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

Back of the Envelope

VOLUME 3, ISSUE 2
SUMMER 2006

See Page 12:

Inside this issue:

Building an Aerobatics Aircraft for JSBSim : The Su-26

Enrique Laso Leon

What's the point?

Pure unlimited aerobatic aircraft have seldom been modeled in any mainstream simulator. Many will argue that the Extra 300 introduced in *Microsoft® Flight simulator-98* is a clear proof of the contrary, yet the *FS* flight model was (and still is) too limited to provide a decent sensation of flying. The most recent aerobatics planes correctly modeled are the *SF260* and *Spitfire MkXIV* from *Real Air®* but those are not dedicated "stunt airplanes".

The last serious attempt to model this category of aircraft was the first Flight Unlimited simulator back in 1996! It was nicely packaged with a fine tutorial (remember, this was at a time when games came with a manual) and allowed one to fly machines ranging from an aerobatic sailplane to the Su-31 unlimited aerobatic aircraft.

Many reasons can be found for the lack of interest in this category. The most obvious is that aerobatic aircraft are limited to Visual Flight Rules since instruments are heavy and sensitive to the kind of flying involved. What's worse, they have a very short range because a routine of 10 minutes of high-g maneuvers will bring the pilot to his (or her) knees, and the less fuel, the better for maneuverability. Finally, performing a correct maneuver requires a lot of training (first loops look awful when replayed!).

All this being said, it is an extremely rewarding way to fly as it gives a sense of the third dimension with fine management of potential vs. kinetic energy being the key.

Flight modeling is the most important part when recreating the feeling of aerobatic flight. Most of the maneuvers are performed inside or near a stall. Asymmetric stalls are also a key ingredient of maneuvers such as the snap roll. JSBSim already provides a fine experience of flying and, being open source, has the potential to evolve to give the whole envelope.

Why did I choose the Su-26? Let us say that it is a matter of personal taste. Sure enough, there are better aircraft around (Cap-232 or the superlative Su-31 come to my mind), but one can hardly fight his aesthetic judgment!

Modeling the Beast

Aerobatic aircraft have plenty of features that make them easy to model (which is good since this is my first add-on aircraft for FlightGear):

- Straight wing with little taper ratio (this allows using the lifting line theory for 3D wing derivations (Prandtl)
- Symmetric Airfoils
- Low Mach number (incompressible flow)
- Reciprocating engines
- No high lift devices (lift is obtained through thick airfoils)
- Simple avionics

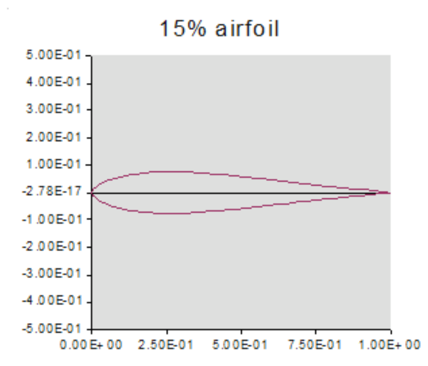
Nevertheless as I wrote before, the post-stall envelope must be modeled for the magic to begin and this is were it gets tough.

I did not have explicit references for the SU-26 airfoil. The only data I could get was a relative thickness (see *The Incomplete Guide to Airfoil Usage*, http://www.ae.uiuc.edu/m-selig/ads/aircraft.html). So, I started with a typical symmetric airfoil and scaled it to match a relative thickness of 15%. While this may look like an appalling approximation, thin airfoil theory indicate that the thickness law (which is the only variable for a symmetrical

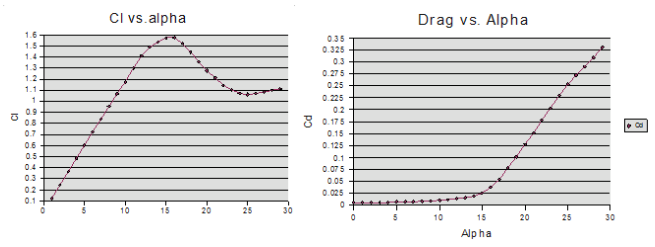
(Continued on page 2)

(Continued from page 1)

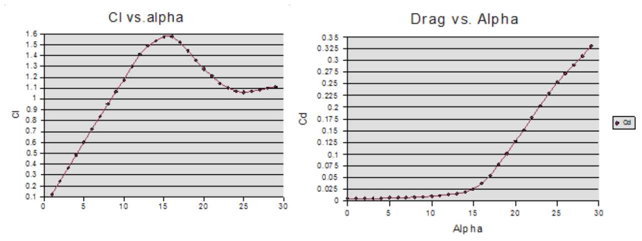
airfoil) only influences pressure repartition, not C_{lift} vs. alpha slope (2*π for incompressible flow). This in turn will change the stall characteristics (abruptness, critical angle, etc.), but I thought the difference would be too small for anybody to notice, provided the airfoil had a rounded leading edge:



The software used to compute the stall characteristics was xfoil (http://web.mit.edu/drela/Public/web/xfoil) which gave polar curves up to 30 degrees of angle of attack.

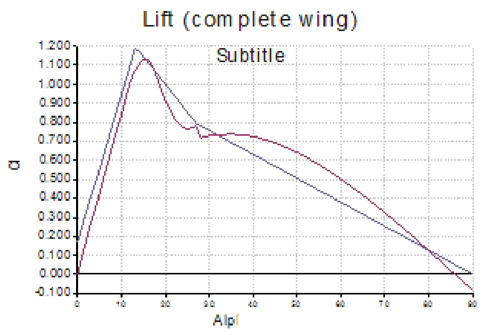


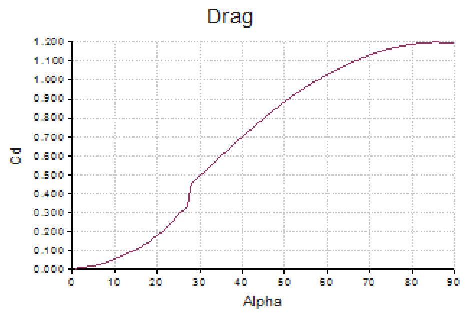
Post stall behavior derivations was found from an article written for the blades of wind turbines (Stall coefficient, aerodynamic airfoil coefficients at large angle of attack, C. Lindenburg réf. ECN-RX-001-04 Energy research center of the Netherlands).



These computations were made using Open Office Calc v2.0 (any spreadsheet would do for that matter). It should be noted that the curve from 0 to 30 degrees (xfoil) matches quite well with the curve for AOA higher than 30 degrees (post stall).

Derivations for the finite span wing was obtained using the Prandtl theory of the (vortex) lifting line (the one that gives the induced drag as a function of the aspect ratio). For post-stall behavior the article mentioned above provides a correction for finite span. The combination of both provides the following coefficient curves :





It is interesting to notice that the values given by Aeromatic (blue curve on the lift diagram, see www.jsbsim.org for Aeromatic) are not that far from the complete derivation. They would even be better if the airfoil camber effects could be taken into account (at zero camber, i.e. symmetrical airfoil lift at zero incidence is zero).

Drag at zero incidence was increased by imposing a minimum value in order to model fuselage drag (very very crude).

The result of all this is an aircraft accomplishing simple aerobatics neatly (barrel rolls, loops, Cuban eights) and even complex ones such as the spin with a nice feel. The latter came as a surprise since JSBSim comes without asymmetric stall effects so far. Snap rolls, while being possible, are somewhat slow and induce a large loss of altitude. Stall on the other hand is extremely brisk with g-loading dropping suddenly as elevator authority is huge (purposely in order to get adequate control all through the envelope)

Future Improvements

The beta version of the Su-26 will be released sometime soon in order to get a first feedback (hopefully from real life pilots, too!).

The model can stand a lot of improvement with

(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)

the current definition of JSBSim. The next step is to correctly model the moment coefficient of the whole aircraft (that is, taking the wing and tail separately) for the whole AOA range. This does not require a lot of additional theory, just to take into account the direction of motion (the "focus" is at a quarter of the chord for low incidence, but measured in the direction of the flow !).

Fine tuning of the propeller is really lacking for now. Particularly thrust in the low speed envelope is abundant, making landings quite difficult. Other improvements could come from evolutions of JSBSim, itself. In order to model post stall behavior, asymmetric effects could be taken into account, especially since propeller airplanes will tend to depart in asymmetric stall due to engine torque and P-factor.

Furthermore variations of lift along span would add an extra feeling, since loss of aileron efficiency at low speed is related to wingtips being more loaded than roots for tapered or swept wings. But in turn that would have little influence on aerobatics airplanes as they often have full span ailerons to handle this problem! ▲

Scripting Multiple Runs in JSBSim

Agostino De Marco

A couple of months ago I began to think over the following idea: *One may have the need to run JSBSim multiple times and compare the results; would it be worthwhile to implement a new capability into the scripting language supported by JSBSim's FGScript class that would make it possible to launch one or more successive simulations with one single script?*

After all, in each of the many scripting languages available today we always find the following elements: conditions (if-then-else clauses), selection (switch clauses), iterations (for clauses), inclusion of blocks of code (include/import clause) etc. Then, what if we had something vaguely similar in JSBSim scripts?

