AIAA Modeling and Simulation Technologies Conference
10 - 13 August 2009, Chicago, Illinois

AIAA 2009-5699

Progress on and Usage of the Open Source Flight Dynamics Model Software Library, JSBSim

Jon S. Berndt*
Engineering and Science Contract Group / Jacobs, Houston, Texas, 77573

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

JSBSim is an open source software (OSS) flight dynamics model that can be incorporated into a larger flight simulation architecture (such as FlightGear, or OpenEaagles). It can also be run as a standalone batch application when linked with a stub routine. Since 2004, when JSBSim was formally introduced at the Modeling and Simulation Technology conference, many advances have taken place, and a variety of uses have been demonstrated. This paper will present updates on project status, an overview of XML configuration file format enhancements, details on recent improvements and design choices, and some basic examples of use. A discussion about interfacing JSBSim with Matlab as a Mex-Function or Simulink S-Function is included, followed by a deeper look at a representative usage case study.

I. Introduction

JSBSim is a high-fidelity, 6-DoF (Degree-of-Freedom), general purpose, flight dynamics model software library written in the C++ programming languages. The library routines propagate the simulated state of an aircraft given inputs provided via a script or issued from a larger simulation application. The inputs can be processed through arbitrary flight control laws, with the outputs generated being used to control the aircraft. Aircraft control and other systems, engines, etc. are all defined in various files in a codified XML format.

The library consists of approximately 70,000 text lines in 185 files. The total lines of program code is estimated at about 50,000 lines. Begun in 1997, JSBSim has an international team of active developers and user contributors.

As a design goal, JSBSim has attempted to find a balance between high simulation fidelity and data file simplicity, so the task of simulating the flight of any aerospace vehicle can be done with the minimum specific input possible. Most code changes now are a result of minor code tweaks, or the addition of a few new features, as the user base provides inputs. The code has evolved in response to the way it is used, similar to natural selection. But, in some ways, development has shifted from compiled C++ code towards the creation of common XML data files.

At the time of this writing, JSBSim is releasing candidate source code archives for version 1.0. The numbering of versions is somewhat arbitrary, but it signals – roughly – that the application has been used reliably for some time in a number of applications, that documentation[1] has reached a level where it is fairly thorough and useful, and the feature set is well defined and mature.

II. Overview of the Vehicle Configuration and Script File Format Changes

JSBSim is a data driven simulation. How data driven? Below is the minimum code needed to run JSBSim:

```
#include <FGFDMExec.h>                    // Include the executive header
int main(int argc, char **argv) {         // Pass a script name via argv
   JSBSim::FGFDMExec FDMExec;             // Instantiate the Executive
   bool result = true;                    //
   FDMExec.LoadScript(argv[1]);           // Load a script
   while (result) result = FDMExec.Run(); // Run until the script completes
}
```

* Engineer, Aerothermal and Flight Dynamics Section, 455 East Medical Center Blvd., M/C JE-B3-2 Webster, Texas 77598, AIAA Senior Member *(Citation of the author's employer neither implies nor precludes endorsement by employer of this work, which has been performed without the use of company time. Affiliation cited for contact purposes only.)*

† Assistant Professor, Department of Aerospace Engineering, Naples, Italy, AIAA Senior Member

All vehicle model characteristics (mass properties, aerodynamics, propulsion, etc.), initial condition values, run script directives, and logging directives, are stored in text files. The various configuration files were migrated to a completely valid XML file format to gain the benefits of using XML.[2] Among those benefits are:

- An XML *schema* can be created that defines the structure of a configuration file (e.g. an aircraft configuration file or a script file). The schema can be used to validate an input file before it gets parsed by the application itself.
- An XSL transformation can be used to convert an XML configuration file into another format, such as HTML, for display in a browser.
- There are many freely available XML parsing codes that can be leveraged.

A. Aircraft configuration file format

The aircraft (or, more generally, *vehicle*) configuration file is now structured to permit greater reuse. The general layout of the configuration file format is as follows (newer sections are in **bold** and preceded by an asterisk):

```
<fdm_config>
  <fileheader> … </fileheader>                  <!-- 0 or 1 instance  -->
  <metrics> … </metrics>                        <!-- 1 instance       -->
  <mass_balance> … </mass_balance>              <!-- 1 instance       -->
  <ground_reactions> … </ground_reactions>      <!-- 1 instance       -->
 *<external_reactions> … </external_reactions>  <!-- 0 or 1 instance  -->
 *<buoyant_forces> … </buoyant_forces>          <!-- 0 or 1 instance  -->
  <propulsion> … </propulsion>                  <!-- 0 or 1 instance  -->
 *<system> … </system>                          <!-- 0 to n instances -->
  <autopilot> … </autopilot>                    <!-- 0 or 1 instance  -->
  <flight_control> … </flight_control>          <!-- 0 or 1 instance  -->
  <aerodynamics> … </aerodynamics>              <!-- 1 instance       -->
  <input> … </input>                            <!-- 0 or 1 instance  -->
  <output> … </output>                          <!-- 0 to n instances -->
</fdm_config>
```

The ground reactions, external reactions, buoyant forces, propulsion, system, and aerodynamics elements in the above configuration file outline can each contain a collection of other elements. The propulsion element contains a list of zero or more engines. The ground reaction element contains a list of zero or more contact points. The system element contains grouping of control system components, and so on. Multiple output elements can be specified.

Some of the configuration file elements can refer to a file (through a filename attribute) that may contain the actual definition of the relevant model. The aerodynamics characteristics, individual engine specifications, control systems, and output specifications can exist in separate files that can be included in any aircraft model. This is a relatively recent new capability.

B. Script file format

JSBSim can be controlled in its standalone, batch mode through the use of a script. A JSBSim script (a custom, XML-format file) contains a reference to a vehicle configuration file and an initial conditions file. It also specifies the length of the simulation run in seconds, the frame rate in Hz, and can contain from zero to many *event* elements. Events specify an action or actions to take when a set of conditions are satisfied.

The structure and features of scripts have been enhanced in recent years. Most notably:

- One can now declare new properties (with initial values) that can be manipulated or used as variables.
- In addition to setting a property to a numeric value in an event, one can now set a property to the value returned by a function.
- An event can be directed to be executed each frame, instead of only when a condition is satisfied. For example, a wind table could be provided as a function within an event that is continuously executed, so that a simulated rocket could be affected by a wind magnitude and direction based on altitude.
- An event can include a command to print a notification to the console when it is triggered, as well as the name and value of specified simulation properties.

There are also now properties available that, when set in a script, will cause an action to take place. The available actions are: terminate the sim, trim the aircraft, write a state file, and reset the simulation to the initial conditions.

III. Enhancements

JSBSim development has seen considerable progress since it was presented in 2004 at the AIAA Conference in Rhode Island[3]. The major features changed or added since then include:

- Support for any number of arbitrary systems (electrical, control, etc.) that can be built using the control components supplied in JSBSim (gain, summer, PID controller, switches, etc.),
- Additional control components such as actuators and sensors (with malfunction capability) and an accelerometer have been added,
- Support for airship modeling,
- Support for arbitrary math functions,
- Support for arbitrary external forces,
- Runtime selectable EOM (Equations of Motion) integrators for rotational and translational velocity and position,
- An overhaul of the configuration file format to be more strictly XML compliant,
- An overhaul of the modeling of the EOM and the propagation of the aircraft state to be more readable and more accurate in a rotating Earth frame,
- Refinement of the ground reactions model to reduce or eliminate instability,
- Support for the specification of property values from the command line.

