JOURNAL OF AEROSPACE COMPUTING, INFORMATION, AND COMMUNICATION
Vol. 2, July 2005

Development of a Low-Cost, Multi-Disciplinary Rotorcraft Simulation Facility

Joseph F. Horn,* Derek O. Bridges,† Leonard V. Lopes,‡ and Kenneth S. Brentner§
Department of Aerospace Engineering, The Pennsylvania State University, University Park, PA 16802

A low-cost rotorcraft simulation facility was developed for use in university-based multi-disciplinary research programs. The objective was to develop a flexible and effective research facility with a low initial cost and minimal recurring costs. The simulation facility was constructed entirely from commercially available commodity hardware components. Several computers are linked together via a local area network to form a "graphical cluster," which allows for interaction between multiple computing nodes and multiple displays. The facility uses the open source FlightGear flight simulation code and the U.S. Army/NASA GENHEL flight dynamics model. In addition, a graphical user interface and a suite of data processing and analysis tools were developed using MATLAB. The system is being used for advanced research programs in the areas of flight control design, advanced rotorcraft flight dynamics modeling, and near real-time acoustics simulation. The use of open source software provides needed flexibility and cost-effectiveness, although incomplete or nonexistent documentation presented challenges. Overall, the simulation facility fulfills the needs of the university research environment in the areas of control design and rotorcraft acoustics. Some weaknesses related to control feel were noted, but were found to be acceptable relative to the low-cost nature of the facility.

I. Introduction

The performance, availability, and affordability of computers, display systems, and networking tools are constantly improving. Furthermore, there is an ongoing movement devoted to the development of inexpensive, open source software. Both of these trends have increased the feasibility of developing real-time rotorcraft simulation facilities within a university research environment. Real-time simulation can be a valuable tool within academia. Some universities have even developed high-fidelity simulation facilities (such as the HELIFLIGHT simulator at the University of Liverpool[1] or SIMONA at the Delft University of Technology[2]) capable of performing true handling qualities analyses. A high-end handling qualities simulator often features a motion base, wide field-of-view projection systems, realistic control inceptors, and proprietary software; however, the development of this type of facility is cost-prohibitive for many institutions. The use of a low-cost engineering simulation facility can still be quite valuable for both education and graduate research. Even with limited fidelity, a real-time simulation is quite useful for the development of flight dynamics models and flight control systems, because it provides timely visualization of aircraft response and a reasonable assessment of the handling characteristics. Real-time simulation is also helpful for education, since students can more readily grasp concepts related to flight dynamics and feedback control by

Received 04 October 2004; revision received 7 April 2005; accepted for publication 10 May 2005.

* Senior Member, Assistant Professor, joehorn@psu.edu
† Student Member, Graduate Research Assistant, dbridges@psu.edu
‡ Student Member, Graduate Research Assistant, lvlopes@psu.edu
§ Senior Member, Associate Professor, ksbrentner@psu.edu

directly observing these concepts in the simulator. For example, students can observe a phugoid oscillation or see how feedback control can stabilize an inherently unstable aircraft. In addition to the traditional disciplines associated with flight simulation (e.g. flight dynamics, control design, handling qualities), it is envisioned that a real-time simulation facility might also be used for research related to other aerospace disciplines such as aerodynamics and acoustics. This paper presents the development of the Multi-Disciplinary Rotorcraft Simulation Facility (MDRSF) at Penn State University.

Several examples of low-cost, real-time engineering simulation tools are already present throughout the rotorcraft community. NASA Ames Research Center has developed the Real-time Interactive Prototype Technology Integration/Development Environment (RIPTIDE).[3] This software package provides an integrated environment for development of mathematical models, control system designs, and cockpit-display concepts. RIPTIDE is currently being used at the Georgia Institute of Technology to study Carefree Maneuvering technology.[4] In addition, Boeing Helicopters has developed its own distributed simulation architecture using low-cost PC/Linux clusters.[5]

At Penn State University, the MDRSF was designed with five main applications in mind:

1. Education in rotorcraft stability and control.
2. Flight controls and cueing system research.
3. Advanced flight dynamics modeling.
4. Rotorcraft acoustics research.
5. Real-time visualization of complex flow fields.

Since the facility was designed to have maximum flexibility at minimal cost, it is composed entirely from commodity hardware components and open source software. It incorporates a cluster of PC systems that can communicate through a variety of network protocols. The cluster is highly expandable and can incorporate a number of different types of nodes, each of which can execute computational or graphical operations or a combination of the two. The graphical nodes of the simulator use the SUSE Linux operating system running FlightGear flight simulator software. FlightGear is an open source, multi-platform, cooperative flight simulator development project.[6] The source code for the entire project is publicly available at flightgear.org and is licensed under the GNU General Public License. The FlightGear code has already been utilized in other university research projects.[7,8] For the MDRSF, FlightGear acts primarily as a graphics engine while the U.S. Army/NASA Ames GENHEL code[9] is used for the flight dynamics mathematical model. The GENHEL code runs on a separate computer by two-way transmission of data via network sockets. Several upgrades to the GENHEL model have been implemented to allow increased functionality and improved user interface as described below.

II. Implementation

The main objective in the design of the MDRSF was to develop a highly flexible real-time simulation tool for a relatively low cost. The initial budget was limited to \$45,000 and the system design was required to minimize large recurring costs (i.e., annual software licenses and maintenance fees). In order to meet these budgetary constraints, the system does not use any custom hardware components and all software is either free or covered under existing departmental site licenses. It is composed entirely of commercial off the shelf (COTS) hardware components, including the computers, plasma displays, LCD monitors, sound system, and control inceptors. The system also uses open source software, including the SUSE Linux operating system, GNU compilers, and the FlightGear flight simulation code, all of which were obtained at no cost. In order to model the rotorcraft flight dynamics, a version of the GENHEL code (provided to Penn State by NASA Ames) is used. All other software, including the Microsoft Windows 2000 operating system and MATLAB, is available at no cost under departmental site licenses. Multiple computers are connected in a ‘graphical cluster’ using network socket communications to provide multiple out-the-window displays (Fig. 1), cockpit instrument displays (Fig. 2), and an operator station (Fig. 3). Due to the cost constraints, it was clear that a motion base and high-fidelity control loaders would be prohibitively expensive. For example, a current published price for a low-cost 2 DOF motion platform is listed as \$18,000.[10] While motion cueing is important for certain handling qualities analyses,[11] it was not considered essential for the applications discussed above, and such a system would use at least 40% of the total budget.

