2006-2157: EXPOSING AEROSPACE ENGINEERING STUDENTS TO FLIGHT SIMULATION SOFTWARE, HARDWARE AND SYSTEMS INTEGRATION

Lawrence Boyer, St. Louis University

Assistant Professor of Aerospace and Mechanical Engineering at Parks College of Engineering, Aviation and Technology.

Dane Johnston, St. Louis University

Senior Aerospace Engineering student at Parks College of Engineering, Aviation and Technology.

Wesley Karmazin, St. Louis University

Senior Aerospace Engineering student at Parks College of Engineering, Aviation and Technology.

Exposing Aerospace Engineering Students to Flight Simulation Software, Hardware and Systems Integration

Abstract

Aerospace Engineering students are exposed to software and hardware in the Flight Simulation course at Parks College to familiarize them with an Aero Engineer's view of the world of real-time, pilot-in-the-loop flight simulation, impart some skills that could be useful to them should they go into this industry and reinforce their knowledge of flight dynamics. This course has spawned an interesting student project which is the main focus of this paper – the development of a PC-based mobile flight simulator.

Introduction

With respect to the Flight Simulation course at Parks, this paper reports

1) the use of student written programs which introduce them to the basics of flight simulation equations of motion and numerical integration.

2) the use of Parks' Engineering Flight Simulator (EFS)

3) the upgrading of the Parks' Engineering Flight Simulator Visual Image Generator with Flight Gear™.

4) the construction and systems integration effort which will produce a new, mobile, X-Plane™-based flight simulator done entirely as a student project with department funds. This section may be of interest to other schools interested in developing a low cost simulator as a project and educational tool for their students.

1 Student Written Programs

In the Flight Simulation course at Parks College, students are often required to write a program (C, C++ or Matlab M-file) which introduces them to the basics of flights simulation equations of motion and numerical integration. This program may only involve the longitudinal degrees of freedom (pitch, speed and altitude). In the program the Euler angle pitch rate is integrated to get pitch angle. But students are introduced to the quaternion method used in full flight simulators which avoids the singularity at 90 degrees of pitch. Please see Appendix A for an example of a student written program. In contrast, the flight model in the EFS is "industrial strength".

2 Use of Engineering Flight Simulator

The Parks College Engineering Flight Simulator is used in a variety of ways

a) to give AE students some "flight" experience, albeit simulated flight
b) to provide a platform on which AE students may implement a new aircraft design for flight validation
c) to provide a "Guinea Pig" on which AE students may do some software development

Figure 1. Engineering Flight Simulator with Lab lights on.

The EFS has a standard instrument cluster, VOR, ADF and ILS (no GPS). Students may shoot a visual or instrument approach to a land runway or an aircraft carrier or just “fly”. See Figures 1 and 2.

Figure 2. EFS Instrument Panel and out-the-window visual scene with lab darkened.