A. Arbitrary Systems Modeling

Since the beginning, JSBSim has provided a way to model arbitrary flight control laws for an aircraft model through user specification of a chain of linked control components in the flight control section of the aircraft configuration file (an XML text file). A simple example of such a chain of components is shown:

```
<pure_gain name="fcs/roll_rate_norm">
  <input> velocities/p-aero-rad_sec </input>
  <gain> 0.31821 </gain>
</pure_gain>

<summer name="fcs/roll_trim_error">
  <input> fcs/aileron-cmd-norm </input>
  <input> -fcs/roll-rate-norm </input>
</summer>

<switch name="fcs/aileron_pid_trigger">
  <default value="1"/>
  <test value="0">
    velocities/vc-kts lt 20.0
  </test>
</switch>

<pid name="fcs/roll_rate_pid">
  <trigger> fcs/aileron-pid-trigger </trigger>
  <input> fcs/roll-trim-error </input>
  <kp>  3.00000 </kp>
  <ki>  0.00050 </ki>
  <kd> -0.00125 </kd>
  <clipto>
    <min> -1 </min>
    <max>  1 </max>
  </clipto>
</pid>
```

In the above example, where each component (pure_gain, summer, switch, and PID) is instantiated and named when parsed by the application at runtime, one should notice that consecutive components usually refer to

previously defined component names for inputs. Note that in XML items declared within angle brackets (< >) are referred to as *elements*.

This component-based approach works well for defining detailed control laws for an aircraft in a text file for later instantiation at runtime. It allows complete control for the flight model developer; at the same time it allows the source code to be generic.

Over time, it has been seen that users are modeling various types of systems with the components (electrical systems, for instance). So, the decision was made to create a new XML element that could be placed in the aircraft configuration file to group components for *systems* modeling. Furthermore, support was added to the new system element to allow referencing a separate file rather than to have the component definitions specified inline. That approach allows standard system models to be created and shared between aircraft models. The following snippet shows how an F-4N aircraft flight model incorporates several system models that are defined in other files (the ".xml" extension is optional):

```
<system file="holdback"/>
<system file="hook"/>
<system file="catapult"/>
<system file="BLC"/>
<system file="gear"/>
<system file="flaps"/>
<system file="speedbrakes"/>
<system file="FCS-pitch"/>
<system file="FCS-roll"/>
<system file="FCS-yaw"/>
<system file="NWS"/>
```

With the introduction of the "system" element, the "flight_control" and "autopilot" elements have now effectively become superfluous, though support will remain for backwards compatibility and readability.

Since the set of control components has stabilized and matured, development has noticeably shifted away from source coding towards configuration file coding. In addition to the systems shown above (included with the F-4N model), two other general purpose system models are under development and in use. One is called GNCUtilities, and as the name suggests it contains general purpose GNC (Guidance, Navigation, and Control) functions, such as a calculation of the great circle heading and distance to a specified latitude and longitude using the Haversine formulas, and a calculation for the smallest included angle to a heading from the current heading. The other general purpose system model contains simple autopilot functions (currently only roll and heading modes are implemented).

The system element also allows for "variables" to be introduced within it and optionally assigned initial values. Variables in JSBSim are more commonly known as "properties" – public text names that are bound at program initialization to class members or get/set functions that are used in configuration files and control component specifications. If a system is defined in its own file, any properties that are defined within the file and given initial values can have those initial values overridden in the main aircraft configuration file. For instance, the Autopilot.xml file contains property declarations with initial values that represent limits that are used in the roll control channel:

```
<!-- Initial constants (can be overridden in the aircraft file) -->

<property value="0.524"> guidance/roll-angle-limit </property>
<property value="0.174"> guidance/roll-rate-limit </property>
```

Within an aircraft configuration file, the Autopilot.xml file can be referenced in a system element as follows, overriding the values given in the Autopilot.xml file:

```
<system file="Autopilot">
  <property value="1.048"> guidance/roll-angle-limit </property>
  <property value="0.348"> guidance/roll-rate-limit </property>
</system>
```

The system element enhancement has provided flexibility and a powerful capability for modeling multiple complex systems in JSBSim, and the design encourages reuse of entire systems defined in XML files.

B. Support for Arbitrary Math Functions

JSBSim supports the specification of arbitrary math functions in several places. Numeric values, property values, and values returned by lookup tables can be operated on by trigonometric, algebraic, and other functions. The function specification format used by JSBSim is similar to MathML but simpler, since JSBSim does not have to specifically support the proper *display* of the math functions. Here's an example of a function:

```
<function name="aero/force/LDe">
    <description> Lift from elevator deflection </description>
    <product>
        <property> aero/qbar-psf </property>
        <property> metrics/Sw-sqft </property>
        <property> fcs/elevator-pos-rad </property>
        <table>
            <independentVar> velocities/mach </independentVar>
            <tableData>
                0.0000  1.0000
                0.6000  1.0500
                1.0000  1.1500
                1.2000  1.0000
                1.6000  0.6600
                2.0000  0.5000
                2.4000  0.4000
                3.0000  0.3100
                5.6000  0.2100
                6.0000  0.2000
                9.0000  0.2000
            </tableData>
        </table>
    </product>
</function>
```

The above example shows the definition of the lift force due to the elevator deflection. In fact, all aerodynamic forces and moments are defined using functions. Functions can also be used to define specific algorithms in the *system* specification, when none of the existing control components are suitable.

Note that JSBSim currently supports up to 3-dimensional tables, with linear interpolation and no extrapolation.

C. Lighter-Than-Air Vehicle Modeling

Modeling of buoyancy has been added, allowing modeling of hot air balloons, buoyancy-assisted vehicles, and zeppelins. Individually positioned gas cells and ballonets can be specified for a vehicle. The gas cells can currently be specified as holding Hydrogen, Helium, or air, and relief valves can be defined. Heat transfer between the environment and a gas cell can be defined by a function. Virtual mass is not specifically supported in program code (at this time), but an experimental capability is provided by a system definition file.

D. External Forces

Beginning a couple of years ago, some users in the FlightGear community and elsewhere expressed interest in flying various scenarios that could not be modeled in JSBSim at the time. Users wanted the ability to apply arbitrary forces and resultant moments to an airframe in order to model scenarios such as aircraft carrier operations (catapult, hook and wire), tow ropes, parachutes, etc. The solution was to provide an external forces class and an external reactions manager class to manage an array of external force class instances.

The external forces class encapsulates the location, direction (in body, wind, or local axes). And magnitude of a force acting on the vehicle structure. The force direction can be changed at any time. The force magnitude can be defined by a function or set via a property name. For example, a simple parachute attached to a two foot diameter ball could be defined as follows:

```
<external_reactions>
  <!-- "Declare" the reefing property -->
```

```
  <property> fcs/parachute_reef_pos_norm </property>

  <force name="parachute" frame="WIND">
    <function>
      <product>
        <property>aero/qbar-psf</property>
        <property>fcs/parachute_reef_pos_norm</property>
        <value> 1.0 </value>  <!-- Full drag coefficient -->
        <value> 20.0 </value> <!-- Full parachute area -->
      </product>
    </function>

    <!-- The location below is in structural frame (x positive aft),
         so this location describes a point 1 foot aft of the origin.
         In this case, the origin is the center. -->
    <location unit="FT">
      <x>1</x>
      <y>0</y>
      <z>0</z>
    </location>

    <!-- The direction describes a unit vector. In this case, since
         the selected frame is the WIND frame, the "-1" x component
         describes a direction exactly opposite of the direction
         into the wind vector. That is, the direction specified below
         is the direction that the drag force acts in. -->
    <direction>
      <x>-1</x>
      <y>0</y>
      <z>0</z>
    </direction>
  </force>
</external_reactions>
```

The reefing property is a normalized value that reflects the amount the parachute canopy is expanded or opened (0 = closed, 1 = fully opened). The reefing rate can therefore be defined by a script or in a flight control or system definition..

The external forces capability added is not explicitly limited to externally applied forces. It provides a way to introduce arbitrary forces on the airframe which then introduce moments.