As someone said in the development-issues mailing list, there are many reasons for wanting multiple runs from a simulation. When designing or modifying a control or guidance system, one may want to evaluate the design at multiple test points within the flight envelope. Having the ability to set up a file with a number of initial conditions, trims, etc. would allow the user to do multiple tests without having to play around with a bunch of files.

Thinking at the simplest level, suppose that one has prepared N scripts, for example, *737_runway_1.xml, 737_runway_2.xml, ... , 737_runway_N.xml,* that clearly let us guess that the user is trying to simulate different variations of a take off run for the B737 aircraft. According to the above idea, he would like to have the chance to run JSBSim through each of the above cases by using a single script, the "driver", which could be something like:

```
<runscript name="B737 takeoff runs" action="MULTIPLE">
  <script file="737_runway_1.xml">
  <script file="737_runway_2.xml">
  ...
  <script file="737_runway_N.xml">

  <!-- this is just a naive attempt
       to define something... -->
</runscript>
```

A first problem arises here. After each run this approach should incorporate a "post-run" step to save the output (if specified), e.g. B737_datalog.csv, in a number of unique files, i.e. B737_datalog_1.csv, B737_datalog_2.csv, ... , B737_datalog_N.csv.

From the coders point of view the development of advanced scripting features would of course become quite demanding, as this job would be like implementing a sort of interpreter on top of the main JSBSim structure.

Actually, looking more closely at the kind of code that JSBSim is and how it is intended by the developers, even if the above "capability" would be nice to have, one can comment that (quoting Jon Berndt) this is the kind of thing that, in spite of being explicitly provided for in the scripting language, should be done externally, through a feature in JSBSimCommander or via a shell script. In fact, an application like JSBSim Commander, being that just a "Commander," will hopefully have that ability sooner or later.

Finally I convinced myself that the capability to launch multiple runs of JSBSim, collect output data appropriately, and prepare plots for the analysis should not be required to the flight dynamic model, because this is not his job, but should rather be implemented outside via a shell script. And that's what I did. Eventually, I will also to have an animation (i.e. an .avi file) of the simulation, automatically generated.

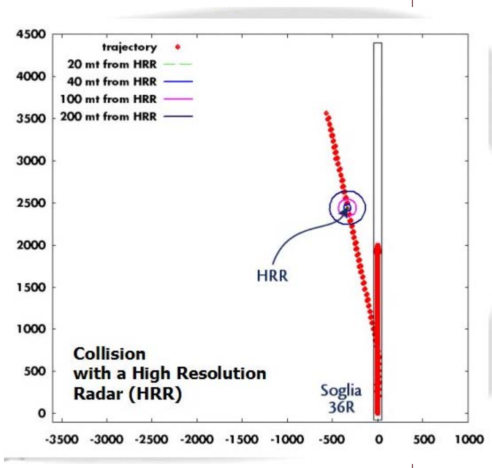
Multiple runs done in this way are really helping me in the analysis of a number of variants of standard flight maneuvers starting from steady equilibrated conditions, in flight and on ground.

Thanks to a grant from ENAV, the Italian authority on flight safety, I have analyzed a number of take off runs from Milano Linate air-

(Continued on page 4)

(Continued from page 3)

port. I have investigated on the possible failures during take-off runs and consequent inadequate pilot reactions/maneuvers that end up with the airplane colliding with a particular radar tower, which is going to be build in the vicinity of the main runway.



For the sake of brevity, I will discuss below, with an annotated example, how I have tackled the most relevant problem from JSBSim users' perspective: running JSBSim a number of times in a row through a shell script.

I will reserve some final observations as hints for the development of similar scripts that, from logged output files, generate relevant plots and automatically produce a document collecting them. This involves the use of well known, freely available, programs like Gnuplot and LaTeX.

Before going on further we have to note that the 0.9.11 version of JSBSim "suspends" the simulation, i.e. the time increment is set to zero when a crash is detected. Consequently, when JSBSim is launched as a stand-alone batched application and the simulation ends up with the aircraft crashing on the ground the program enters into an infinite idle loop and the user must manually kill the corresponding process.

The need to work with failure situations during take-off and with scripted runs has required me to modify a little bit JSBSim's main loop in order to have the program execution terminated each time, with no possibility of suspension.

The code snippet In Code Sample 2 illustrates the modifications I made in JSBSim.cpp.

Multiple runs of JSBSim are now easier to achieve with a bash (Bourne-Again SHell) shell script. Bash is a command language interpreter that executes commands read from the standard input or from a file. Bash also incorporates useful features from the Korn and C shells (ksh and csh), popular interpreters from the Unix world. Bash is freely available for all Linux distributions, for Unix and for Windows (Cygwin).

The basic idea behind multiple runs via a bash script (that I'll call "multiple-run.sh") is to concentrate in a given directory a number of JSBSim scripts, e.g. "B737_script1.xml", "B737_script2.xml", "B737_script3.xml", etc., and have a tool that performs the following tasks:

1. list the eligible xml files,
2. put them into an array,
3. generate a unique ID for each of them, and
4. launch JSBSim as many times as the number of cases listed.

The unique ID will help to rename and manipulate the output files further. A possible "multiple-run.sh" is listed in Script 1 (see page 5), with appropriate annotations.

It is a good idea to put the script where JSBSim.exe resides and give it the usual attributes (execute permission) like the following command does:

```
chmod u+rx ./multiple-run.sh
```

Code Sample 2

```
// *** CYCLIC EXECUTION LOOP, AND MESSAGE READING *** //

while (result) {
  while (FDMExec->ReadMessage()) {
    msg = FDMExec->ProcessMessage();

    switch (msg->type) {
      case JSBSim::FGJSBBase::Message::eText:
      {
        cout << msg->messageId << ": " << msg->text << endl;

// I just warn the user the simulation will be stopped anyway
#if defined(AGO_NO_SUSPEND) // agodemar
      string::size_type loc = msg->text.find( "Crash Detected: Simulation STOP", 0 );
      if ( loc != string::npos )
        cout << "............. simulation will be stopped !!!" << endl;
#endif
        break;
      }
    }
// ...
      if ( ! FDMExec->Holding()) {
        if ( ! realtime ) { // IF THIS IS NOT REALTIME MODE, IT IS BATCH
          result = FDMExec->Run();

// Here I break and stop the loop avoiding program suspension
#if defined(AGO_NO_SUSPEND)
          if ( FDMExec->GetState()->Getdt()==0.0 ) // check if SUSPENDED after CRASH
            break;
#endif
```

The multiple runs will be launched with the command,

```
./multiple-run.sh
```

[The command must be run in the directory where the shell script resides.]

Once all JSBSim runs are done (according to the contents of the directory <JSBSim root>/test/scripts) the user will find a bunch of output csv files in the JSBSim root dir. Their names start with a unique numeric code as produced in the script main loop.

At this point, it is a straightforward task to prepare a script that lists all

(Continued on page 6)

```
#!/bin/bash

# The sha-bang (#!) at the head of the script tells the system
# that the file is a set of commands to be fed to the indicated command interpreter,
# in this case /bin/bash. Thus #! is actually a two-byte magic number
# that designates here an executable shell script

echo "--------------------------------------------------"
echo " Multiple JSBSim runs "
echo "--------------------------------------------------"
echo

# JSBSim root dir, executable name, and working directory
# (change them conveniently for your system)

JSBSim_ROOT="l:/agodemar/jsbsim/JSBSim-0.9.11_dotNET2005/JSBSim/"

JSBSim_EXEC=$JSBSim_ROOT"JSBSim.exe"

WORK_DIR="test/scripts/"
# Note: must be a relative path to JSBSim root dir !!
# we will put here the scripts "B737_script1.xml", "B737_script2.xml", ... etc.

#----------------------------------------------------------------------------------
# collect script file names (from WORK_DIR)
#----------------------------------------------------------------------------------

SCRIPT_FILES0=`ls "$JSBSim_ROOT$WORK_DIR"B737_script*.xml`

# The back ticks `...` return the result of the system command, ls, into a string.
# File names returned here do not include the path.

#----------------------------------------------------------------------------------
# give the scripts a proper name, including the path
#----------------------------------------------------------------------------------

SCRIPT_FILES=
for file in $SCRIPT_FILES0
do
         file="$WORK_DIR/"$file                # prepend the appropriate path
         #echo $file
         SCRIPT_FILES="$SCRIPT_FILES"$file" "  # collect the names
done
echo "scripts: "$SCRIPT_FILES
echo

#----------------------------------------------------------------------------------
# run JSBSim multiple times
# and generate the unique ID for each run
#----------------------------------------------------------------------------------

prefix=
PREFIXES=

for scriptfile in $SCRIPT_FILES
do

  if [ ! -e "$scriptfile" ]               # Check if file exists.
  then
    echo "$scriptfile does not exist."; echo
    continue                              # On to next scriptfile in the list.
  fi

  # File exists, print name and size

  ls -l $JSBSim_ROOT$scriptfile | awk '{ print $9 "         file size: " $5 }'

  # This is an example of piping (<command 1>|<command 2>):
  # $JSBSim_ROOT$scriptfile expands to the name of the current JSBSim script that is to
  # be used for the simulation. Command ls -l returns a number of fields, whose 9th and 5th
  # are printed by the awk utility (actually on cygwin gawk, Gnu awk, is invoked)

  # Now generate a unique id by using the date command
  prefix=$(date +%N)

  # "+%s" option (GNU-specific): seconds since 1970-01-01 00:00:00 UTC
  # "+%N"                      : nanoseconds (000000000..999999999)

  # The following alternative would strip off leading and trailing zeroes, if present.
  #    prefix=`date +%N | sed -e 's/000$//' -e 's/^0//'`
  # by using sed

  # store ID in an array
```

(Continued on page 7)

(Continued from page 4)
these newly created csv file (for example by looking for the XXXX_jsbsimrun.log). Retrieving their unique prefix, e.g. like in
00091110230_B737_velocities.cvs
000622010230_B737_velocities.cvs etc, may prove useful in plotting the results.