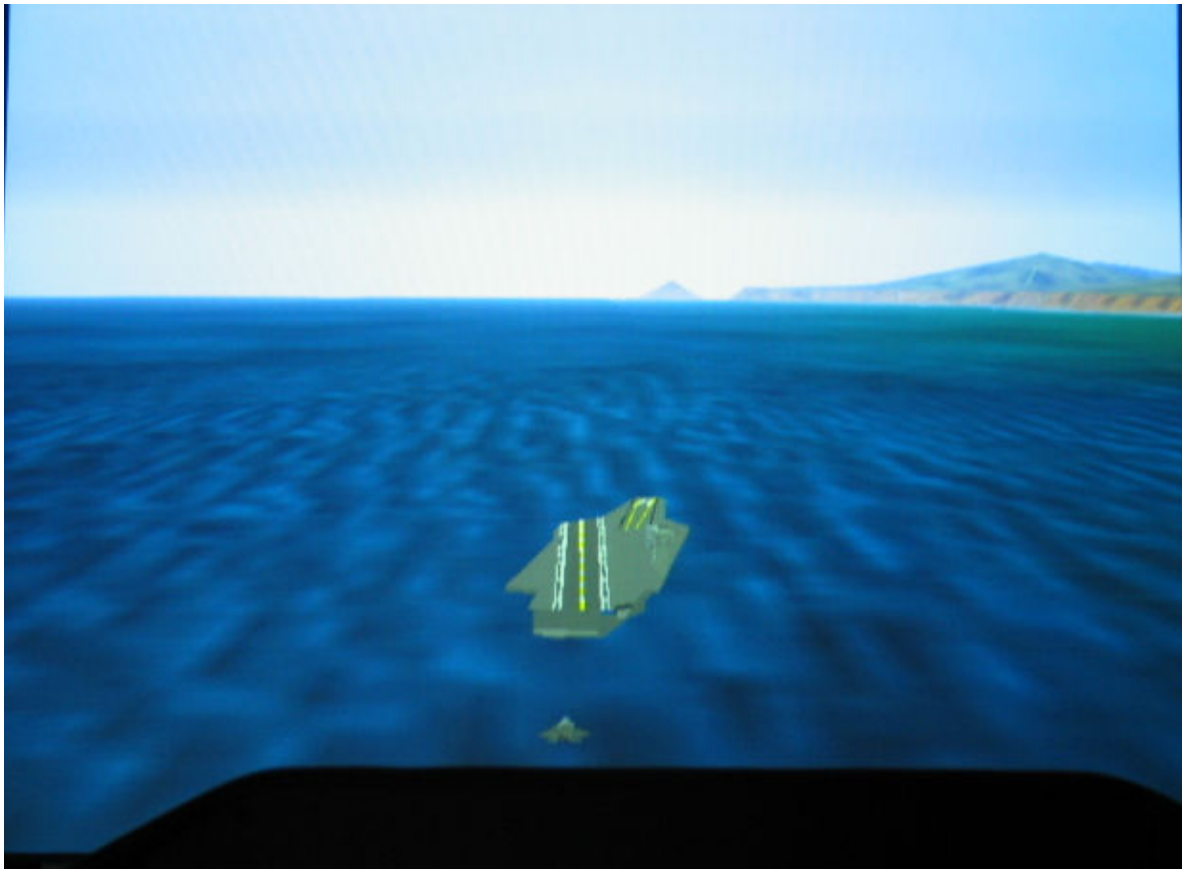

Figure 3. EFS out the window visual scene on approach to aircraft carrier

The EFS has a simulated arresting hook on the aircraft and arresting cable on the aircraft carrier deck. Catching the hook is not easy and students enjoy the challenge.

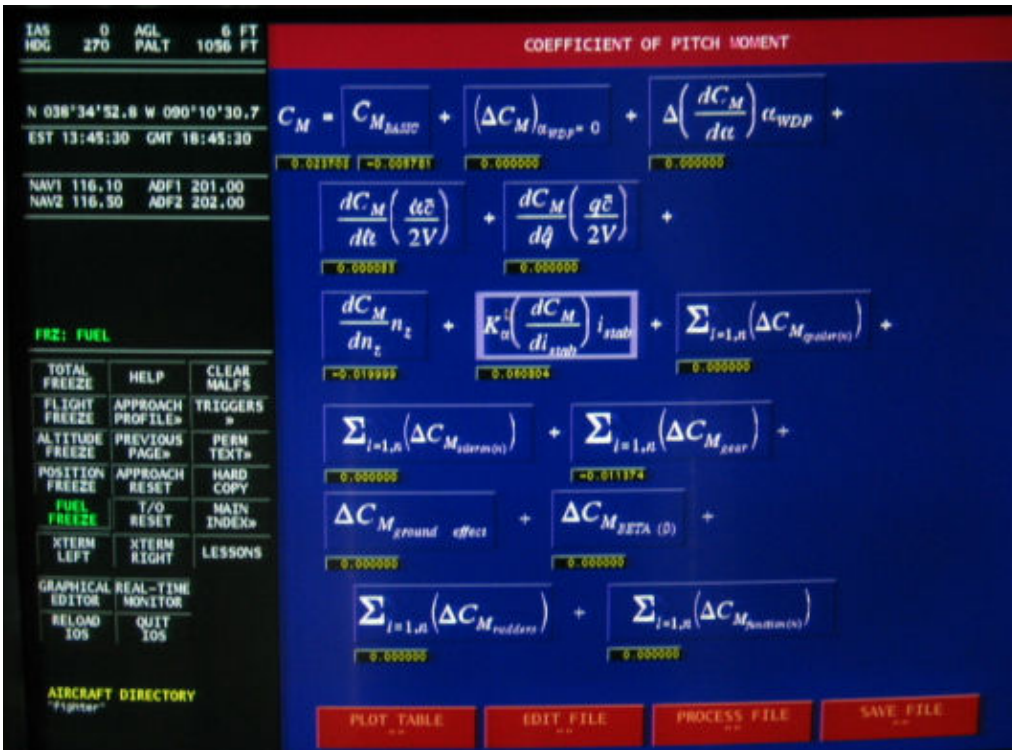

Figure 4. Pitching Moment Aero Model Development page on EFS IOS.

Implementing a new aircraft design in the EFS is labor and data intensive. Aero coefficient and stability derivative data for the aircraft are needed to obtain a complete aerodynamic model. Figure 4 shows a sample page used in the data entry process for a new or updated aero model. This is often too time consuming to be useful in a single semester and is one motivation for building the X-Plane™-based Mobile Flight Simulator presented later in this paper as a student project.

Figure 5. EFS Flight Test Data Recording capability.

Time History data can be recorded, plotted and printed on the EFS which is useful in teaching flight testing techniques and practices.

Figure 6. Moving Model as seen from pilot seat.

There have been many opportunities for students to add features to the EFS since all the source code on the host computer where the flight model and IOS software resides is accessible. This

facilitates software updates at the source code level (Fortran and C). A sample project resulting from the effort of a team of students in the past was the creation of the Moving Model IOS page seen in Figure 7 and the associated model residing in the host computer which results in the display of an aircraft as seen in Figure 6. The Moving Model is an aircraft which appears only on the visual scene to provide the ownship pilot the opportunity to fly formation with an aircraft on a stable flight path. The flight path followed is a constant altitude race track pattern of user selectable leg length. Selecting the REPOSITION button, initializes the moving model 1,000 feet ahead of and at the same altitude, heading and airspeed as the ownship.