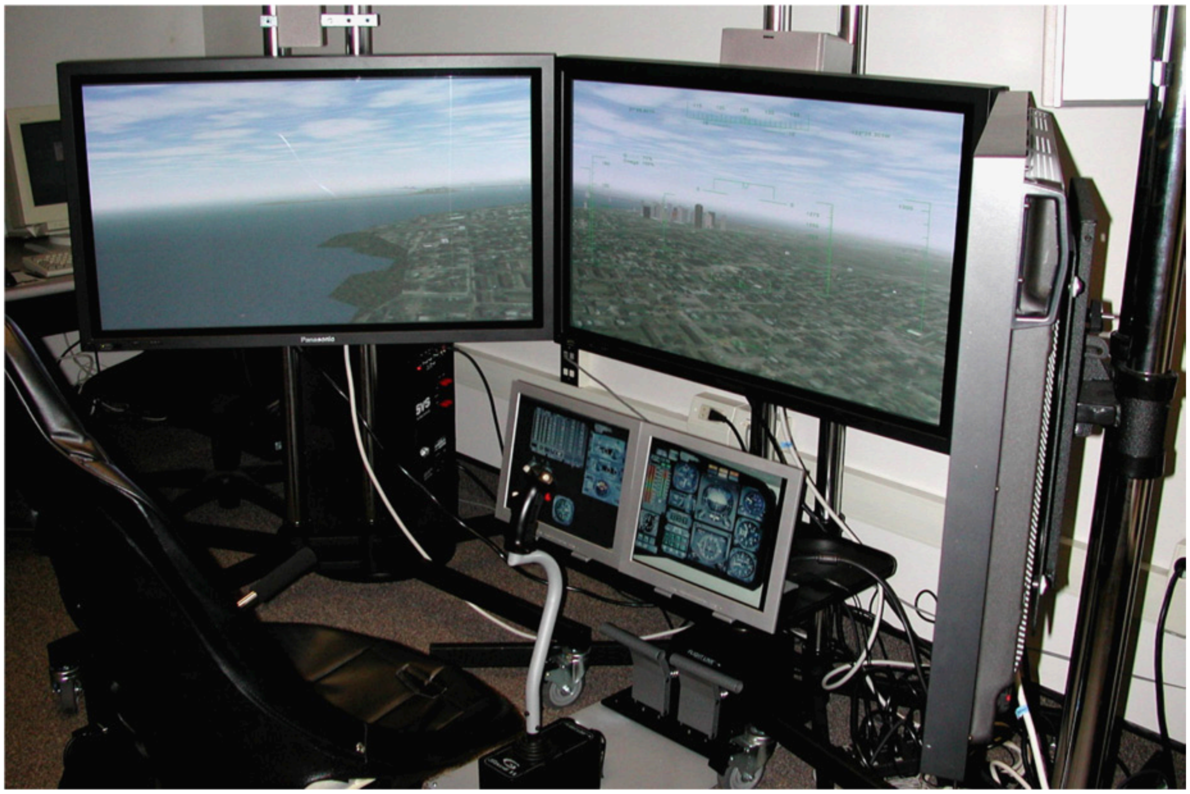


Fig. 1 Penn State Rotorcraft Simulator (Cockpit).

A. Hardware Implementation

The hardware architecture schematic, shown in Fig. 4, illustrates the interconnections of all existing and planned components of the simulator. A detailed description of the hardware components and their arrangement is presented below.

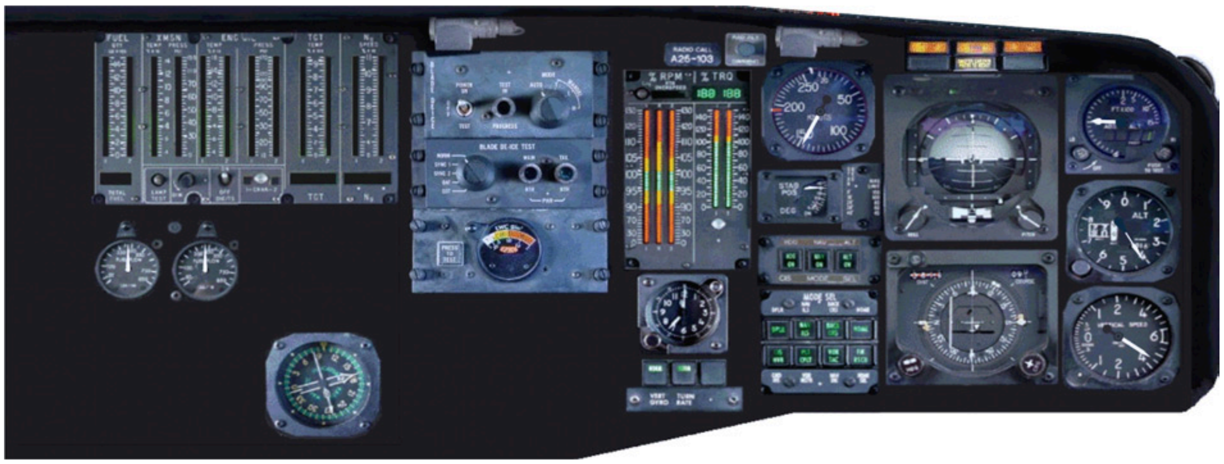


Fig. 2 Penn State Rotorcraft Simulator (Cockpit Instruments).