Suppose one has prepared the simple Gnuplot script:

```
#------------------------------------------------------------
#Gnuplot script
#------------------------------------------------------------
# set the terminal and the output file
set term postscript color enhanced linewidth 2.0
set out "CHANGEME_B737_velocities.ps"
set key top left
# plot from file
plot \
"CHANGEME_B737_velocities.csv" u 1:($13/10) w l t "V_C [kts/10]",\
"CHANGEME_B737_position.csv" u 1:($2/100) w l t "h [ft/100]",\
"CHANGEME_B737_aerodeflections.csv" u 1:($4*10) w l t "delta_e [deg*10]",\
"CHANGEME_B737_attitude.csv" u 1:($3*180.0/pi) w l t "theta [deg]"
set out # free the output
#    EOF
#------------------------------------------------------------
```

Gnuplot script, “template-velocities.plt”

One can parse the Gnuplot script with sed, substitute the string "CHANGEME" with the unique IDs retrieved from the list of JSBSim output files, and produce a unique plotting script, e.g.
00091110230_B737_velocities.plt
000622010230_B737_velocities.plt, etc. The corresponding postscript figure will be produced by simply launching Gnuplot with the appropriate plotting script on the command line:

```
gnuplot 00091110230_B737_velocities.plt
gnuplot 000622010230_B737_velocities.plt
…
```

It is worth making this procedure automatic like the one that launches JSBSim multiple times.

The same ideas apply to the generation of a LaTex document: scan for *.ps, retrieve the unique prefixes, parse a template document, generate a unique one, collect all, and finally run through the LaTeX compiler to obtain a final postscript or pdf document with all the desired plots for all the simulated cases. ▲

Scripting Changes in JSBSim

Jon S. Berndt

Substantial changes have been made to the scripting capability for JSBSim. In doing so, the version number has been incremented to 0.9.12.

The scripting changes were needed for some testing and debugging. Some of the items had been getting thought about for a year or two.

It should not be too difficult to modify existing scripts to run using the new format. There are new attributes and keywords for some of the elements (example below). The major changes are:

- The "when" element is changed in name to "event"
- The conditional test[s] that must be fulfilled for a set of actions to take place is handled by the FGCondition class from the flight controls code. The class now moves to the math/ subdirectory as it will have even more use in the near future.
- The "persistent" element is now an attribute of the event element.
- You can specify to "notify" when an event is triggered, which results in a message being printed out.

(Continued on page 8)

```
<?xml version="1.0"?>
<runscript name="C172-01A takeoff run">
  <!-- This run is for testing the C172 altitude hold autopilot -->
  <use aircraft="c172x" initialize="reset00"/>
  <run start="0.0" end="3000" dt="0.0083333">
    <event name="engine start">
      <notify/>
      <condition> sim-time-sec >= 0.25 </condition>
      <set name="fcs/throttle-cmd-norm" value="1.0" action="FG_RAMP" tc="0.5"/>
      <set name="fcs/mixture-cmd-norm" value="0.87" action="FG_RAMP" tc="0.5"/>
      <set name="propulsion/magneto_cmd" value="3"/>
      <set name="propulsion/starter_cmd" value="1"/>
    </event>
</runscript>
```

(Continued from page 5, Scripting Multiple Runs in JSBSim)

```
  PREFIXES=$PREFIXES" "$prefix

  echo "running JSBSim ..."

  JSBSim_RUN_CMD=$JSBSim_EXEC" --root="$JSBSim_ROOT" --script="$scriptfile

  # ...here only assembles the command line for launching JSBSim

  echo $JSBSim_RUN_CMD

  echo "writing log in file: "$prefix"_jsbsimrun.log"

  $JSBSim_RUN_CMD > $prefix"_jsbsimrun.log"

  # ...here the command line is actually launched and the output redirected into a
  # unique log file
  # Note: this log file, actually its name, will be useful in post processing
  #       task by other scripts

  # Next is a list of output files pre-defined appropriately in the aircraft
  # config file.
  # I assume here that the scripts deal with a single aircraft type, i.e. "737.xml".
  # The output properties are specified in that aircraft config file and are logged
  # to different files for convenience of analysis

  OUTPUT_FILES="B737_accelerations.csv
              B737_attitude.csv
              B737_forcesmoments.csv
              B737_position.csv
              B737_velocities.csv
              B737_aerodeflections.csv
              B737_commands.csv
              B737_gear.csv
              B737_propulsion.csv
              B737_animation.csv"

  echo "moving output files ..."

  # Now the current JSBSim output files are renamed into unique file names by prepending
  # the id generated above

  MOVE_CMD=

  for outputfile in $OUTPUT_FILES
  do
      MOVE_CMD="mv $JSBSim_ROOT$outputfile"" "$JSBSim_ROOT$prefix"_"$outputfile
      echo $MOVE_CMD
      $MOVE_CMD
  done

  echo
done

# summary
echo "--------------------------------------------------"
echo "Summary of runs"
echo "--------------------------------------------------"

# count runs
n=0
for logprefix in $PREFIXES
do
  let "n = $n + 1"
done

echo "N. of JSBSim runs: "$n
echo -n "Run log-files: "
for logprefix in $PREFIXES
do
  echo -n $logprefix".log "
done
echo
echo "--------------------------------------------------"

exit 0
```

(Continued from page 6)

- Events can be named.
- A delay can be specified for an event, so that it executes a certain number of seconds after it is triggered.
- An event can be a notify-only event. That is, you do not have to specify actions for an event, it can exist to notify the user of some kind of event.
- The "use" command combines selection of the initialization file and the aircraft.
- Properties can be defined in a script, so that events can be repeated, etc.

In order to help convert scripts from the old format to the new one, an XSL transformation has been created and is available for download at:

http://jsbsim.sourceforge.net/convert_script.xsl

Additionally, an XSL script has been created for the script itself, so double-clicking on the script in Microsoft Explorer (for instance) or other file manager application should bring up the file in a browser, formatted to be more easily human readable (see accompanying illustration). See:

http://jsbsim.sourceforge.net/JSBSimScript.xsl

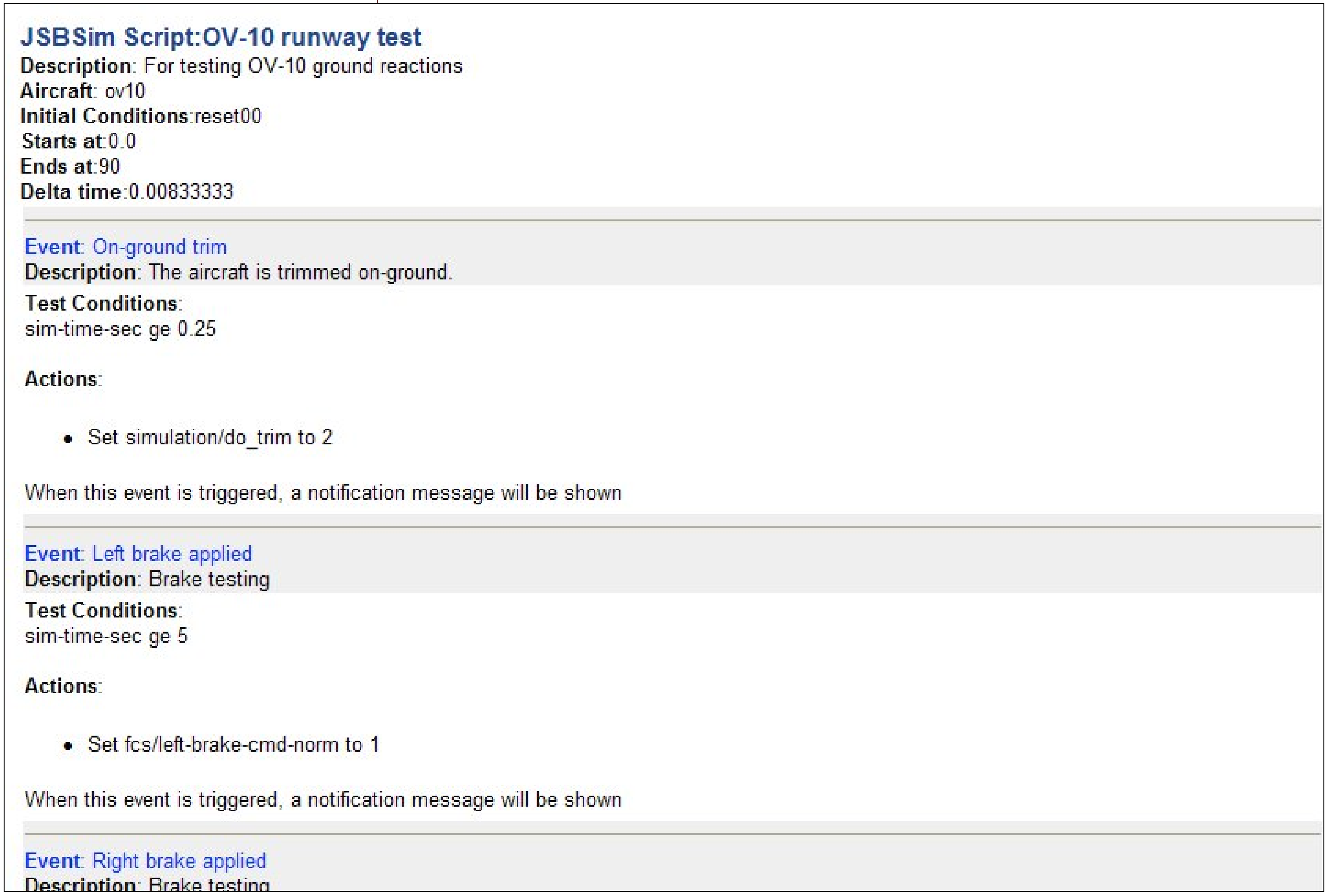
An XML Schema has also been written that allows a JSBSim script to be validated. The schema is located at the JSBSim web site at:

http://jsbsim.sourceforge.net/JSBSimScript.xsd

The conditional tests are now modeled using a "standard" JSBSim conditional construct, that is also used in the flight control switch component. When coupled with the new property declaration capability and the notify flag, some interesting effects can be achieved. One of those is a repeating notification that is printed at time or altitude intervals. For example:

```
<run start="0" end="100" dt="0.0083333">
  <property>
   simulation/notify-time-trigger
  </property>
…
  <event name="Time Notify"
        type="FG_DELTA" persistent="true">
   <description>Interval</description>
   <notify/>
   <condition>
    sim-time-sec >= simulation/notify-time-trigger
   </condition>
   <set name="simulation/notify-time-trigger"
     value="100" type="FG_DELTA"/>
  </event>
</run>
```

The above construct will print out a message every 100 seconds. ▲



JSBSim Script:OV-10 runway test
Description: For testing OV-10 ground reactions
Aircraft: ov10
Initial Conditions:reset00
Starts at:0.0
Ends at:90
Delta time:0.00833333

Event: On-ground trim
Description: The aircraft is trimmed on-ground.
Test Conditions:
sim-time-sec ge 0.25

Actions:

- Set simulation/do_trim to 2

When this event is triggered, a notification message will be shown

Event: Left brake applied
Description: Brake testing
Test Conditions:
sim-time-sec ge 5

Actions:

- Set fcs/left-brake-cmd-norm to 1

When this event is triggered, a notification message will be shown

Event: Right brake applied
Description: Brake testing

JSBSim and Microsoft Flight Simulator: The FGJSB Project

Oleksiy Frolov, Majestic Software

FSJSB project was started by Majestic Software (www.majesticsoftware.com) with a goal to integrate a better and more predictable JSBSim FDE into Microsoft Flight Simulator 2004 to achieve:

- More realistic aircraft handing
- Utilization of the Microsoft flight simulator 2004 scenery and environment engines for better debugging and testing the JSBSim format aircrafts
- Adding the flexibility to the aircraft add-ons development for Microsoft Flight Simulator, which default MSFS2004 FDE could not provide. This includes the separate axis for the steering and rudder operation, JSBSim scripting, autoflight functions, as well as correct ground behaviors of the aircraft.

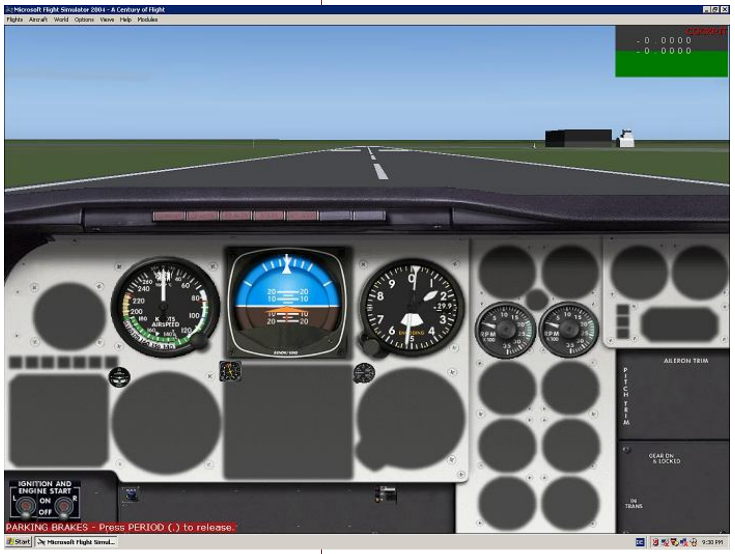
Project Structure

The project executable is wrapped within a single MSFS2004 format gauge file (FSJSB.gau) and provides 3 main functions:

- Starting, stopping and managing the built-in JSBSim engine,
- Interfacing between the JSBSim engine and the MSFS (requires a registered Pete Dowson's FSUIPC utility),
- Synchronizing the JSBSim and MSFS2004 execution.

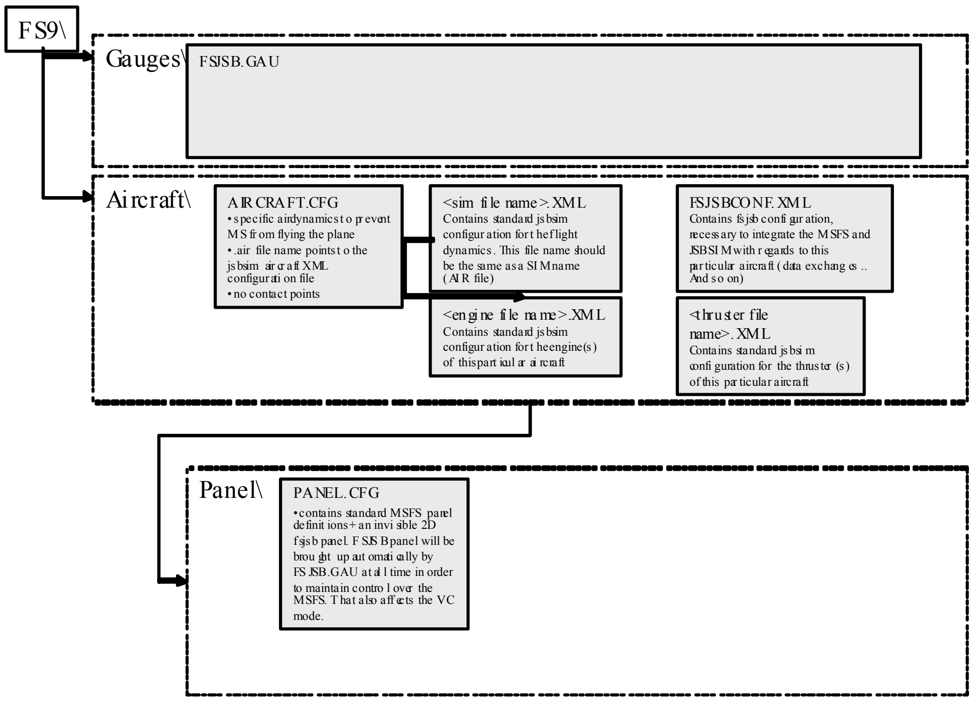
Project modules

JSBSim wrapper: instantiates and manages execution of the JSBSim engine (class instance) , using windows multimedia timer running at a frequency of 124 Hz.


Microsoft Flight Simulator 2004 - A Century of Flight
AIRSPEED
AILERON TRIM
PITCH TRIM
GEAR DN & LOCKED
IN TRANS
IGNITION AND ENGINE START
ON
OFF
PARKING BRAKES - Press PERIOD (.) to release.
Start

Interface module: Manages a data exchange between Microsoft Flight Simulator and the JSBSim wrapper. The interface module configuration is stored in an external file (namely FSJSB. XML) in the propriety XML format, allowing the specification of the data flow as well as simple mathematical transformations. Interface module also provides means to direct the information flow to the XML format MSFS2004 gauges in order to bypass the internal Microsoft FDE and display the JSBSim data at the cockpit gauges directly.

The FSJSB project is currently at the beta stage, and is free for testing and utilization in the research and non-commercial applications (to obtain the distributable, please contact support@majesticsoftware.com).



AIRCRAFT.CFG
- specific airdynamics to prevent MS from flying the plane
- .air file name points to the jsbsim aircraft XML configuration file
- no contact points

<sim file name>.XML
Contains standard jsbsim configuration for the flight dynamics. This file name should be the same as a SIM name (AIR file)

<engine file name>.XML
Contains standard jsbsim configuration for the engine(s) of this particular aircraft

FSJSBCONF.XML
Contains fsjsb configuration, necessary to integrate the MSFS and JSBSIM with regards to this particular aircraft (data exchanges .. And so on)

<thruster file name>.XML
Contains standard jsbsim configuration for the thruster (s) of this particular aircraft

PANEL.CFG
- contains standard MSFS panel definitions + an invisible 2D fsjsb panel. FSJSB panel will be brought up automatically by FSJSB.GAU at all time in order to maintain control over the MSFS. That also affects the VC mode.