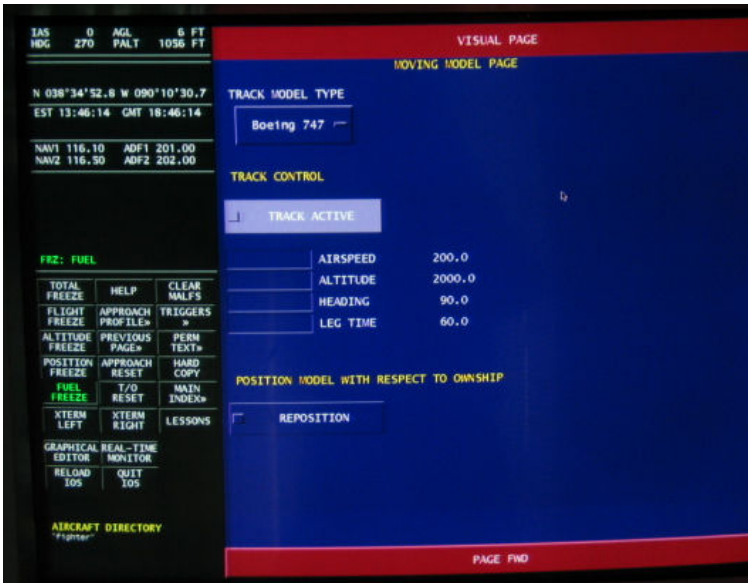

Figure 7. Moving Model IOS page.

3 Upgrading of EFS Visual IG with PC Running Flight Gear™.

The current visual image generator (IG) on the EFS only provides a single scene some 10x10 miles and the quality of the scene is marginal. It is hoped that replacing the IG with a PC running Flight Gear™ connected via local area network to the host will facilitate a low-cost out-the-window scene. This project began in January, 2006. As of early March, insufficient progress has been made on this project to report any details.

4 Mobile Flight Simulator – A Special Project by 2 Aerospace Engineering Students

4.1 Software

4.1.1 Software Selection

X-Plane™ was chosen for this project after a Boeing-funded study of several, publicly available flight simulator applications. It offered the fullest package by way of customizable scenery and offered the most flexibility in terms of custom designs, and had what was considered to be the most accurate flight model. X-Plane™ also offers the unique feature of fully adaptable

environmental features, such as density, pressure, temperature, and other atmospheric properties including all weather effects. The fact that X-Plane™ is an actively supported application, that still undergoes periodic upgrades and has support available also played heavily into our choice. The software runs for approximately $50 and right out of the box, X-Plane™ offered the multiple computer support that other titles relied on third party or community application development for which was crucial for the overall integration of the flight simulation experience.

Figure 8. Sample X-Plane™ Visual Image

4.1.2 Blade Element Momentum Theory

X-Plane™ is based around the Blade Element Momentum Theory (BEMT) developed in the 19th Century to determine the behavior of propellers. It provides a means to determine forces and moments by assuming the blade as composed of a number of aerodynamically independent cross-sections, whose characteristics are the same as a blade at a proper angle of attack. Therefore the operation of a cross-section is indirectly related to that of a two-dimensional airfoil.

X-Plane™ uses this theory over the entirety of an aircraft. By sectionalizing the fuselage, wings, tail, control surfaces, and other independent surfaces, individual forces are then calculated in real time into a resultant force that is worked into a function of velocity to give an accurate real time flight model.
According to the creators, the software calculates its flight model using BEMT in the following order:

a) Element Break-Down

X-Plane™ breaks the wing(s), horizontal stabilizer, vertical stabilizer(s), and propeller(s) (if equipped) down into a finite number of elements.

b) Velocity Determination

The aircraft linear and angular velocities, along with the longitudinal, lateral, and vertical arms of each element are considered to find the velocity vector of each element. Downwash, prop wash, and induced angle of attack from lift-augmentation devices are all considered when

finding the velocity vector of each element. Done twice per cycle.

c) Coefficient Determination

X-Plane™ applies finite wing lift-slope reduction, finite-wing CLmax reduction, finite-wing induced drag, and finite-wing moment reduction appropriate to the aspect ratio, taper ratio, and sweep of the wing, horizontal stabilizer, vertical stabilizer, or propeller blade in question. Compressible flow effects are considered using Prandtl-Glauert.

d) Force Build-Up

Using the coefficients just determined in step 3, areas determined during step 1, and dynamic pressures (determined separately for each element based on aircraft speed, altitude, temperature, prop wash and wing sweep), the forces are found and summed for the entire aircraft. Forces are then divided by the aircraft mass for linear accelerations, and moments of inertia for angular accelerations.

This process is repeated 15 times per second and is advantageous over previous flight simulation styles where as the flight model must fully recalculated by the designer to view minor changes, where as the X-Plane™ flight model can instantly show changes in design an their effects.

4.1.3 Design Software

X-Plane™ uses its own built in "Plane Maker" to create its flight model from a 3D mold. The user creates the fuselage and wings through multiple cross sections to create the structure of the plane. The power plants, control surfaces, control geometries, artificial stability, weight, and CG values are entered numerically. The design software gives minimal structural inputs by way of giving general fracture values (in acceleration units, g's) for overall surfaces, i.e. wings, landing gear. This allows the student to get an idea of how an aircraft will fly with out in depth structural considerations.