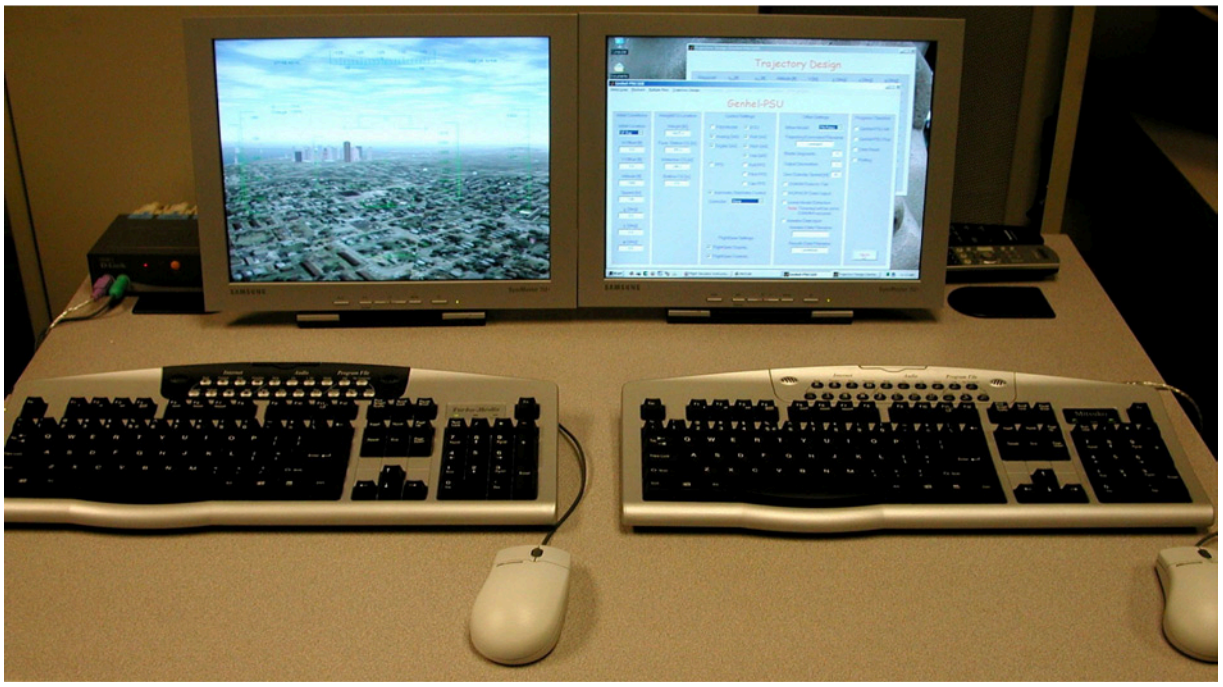


Fig. 3 Penn State Rotorcraft Simulator (Operator Station).

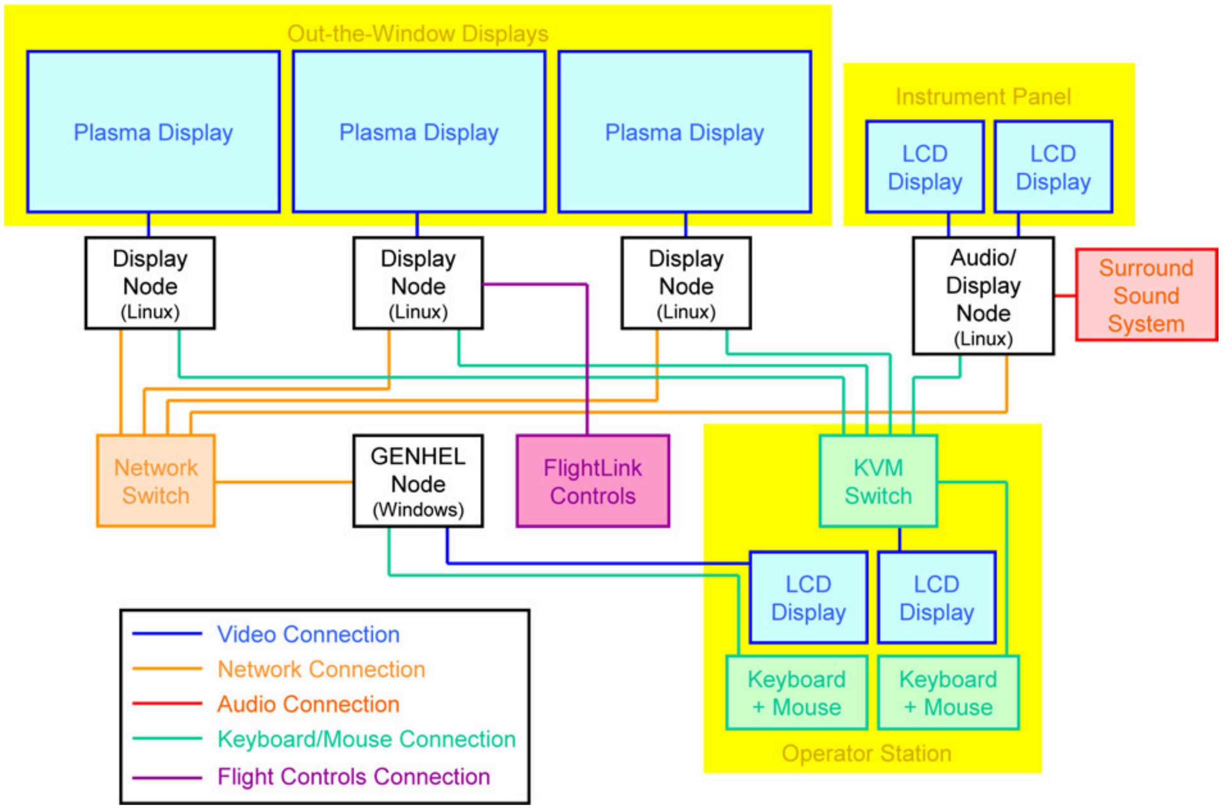


Fig. 4 Hardware Schematic.

1. Displays

For the primary out-the-window graphics, three Panasonic 42-inch HDTV flat-panel plasma displays are mounted on floor stands. Compared to a projector system, plasma displays offer a number of advantages, including superior definition and brightness (especially when room lighting cannot be easily controlled), a smaller footprint (no screens or other display surfaces are required), and maximum flexibility due to the moveable floor stands. Furthermore, the cost of plasma displays has decreased significantly in the last several years, resulting in an affordable alternative to a projector system. There are some disadvantages to using plasma displays, including motion parallax and reduced field-of-view; however, these were outweighed by advantages, especially due to the fact that multiple plasma displays are arranged to provide a wide wrap-around view.

In addition, four Samsung 15-inch, high-contrast LCD monitors are available for auxiliary displays. Currently, two monitors are mounted below the center plasma screen to display cockpit instruments and two monitors are used as consoles for the operator station. The operator station consoles can monitor the out-the-window displays, in addition to displaying information related to preprocessing and postprocessing of simulation results. Display selection for the operator station is performed using a 4-port KVM (keyboard, video, mouse) switch. LCD monitors were chosen since they require minimal desk space and power with a price comparable to CRT displays. In the future, additional LCD monitors may be included to simulate the chin windows of the helicopter, which would improve realism from the pilot's point of view.

2. Computer Hardware

The computers used for the simulator are typical high-end PC computers with NVIDIA GeForce 4 Ti4600 AGP graphics cards, with simultaneous analog and digital video capability. All of the nodes have 100 Mbps Fast Ethernet network cards to connect with the private network through an 8-port switch; the master node has an additional 100 Mbps Fast Ethernet network card to link the internal network to the Internet. The audio node computer also contains a quality sound card to interface with the sound system, described separately. The computers are mounted in a 25U cabinet in order to minimize the footprint of the facility.