News

JSBSim Moves to Lesser GPL License

After considering the prospect for almost two years, JSBSim has changed its license to the LGPL (Lesser General Public License). Why did we do this? The use of the LGPL is generally discouraged, in favor of the regular GPL, except in certain cases. From the Gnu web site: "*For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.*"

For more information, see the full text of the license at: http://www.gnu.org/licenses/lgpl.html.

PID Component Added to JSBSim Flight Control System Components

A proportional-integral-derivative control component has been added to the set of JSBSim flight control components. The component can represent any of the three control actions by itself or in any combination. Also, a trigger property can be specified that—if the value of that property becomes non-zero—the integrator inputs will be set to zero. This will prevent integrator wind-up.

The format for specifying the PID component is as follows:

```
<pid name="name">
 <input> property </input>
 <kp> p_value </kp>
 <ki> i_value </ki>
 <kd> d_value </kd>
 <trigger> property </trigger>
 <clipto> <min> value </min>
          <max> value </max> </clipto>
</pid>
```

Example:

```
<pid name="fcs/altitude-hold-pid">
 <input> fcs/ap-alt-hold-switch </input>
 <kp> 0.031 </kp>
 <ki> 0.000028 </ki>
 <trigger> fcs/windup-trigger </trigger>
 <clipto> <min>-1.0</min>
          <max> 1.0</max> </clipto>
</pid>
```

JSBSim Version 0.9.12 Released

A new version of JSBSim has been released (or will be shortly). The new version will include various fixes and additions, including the PID component mentioned above, the new scripting capabilities, some bug fixes, etc. You can download the new source and/or executable at www.jsbsim.org.

New Integrators Implemented in JSBSim

In recent testing, it was discovered that a tumbling, unpowered, vehicle following a ballistic trajectory would not hit the ground, but instead would oscillate about a particular altitude. Further investigation revealed that the Euler integrator used in JSBSim was not accurately propagating the state. [It should be noted that this was an unusual situation, and that in normal use with conventional aircraft, the use of an Euler integrator is usually fine.]

By shrinking the time step (down to about 1/5000 seconds), a solution was arrived at. However, integrating at 5000 Hz is not a viable solution in most cases! So, several additional single-pass integration schemes were added to FGPropagate: Trapezoidal, 2nd order Adams-Bashforth, and 3rd order Adams Bashforth (the current, experimental, default):

$$y_{t+1} = y_t + \frac{\Delta t}{12}[23\dot{y}_t - 16\dot{y}_{t-1} + 5\dot{y}_{t-2}]$$

This integrator was able to provide a solution for the unusual case above using a simulation execution rate of 120 Hz.

Landing Gear Refinements Continue

Efforts continue towards reducing jitter sometimes found in aircraft flight models. When the aircraft is stationary on the ground (V=0) the slip/skid angle is undefined, and that can cause problems. JSBSim has mostly eliminated that specific problem, but there remains a problem if brakes are applied while at rest, or at very low speed.

The techniques applied to reduce jitter and motion in JSBSim are:

- Coriolis acceleration is not applied while there is weight-on-wheels (WOW)
- Wind effects are only applied as the aircraft gains speed
- Centrifugal acceleration is not applied until the aircraft leaves the runway
- Wheel/brake forces are faded out at very small velocity
- A filter is applied to wheel slip angle and wheel forces at very low velocity

Proper choice of landing gear spring and damping coefficients also helps to obtain the best gear performance. ▲

Modeling Aerodynamic Moments in JSBSim

Jon Berndt

When modeling aerodynamic forces and moments in JSBSim, one might be tempted to simply copy the force and moment coefficients from a textbook or paper that describes flight test data for the aircraft in question. It is important to understand where the data comes from, how it was obtained, and – most importantly – what it really represents.

This was made clear recently when a JSBSim user was modeling the flight of a rocket, using aerodynamic properties calculated by the Missile DATCOM program. The user had set up Missile DATCOM to return moment coefficients that were *referenced to* the CG (center of gravity). A question was raised, however, because there is program code in JSBSim that looks like this:

```
vForces = State->GetTs2b()*vFs;

vDXYZcg = MassBalance->StructuralToBody(Aircraft->GetXYZrp() +
                                                          vDeltaRP);

vMoments = vDXYZcg*vForces; // M = r X F

for (axis_ctr = 0; axis_ctr < 3; axis_ctr++) {
  for (ctr = 0; ctr < Coeff[axis_ctr+3].size(); ctr++) {
    vMoments(axis_ctr+1) += Coeff[axis_ctr+3][ctr]->GetValue();
  }
}
```

This may appear confusing at first, because the moments (vMoments) are calculated, then – apparently – written over just a few lines later. Upon closer examination, one can see that the moments are actually added together (see the "+=" operator in the vMoments equation). The moments are initially set to a value:

```
vMoments = vDXYZcg * vForces
```

which represents,

```
M = r × F
```

where *r* represents a radius vector from the ARP (aerodynamic reference point) to the CG. The *F* represents the body force vector (lift, drag, and side forces transformed to the body axes) acting on the CG. This *couple* is later summed with the total *aerodynamic* moment in the nested *for* loop as seen above.

What is the ARP, and why is it necessary? How does it relate to the data I am using to model an aircraft (or rocket) – the data I find in textbooks and tech reports, or from flight test, or from a program such as Missile DATCOM?

If one integrates the pressure difference over the upper and lower surfaces of an airfoil, the net lift is determined. The net lift acts at the center of pressure (this is <u>not</u> the same as the aerodynamic center). If one chooses a point on the airfoil, one can also determine the moment produced by the pressure distribution over the airfoil – however, the moment will be different about any unique point on the airfoil. The most commonly selected point to reference the moment to is the wing quarter chord point. At this point, most airfoils (at subsonic velocities) will have a constant moment coefficient, independent of angle of attack (at least below the stall).

To bring this into perspective for a real aircraft, and data taken from a technical report, consider this excerpt from NASA Technical Memorandum 72863, "Stability and Control Derivative Estimates Obtained from Flight Data for the Beech 99 Aircraft":

"*The derivative* $C_{m_{\alpha,c/4}}$ *must be resolved to a flight center of gravity position of 26 percent of the mean aerodynamic chord. This is accomplished in the following equation:*

$$C_{m_\alpha} = C_{m_{\alpha,c/4}} - C_{L_\alpha}\left(X_{CG,\,flight} - X_{c/4}\right)$$

So, the moment coefficient data given in this technical report is referenced to the initial flight CG at 26% mac. The ARP should be set to this point. When the fuel burns off and the CG changes, it will automatically be taken into account.

This feature in JSBSim can be handy in modeling the aerodynamics of a rocket, which might reach a very high velocity, and see the center of pressure vary widely as it passes through mach 1. In fact, I have recently seen one rocket modeled in JSBSim where the lift, side, and drag forces—acting at the ARP—were the sole means responsible for modeling the moments on the vehicle. ▲

Wingtip Vortices—The beautiful image below illustrates trailing wingtip vortices exceptionally well (in addition to being an exceptional photograph). This photograph is of a Boeing 777 approaching London's Gatwick airport. The image was taken on July 10, 2006, by Steve Morris, AirTeamImages. The photograph is reprinted here

For more information on OpenEaagles see www.openeaagles.org

OpenEaagles Simulation Framework Utilizing JSBSim

Douglas Hodson, Chris Buell

OpenEaagles is an open source C++ framework designed to support the rapid construction of virtual (human-in-the-loop) and constructive simulation applications. It has been used extensively to build DIS compliant distributed simulation systems. It is based upon EAAGLES (Extensible Architecture for the Analysis and Generation of Linked Simulations), a popular simulation framework developed and maintained by the U.S. Air Force to support a multitude of simulation activities.

As a framework, OpenEaagles provides a design pattern for how to construct a simulation. The goal is to provide an application developer a solid foundation so that robust, scalable, virtual, constructive, stand-alone, and distributed simulation applications can easily be built. It leverages modern object-oriented software design principles while incorporating fundamental real-time system design techniques to meet human interaction requirements.

By providing abstract representations of system components (that the object-oriented design philosophy promotes), multiple levels of fidelity can be easily intermixed and selected for optimal run-time performance. Abstract representations of systems allow a developer to tune the application to run efficiently so that human-in-the-loop interaction latency deadlines can be met. On the flip side, constructive-only simulation applications, that do not need to meet time-critical deadlines, can use models with even higher levels of fidelity.

The bulk of the development for the EAAGLES software has been performed at the Simulation and Analysis Facility (SIMAF) located at Wright Patterson AFB, Ohio. SIMAF participates in a number of distributed events each year. The vast majority of the distributed simulation software used in the facility has been "home grown" utilizing the EAAGLES framework. Applications built utilizing the framework include cockpits (F-16), ground control stations (Predator MQ-9), threat Integrated Air Defense Systems (IADS) and a futuristic battle manager.

OpenEaagles has been released as public domain code. This was done to encourage its use throughout the community. OpenEaagles closely tracks and incorporates new enhancements to EAAGLES, but does not include some functionality.