E. Landing Gear Modeling: Problem and Solution

Modeling an aircraft at rest while supported by landing gear can cause instability in simulations if it is not done with care. This problem has historically been dealt with in several ways, including through the use of the tricycle approximation[4].

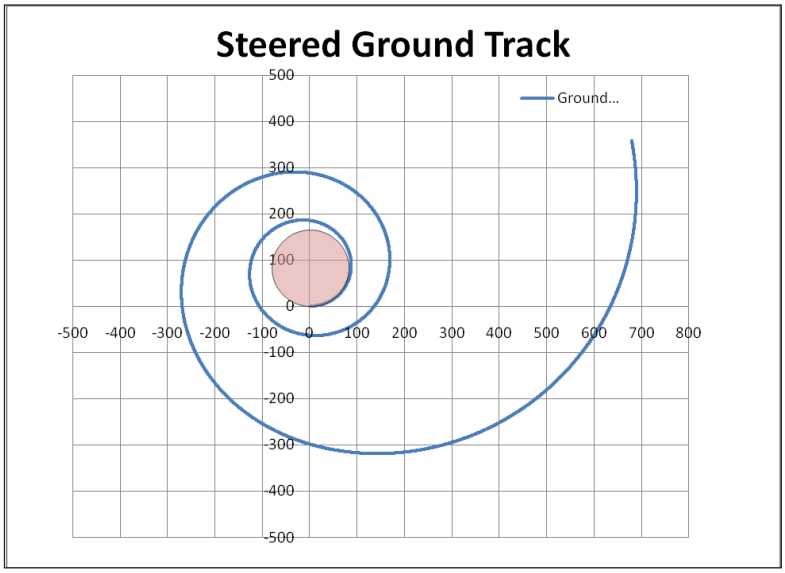

Figure 1. Ground track of a jet aircraft accelerating from rest. Shaded circle is the minimum turn radius. Nose gear is steered 8 degrees, left. X/Y Distance measurements are specified in feet.

In early versions of JSBSim, individual tires were modeled as contact points for which the side force (parallel to the aircraft Y axis) was simply a function of the friction coefficient, which went from 0.0 to a maximum of the static friction coefficient (as the tire was steered), and then back down to a steady state value defined by the dynamic friction coefficient. With this model, when running JSBSim with the aircraft at rest on the runway, the aircraft was seen to slowly drift, and to exhibit a rapid, small amplitude, angular oscillation.

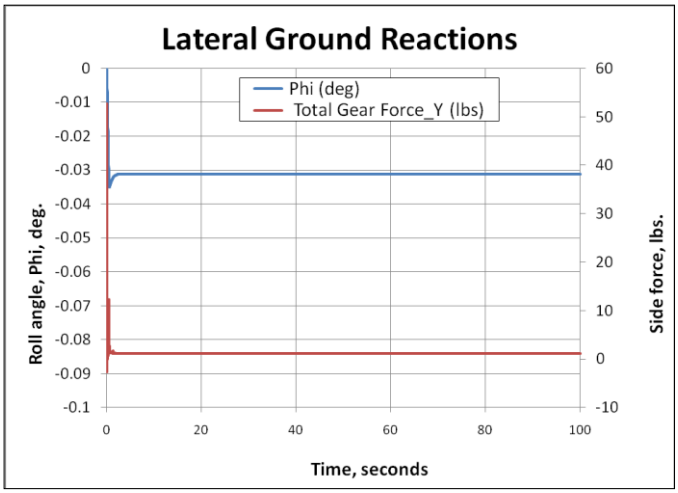

Figure 2. Time history of lateral ground reactions and roll angle with fade-in of forces from 0.0 to 0.3 ft/sec.

The landing gear model has been improved mainly through two approaches: using Pacejka's "Magic Formula"[5] for calculating lateral forces due to steering or dynamics, and the fading out of ground reaction forces when velocity is very near zero. Figure 1 shows that the initial turn radius is the same as the geometrically-determined tricycle approximation (calculated from the wheelbase and the front tire steering angle), and that the turn radius enlarges with velocity. Figure 2 shows the aircraft resting on the ground without drifting or oscillating after being dropped onto the runway.

F. Simulation Performance / CPU Loading

Since JSBSim is a generic flight dynamics software library, and aircraft are fully defined in text files, one might ask if runtime performance suffers due to the high degree of generality that the code must support. Various code profiling tools (valgrind, cachegrind, etc.) have been employed to find bottlenecks in the code, and compiler directives have been set to maximize performance. On the author's machine, the "-O9" compiler optimization flag is set, as well as the –march=nocona architecture flag. The resulting standalone JSBSim application is 2.4 MB in size. When a simulated 3 hour test run was made on a 64-bit machine with an Intel quad-core CPU running at 2.66 GHz under Microsoft Windows Vista, the entire run took 43 seconds – 250 times faster than realtime. For the test run, a 50 Hz. frame rate was used, rotational EOM (rates and positions) were propagated with a 2nd order Adams Bashforth integrator, and translational EOM used a trapezoidal integrator.

IV. Interfacing JSBSim with Matlab/Simulink

Users of Matlab/Simulink can rely on a well established interface to FlightGear. This capability is provided by the MathWorks "Aerospace Blockset" product. Air vehicles implemented by Simulink flight dynamics models are easily animated by sending the simulated flight data to a properly configured FlightGear session, which is used as a flight visualization server. From the standpoint of the open source community, the fact that MathWorks decided to release and support the FlightGear interface (in sync with new FlightGear releases) did sound like a very important point, which enhanced the already widespread success of FlightGear.

Recently Agostino De Marco provided an example of a MEX-function that enables Matlab users to access JSBSim capabilities from the Matlab command window. This function has been named *MexJSBSim*. The acronym "MEX" in MexJSBSim stands for Matlab EXecutable. MEX-files are dynamically linked subroutines that can be produced from C, C++ or Fortran source. When compiled and made visible to the the environment, MEX-functions (or "MEX-files") can be run from within Matlab in the same way as Matlab M-files or built-in functions. These external interface functions provide functionality to transfer data between MEX-files and Matlab.

The MEX-function *MexJSBSim* has been developed with the purpose of establishing a sort of standardized way of using a JSBSim::FGFDMExec object from the Matlab command line. Currently, this project is developed under the Windows platform. Yet, the source code is platform-independent and the Mex-function can be compiled on other platforms.[‡] MexJSBSim is an attempt to provide control engineers a direct access to a well established, high fidelity, flight dynamics model like JSBSim, enabling them to concentrate their work on automatic control logic and autopilot design.

[‡] See the MexJSBSim web site at http://www.dias.unina.it/demarco/Work/JSBSim_Matlab [cited 1 August 2009]