4.1.4 Design Data

The X-Plane™ software outputs a large amount of useful data for the student to use in the design process. All major variables can be tabulated, such as, coefficients of lift and drag, wing loading, G forces, speed, wind speed, center of gravity, aerodynamic center, all aerodynamic and engine forces, ground effect, all wash conditions, stability derivatives, and all control settings. These can be presented in a table format based upon cycles per second (which can be adjusted). This can be used for many entry level design choices to quickly change aerodynamic variables and to see their effects upon the overall design.

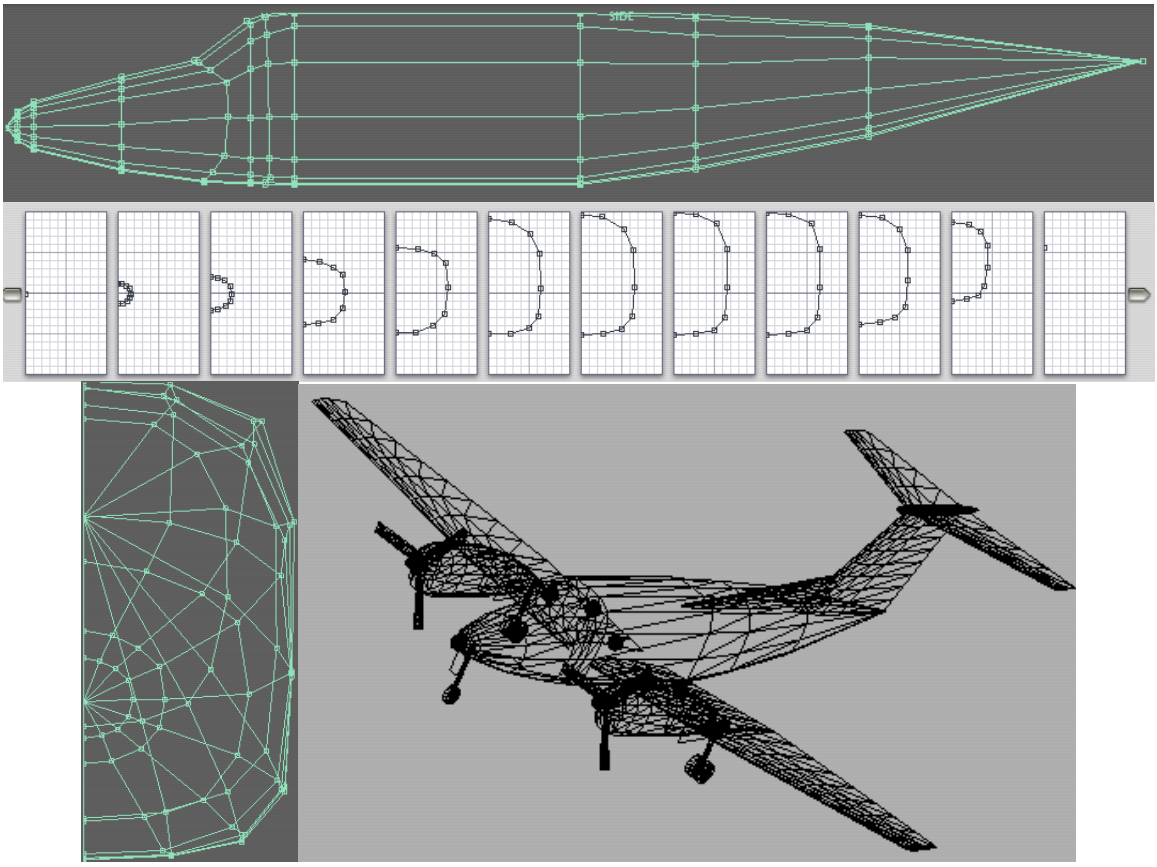

Figure 9. X-Plane™ Plane Maker

4.1.5 Ease of Use

X-Plane™ uses a simple interface to present the data to be modified and setting up the control systems for the design. This allows a student to spend more time modifying their aircraft/design instead of learning a difficult interface as most are presented in more dated or "professional" styled flight simulators. The ability to quickly change variables without spending a large amount of time moving through the interface helps reduce frustrations and increase productivity. This helps greatly in a learning environment when generally it is more important to grapple with the concepts vs. the software.

4.2 Hardware

4.2.1 Hardware Goals

Hardware was the original motivating force behind this project. Saint Louis University is fortunate to have an Opinicus Corporation (Clearwater, FL) engineering simulator on campus, which has served design students well for the last ten or so years. Unfortunately, it is based on custom hardware running a Unix operating system and a proprietary simulation program. It is becoming harder and harder to maintain this system, as the system is no longer officially supported. After the Opinicus simulator was down for almost a semester, it became clear that

another option was needed and that with the rapid advances in computer technology at the consumer level, it could be done much cheaper, with a very robust maintenance/upgrade path.

Both of the students involved are aerospace engineering majors with very strong interests in computer technology. It was abundantly clear that the capabilities of the Opinicus could be easily exceeded on the hardware side by Commercial, Off-The-Shelf (COTS) equipment. Not only does this drastically reduce the cost, but maintainability and upgradeability are dramatically improved as well.