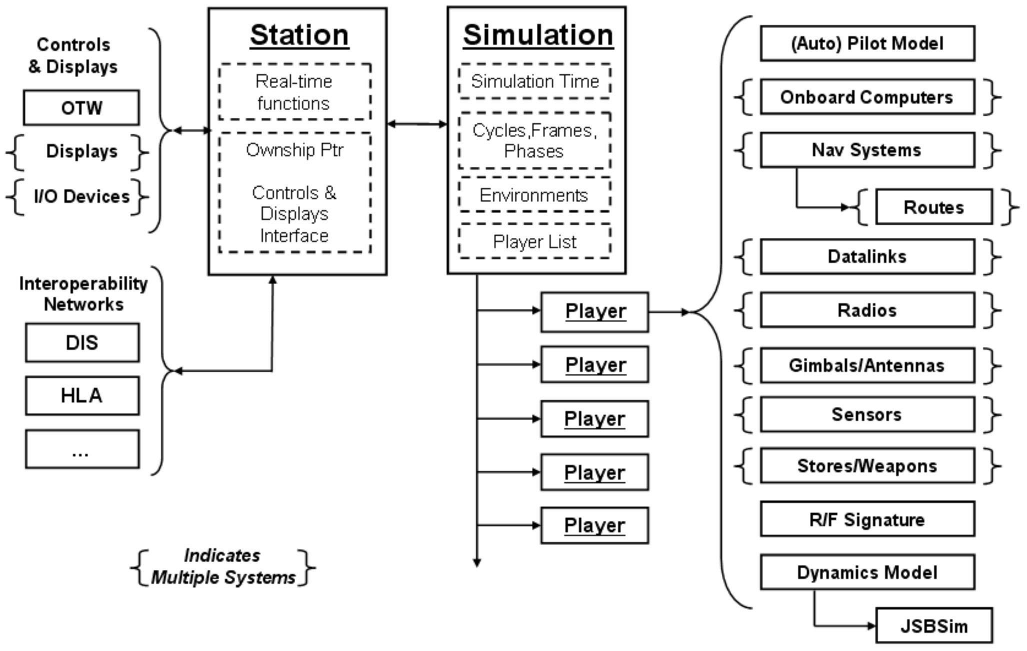


Figure 1. The OpenEaagles Simulation Framework

Interfacing to JSBSim

OpenEaagles is a framework that serves as a simulation design pattern as shown in Figure 1. Most of the elements that are needed to build a full-up simulation application exist within the framework, with one notable exception, the function main(). This function resides with the executable application, not the framework.

Typically main() is closely associated with the Station class which connects the simulation models to graphics and device I/O. The Station class also contains a Simulation class which takes care of basic simulation activities such as managing a list of Players/Entities of interest. A Player can have many components and systems attached to it. Of

(Continued on page 13)

(Continued from page 12)

particular interest related to JSBSim is a Players dynamics class. Each Player can have an associated dynamic model.

OpenEaagles provides a dynamic model that serves as an interface class, but unfortunately, specific flight models could not be included in the public domain release. Because JSBSim is a mature model it made sense for the OpenEaagles project to utilize it. Since JSBSim is also coded in C++, extending the included dynamics class to utilize JSBSim was fairly straightforward.

Since JSBSim can be configured to represent several aero propulsion systems, we have refined the OpenEaagles dynamics class to include features never before considered. For example, most of the aerodynamic models we have encountered, treat multiple engines as one. We did include the capability to model multiple engines in the dynamics class but it had never been tested. Attributes associated with engines other than jet turbines also needed to be considered. Specifically, the Propeller and TurboProp engines which requires additional controls (fuel mixture, prop pitch, etc).

Calling JSBSim

OpenEaagles is a frame-based system in which code is partitioned to support the development of real-time systems. As a frame-based system, delta time is passed as an argument to models so proper calculations involving time can be performed. Having models rely on delta time for calculation means the frequency of the entire system can change without having to change each and every model (so long as Nyquist rates are met). Additional time related information is recorded in terms of cycles (16 frames or sometimes called a major frame) and phases. Phases sequence the flow of data throughout a model. Four phases are currently defined:

- Dynamics -- update player or system dynamics including aerodynamic, propulsion, and sensor positions (e.g., antennas, IR seekers).
- Transmit -- R/F emissions, which may contain datalink messages, are sent during this phase. The parameters for the R/F range equation, which include transmitter power, antenna pattern, gains and losses, are computed.
- Receive -- Incoming emissions are processed and filtered, and the detection reports or datalink messages are queued for processing.
- Process -- Used to process datalink messages, sensor detection reports and tracks, and to update state machines, on-board computers, shoot lists, guidance computers, autopilots or any other player or system decision logic.

JSBSim is currently being called in the dynamics phase at a rate of 50Hz.

Building Applications

Building an application with OpenEaagles consists of extending select classes of interest and writing a mainline. Typically, the mainline will call an OpenEaagles provided parser that will read an input file, and create the object hierarchy which is the simulation application. The input file describes in a simple language, the objects to create and the attributes to set.

Wherever a Player is defined, the creation of a JSBSimModel object (which was subclassed off of the DynamicsModel class) can be specified as the dynamics model of choice. The OpenEaagles inputs associated with the JSBSimModel class include the root directory for the JSBSim data files and a string that specifies the model of interest. With this information the interface class will call JSBSim, thus allowing it to process its own input files.

Thanks to C++ and the object-oriented nature of OpenEaagles and JSBSim, multiple instantiations of JSBSim (i.e. multiple Players) can easily be created and utilized within the same simulation application.

We see JSBSim being utilized as a high fidelity aero model driven by a human operator. The example fighter cockpit application, as shown in Figure 2, is a full-up simulation application where the pilot is controlling or flying one of the players in the player list. All stick and throttle inputs as well as graphic outputs are associated with the Player through the use of an "ownship" pointer that exists in the Station class. This connection is typically made by extending the Station class and customizing it for the application being built.

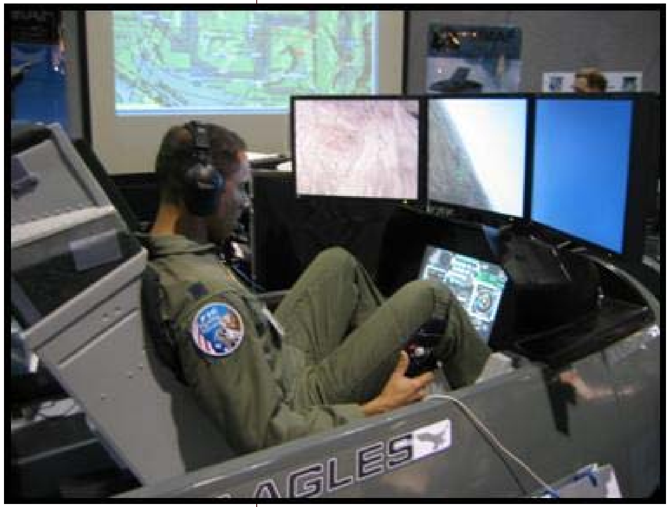


Figure 2

Examples such as this will be demonstrated at I/ITSEC 2006 (Interservice/Industry Training, Simulation & Education Conference). This year JSBSim will be the dynamics model of choice.

Final Thoughts

The OpenEaagles development team is convinced that we should utilize JSBSim as an important 3rd party tool, not only to fill a void in OpenEaagles, but also as an addition to other flight dynamics models currently utilized within the Air Force. We are excited by the possibilities of modeling a more diverse set of air vehicles. ▲

The 2006 AIAA Modeling and Simulation Conference in Keystone, Colorado

Jon S. Berndt

I had the opportunity to attend the 2006 American Institute of Aeronautics and Astronautics (AIAA) Modeling and Simulation Technology (MST) Conference this year. It was held at Keystone in Colorado. The location was breathtaking, and the conference was great. The AIAA conference held in August of each year is really a series of four parallel conferences: the Guidance, Navigation, and Control (GNC) Conference; the Atmospheric Flight Mechanics (AFM) Conference; the AIAA/AAS Astrodynamics Specialist Conference, and the MST Conference, as mentioned. When you register for one conference, you gain access to all the conferences.

This year, among the several workshops given, there was one that demonstrated the use of AERO-ML (a.k.a. DAVE-ML). AERO-ML is a pending AIAA standard that specifies a formal data format as a medium for exchanging aircraft aerodynamic characteristics between different simulations. The workshop speaker was Bruce Jackson (NASA Langley, "LaRCSim" author). I attended this interesting workshop, and even said a few words about the use of XML by JSBSim. Also present at the workshop was Geoff Bryan, an engineer from the Defence Science and Technology Organisation (DSTO), a government research organization in Australia. DSTO has created a software library in C++ called *Janus* that reads files in AERO-ML format. That library has been recently open sourced and released. You can read more about Janus (and request a copy) here: http://www.dsto.defence.gov.au/research/4675/. The file format for JSBSim was an early example of the use of XML in describing aircraft aerodynamics. In turn, JSBSim has incorporated some of the features developed for AERO-ML, moving the JSBSim format closer to this emerging standard.