3. Flight Controls

Rotary wing flight controls, including a cyclic stick, collective lever, seat, and pedals, were purchased from FlightLink Aviation Training Devices. The cyclic stick and pedals have a built in spring gradient, but do not have re-centering capability (force-trim release). The spring gradient, friction, damping, and breakout characteristics of the stick and pedals are not adjustable, and therefore do not necessarily represent the characteristics of the simulated aircraft. The collective lever has variable friction, but no back-drive. The cyclic stick also features a number of buttons that can serve different functions in the simulation. The lack of back-drive, re-centering, and capability to program feel characteristics are significant drawbacks. The FlightLink controls were chosen since they were the best rotary-wing controls available for a relatively low cost ($2,000). More realistic rotorcraft simulation control inceptors with programmable control loading systems are produced in relatively small numbers and the cost runs in the tens of thousands of dollars. On the other hand commercially available multi-axis joysticks cost approximately $100. These can be used to simulate a 3-axis sidestick inceptor, but their limited range of control travel and low spring forces make them unsuitable for representation of a center stick.

4. Sound System

An advanced sound system was developed primarily for research on rotorcraft acoustics simulations. Currently, the system is mainly used to generate playback of external noise simulations, although an internal noise component of the simulation may be added in the future. The system uses an off-the-shelf Sony 6.1 surround sound home theater system with a Dolby Digital/DTS receiver. The surround sound capability of the system has not been utilized in the initial phase of the acoustic simulation. To generate the characteristic low-frequency rotor noise, an SVS 16-46 CS-Plus subwoofer with a built in 525-watt BASH amplifier is used. This subwoofer has a low-frequency response of approximately 16 Hz.

B. Software Implementation

The simulator incorporates several different software packages running on different nodes; these include FlightGear, the flight simulation software, GENHEL, the flight dynamics model, CHARM, a free wake model that is used to calculate rotor inflow, MATLAB™, which is used to provide a graphical user interface (GUI) for the simulator, and PSU-WOPWOP, the rotor noise prediction code. The schematic in Fig. 5 illustrates how data is transferred between these different programs. A further description of the key software components and their implementation is given below.

1. Graphical Cluster

The flight simulator has been configured in an arrangement we refer to as a 'graphical cluster.' The design is based on the 'Beowulf cluster,' a widely used multi-computer architecture in which several commodity computers (typically PC's) are linked together within a private network to provide large-scale parallel computations.[12] In the current flight simulator, the nodes are connected together in the same way as a traditional Beowulf cluster, but each node has graphical capabilities and provides a visual output. In this arrangement, each node is a graphical workstation containing a high-end graphics card supporting multiple display outputs. A master node acts as the gateway to the external network and distributes computational and/or graphical tasks to the slave nodes. Due to the modular nature of the Beowulf architecture, other nodes can easily be added to the private network; for example, future additions include a dedicated audio node and flight dynamics nodes that each operate simultaneously on separate aspects of the simulation model, such as the CHARM free-wake aerodynamics module (described below), ship airwake solutions, or additional flight controls systems.

2. FlightGear (Flight Simulation Software)

One of the key software components of the MDRSF is the FlightGear simulation package, which acts as the primary graphics engine, coordinates network communication between subsystems, and reads in pilot inputs. FlightGear is a no-cost, open source simulation code that is portable across many different processors and operating systems. The code is written in the C++ programming language and uses OpenGL graphics. It is also linked with the no-cost,

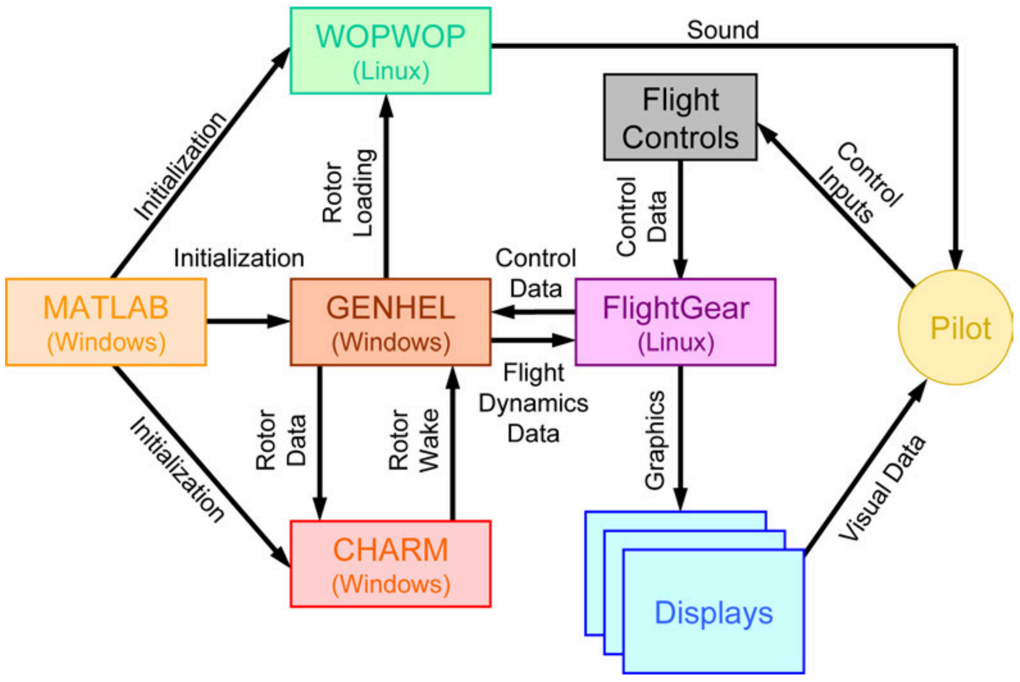


Fig. 5 Software Schematic/Data Flowchart.

open source SimGear and PLIB simulation and gaming libraries. For this application, FlightGear version 0.9.4 is compiled and run on computers with the SUSE Linux 9.0 operating system.

FlightGear supports the use of multiple displays to form a panoramic, or wrap-around, view. An unlimited number of display channels are permitted, where each channel is a separate process running on a separate computer, communicating with the master node via network sockets. The network socket communication capability is built into the FlightGear software. The FlightGear processes on the display computer operate in a master/slave mode, in which the slave computers wait for state information from the master node before updating the graphics display. For the network communication, FlightGear offers a choice between the User Datagram Protocol (UDP) and the Transmission Control Protocol (TCP). UDP was chosen over TCP for data distribution in the simulator because UDP produces less network overhead and includes packet headers that contain fewer bytes compared to TCP.