4.2.2 Computer Equipment

Fortunately**,** for quite some time now, the commercial computer market has generally been driven by games. Gamers demand the most from their hardware, 3D visual processing, plenty of processing power for a game's "AI," standards in controller connections, etc, while demanding lower costs and upgradeability. This all plays nicely into the simulator project as, coupled with the right software, a "gaming" computer is well suited to meet our goals. This is the high-end of the consumer market, but it is still much more affordable than equipment from the professional side. Incidentally, the line between consumer and professional in the current computer market is very blurry, with many going as far as to label high-end gaming machines "pro-sumer" equipment. In the factory-built segment, this refers to products such as Dell's XPS line, or systems from "boutique builders" like Alienware, Falcon Northwest, or VooDooPC. All of these share a premium price however; as you have to pay extra for the custom options and lower order volume. Drawing on these students' experience with custom computer hardware, however, the machines for the project were ordered a piece at a time from internet vendors, and assembled on site, allowing for the greatest customization without the custom-build cost overhead. This method allows for the greatest cost/performance ratio, but does not offer the same warranty and service generally found with the premium factory systems. Instead each individual part is warranted through its individual manufacturer. For that small inconvenience, it is estimated that the hardware cost of the simulator is roughly one-third that of a similarly equipped Dell machine. The hardware specifications are listed below:

AMD Athlon64 3200+ processors
1GB of Kingston Ram
nVidia based 7800GTX PCI-E video card for the visualization
nVidia based 6600GT PCI-E video cards for the support machines
nVidia based socket 939 (nForce 4) chipset motherboards
80GB Harddrives

This configuration allows all of the major system components (CPU, Video Card, Memory, HDD) to be easily swapped with new off the shelf parts if and when they are needed. The simulator system is comprised of 1 visualization machine, and 2 support machines, one to run the cockpit display and serve as master/data recorder, and an instructor's station that allows the introduction of flight environment variations and aircraft system malfunctions (weather, equipment failure, etc) to be changed on the fly.

4.2.3 Control Equipment

The same thought process was given to the control equipment. The Opinicus simulator relies on a custom control interface that suffers the same shortcoming as the rest of the hardware: it is a custom interface and not easily maintainable. It does however provide 3 axis control loading, something not included in the new simulator at this time. While the gaming market is booming, it has in recent years turned away from the lofty years of the flight simulators in the late 90's. Flight Simulator enthusiasts are generally only provided with a handful of titles every year, and the flight simulator market segment is only a fraction of what it once was. This has most affected the controller market, as the demand for high-fidelity flight controls has diminished quite rapidly. Fortunately, controls suitable to meet the goals of the project were sourced, and offer the ubiquitous USB interface, meaning that upgrades are simple, quick, and hassle-free. The controls chosen for the project are listed below:

CH Products Flight Yoke, 6-Throttle Quadrant, and Pro Pedals to replicate a general aviation setup, single or double engine. (Up to 6 engines are easily configured but this setup lacks a mixture/prop control for each engine)

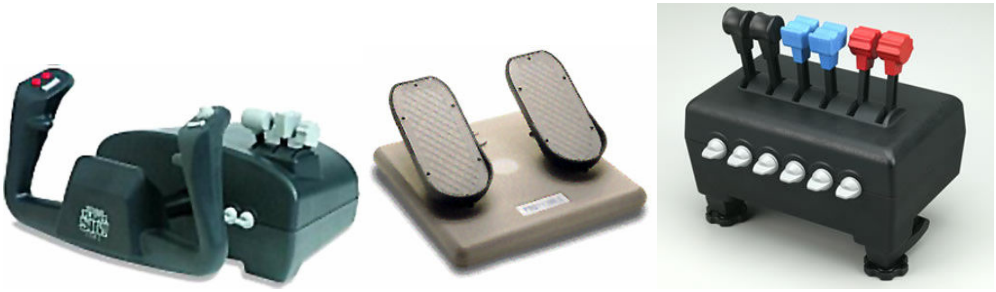

Figure 10. CH products control equipment

Thrustmaster Cougar. This is an all metal stick and throttle setup based on the Block 52 F-16 Hands On Throttle and Stick system. It allows for systems integration development (programming the stick and throttle) and a military-feel single throttle for multiple engines. It is used in conjunction with the rudder pedal system from CH.

Figure 11. Thrustmaster Cougars

Between these two sets of controls, and the interactive cockpit display, students can configure a cockpit for almost any type of aircraft imaginable.

4.2.4 Mobility and Ergonomics

The simulator was designed from the start to be mobile, allowing flexibility in its use and implementation. It can be moved fairly quickly from a design lab environment, to a classroom, allowing it to be used both as a design tool and an instructional aid for engineers who have little to no flight experience. The computer systems are housed in a cockpit mockup that allows ease of transportation, and a more authentic feel. This allows the device to be moved into any classroom equipped with a projector, and be instantly fully operational. Consideration was given to ergonomics, with several students volunteering to be measured in various seat/pedal/control configurations. Using basic geometry, with the classroom display screen and the configured view angle of the simulator software, we came to dimensions that accurately represent the view over an instrument panel from a general aviation aircraft. However, anyone that has flown a simulator before has experienced, having the pilot's view limited to just forward, or having to use hat switches to "drive" your field of view around is a major shortcoming. This problem was easily overcome in the design with another piece of professional equipment that has been recently commercialized. The TrackIR system from NaturalPoint is a very affordable head tracking device that allows for a full six degrees of freedom (left/right, up/down, cant, tilt, and zoom). The device exaggerates the pilot's head movement, so looking from the center to the right side of the screen is analogous to looking over the right shoulder, so one does not have to remove the eyes from the screen.