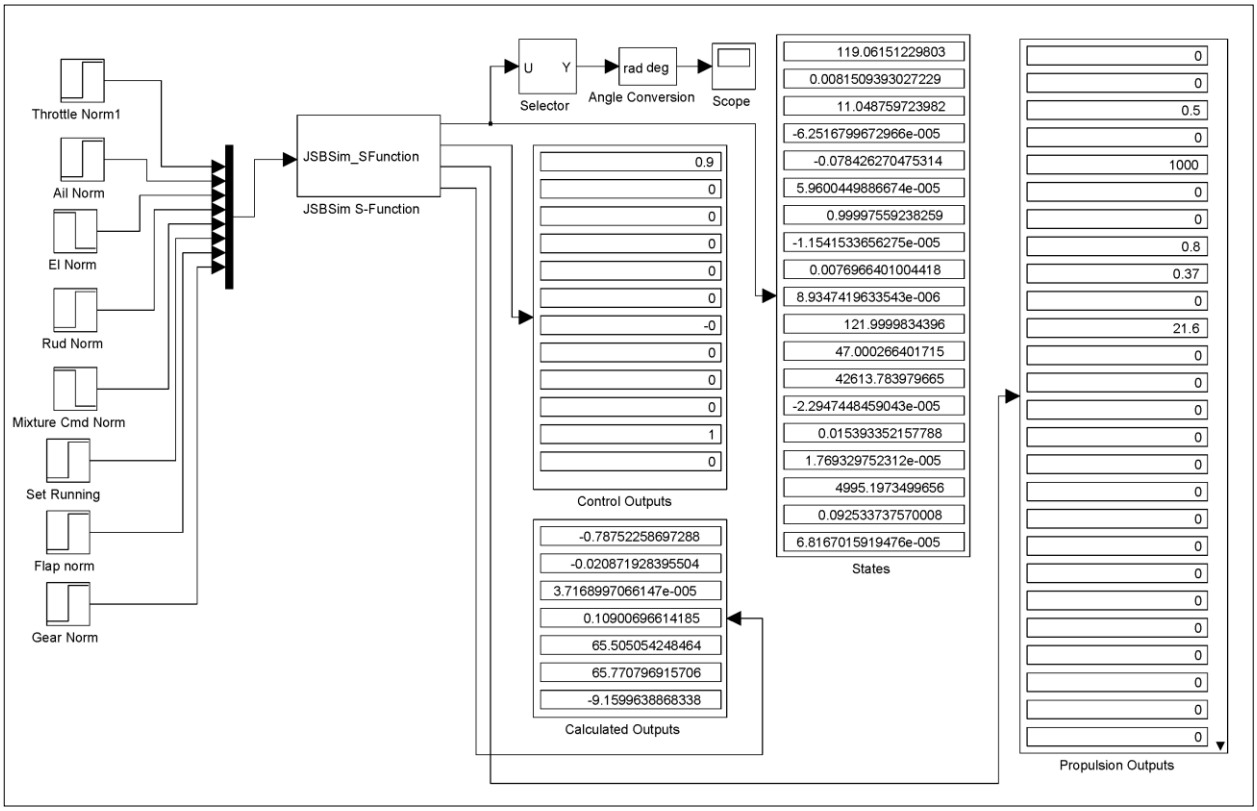

Figure 3. JSBSim S-function GUI.

Evolving from the MexJSBSim code, a JSBSim S-function has been developed. Matlab S-functions (system-functions) provide a powerful mechanism for extending the capabilities of the Simulink environment. An S-function is a computer language description of a Simulink block written in Matlab, C, C++, Ada, or Fortran. C, C++, Ada, and Fortran S-functions are compiled as MEX-files. As with other MEX-files, S-functions are dynamically linked subroutines that the Matlab interpreter can automatically load and execute. S-functions use a special calling syntax called the S-function API, that enables users to interact with the Simulink engine. This interaction is very similar to the interaction that takes place between the engine and built-in Simulink blocks.

In practice, the JSBSim S-function is a custom made Simulink block and is usable as other built-in blocks. Figure 3 shows an example of Simulink project that uses the JSBSim S-function.

Both the MEX function, *MexJSBSim*, and the S-function represent two effective examples of integration of a complex C++ code with the Matlab application. These allow the JSBSim users that have access to Matlab to bring all the advantages offered by JSBSim into that sophisticated and powerful computing environment.

V. Examples of Use

A. FlightGear

The earliest and best known use of JSBSim is within the open source *FlightGear* flight simulator[§]. FlightGear features several selectable flight dynamics models: LaRCSim (Langley Research Center Simulator), JSBSim, YASim ("Yet Another Simulator"), and a derivative of LaRCSim produced by Michael Selig at the University of Illinois at Urbana-Champagne. Input from the FlightGear community has driven many of the enhancements to JSBSim. While JSBSim is incorporated as a native part of FlightGear, a recently added feature is the ability to run JSBSim separately, communicate with FlightGear over a socket, and use FlightGear as visual scene server to a JSBSim instance.

[§] See the FlightGear project web site: http://www.flightgear.org

B. OpenEaagles

OpenEaagles (Open Extensible Architecture for the Analysis and Generation of Linked Simulations) is – according to their web site[**] - "a multi-platform simulation framework (see Figure 4) targeted to help simulation engineers and software developers rapidly prototype and build robust, scalable, virtual, constructive, stand-alone, and distributed simulation applications."

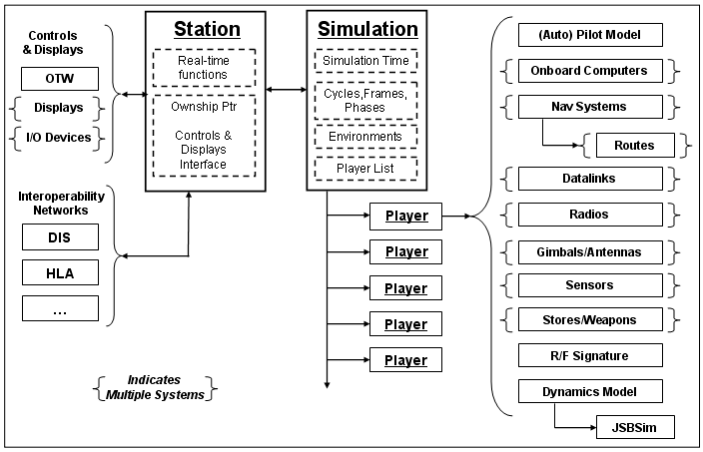

Figure 4. OpenEaagles architecture showing JSBSim (at bottom right).

C. ArenaLogic Simulator

ArenaLogic has made various uses of JSBSim during development of military simulator products, and is currently investigating the use of JSBSim to drive modeling of missiles and ordnance, as well as target aircraft.

D. VirtualAir

VirtualAir[††] is an open source air traffic simulation framework, using Modeling and Simulation HLA (High Level Architecture) to integrate various driver flight models, including X-Plane,[‡‡] Microsoft Flight Simulator,[§§] FlightGear, and JSBSim (through a Python interface to JSBSim[***]). JSBSim has been integrated with VirtualAir at this time, but is still undergoing testing.

E. U. S. Department of Transportation

In work done with and for the U.S. D.O.T., a human pilot math model was developed using JSBSim as the 6-DoF (six degree-of-freedom) simulation core.[6]

F. Aerocross Systems Echo Hawk

JSBSim is used for Hardware-in-the-Loop (HITL) testing of the Aerocross Echo Hawk UAV. Custom code was written to interface with the flight hardware (PC/104-based system) via RS-232/422/485, proportional analog I/O, discrete I/O, and sockets, but the core simulation code was unaltered JSBSim code. Pilot/operator training also relies on JSBSim as the 6-DoF code.

Figure 5. Aerocross Systems Echo Hawk

G. duPont Aerospace Company

JSBSim was used at duPont Aerospace Company along with Matlab for real-time HITL simulation and pilot/operator training. Rex duPont of duPont Aerospace Company explains,[†††]

> In the 1990s duPont Aerospace Company was building an airplane to test its concept for a vertical takeoff fan jet transport plane. We had developed a Microsoft Windows-based flight simulator that we had used to test the flying qualities of the proposed craft. However, we needed a simulation that we could use in real time so that we could test the flight characteristics on a full-sized mockup with the flight actuators operating in the loop. We settled on the FlightGear simulator, using the JSBSim flight dynamics model because we could get the full code, it was nicely organized so that we could create new subprograms to match our aircraft, and support was readily available.

[**] See the OpenEaagles project web site: http://www.openeaagles.org
[††] See the VirtualAir project web site at http://www.sourceforge.net/projects/virtualair
[‡‡] See the X-Plane web site at http://www.x-plane.com
[§§] See the Microsoft Flight Simulator web site at http://www.microsoft.com/games/flightsimulatorx/
[***] JSBSim-Python web site: http://sourceforge.net/projects/jsbsim-python/
[†††] Private email correspondence with Rex duPont, 22 August 2009.

