the same time my flight did. As it turned out, Charley arrived a few hours earlier than my flight did, and was nothing more than a slight depression when it passed through. It was very overcast with a low ceiling as we approached the runway. As the houses became visible when we dropped down below the cloud deck only several hundred feet AGL, I knew I was in the northeast. It was beautiful.

Monday morning started early for speakers, with a continental breakfast and introductions. The session chair for my speaking slot was Michael Madden (NASA Langley), who has written several papers presenting LaSRS++ (the Langley Standard Real-time Simulation in C++) and uses of it. Also speaking in the session was Patricia Hawley, who was presenting a paper on an OO simulation architecture.

One of the notable presentations in the morning was by Ian McManus, from Queensland University of Technology, Brisbane, Australia. Mr. McManus presented the paper, "A 'Hardware in Loop' Simulation Environment for UAVs Operating in Civilian Airspace." To my surprise, I found that they were using FlightGear as the front end for visuals, Matlab and Simulink with the AeroSim blockset (from u-dynamics.com) and some of the JSBSim aircraft models (which can be read through the AeroSim blockset). In speaking with a few of the attendees, I found great excitement about open source software, and FlightGear in particular.

Finally, after lunch came the moment I had been waiting for months to experience. I presented JSBSim to an interested audience


AIAA
AIAA Modeling and Simulation Technologies Conference and Exhibit
Jon Berndt
League City, TX
SPEAKER
JSBSim: An Open Source Flight Dynamics Model in C++

of about 40 people. The presentation went much better than in practice – I felt very much at ease despite worrying about my abilities as a speaker. The presentation went about 25 minutes, and there were several good questions asked afterwards. [The set of PowerPoint slides I presented are online at

(Continued on page 6)

The Proportional-Integral-Derivative Controller

By Roy Vegard Ovesen & Jon Berndt

One simple way to control something is to use feedback control. In feedback control, one looks at the value to be controlled, compares this value with the desired value, and adjusts the control input accordingly.

Take driving an automobile as an example. We look at the speedometer, compare the speed with (for example) the speed limit, and we adjust the gas pedal appropriately. If we are driving slower than the limit we push down on the gas pedal. If we are driving faster than the limit, we let go of the gas pedal or alternatively push down on the brake pedal. Our everyday lives are full of examples of feedback control.

Proportional-Integral-Derivative controller

A very commonly used feedback controller is the PID controller. It calculates an output based on a proportional gain, the integral, and the derivative of the error signal. The error signal is the difference between the *desired* (*reference*, or *target*) value and the *input* (or actual *sensed*) value:

$$e = r - i$$

where e is the error signal, r is the reference and i is the input signal. First, let's discuss the individual control elements.

Proportional element

In proportional control, the output is proportional to the error signal. Let us imagine that the proportional *gain* is 1. This means that the output from the controller is equal to the error signal, and consequently a large error signal input to the proportional controller will result in a large controller output. A proportional controller is essentially an amplifier with an adjustable gain.

Integral element

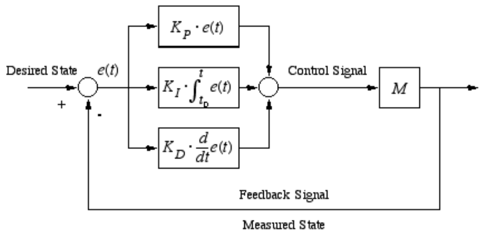
In integral control, the output is *changed* at a rate proportional to the error signal input. If the error is zero, then the integral control output will be stationary (not necessarily zero, though). If the error input is 1 and the integral controller gain is 1, then the output will have a slope of 1. If we start at time zero with the integrator output at zero, then in this example the integrator output would be 1 at elapsed time one second, 2 after two seconds, and so on. Integral controllers can account for a bias or drift in a system.

Derivative element

Derivative control (or, *rate control*) is not used alone, because it only acts during transient periods. In derivative control the magnitude of the controller output is proportional to the rate of change of the error that is input to it. Derivative control has an anticipatory quality. However, care should be used with this control action, because it can amplify noise, and can lead to saturation of the output (for instance, it can cause an aerosurface to hit its limit) - though both integral and proportional can cause this too.

The figure below shows a diagram of a PID controller. [Note that control systems are typically made up of many more components than a PID controller. The control system for an aircraft such as the F-16 or space shuttle, for instance, includes switch, scheduled gain, hysteresis, deadband, gain, summer, etc. control components. JSBSim can model all of the various components.]