There were quite a number of very interesting papers presented during the conference, which ran from August 21 through August 24. Among the more interesting papers submitted were:

- A Generic Multibody Parachute Simulation Model
- Adjusting a Helicopter Rotor Blade Element Model to Match Sparse Criteria Data
- Creating Flight Simulator Landing Gear Models Using Multidomain Modeling Tools
- Development and Qualification of a FNPT II Helicopter Simulator - Lessons Learned
- Development of a Pilot-in-the-Loop Aircraft Simulation Laboratory
- Distributed Real Time Simulation Using DIS and XML
- Flexible Uses of Simulation Tools in an Academic Environment

(Continued on page 15)

AIAA Modeling and Simulation Technologies Conference and Exhibit 2007 Call For Papers Now Open!

The call for papers for the **AIAA Modeling and Simulation Technologies Conference** is now open! Technical Papers are now being solicited in the following suggested topic areas:

- Best Student Paper
- Unmanned Aerospace Vehicles and Unmanned Systems
- Aircraft Dynamics
- Aircraft Flying Qualities
- Projectile and Missile Dynamics and Aerodynamics
- Reentry and Aeroassist Vehicle Technology
- Reusable Launch Vehicles
- Unsteady and High Angle-of-Attack Aerodynamics
- Linear and Nonlinear Equations of Motion
- Atmospheric Flight Mechanics Education
- Vehicle Flight Test
- Invited Sessions and Workshops

This list is intended to provide ideas for papers and is not meant to limit papers to mentioned topics. Prospective authors are invited to electronically submit abstracts of 500–1000 words.
The abstract deadline is **1 February 2007.**

AIAA Guidance, Navigation and Control Conference and Exhibit
AIAA Modeling and Simulation Technologies Conference and Exhibit
AIAA Atmospheric Flight Mechanics Conference and Exhibit
20-23 August 2007
Marriott Hilton Head Beach and Golf Resort
Hilton Head, South Carolina

To view the full call for papers, or to submit a paper visit **http://www.aiaa.org/events/mst**

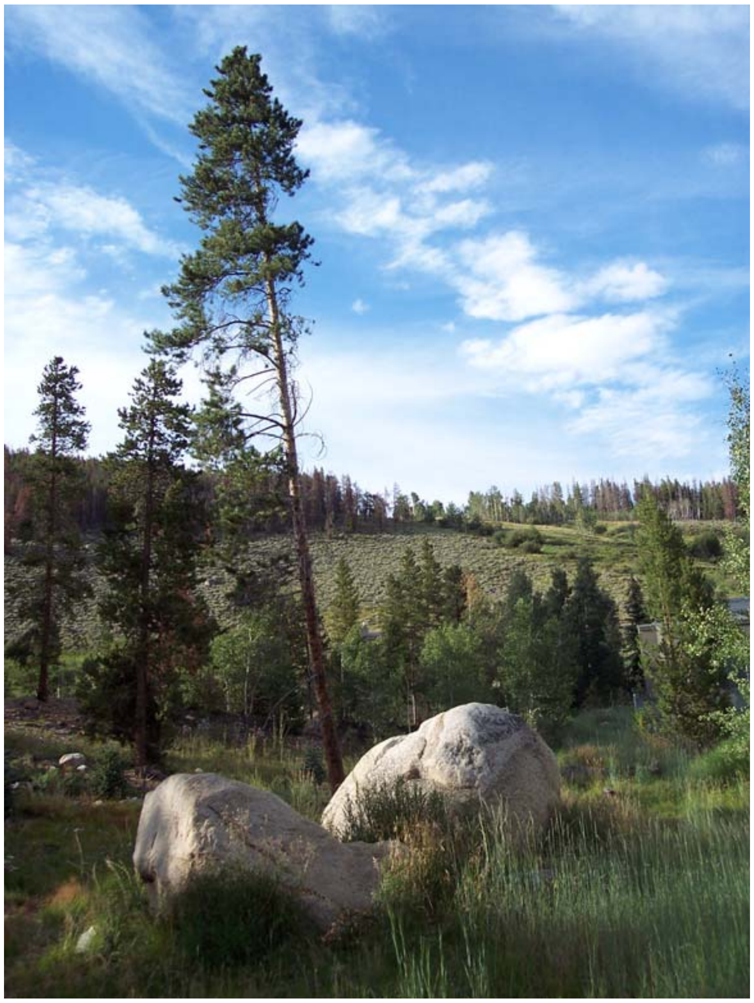
(Continued from page 14)

- Ground Dynamics Model Validation by Use of Landing Flight Test Data
- High Fidelity Landing Gear Modeling for Real-Time Simulation
- Modeling of Apparent Mass Effects for the Real-Time Simulation of a Hybrid Airship
- Modeling, Simulation and Rapid Prototyping of an Unmanned Mini-Helicopter
- Simulation of Airship Dynamics
- The Development of the Tri-Turbofan Airship Model for Autonomous Flight Control Research
- … and many others.

There were also quite a few papers submitted for UAV modeling and simulation, which is of course a hot topic now.

Beginning at 8:00 in the morning on each of the four days, a keynote speech was given in one of the four conference disciplines (GNC, MST, AFM, and AS). Following that, papers covering related topics were presented in a session. It was sometimes hard to choose among individual presentations to attend. For this event, some of the presentation rooms were not in the same building, which required a five or ten minute walk between presentations. Advance and alternate planning is important at conferences.

Above and below: The setting for the 2006 AIAA GNC/MST/AFM/AS conferences was in Keystone, Colorado. At the beginning of the week, we were told that it might rain every day. Instead, it was sunny almost every day. Perfect!



On the last full day of the conference I was invited to attend the Modeling and Simulation Technical Committee (M&S TC) meeting. Within AIAA are a number of Technical Committees. From the AIAA web site:

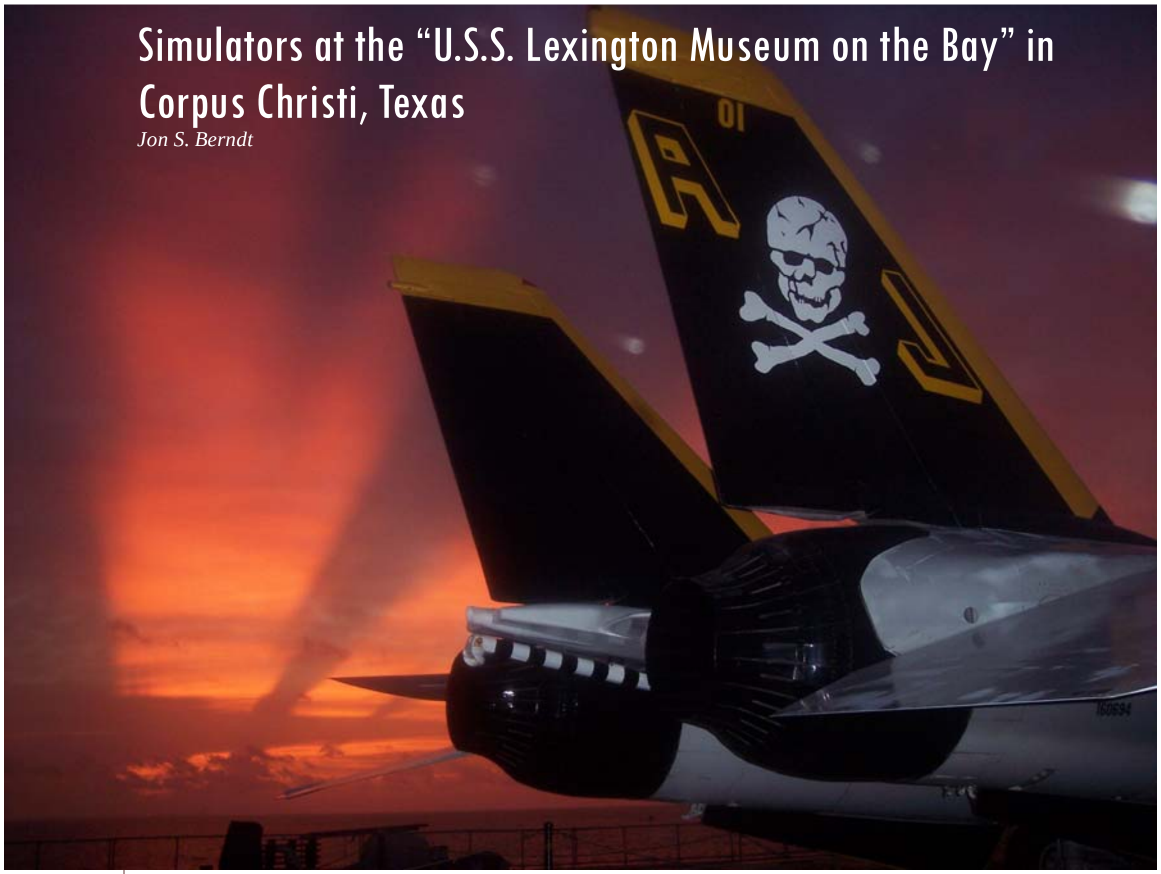
> *AIAA Technical Committees (TCs) bring together experts in their fields and given them the opportunity to exchange knowledge and get to know their colleagues from around the globe. These committees participate in numerous activities: they develop and administer over 20 technical conferences each year; conduct professional development courses, produce books, and work with K-12 students to promote an interest in engineering education. And that's just the start. The TCs also honor technical leadership through Technical and Best Paper awards; serve as journal and book reviewers; formulate technology assessment packages for the non-technical public, and even develop and judge college student design contests.*

I am nominated for membership in the M&S TC. The process and guidelines for membership are being reworked at this time. I plan to attend the next M&S TC meeting in December.