We developed simultaneously a Matlab simulator for the use in developing more effective autopilot guidance systems, since our primary task became to take the aircraft off using the autopilot alone and to hover in place for 30 seconds. This would show definitively that the control system was sufficiently robust. Therefore, we built into each relevant module of the Matlab simulator and the JSBSim derivative simulator a series of unit tests that provided a sequence of inputs to each model that could be cross verified to ensure that the two systems stayed in sync.

We used the JSBSim system to test a number of dynamic issues that were not easily testable with the Matlab model, especially issues involving pilot feel and the controllability during transition to and from hover. These issues are hard to evaluate in the pure control-system world of Matlab because during transition the underlying force structure is continuously varying as aerodynamic forces become more important and pure thrust control forces less.

Figure 6. duPont Aerospace DP-1 in a tethered hover test. (Courtesy of duPont Aerospace Company)

We also did parametric studies on such issues as the sensitivity of the key parameters in the aerodynamic simulation to possible errors in estimation. These were done by having the pilot fly a series of standard maneuvers designed to test the aircraft's response when one or more parameter was degraded by as much as 50% (without the pilot knowing which one was changed).

We simulated various servo bandwidths as well, testing to see at what point the flying characteristics became unacceptable. This helped define the characteristics needed. What pilots desire from control systems is almost always different from the optimal theoretical parameters.

Additionally, we developed a number of HUD display systems that facilitated operations during hover, where very precise control of ground speed is required. Eventually we achieved a system that allowed a young engineer who did not even have a pilot's license to take off and hold a hover at constant altitude to within one foot for over 30 seconds.

We finally achieved our goal of autopilot controlled take-off and hover in two flights of approximately 45 seconds duration on September 30, 2007. Both flights were terminated because one of the engines ran out of fuel, rather than for any control problems.

H. MITRE Air Traffic Studies

JSBSim is being used at MITRE in developing a 6-DoF simulation of the FMS (Flight Management System) behavior during CDAs (Continuous Descent Arrival) and OPDs (Optimized Profile Descent). Both the standalone version of JSBSim (for batch runs) and a version integrated with FlightGear have been used. Additional control system components have been created to support specific lateral and vertical navigation studies. JSBSim has also been extended to handle output of messages over a socket to another application used at MITRE which provides a view similar to what an Air Traffic Control operator would see.

I. University of Naples

The University of Naples has a motion base flying/driving simulator[7] that is driven by FlightGear and JSBSim. The simulator has a three-screen visual presentation that provides a 190 degree field-of-view. The JSBSim source code was modified to provide a force feedback capability.

JSBSim has been used at the University of Naples as a tool supporting risk level evaluations of near-ground flight operations. One of the practical problems considered in those collision risk studies consisted in the evaluation of threat posed to flight operations when a new obstacle (such as a building or radar tower) is placed inside the airport area. The risk evaluation has been performed by varying the obstacle geometry and location.

The risk assessment procedure has been based on the analysis of statistical deviations of aircraft trajectories from the “normal” flight path, evaluating the probability of a generic trajectory to cross a given “protection” area enveloping the potential obstacle. Within this framework, the operational scenario has been formally described and implemented in order to run multiple computer simulations.

VI. An Example of Use in Forensic Engineering

In this section we discuss in more depth an example of the use of JSBSim. This example is in the reconstruction, simulation, and analysis of a flight accident. The current ability to produce advanced and realistic animations is a valuable aid to investigators in analyzing accident trajectories. The subject of this section can be viewed as a typical application of what is known as "forensic engineering," i.e. the application of scientific and engineering knowledge to legal matters, such as accident reconstruction[8].

A. The Right, Super-Slow, Tonneau Accident

In 2001 a flight accident occurred in Italy with a two-seat, competition sailplane, the Grob G103C Twin Astir. The design of this aerobatic sailplane dates back to the 1970s. The airplane has a weight (MTOW) of 600 kg, a wing aspect-ratio of 16.95, a wing surface of 18.172 m^2, a wing span of 17.55 m, and an overall length of 8.2 m (see Figure 7).

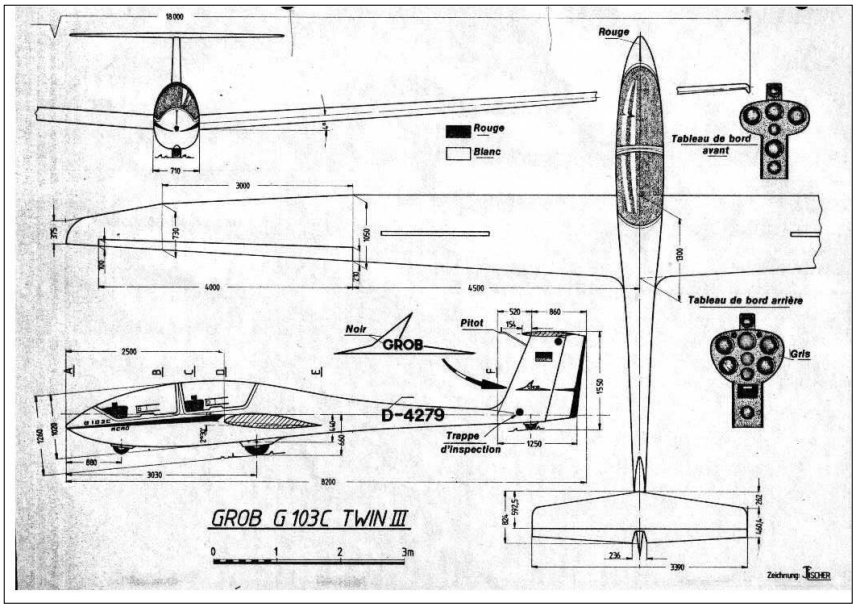

Figure 7. Grob Sailplane.

The day the aircraft crashed, the pilots were rehearsing the typical sequence of maneuvers required in one of the Italian national contests. The accident occurred during the execution of an aerobatic figure called the "right super-slow tonneau," at an altitude of approximately 500 m above ground level. This maneuver requires the pilot to perform a complete rotation about the airplane's longitudinal axis (roll axis) in a span of not less than 10 seconds. For this particular sailplane (belonging to the class of trainers known as Club), due to the high rolling moment of inertia, normally the super-slow tonneau cannot be completed in less than 10 to 15 seconds. During this maneuver the sailplane CG trajectory is not strictly a straight line, but it is a three-dimensional curve. Typically, this maneuver ends with a loss of altitude of about 80 m.

The sketch of Figure 8 illustrates the complete three-dimensional evolution, starting and ending with the aircraft in wings level conditions. Figure 9 shows the simulated time histories of Euler angles ψ, θ, and φ for this maneuver, produced by JSBSim. Many witnesses have confirmed that the rear fuselage beam snapped when the sailplane approached the second knife-edge, causing the unrecoverable fall that killed the two pilots. The fatal condition can be visualized by inspecting Figure 8 (the failure occurred after a rotation of approximately 270 deg) and observing the markers on the roll angle curve in Figure 9 (roll markers are at 45°

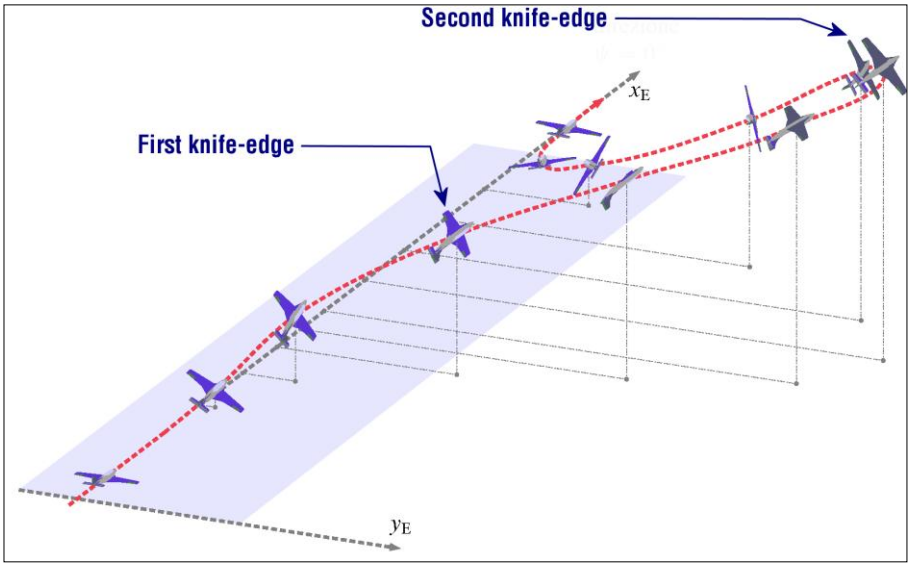

Figure 8. Super-slow Tonneau Maneuver.

increments).