4.3 Systems Integration

This turned out to be one of the easier elements of the design process. The full integration is done within the framework touched upon in section 2.3, that houses the computer equipment and flight controls, simulates a cockpit environment, and makes the entire apparatus mobile. The COTS nature of the hardware makes the integration fairly simple. The controls are plug and play with the simulator software and the operating system, the buttons on the control panel are run through a keyboard emulator that makes them plug and play, and the machines themselves are standard pro-sumer level desktop computers. All of the accessories connect through the common USB standard, so wiring of the forward section to the computer section is limited to several USB extensions and one DVI monitor connection. The physical controls are almost limitlessly programmable and configurable within the simulator, or on an even more advanced level within their own software, all easily done by any student remotely familiar with the basic Windows user interface. Combined with the customizability of the interactive control panel within X-Plane™, the design student has the option to create any cockpit configuration they desire. All of this is done while retaining access to all of the components, and still allowing for easy replacement/upgrade. Overall the integrated nature of the finished simulator will allow for a much higher fidelity design experience that is much more accessible to the design student than the Opinicus simulator.

4.4 Uses

The new, X-Plane™ based, Mobile Flight Simulator will be utilized in ways similar to the existing Opinicus Engineering Flight Simulator. The following table shows a summary of the

courses and tasks. Enrolled students will be shown the task, given the opportunity to act as "pilot" to accomplish the task and analyze the resulting data (in some cases these data are time history plots).

Courses	Tasks
Intro to Aeronautics	Aircraft Familiarization - Flight Instruments - Flight Controls - Taxiing - Takeoff - Climb - Turn - Navigation - Approach - Landing
Airplane Performance	Thrust or Power Required and Available vs. Speed Climb Performance
Airplane Stability and Control	Longitudinal Dynamics (Short Period, Phugoid) Lateral Directional Dynamics (Dutch Roll, Spiral Stability) Crosswind Landing
Capstone Design	Implementing a new aircraft design in the simulator
Flight Simulation (elective)	Simulator Architecture Software Engineering for AEs

5 Conclusions

The Flight Simulation course along with the AE department's Engineering Flight Simulator, has spawned the development of a PC-based mobile flight simulator utilizing X-Plane™. This project is in its final stages as of early March 2006 and promises to be a very popular and useful educational tool for Aerospace Engineering students.

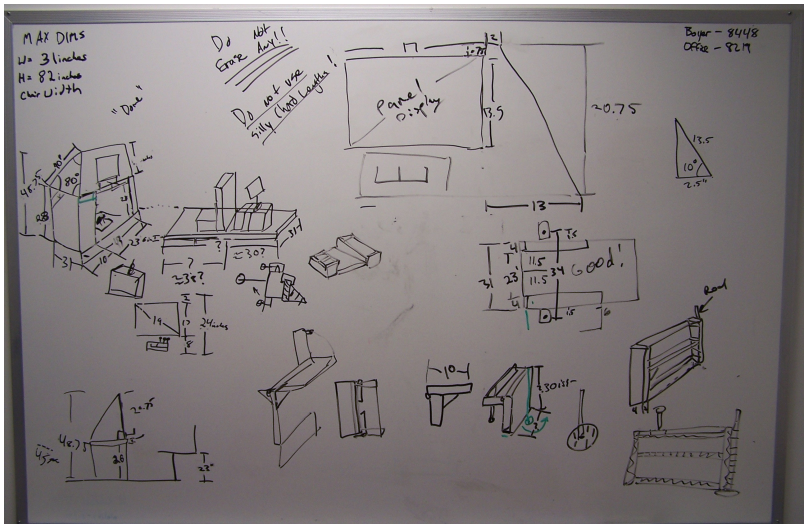

Figure 12. Mobile Flight Simulator working sketches on white board.

Figure 13. Mobile Flight Simulator Early Concept Rendering without façade.