As for external graphics, the base package that comes installed with FlightGear provides a fairly large terrain database based on satellite photos, mainly centered around San Francisco. Most of the terrain is simply an appropriate section of a satellite photo mapped onto the two-dimensional ground surface. Three-dimensional models are also used to model buildings and other large objects; in most places, the building models are placed randomly, but accurate models for the Golden Gate Bridge, skyscrapers in downtown San Francisco, and the terminal, control tower, and runways at San Francisco International Airport are provided. FlightGear also allows for modification of the graphical database, supporting a variety of different formats for inserting graphical models. For example, a visual model of a HH-60J Coast Guard helicopter was used to provide a third-person view of the aircraft, and a model of an LHA-class ship (shown in Fig. 6) was added for use in studying shipboard operations. The capability to include 3-D graphical

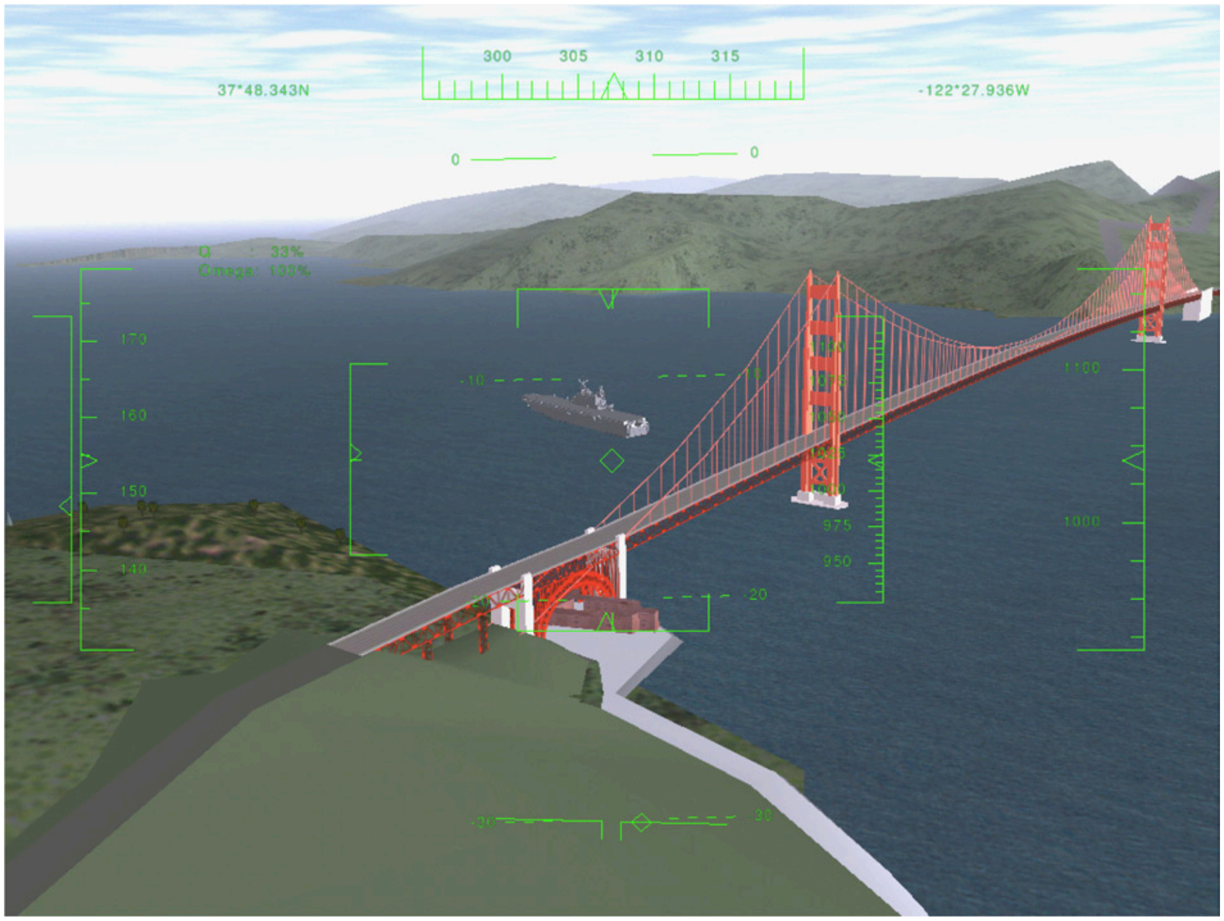


Fig. 6 Simulator Display with LHA-Class Ship.

models for adding visual detail at low altitude is particularly important for rotorcraft, which fly lower and slower than fixed-wing aircraft.

FlightGear uses two native protocols for interfacing with an external flight dynamics model. The native FDM protocol is used to receive flight dynamics data (position, attitude, engine conditions, etc.) from the external flight dynamics model. The native controls protocol is used to send the pilot control inputs (cyclic, pedals, collective, and discretes) to the external flight dynamics model. Two C++ header files define these two native protocol classes, which are essentially data structures containing all of the data passed between FlightGear and the external flight dynamics model. The header files are also linked with GENHEL (the flight dynamics model for the MDRSF), along with a function to send and receive data via network sockets.

The primary downside to FlightGear, as with other open source software, is that it lacks the extensive documentation and support that is often available with commercially purchased software. There is some online documentation available, but some of this documentation is out of date, although it is marked as such, and many advanced features (including the capability to interface with external flight dynamics models, as discussed below) are not well documented. This is partially alleviated by the fact that the code is open source; direct examination of the source code can often provide a better understanding than a help file. Furthermore, the FlightGear development community itself is open and the developers freely exchange information about the code with advanced users and other developers, mainly through electronic mailing lists and the FlightGear website.

3. GENHEL Flight Dynamics Model

The flight dynamics model selected for the simulator is the U.S. Army/NASA Ames GENHEL model of the UH-60A Black Hawk.[9] This is a well-established Fortran-based simulation code made available to Penn State by NASA Ames Rotorcraft Division. The code presents an accurate representation of the flight dynamics of the UH-60A Black Hawk, but does not provide any support for out-the-window graphics or direct input for pilot controls. Furthermore, the code operates from input files and does not have a graphical user interface. Thus, the main software development work involved with this project was to interface the GENHEL mathematical model with FlightGear and to provide an improved user interface for the operator.

As described above, one of the features of FlightGear is that it can use its built-in network socket communications to transmit data to and from an external flight model running on a separate computer. Since the FlightGear code is open source, it was possible to adapt the network communication portion for use in GENHEL (as C++ code in a Fortran wrapper). The original GENHEL Fortran code remains essentially unchanged, so no further validation is required. This also contributes to the modular nature of the simulator in that the GENHEL code can be modified and recompiled independently of any other software. The FlightGear code also required minor modifications to add rotorcraft-related variables to the data passed from GENHEL.