Typically, a Grob G103C enters the tonneau with a velocity of 180 km/h (about 97 kts), with a slight pitch-down attitude (pitching angle θ negative). From this condition the pilot pulls the stick, the airplane rotates nose-up and climbs. As soon as θ reaches a value of 5 to 7 degrees above the horizon, the required roll rotation is started off by moving the stick progressively to the maximum right position. The roll commences and when the manuever finally approaches the second knife-edge (270 degrees roll) the pilot points the fuselage nose down moderately by acting simultaneously on elevator and rudder. From this point the speed increases as observed in Figure 10.

During the development of the whole maneuver, it is at the moment the airplane approaches the second knife-edge that both the required stick and rudder pedal forces are higher. In this situation, the elevator is deflected at the 100% (forward), ailerons at 60% (right) and rudder at 50% (right).

Figure 11 shows the simulated time history of altitude above the ground. Passing from the second knife-edge to the straight wings-level flight the right pedal is fully deflected (from the same side of the stick). During the last 60 degrees of rotation before re-leveling wings, the stick needs to be pulled again to avoid nose-down attitudes. This helps keeping a moderate speed in order to enter correctly in the next figure of the aerobatic program. If the maneuver is correctly performed, the final speed at the end of the tonneau is approximately the same as the entering speed (from 150 to 180 km/h).

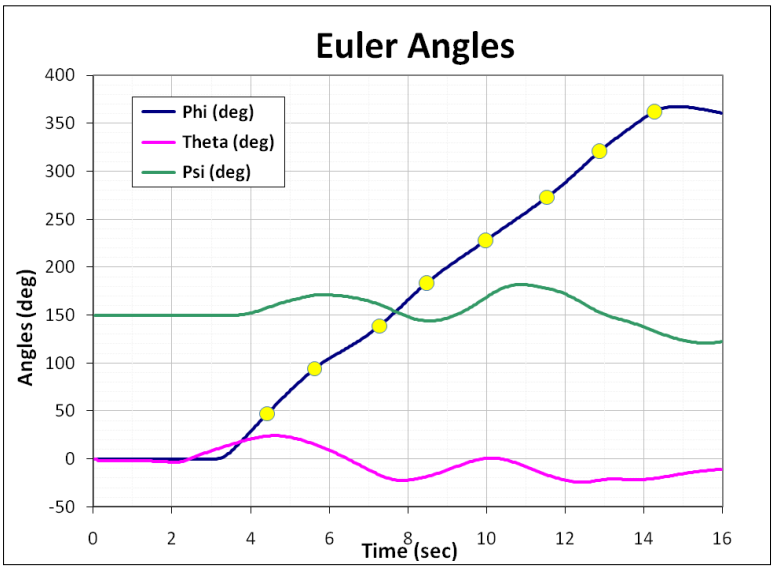

Figure 9. Time history (from a JSBSim run) of the Euler angles for a complete super-slow tonneau maneuver.

Figure 12 shows the detailed time history of the normalized aero-surface deflections prescribed as inputs to JSBSim. These laws have been determined through pilot interviews and by subsequent trial and error, taking as a reference a number of key conditions to be met during the development of the entire maneuver (for instance, the speed loss and the altitude loss, combined with roll angle time history, the time duration of a complete roll, etc.). It is clear that the histories of commands that determine the simulated maneuver depend strongly on the aerodynamic model of the airplane, and especially on each aero-surface estimated control effectiveness (aerodynamic control derivatives). [Note: since this study was performed, support for functions has been added to the scripting capability. Another approach to providing "virtual pilot" inputs to perform the tonneau maneuver would be to manually fly the same maneuver in FlightGear while logging the control inputs. The inputs could then be turned into a table and included within a function in a script event.]

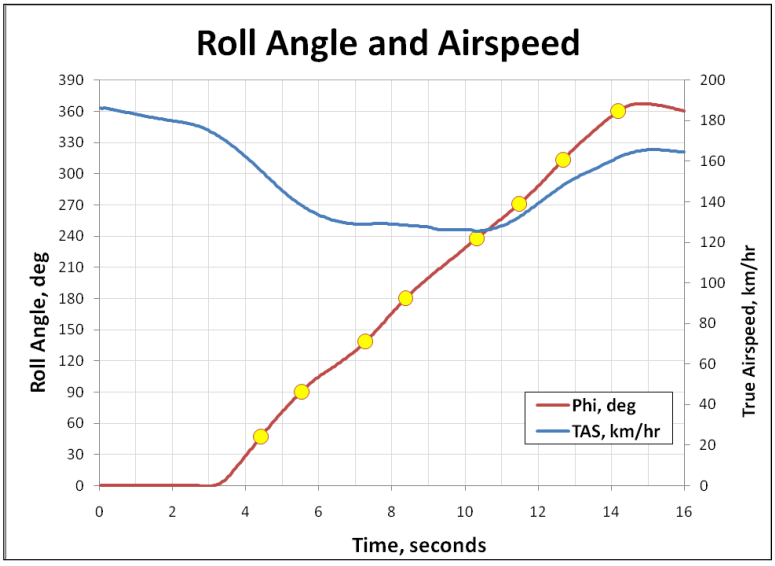

Figure 10. Time history (from a JSBSim run) of roll angle and true airspeed (TAS) for the complete super-slow tonneau maneuver.

B. Simulating the Maneuver with JSBSim

A detailed model of the sailplane has been produced in the JSBSim input format with the aid of several tools. The

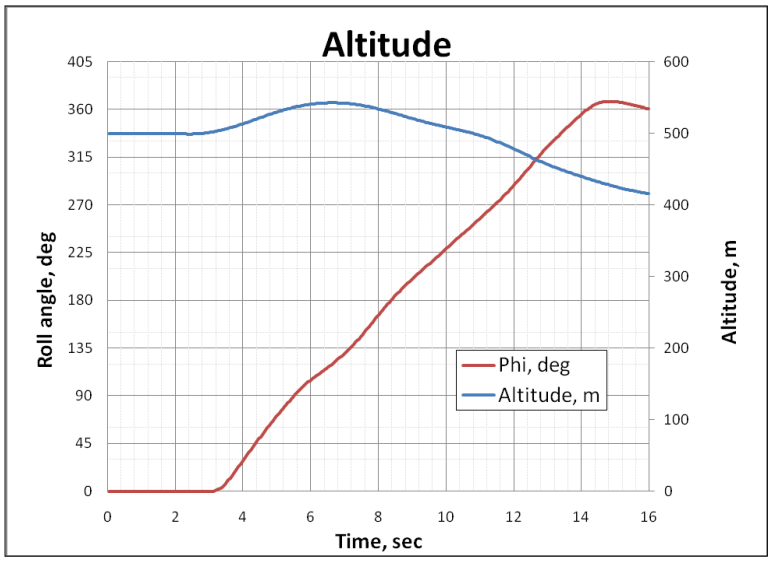

Figure 11. Time history (from a JSBSim run) for a complete super slow tonneau maneuver.

aerodynamics of wing and empennages has been analyzed with the well known two-dimensional viscous/inviscid code XFoil[9], combined with the open source software XFLR5[10]. The latter is a GUI (Graphical User Interface) application that allows for calculation of aerodynamic characteristics of a complete wing, that means including 3D effects and induced drag, in the presence of a fuselage and of tail surfaces. The XFLR5 code uses lifting line theory (LLT) in the case of wings with a simple geometry. It can switch to a vortex lattice method (VLM) or a panel method (PM) in order to include the effects of sweep or dihedral and of the presence of other bodies in the aerodynamic configuration. The program enables the user to generate a wing geometry with different geometrical sections, allowing different airfoils for each section. All aerodynamic characteristics of the airfoils used in the wing, in the whole range of Reynolds numbers and angles of attack, must be pre-calculated by XFOIL and be available in a database. For this task XFLR5 embeds a rewritten version of XFoil, which is nicely integrated into the main GUI application as a sub-tool that can be launched through one of the many menu options.