The error signal goes through each element of the PID controller in parallel (not serially), and calculates the output signal



from the error signal according to this formula:

$$o = o_0 + K_p \cdot e + K_i \int e \cdot dt + K_d \cdot \dot{e}$$

The first element o_0 is the output bias. This constant value corresponds to a working point for the controller. When the error signal e is zero the output o is equal to o_0. For the sake of simplicity, we will imagine

(Continued on page 3)

News Items

Development and coding is well under way for the upcoming move to the use of a more robust and proper XML parser (based on SimGear's EasyXML wrapper of eXpat by David Megginson). A new C++ class has been created that extends EasyXML for JSBSim use. Also written is a utility class that encapsulates an XML element; this class is used by the EasyXML-derived class, FGXMLParse. There is of course an upcoming change to the JSBSim configuration file format. The new format will also extend the capabilities of JSBSim. Math functions will be allowed in flight control component specifications as well as in aerodynamic coefficient definitions, etc.

Also under way is an evaluation of the Cessna C-172r Skyhawk flight model. Performance figures given on the manufacturer's web site (as well as from other sources) are being compared with flight model performance. Adjustments have been made, and testing and adjustments will continue until the model performs within tolerances. But, what tolerances? That's another question. The FAA publishes requirements that must be met by a simulator in order to be certified at a particular level. To what level of accuracy JSBSim should match actual performance is currently being discussed.

— JSB

(Continued from page 2)

that the output bias is zero. K_p is the proportional element. The proportional gain K_p is multiplied with the error signal. $K_i \int e \cdot dt$ is the integral element. The integral gain K_i is multiplied with the integral from zero up to the present time of the error signal. Finally, we have the derivative element $K_d \cdot \dot{e}$. The derivative gain K_d is multiplied with the time-derivative of the error signal. Now imagine that we set both the integral gain K_i and the derivate gain K_d to zero. The PID formula is reduced to only the proportional element K_p. The controller is then called simply a P controller because it only contains the P part of PID.

The control system for an aircraft such as the F-16 or space shuttle, for instance, includes switch, scheduled gain, hysteresis, deadband, gain, summer, etc. control components. JSBSim can model all of the various components.

Making a PID Controller Using the JSBSim Flight Control System Components

Of the many components that comprise the set used to model flight control systems in JSBSim, three are used to model a PID controller: the Integral, Gain, and Filter components (specifically, the lead-lag filter component). Below is the JSBSim definition for an example PID controller (this definition is based on the v1.0 JSBSim configuration file format – v2.0 is pending), with a property, "roll-error" as the input to all three controller elements:

```
<COMPONENT NAME="Proportional controller" TYPE="PURE_GAIN">
  INPUT     roll-error
  GAIN      2.0          <-- This is Kp -->
</COMPONENT>

<COMPONENT NAME="Integral control" TYPE="INTEGRATOR">
  INPUT     roll-error
  C1        0.125        <-- This is Ki -->
</COMPONENT>

<COMPONENT NAME="Derivative control" TYPE="LEAD_LAG_FILTER">
  INPUT     roll-error
  C1        0.1          <-- This is Kd -->
  C4        1.0
</COMPONENT>

<COMPONENT NAME="PID control summer" TYPE="SUMMER">
  INPUT     proportional-controller
  INPUT     integral-controller
  INPUT     derivative-controller
</COMPONENT>
```

In Depth: DATCOM+

By Bill Galbraith

I was looking for a way to generate a model that was fairly close to actual aircraft performance with a minimum of cost.

I was asked to write an article about DATCOM+ for this newsletter. Now, I could spew off the technical description of what DATCOM does, its history with the Air Force, and expound on all of its attributes, but you can read about those somewhere else. (See below for a link to the DATCOM User's Manual). Instead, I thought I'd say a few hundred words about my experience with DATCOM.

I build flight simulators for a living, for airplanes and helicopters, mainly for the military, but sometimes for commercial applications. Sometimes, the trainers are simple devices that just need to fly something like the target aircraft, other times it's a full-up flight model. The manufacturers sometimes have their flight models available, but they are usually very expensive. For a simple $500,000 trainer, you just don't have the budget to buy their flight model. You also have to raise an eyebrow at their flight model when you see it was developed before the airplane was built, and during initial testing, the configuration was changed to improve flight characteristics. Was the model refined after that design change? Sometimes, you just can't get these answers.