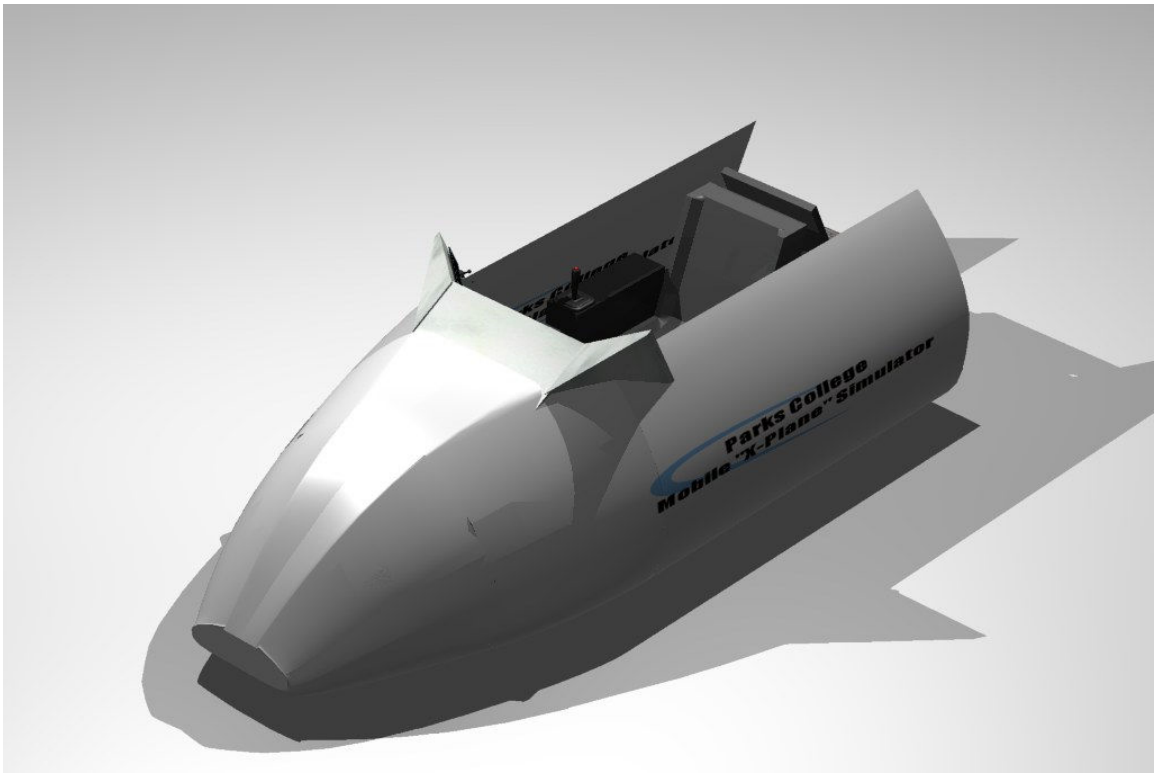

Figure 14. Mobile Flight Simulator Rendering with all the finishing touches.