The GENHEL code is compiled and executed with Compaq Visual Fortran 6.6 and Visual Studio 6.0 under the Windows 2000 operating system. The Visual Studio environment is used because it provides debugging features and allows easy compilation of mixed-language code (needed to include the C++ network interface). The fact that GENHEL and FlightGear run on different operating systems does not present any major obstacles.

In addition to the network interface with FlightGear, a number of modifications and improvements have been made to the original GENHEL code. The most notable of these are listed below:

1. A modified main executive was added to control execution of the GENHEL model. The main GENHEL model is a separate subroutine partitioned from the main executive. The main executive controls timing for real-time operation, transmission of data to and from FlightGear, and can be used to implement modified flight control laws.
2. Prescribed pilot control inputs can be read in from files into the main executable in order to run check cases.
3. A pilot model is available in the main executable in order to simulate prescribed flight trajectories read from a file. The pilot model is based on an optimal control model (OCM) representation of the human pilot,[13] and uses airspeed scheduled multi-input, multi-output (MIMO) control laws to autonomously track a three-dimensional trajectory.
4. The code was modified to allow variation in the number of blade segments for the main rotor. A larger number of blade segments are necessary for acoustics calculations.

5. The CHARM free wake module[14] has been integrated with the GENHEL code. The CHARM free wake can be used for calculations related to inflow and rotor-tail aerodynamic interaction.
6. A subroutine has been added to allow output of blade loading data compatible with the PSU-WOPWOP acoustics code.[15–17] This enables the acoustics playback discussed later in this paper.
7. Computational fluid dynamics (CFD) solutions of a ship airwake can be read in from a file in order to accurately simulate helicopter flight in shipboard operations.[13]
8. The models of the stability augmentation system (SAS) and flight path stabilization system (FPS) have been modified so that individual axes can be turned on or off.

4. *Execution of FlightGear and GENHEL*

As shown in Fig. 4, FlightGear must be executed on several different graphical nodes, and a number of network communications between different executions of FlightGear and the GENHEL mathematical model need to be established. A master graphical node reads the pilot control inputs, handles communications with GENHEL, and sends data to the slave nodes. Meanwhile, the slave nodes simply receive data from the master node and drive the out-the-window displays and cockpit instruments. This complex execution of the FlightGear software is automated using shell scripts and the extensive array of command line options available in FlightGear, as described below:

1. The *execute* script on the master node starts the FlightGear program on all of the slave nodes remotely, using the *fgfsnode* scripts shown below. In addition, it starts FlightGear on the master node with instructions to receive flight dynamics data from the GENHEL node, transmit pilot control data to the GENHEL node, and send flight dynamics data to the three slave nodes driving the out-the-window and instrument panel displays.
2. On each slave node, the *fgfsnode* script executes FlightGear with instructions to receive all flight dynamics data from the master node via a designated port. The slave nodes also set different field-of-view options; for example, to set left and right window views.
3. Once the FlightGear software is executed, it runs completely independently of the flight dynamics model. If no flight dynamics data is sent from GENHEL, FlightGear simply displays the last known state of the aircraft, appearing ‘frozen’ while waiting for new data to be sent. When the flight dynamics data stream resumes, FlightGear continues to update the display.

GENHEL can be executed from the Visual Fortran environment or can be initiated from the MATLAB interface as discussed in the following section. The network interface between the GENHEL and FlightGear is designed so that data is exchanged every 30 milliseconds. GENHEL typically operates with a 10 millisecond time step, so data is exchanged every third time step. Real-time execution of the code proceeds as follows:

1. The GENHEL model and network communications with FlightGear are initialized.
2. The flight dynamics data is sent to FlightGear.
3. The pilot control data is received from FlightGear.
4. GENHEL executes 30 milliseconds of flight dynamics calculations (normally 3 time steps).
5. GENHEL waits until the system clock time is incremented exactly 30 milliseconds from the previous time step, then returns to step 2.

Control latency will result due to the sampling rate of the controls (resulting in delay of 0-30 milliseconds), the time to execute the flight dynamics calculations (30 milliseconds), and the time delay to send data back and forth over the network sockets. The UDP network protocol was chosen primarily to minimize the delay due to network communications. The actual control latency was measured by displaying on the HUD the current lateral control position measured by FlightGear and the lateral control position sent back by GENHEL. The pilot executed a sudden lateral stick input and the values displayed were recorded using a standard video camera. Using frame-by-frame playback, the latency could be measured by observing the number of frames required before the value sent back by GENHEL could reach the same value as that measured directly by FlightGear. The total latency was found to be between 67 milliseconds and 100 milliseconds. Considering that the sampling and flight dynamics calculations result in up to 60 milliseconds of time delay, the latency due to network communications is relatively small. Latency could probably be improved by decreasing the time between the data exchanges (exchange data every 10 milliseconds instead of every 30 milliseconds). However, it is not currently possible to do this with real-time execution of GENHEL because there are limitations on the precision of the system clock functions in Visual Fortran.