The modeling of the complete aircraft has been improved and completed with the well known aircraft design tool AAA by DARcorporation[11]. The full model of the G103C includes: an accurate estimation of fuselage aerodynamics on the total pitching and yawing moments, the non-linear effects at high angle of attack and high angles of sideslip, and the estimated stability and control derivatives.

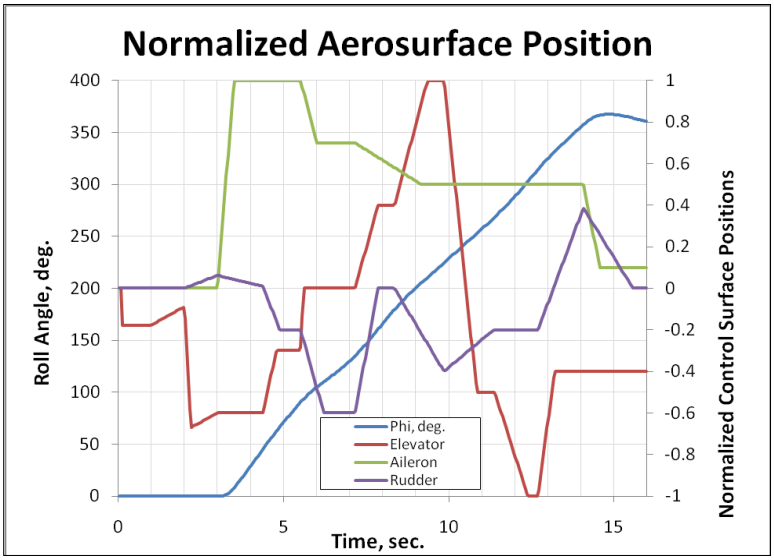

Figure 12. Normalized control surface position inputs.

C. Analysis of Simulated Results

Investigators have proven that a failure of the composite fuselage shell (located near the cockpit) caused the structural failure of the rear fuselage beam. The failure seems to be due to the appearance of a structural instability caused by the combined actions of torsion and bending which – at the second knife-edge of the maneuver – are particularly demanding for the structure.

Assessing the nature of the dynamic loads during this maneuver was required to defend the sailplane manufacturer from charges of poor structural design. Therefore, it was important to calculate the accelerations of the tail empennages during the development of the maneuver in order to correctly estimate the rear fuselage beam loading conditions, accounting for all the inherent inertia effects.

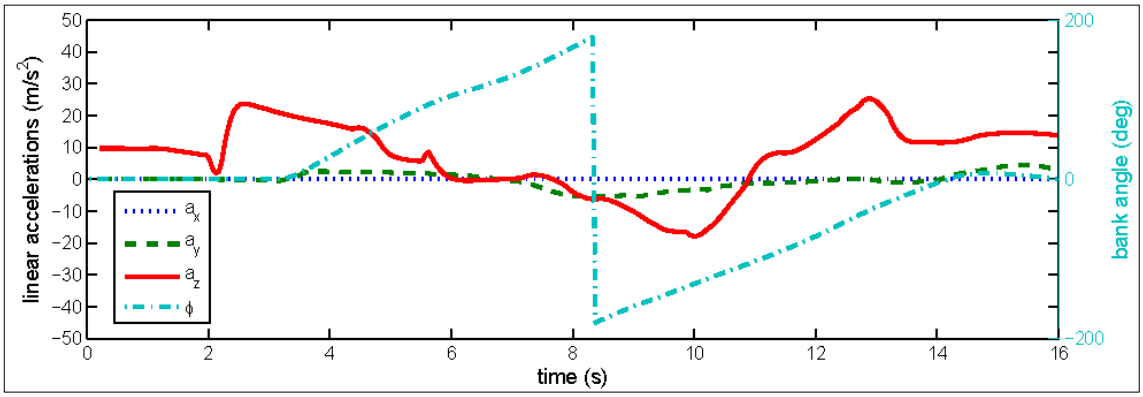

Figure 13. Time history (from a JSBSim run) of tail accelerations for a complete super-slow tonneau maneuver.

Figure 13 shows the simulated time histories of linear accelerations of a point of the sailplane located in the plane of symmetry, near the tail. They are calculated by JSBSim and are used to evaluate the actual time dependent loads on the tail empennage. [Note: the accelerations were calculated through the careful setting of the "pilot eyepoint" position in the configuration file. Accelerations at the pilot position are calculated, based on the linear acceleration of the aircraft, and include rotational effects. In the latest releases of JSBSim, the same results could be obtained using the new accelerometer control system component.]

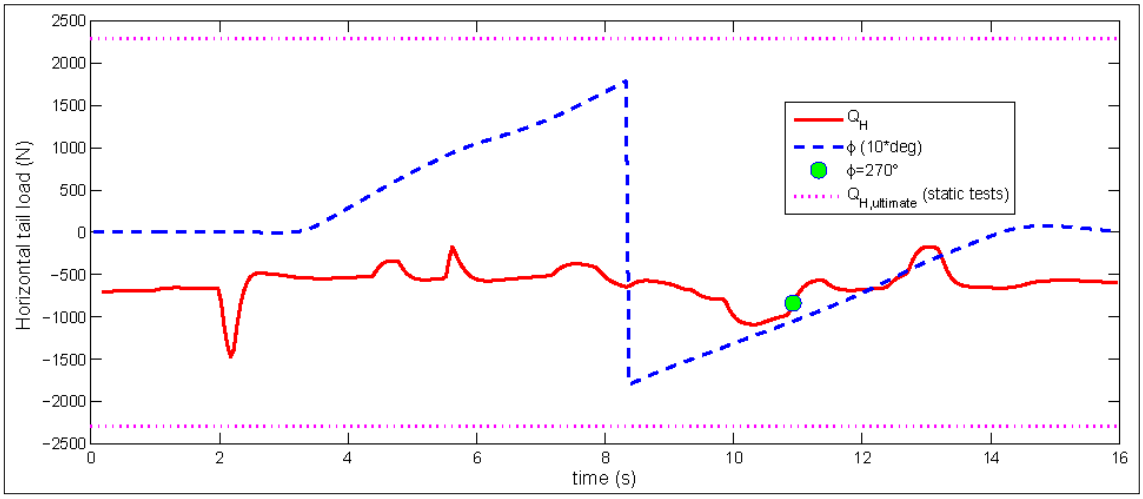

Figure 14. Time history (from a JSBSim run) of horizontal tail loading for a complete super-slow tonneau maneuver.

As it can be observed in Figure 14 (and in a similar plot for vertical tail loads not included here), the maximum dynamic loads are quite less than the effective maximum loads allowed by the structure. This result pointed to the investigation of possible alternative explanations for the accident. It was actually discovered later that the same sailplane experienced a hard landing some time before the fatal event, which required the replacement of the tail wheel and the repair of the rear fuselage tip.