This is what drove me to investigate DATCOM. I was looking for a way to generate a model that was fairly close to actual aircraft performance with a minimum of cost. I was looking towards conventional aircraft in the subsonic flight range, within a nominal flight envelope that would include take-off and landing, climbs, turns, and level flight, with somewhat representative stall effects. My initial investigation was done using a T-34C, a low-wing trainer turboprop aircraft manufactured by Beech Aircraft in the 1970's. I had the Beech Aircraft flight model available, some flight test results from the Navy flight testing, and many of the reports from Patuxent River on this aircraft. I obtained a copy of the DATCOM Fortran code, got it compiled and cleaned up so it was useful. I built a rough DATCOM model of the T-34, and was very impressed with the results, enough so that I felt it was worth further investigation.

DATCOM uses an incremental build-up method, starting with wings and fuselage, adding ailerons and flaps, then a tail with elevator. A DATCOM note mentions that the moveable surfaces on a lifting surface are assumed to be on the aftmost lifting surface. Therefore, ailerons and flaps have to be added before the tail and elevator. For more exotic designs, this input file will have to be restructured. Along the way, I realized that the input format was pretty user-abusive, so I built a small AWK script that would allow me to embed comments into my DATCOM input file, and the AWK script would strip out my comments and feed the results into the DATCOM program. These comments included the description from the DATCOM User's Manual, as well as notes that I left when I figured out some particularly vague items. I did note some short-comings in the DATCOM output, mainly in the directional axis, but those were small problems compared to the information that DATCOM produced.

The next simulator that I worked on was a Cessna Citation II. The flight model and flight testing was being done concurrently by a brilliant engineer, Keith Richards [Ed: no – not *that* Keith Richards] of Alpha Systems Engineering. With thousands of hours invested in that project, he had an FAA Level B tested flight model and flight test results. Sounded like a good target for more experiments. I built a DATCOM input file for the Citation, which was made much easier by the fact that the input file had lots of comments in it. The output format was becoming difficult to handle, as I was doing iterations to refine my model, and grew tired of transferring data from their output format to what I needed. I dug through the horribly-commented Fortran code and figured out where I needed hooks in order to extract data into files in the format that I needed it. I output it in a format that allowed me to plot the coefficients against those from the Citation simulator, and once again, found good concurrence. I added another output module, to output the coefficients in a format to be included in the Citation simulation. There again, the results of the simulation were quite impressive.

Then, along came JSBSim. Here was a flight model that I was interested in, and

(Continued on page 5)

(Continued from page 4)

there seemed to be many people in the JSBSim community interested in DATCOM, so I figured it was good swap. With some small modifications, I was able to output the coefficients in the JSBSIM XML file format. All of the data for a JSBSim model is not available in the DATCOM input file, so the generated XML file needs to be modified before it can be used in JSBSim, but it's a pretty good start, especially towards the aerodynamic coefficients.

I've made my DATCOM+ program available to the JSBSIM community, the executable running in a CMD window of Windows XP or under Cygwin. DATCOM+ is a superset of DATCOM, that superset being the user-friendly input format and the JSBSim-formatted output. For now, I'm calling it an alpha-beta release. The beta portion is the DATCOM part itself. I feel good about the DATCOM code being good, but we don't handle all the interesting cases yet, such as near-ground effects and propulsion effects. The alpha part is the JSBSim output. I'm new to the JSBSim world, and the output format can be refined further, but it's a decent start if you are interested in building a model of a particular aircraft.

I am presently working on an Automatic Fidelity Test (AFT) system in C++ using JSBSim, so that flight models can be evaluated against flight test data or published performance data. Future work in DATCOM+ includes near-ground and propulsion effects, a database of DATCOM models, and better output for JSBSim-format files. Your suggestions are welcome.

You can find my DATCOM+ program and the USAF DATCOM User's manual on my web site at www.holycows.net/datcom.

JSBSim Reference Frames

by Mathias Fröhlich & Jon Berndt

There are several different coordinate frames used in JSBSim. One question that was raised at the end of my presentation of JSBSim at the AIAA MST Conference dealt with just that. It's important to be aware of the various coordinate systems that are used. This note describes each of the frames:

<u>Earth-Centered Inertial (ECI)</u> The origin for the ECI frame is fixed at the center of the earth, with the Z axis towards the north pole. It does not rotate with the earth. This frame never shows up directly in the equations of motion. It is assumed to be an inertial frame, neglecting the movement of the earth around the sun and larger scale movements.