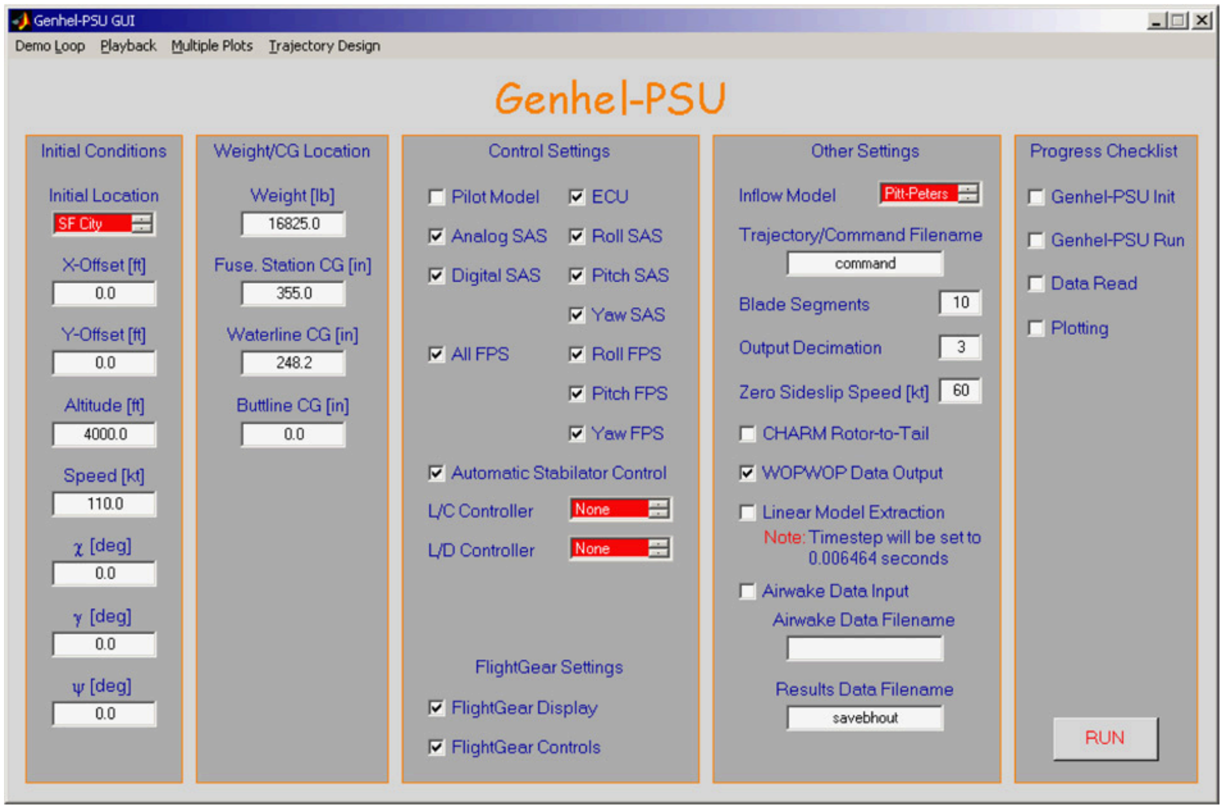
5. MATLAB Interface

In order to provide a straightforward interface for the simulator operator, a graphical user interface (GUI) has been designed in MATLAB 6.5.[19] The GUI (shown on the operator station monitor in Fig. 3 and in Fig. 7 below) allows the simulator operator to set a number of options for the GENHEL code, such as:

1. Initial aircraft states, including position, altitude, heading, and airspeed.
2. Aircraft load properties, such as weight, inertia, and center of gravity location.
3. Flight control settings, including SAS, FPS, and user-defined control laws.
4. Mode of operation: real-time piloted simulation, real-time display of a flight trajectory flown by the pilot model, or real-time display of a flight with control inputs specified from an input file.
5. Simulation-related variables, such as inflow model, number of main rotor blade segments, ship airwake input data, and PSU-WOPWOP output data.

Without the GUI, these options are set by editing a number of text input files. The MATLAB GUI, however, automatically generates these input files and then executes the GENHEL code. After the simulation run is complete, MATLAB processes the output files and provides time history plots of the aircraft states.

The GUI also incorporates a number of related tools, including plotting options for multiple simulation runs, automated trim sweeps, and an interactive routine to design aircraft trajectories for use with the GENHEL pilot model (see Fig. 8). The trajectory design tool allows the user to specify a set of waypoints in three-dimensional space, with specified airspeed, heading, and flight path at each point. Optimal control theory is used to generate a smooth trajectory between waypoints, which minimizes a weighted average of linear acceleration, angular acceleration, and total maneuver time. Once the trajectory is designed, the OCM pilot model can autonomously fly the maneuver, while the simulator provides a graphical display.



Initial Conditions
Initial Location: SF City
X-Offset [ft]: 0.0
Y-Offset [ft]: 0.0
Altitude [ft]: 4000.0
Speed [kt]: 110.0
χ [deg]: 0.0
γ [deg]: 0.0
ψ [deg]: 0.0

Weight/CG Location
Weight [lb]: 16825.0
Fuse. Station CG [in]: 355.0
Waterline CG [in]: 248.2
Buttline CG [in]: 0.0

Control Settings
☐ Pilot Model ☑ ECU
☑ Analog SAS ☑ Roll SAS
☑ Digital SAS ☑ Pitch SAS
☑ Yaw SAS
☑ All FPS ☑ Roll FPS
☑ Pitch FPS
☑ Yaw FPS
☑ Automatic Stabilator Control
L/C Controller: None
L/D Controller: None
FlightGear Settings
☑ FlightGear Display
☑ FlightGear Controls

Other Settings
Inflow Model: Pitt-Peters
Trajectory/Command Filename: command
Blade Segments: 10
Output Decimation: 3
Zero Sideslip Speed [kt]: 60
☐ CHARM Rotor-to-Tail
☑ WOPWOP Data Output
☐ Linear Model Extraction
Note: Timestep will be set to 0.006464 seconds
☐ Airwake Data Input
Airwake Data Filename:
Results Data Filename: savebhout

Progress Checklist
☐ Genhel-PSU Init
☐ Genhel-PSU Run
☐ Data Read
☐ Plotting

Fig. 7 MATLAB-based Graphical User Interface.