VII. Conclusion

JSBSim has found a variety of uses in industry and academia, and has benefited from the exposure, through feature requests, shared expertise, suggestions, and trouble reports.

Future plans may include,

- Adding multi-body capability, allowing, for example, the modeling of multi-stage launch vehicles more seamlessly,
- Adding a formal turbulence model,
- Completing work on adding WGS84. JSBSim currently uses a spherical Earth model.
- Enhance compatibility with additional pre- and post-processing utilities and languages.

Appendix

A. Tonneau Maneuver Script

```
<use aircraft="g103c" initialize="init_tonneau"/>
<run start="0.0" end="16" dt="0.005">

<event name="Trim">
   <description>Trim at the initial conditions state</description>
   <condition>  sim-time-sec gt 0.1  </condition>
   <set name="simulation/do_simple_trim" value="1"/>
</event>

<event name="Trim tab off">
   <description>Set trim tab to zero</description>
   <condition> sim-time-sec gt 1 </condition>
   <set name="fcs/pitch-trim-cmd-norm" value="0" action="FG_RAMP" tc="2"/>
</event>

<event name="Enter the tonneau">
   <condition> sim-time-sec gt 2 </condition>
   <set name="fcs/elevator-cmd-norm" value="-0.6" action="FG_RAMP" tc="0.2"/>
   <set name="fcs/rudder-cmd-norm" value="0.1" action="FG_RAMP" tc="1.6"/>
   </notify>
</event>

<event name="Phase 1, Start rolling">
   <condition> sim-time-sec gt 3 </condition>
   <set name="fcs/aileron-cmd-norm" value="1" action="FG_RAMP" tc="0.5"/>
    <set name="fcs/rudder-cmd-norm" value="0" action="FG_RAMP" tc="1.6"/>
</event>

<event name="Phase 2">
   <description> 45 of 360 </description>
   <condition> attitude/phi-rad gt 0.785 </condition>
   <set name="fcs/elevator-cmd-norm" value="-0.3" action="FG_RAMP" tc="0.4"/>
   <set name="fcs/rudder-cmd-norm" value="-0.2" action="FG_RAMP" tc="0.5"/>
</event>

<event name="Phase 3">
   <description> 90 of 360 </description>
   <condition> attitude/phi-rad gt 1.57 </condition>
   <set name="fcs/elevator-cmd-norm" value="0" action="FG_RAMP" tc="0.1"/>
   <set name="fcs/rudder-cmd-norm" value="-0.6" action="FG_RAMP" tc="0.7"/>
   <set name="fcs/aileron-cmd-norm" value="0.7" action="FG_RAMP" tc="0.5"/>
</event>

<event name="Phase 4">
   <description> 135 of 360 </description>
   <condition> attitude/phi-rad gt 2.35 </condition>
   <set name="fcs/rudder-cmd-norm" value="0" action="FG_RAMP" tc="0.7"/>
   <set name="fcs/aileron-cmd-norm" value="0.5" action="FG_RAMP" tc="2"/>
   <set name="fcs/elevator-cmd-norm" value="0.4" action="FG_RAMP" tc="0.7"/>
</event>
```

```
<event name="Phase 5">
   <description> 180 of 360 </description>
   <condition> attitude/phi-rad gt 3.125 </condition>
   <set name="fcs/rudder-cmd-norm" value="-0.4" action="FG_RAMP" tc="1.5"/>
   <set name="fcs/elevator-cmd-norm" value="1" action="FG_RAMP" tc="1"/>
</event>

<event name="phase 6">
   <description> 225 of 360 </description>
   <condition>
      attitude/phi-rad ge -2.37
      attitude/phi-rad le -2.35
   </condition>
   <set name="fcs/rudder-cmd-norm" value="-0.2" action="FG_RAMP" tc="1.5"/>
   <set name="fcs/elevator-cmd-norm" value="-0.5" action="FG_RAMP" tc="1"/>
</event>

<event name="Phase 7">
   <description> 270 of 360 </description>
   <condition>
      attitude/phi-rad ge -1.6
      attitude/phi-rad le -1.57
   </condition>
   <set name="fcs/elevator-cmd-norm" value="-1" action="FG_RAMP" tc="1"/>
</event>

<event name="Phase 8">
   <description> 315 of 360 </description>
   <condition>
      attitude/phi-rad ge -0.8
      attitude/phi-rad le -0.78
   </condition>
   <set name="fcs/rudder-cmd-norm" value="0.4" action="FG_RAMP" tc="1.4"/>
   <set name="fcs/elevator-cmd-norm" value="-0.4" action="FG_RAMP" tc="0.5"/>
</event>

<event name="Finish tonneau">
   <description> Go wings-level </description>
   <condition>
      attitude/phi-rad ge -0.05
      attitude/phi-rad le -0.01
   </condition>
   <set name="fcs/rudder-cmd-norm" value="0" action="FG_RAMP" tc="1.5"/>
   <set name="fcs/aileron-cmd-norm" value="0.1" action="FG_RAMP" tc="0.5"/>
</event>
```

Acknowledgments

The authors would like to acknowledge the other JSBSim development team members, David Culp, Lee Duke, Mathias Fröhlich, Anders Gidenstam, Erik Hofman, David Megginson, and Tony Peden, as well as the many members of the JSBSim and FlightGear user community who have contributed to the improvement of JSBSim.

References

[1] Berndt, Jon S., et. al.. *JSBSim Reference Manual*, (work in progress), http://www.jsbsim.org/JSBSimReferenceManual.pdf [cited 1 August 2009].

[2] Jackson, E. Bruce and Hildreth, Bruce L., "Flight Dynamic Model Exchange Using XML," AIAA paper 2002-4482, August 2002.

[3] Berndt, Jon S., "JSBSim, An Open Source Flight Dynamics Model in C++," AIAA 2004-4923, *AIAA Modeling and Simulation Technology Conference*, Providence, RI, August, 2004.

[4] Ragsdale, W. A., "A generic landing gear dynamics model for LaSRS++," AIAA-2000-4303, *AIAA Modeling and Simulation Technologies Conference*, Denver, CO, Aug. 14-17, 2000.

[5] Clark, Samuel, *Mechanics of Pneumatic Tires*, US Dept. of Transportation, Aug. 1981, Chapter 1.

[6] Yan Zhang, Seamus McGovern, "Mathematical Models for Human Pilot Maneuvers in Aircraft Flight Simulation", *Proceedings of the ASME International Mechanical Engineering Congress and Exposition, IMECE 2009*, November 13-19, 2009, Lake Buena Vista, FL.

[7] Coiro D. P., De Marco A., Nicolosi F., "A 6DOF Flight Simulation Environment for General Aviation Aircraft with Control Loading Reproduction." AIAA Paper 2007-6364, AIAA Modeling and Simulation Technologies Conference and Exhibit 20-23 August 2007, Hilton Head, South Carolina, USA.

[8] Coiro, D. P., De Marco, A., & Leoncini, P. (2003). "Advanced Accident Flight Path Simulation and Innovative Visual Animation". *Proceedings of ESMC2003 (European Simulation and Modeling Conference).* Naples, Italy: EUROSIS, The European Multidisciplinary Society for Modeling and Simulation Technology.

[9] XFoil, Drela, Mark, and Youngren, Harold, Software Package, Ver. 6.97, April 2007, http://web.mit.edu/drela/Public/web/xfoil/ [cited 2 August 2009]

[10] XFLR5 Airfoil and Wing Analysis Tool, Depperois, A. Software Package, Ver. 4.17, June 2009, http://xflr5.sourceforge.net/ [cited 2 August 2009]

[11] AAA, Advanced Aircraft Analysis, DARCorporation, Software Package, Ver. 3.2, Lawrence, KS.