Finally, I heartily recommend to technically oriented and educated readers to consider attending the Modeling and Simulation Conference. The conferences are well attended and it's a "target rich environment" for modeling and simulation information. I met some wonderful people, and the food and location was first-rate. Next year (see the announcement on page 14) the conference will be held at Hilton Head Beach Resort in South Carolina. ▲

Simulators at the “U.S.S. Lexington Museum on the Bay” in Corpus Christi, Texas

Jon S. Berndt



Above: The sun rises over Corpus Christi Bay in a photograph taken by the author from the flight deck of the U.S.S. Lexington Museum. The tail of an F-14 is in the foreground.

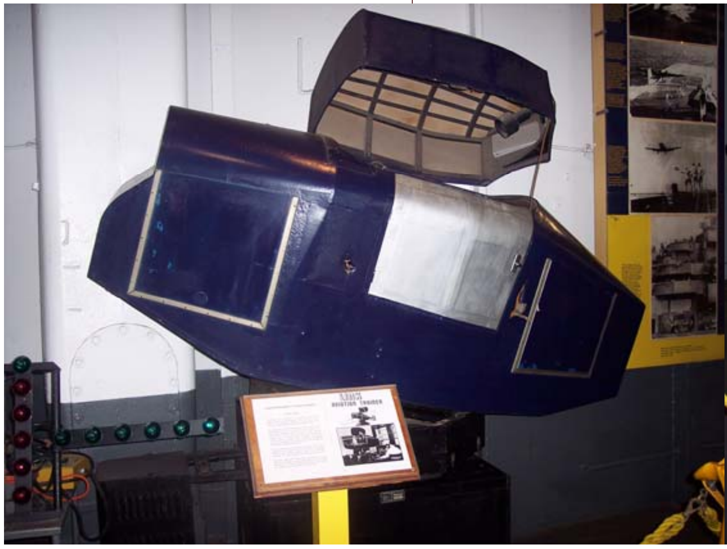
I visited the aircraft carrier U.S.S. Lexington (CV16) Museum recently and stayed onboard overnight as part of a scouting outing with my seven year old son. Visiting the carrier and exploring it with my son was exciting enough (and I wholeheartedly recommend visiting the Lexington museum if you are ever anywhere even close to Corpus Christi, Texas), but it was even more special because my father served on the Lexington exactly fifty years ago, during 1956 and 1957.

To my initial surprise, there were several simulators aboard Lexington. I thought it would make a good article for this season’s newsletter.

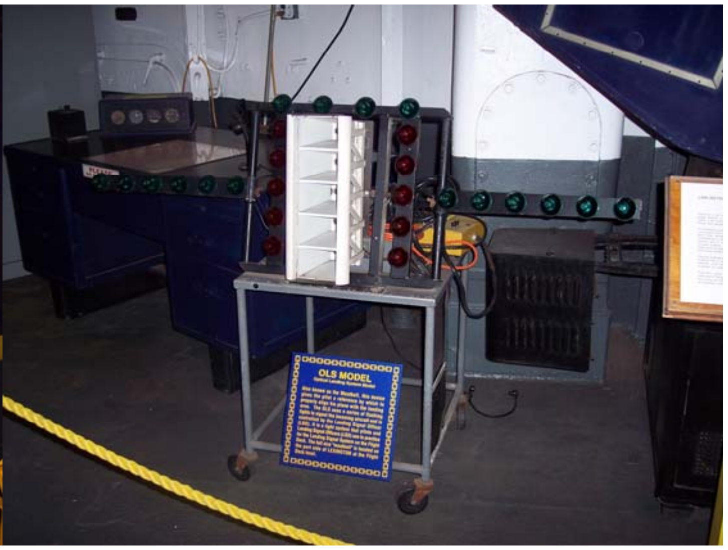
Many users of FlightGear, JSBSim, and other simulators find carrier landings to be a tantalizing challenge, with *night* carrier landings being the pinnacle of challenges.

Onboard the Lexington was an A-7E Corsair II Night Carrier Landing Simulator (NCLS, see photograph, page 15). It is currently being refurbished. For information on the Vought A-7 aircraft, see the outstanding Vought Historical web site at: http://www.vought.com/heritage/. The NCLS were produced by Vought and two were delivered to the Navy in 1971. These were the first simulators to

(Continued on page 17)



The Link Blue Box Pilot Trainer



The “Meatball” training device for LSO’s (Landing Signal Officers)

(Continued from page 16)

combine real-time, out-the-window, computer-generated scenery with a cockpit. The simulator also featured a motion base. A nearby instructor station allowed the instructor to control the scenario. The view presented to the pilot showed the outline of the carrier and the "meatball". These simulators were unveiled at NAS Lemoore on May 5, 1972.

The "meatball" training device (see photograph, page 14) is used by pilots and LSO's (Landing Signal Officers) to familiarize with the device, formally called an OLS (Optical Landing System). The OLS is an arrangement of lights of various colors that incorporates a Fresnel lens. The lights are projected at different angles above the horizon along the glideslope. The lighting pattern appears different to the pilot depending on how the approach is progressing, so he or she can tell where they relative to the ideal glideslope. For more information on the OLS, see the NATOPS LSO Manual at http://navyair.com/LSO_NATOPS_Manual.pdf.

There is an old Link "Blue Box" aboard Lexington. Ed Link built his first pilot trainer in his father's piano and organ factory, completing it in early 1929. During the 1930's, Ed and his brother ran a flying school, selling training lessons after hours in the factory. When the Depression hit, it hit their training business hard. However, when the Army Air Corps started delivering the mail, it became apparent (after a particularly bad bout of weather caused the loss of nearly a dozen pilots) that some kind of training device would be useful. The Army pilots had been trained to fly by watching the ground. They decided to take a closer look at Link's training device. From the Link History web site:

> *On a foggy, misty day in 1934, a group of Army officers awaited Ed's arrival in Newark, New Jersey. Ed was flying in from Binghamton, New York.*
>
> *The officers, convinced that he couldn't make it in such soupy weather, were about to leave. Just as they were about to leave they could hear the sound of an approaching airplane. Within a minute's time an aircraft circled the field and touched down on the runway. It was Ed Link...he had flown in on instruments and demonstrated that effective flight was possible even during adverse weather conditions.*
>
> *The military officials were sold on the promise training to fly by instruments could offer and, shortly thereafter, the Army Air Corps ordered six of his trainers for $3,500 a piece. By the time the order was completed other orders started coming in and Link Aviation Devices, Inc. was formed to meet the increased trainer production demand.*
>
> *The company expanded rapidly, in spite of some facility setbacks in the mid 1930s, and during World War II the ANT-18 Basic Instrument Trainer, known to tens of thousands of fledging pilots as the Blue Box, was standard equipment at every air training school in the United States and Allied nations. In fact, during the war years Link produced over 10,000 Blue Boxes, turning one out every 45 minutes.*

Aboard the Lexington museum is also a motion base simulator that seats 15 or so people, and features a simulated ride in an F-18 on an attack mission. It's convincing enough and my seven year old son loved it.

I found the simulator exhibits (particularly the A-7E simulator) to be interesting. Most of all, though, the experience of exploring the ship, perusing the exhibits, becoming acquainted with the various smells, and walking on the flight deck with my son at sunrise made it a memorable time.

The U.S.S. Lexington Museum can be seen from the air at Google Maps. ▲

The Vought A-7E Carrier Night Landing Trainer cockpit (above, and instructor station (below).

If you've got some time to kill late at night: using maps.google.com, how many aircraft carriers can you find—and *identify*? *It might not be as hard as you'd think it would be. Extra credit: find one "at sea".*

A Boeing 747-45E approaches Vancouver International Airport in British Columbia. This beautiful photograph was taken on July 29, 2006, by Marek Wozniak. Reprinted here with permission of the photographer.

Simulate This! The "Transition"

Aerodynamic trade studies and early design iteration on the Transition were conducted using a vortex-lattice code called AVL written by Prof. Mark Drela at MIT. Airfoil design was also assisted by X-foil (also written by Prof. Drela). These analyses were cross checked with a 1/5th scale model of the Transition that was measured in the MIT Wright Brothers Wind Tunnel -- verifying the predicted stall performance and stability of the Transition design.

Terrafugia, which is derived from the Latin for "escape from the earth," was founded by graduates of the Department of Aeronautics and Astronautics at the Massachusetts Institute of Technology and incorporated in 2006. Currently based in Cambridge, Massachusetts, Terrafugia combines solid aircraft design fundamentals with a focus on creativity and customer service. Terrafugia's mission is the expansion of personal mobility through the practical integration of land and air travel.

People have dreamed of "roadable" aircraft since 1918 when Felix Longobardi was issued the first patent for a vehicle capable of both driving on surface roads and flying through the air. The most well known, and arguably most successful roadable aircraft was developed in the 1950s and 60s by Molt Taylor. There are also many visionaries developing their own concepts for a roadable aircraft. This plurality of concepts shows that there is a perceived need for a vehicle of this type. Unfortunately, the cost/benefit of these vehicles never justified serious financial backing – the *real* need was not sufficiently acute to justify the performance sacrifices of a dual use vehicle. ▲

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