Appendix – Example Student Written C Program

```
// Longitudinal_stability.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include "stdio.h"
#include "math.h"


// Declaration of variables
double AR;
double Mach = 0.158;
double RHO = 0.002378;       //Slugs/ft^3
double h_initial, h;
double delta_elevator, alpha, alpha_initial;
double a = 1116.45; //(ft/min);
double k;
double gust;

double CL, CD, CM, CL_alpha, CD_alpha, CM_alpha, CMq, CM_alpha_dot;
double CL0, CM0, CD0, CM0_wing, CM0_tail, CM0_fuselage ;
double CL_delta_elevator, CM_delta_elevator;
//double CDu, CLu;

// Forces and Moments:

double L, Lx, Lz, M, D, Dx, Dz, thrust;

double W, m, g;  // Weight of aircraft
double Ix, Iy, Iz, Ixz;  // Moments of Inertia

double Q, S, b, c;  // Dynamic rpessure, Planform Area, span and mean Chord
double X_cg;  // Location of Center of Gravity

//velocities, angular rates and angles
double Vtrue_initial, Vtrue,ub, wb, w_earth, q;
double theta, theta_dot, alpha_dot, theta_degrees;

//Accelerations and angular accelerations
double u0_dot, ub_dot,  wb_dot, w_dot, q_dot;

// Time Constants

double elapsed_time = 0.0;
double time_end = 420.0;
```

```
double delta_t;
double k_wb, k_elevator, k_theta, k_thrust;

double tol_w_earth, tol_ub_dot, tol_q_dot, tol_wb_dot, tol_trim_count;
int trim_count,not_trimmed;
int flight_freeze;

// Flight parameters
//******************

// Function for Equations of Motion

void core_eom(void);

main()

{
        // Initializing constants

        h_initial = 1000;       //ft

        // Aircraft Data

        S = 184.0;      // ft^2
        b = 33.4;       // ft
        c = 5.7;        // ft
        AR = b*b/c;     // Aspect ratio
        k = 0.048;      // Correction factor in Drag Coefficient equation

        // Data obtained from sample problem 2.2
        CM0_wing = -0.099;
        CM0_tail = 0.194;
        CM0_fuselage = -0.037;


        X_cg = 0.295 * c;       //  CG @ 29.5% of MAC
        CL = 0.41;                      // Lift Coeficient
        CD = 0.05;                      // Drag coeficient
        CL_alpha = 4.44;        // dCL/dAlpha
        CM_alpha = -0.683;  // dCM/dAlpha
        CMq = -9.96;

        CL0 = 0.26;
        CM0 = CM0_wing + CM0_tail + CM0_fuselage;     // Total CM0 for wing, tail and
fuselage
        CD0 = 0.028;
```

```
CL_delta_elevator = 0.355;  // dCL/ddelta_elev
CM_delta_elevator = -0.923;  // dCM/ddelta_elev

tol_trim_count = 120;
delta_t = 1.0/60.0;

// WEIGHTS AND MOMENTS OF INERTIA

W = 2750.0;  // lbs
g = 32.2;          //        slugs/sec^2
m = W/g;          //        Mass in Slugs


Ix = 1048.0;  //Slug-ft^2
Iy = 3000.0;  //Slug-ft^2
Iz = 3530.0;  //Slug-ft^2

k_wb = 0.025;
k_elevator = 0.004;
k_theta =0.001;
k_thrust = -8;

Ixz = 0.0;

not_trimmed = 1;


Vtrue_initial = Mach * a;       // ft/sec
Vtrue = Vtrue_initial;

alpha_initial = 2/57.2957;       //2 degrees, passed to radians
theta = alpha_initial;

// horizontal and vertical components of initial velocity in body components:

ub = Vtrue * cos(alpha_initial);
wb = Vtrue * sin(alpha_initial);


FILE *output_file_pointer;


// Open file
output_file_pointer=fopen("output_long", "wt");
```

```
rewind(output_file_pointer);

fprintf(output_file_pointer,
                "longitudinal parameters time hystory response *****\n"
                "time(sec)    u(ft/sec)     w(ft/sec)   theta(rad) \n");
                //"trim_count       w_earth       ub_dot       q_dot       wb_dot \n");

flight_freeze = 1;

while (not_trimmed)
{

        // force pitch rate to zero while trimming
        q = 0.0;

        //force altitude to stay at initial value
        h = h_initial;

        //Control vertical speed with pitch angle
        theta = theta + k_theta * w_earth;

        //control longitudinal acceleration with thrust
        thrust = thrust + k_thrust * ub_dot;

        //control pitch acceleration with elevator
        delta_elevator = delta_elevator + k_elevator *q_dot;

        // control vertical acceleration with angle of attack (indirectly)
        wb = wb + k_wb * wb_dot;

        //Vtrue_initial = Vtrue_initial * cos(theta);

        //now make sure true airspeed vector is correct magnitude
        ub = sqrt(Vtrue_initial*Vtrue_initial - wb*wb);

        // assigning values to the tolerances:
        tol_w_earth     = 0.005;
        tol_ub_dot =0.001;
        tol_q_dot= 0.003;
        tol_wb_dot = 0.002;

        //*****************************************************************

        if (fabs(w_earth) < tol_w_earth &&
         fabs(ub_dot) < tol_ub_dot &&
         fabs(q_dot) < tol_q_dot &&
```

```
                fabs(wb_dot) < tol_wb_dot)

                trim_count = trim_count + 1;
               else
                trim_count = 0;

               if (trim_count > tol_trim_count)
                not_trimmed = 0;
               else
                not_trimmed = 1;

                core_eom();

        }               // End of trim while loop.

               // RUN

               flight_freeze = 0;

               while (elapsed_time <= time_end)
               {
                       if (elapsed_time >= 30 &&
                               elapsed_time < 30.1)
                               gust = 15;      // Gust in ft/sec
                       else
                               gust = 0;

                       wb = wb + gust;

                       core_eom();

                       theta_degrees = theta*57.3;

                               fprintf(output_file_pointer, "%8.4f %8.4f %8.4f %8.4f \n",
                                       elapsed_time, Vtrue, theta_degrees, h);
                                //elapsed_time, w_earth, ub_dot, q_dot, wb_dot);
                                       elapsed_time = elapsed_time + delta_t;


               }

        return 0;
}

void core_eom(void)
{
```

```
Q = 0.5*RHO*(Vtrue*Vtrue);

        // equations of motion

        //Set q to 0:

        //q = 0.0;

        // Determination of AOA

        alpha = tan (wb/ub);

        CL = CL0 + CL_alpha*alpha + CL_delta_elevator * delta_elevator;
        CD = CD0 + k*CL*CL;
        CM = CM0 + CM_alpha*alpha + CM_delta_elevator*delta_elevator +
CMq*(q*c/(2*Vtrue)); //+ CM_alpha_dot*(alpha_dot *c/(2*Vtrue));

        L = CL*(Q*S);
        D = CD*(Q*S);
        M = CM*(Q*S*c);

        // rate of pitching change

        q_dot = M/Iy;

        // Expressing Lift and drag in x and z body components:

        // Lift
        Lx = L * sin(alpha);
        Lz = -L * cos(alpha);

        //Drag
        Dx = -D * cos(alpha);
        Dz = -D * sin(alpha);

        //Accelerations:

        ub_dot = (Dx + thrust + Lx + W*(-sin(theta)))/m;
        wb_dot = (Dz + Lz + W*(cos(theta)))/m;


        // Integrating for the longitudinal velocity
        ub_dot = ub_dot - q*wb;

        // Integratin for the vertical velocity
        wb_dot = wb_dot + q*ub;                 //az + q*ub, or acceleration in the z direction
```

```
        // Vertical velocity with respect to earth:

        w_earth = ub*(-sin(theta)) + wb*cos(theta);

        //Keep altitude
        h = h + w_earth*delta_t;

        //Pitch rate
        q = q + q_dot*delta_t;//until it gets close enough to zero

        theta_dot = q;  //Same as above

        if (!flight_freeze)
        {

                //if (elapsed_time == 60)
                //      gust = 150;     // Gust in ft/sec
                //else
                //      gust = 0;

                ub = ub + ub_dot*delta_t;
                wb = wb + wb_dot*delta_t;
                Vtrue = sqrt(ub*ub +wb*wb);
        }

        theta = theta +theta_dot*delta_t;
}
```

END OF EXAMPLE STUDENT WRITTEN PROGRAM **********