<u>Earth-Centered Rotating (ECR)</u> The origin for the ECR frame is fixed at the earth. The X axis points towards a latitude and longitude of zero. The Z axis points towards the north pole. The frame *does* rotate with the earth. The state value for position (latitude, longitude, altitude) in the equations of motion is expressed in this frame.

<u>Local (LVLH)</u> This frame is usually referred to as simply the Local frame, but the complete and conventional label given is Local Vertical / Local Horizontal. This frame moves with the aircraft. The Z axis is coincident with a line drawn from the aircraft to the center of the Earth, and positive down. The Y axis points east and the X axis points north, with the origin located at the aircraft center of gravity. The X and Y axes are tangent to the earth surface plane at all times.

<u>Structural</u> The structural frame is the frame in which the manufacturer typically uses to refer to the location of elements such as landing gear, the mean aerodynamic chord, the CG, etc. The terms station, butt line, and waterline are often used to refer to the X, Y, and Z locations, respectively. In this frame, the X axis is parallel to the fuselage centerline and is positive aft, the Y axis is positive towards the right (imagine yourself sitting in the cockpit), and the Z axis completes the triad, positive upwards. The origin is often placed on the fuselage centerline at or in front of the nose.

<u>Body</u> The body frame is the frame in which vehicle velocities are reported (U, V, and W velocities and accelerations). The body frame is similar to the structural frame, except that the X axis is positive forward, and the Z axis is positive downwards (the body frame is rotated 180 degrees about the structural Y axis). The origin of the body frame is at the vehicle CG.

Note that JSBSim reports latitude, longitude, and altitude in geocentric coordinates – *not* geodetic. See the sidebar at right for a definition of the difference.

Geocentric and Geodetic: What's the difference?

*The **geocentric** latitude is given by the angle between the plane of the equator and the radial drawn from the center of the earth to the point of interest (e.g. an aircraft). Note that because the earth is not spherical, but ellipsoidal, the line from the center of the earth to the point of interest may not be normal (perpendicular to) to the surface of the Earth.*

*The **geodetic** latitude is given by the angle made by the equatorial plane and a line passing through the point of interest and which is also perpendicular to the surface of the earth. The geodetic latitude is a little more than the geocentric latitude. The geodetic altitude is also a little less than the geocentric altitude. The geodetic position corresponds to what you would see on a map.*

Highlighted References

Online:

Dr. Wayne Durham's online textbook for his Aircraft Dynamics and Control class at Virginia Tech:

http://www.aoe.vt.edu/~durham/AOE5214

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

(Continued from page 1)
the JSBSim web site at www.jsbsim.org - select the Documentation link.] Several people whose names I recognized were in the audience, including Bruce Jackson (author of LaRCSim and more recently a key collaborator on the proposed DAVE-ML AIAA flight simulation exchange format specification), and Dr. Peter Zipfel (http://www.aiaa.org/store/storeproductdetail.cfm?id=1198). Several people approached me for more information after the presentation. I found out that the paper, "JSBSim: An Open Source Flight Dynamics Model in C++", was among the several candidate papers nominated for Best Paper award for the Modeling and Simulation conference. The winner will not be known for a while.

The keynote speaker on Tuesday at lunch was Peter Diamondis, the motivating force behind the X-Prize and several other business ventures. He spoke on the upcoming X-Prize flight attempts, his new NoGravity venture, etc. (see www.nogravity.com)

Tuesday night I attended a banquet at the Bella Vista restaurant along the river where, later on in the evening, fire-tenders on water gondolas lit torches along the riverfront, as music echoed across the water – WaterFire, Providence. It was a well-attended event, and I enjoyed it very much. I ended up knowing a surprising (to me) number of people attending the event and had some very enjoyable dinner conversation. I also bumped into a JSBSim mailing list member, Nick Hein, and had an enjoyable conversation about various simulation topics.

Next year's conference will be held in San Francisco. For more information, visit www.aiaa.org. -JSB

Next Issue:

"Back of the Envelope" is a new communication tool written 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: jsb@hal-pc.org.

Some possible topics for future issues includes:

- The Property System
- JSBSim Configuration Files in XML
- Integrating the Equations of Motion in JSBSim
- Scripting JSBSim runs