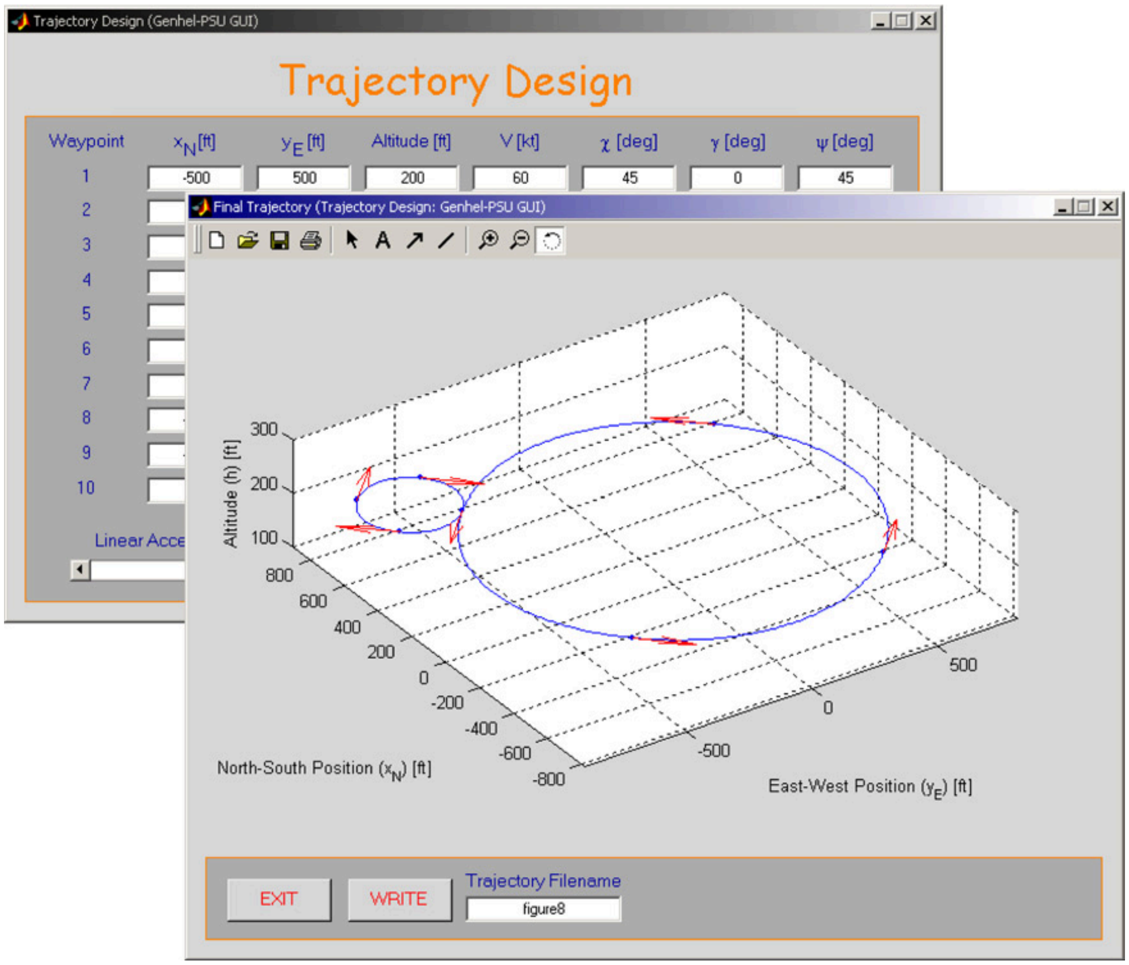


Fig. 8 Trajectory Design Graphical User Interface.

6. *Acoustics Simulation Interface*

During flight simulation, the aircraft position and attitude, along with the loading and motion (flap, lag, and pitch) of the main rotor blades can be recorded throughout the maneuver, so the noise at a single observer location can be calculated and heard in a 'playback' mode; tail rotor loading and motion data is currently not saved, but could be utilized in future implementations. The software that is used to calculate the noise is the PSU-WOPWOP acoustic prediction code.[15–18] PSU-WOPWOP is based upon the same theoretical basis as the NASA WOPWOP code,[20] but can be used to predict the noise of maneuvering rotorcraft with multiple rotors. PSU-WOPWOP, which is written in Fortran 95 with an object-oriented design, uses a source-time-dominant algorithm to compute the noise from Farassat's Formulation 1A[21] of the Ffowcs Williams-Hawkings (FW-H) equation[22] at an arbitrary observer location. PSU-WOPWOP also has an option to use a compact-chordwise loading model that is well suited to the output of GENHEL—i.e., it only requires blade loading as a function of span and rotor azimuth. This approximation is faster than using full surface pressure integration. The combination of the source-time-dominant algorithm and the compact chordwise loading noise model enables PSU-WOPWOP to run approximately 50 times faster than a maneuver version of the NASA WOPWOP code (for stationary observers). Although the current implementation of PSU-WOPWOP only works in a 'playback' mode, the speed of PSU-WOPWOP makes the coupling of real-time acoustic prediction and flight dynamics a possibility.

After calculating the acoustic pressure (which is shown as a time history in Fig. 9), PSU-WOPWOP converts the pressure signal into an audio file format. The maneuver is then played back with video (from the point of view of the

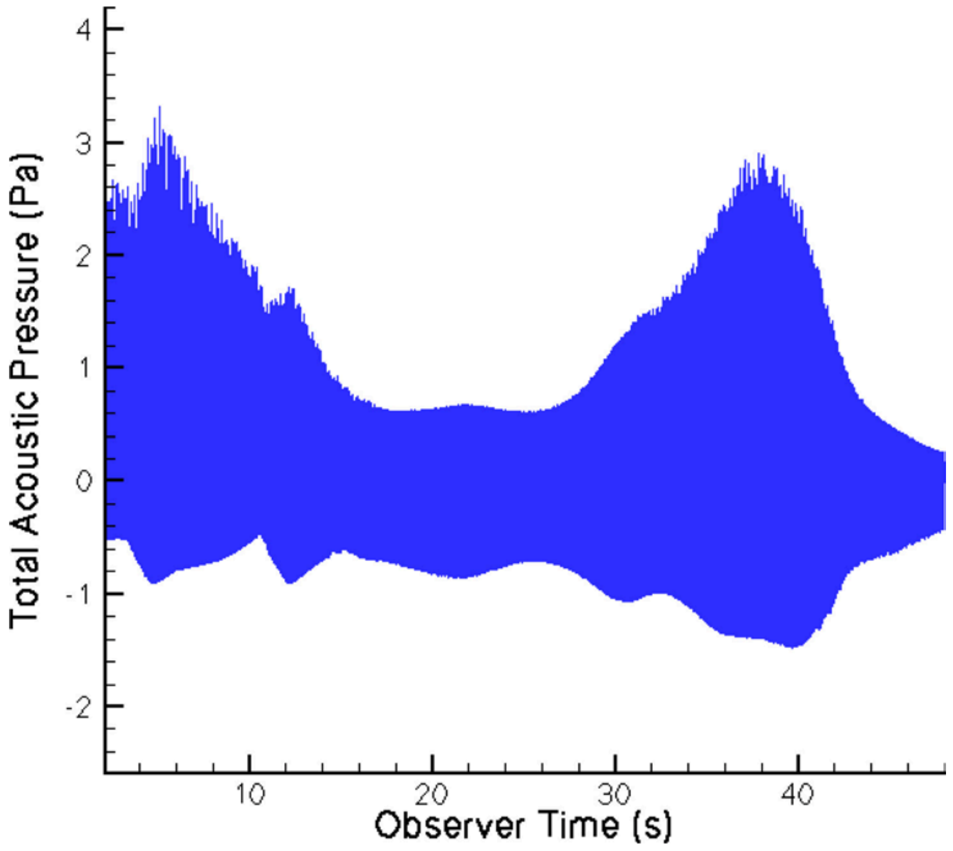


Fig. 9 Acoustic Pressure Time History.

stationary observer) shown on the out-the-window displays by FlightGear and sound from PSU-WOPWOP played over the sound system simultaneously.

III. Cost Analysis

As mentioned above, the main objective of the design of the MDRSF was to build a highly flexible real-time simulation tool with relatively small initial and recurring costs. Two features in particular that contribute to the cost-effectiveness of the simulator are the application of no-cost, open source software, where feasible, and the use of commercial off the shelf (COTS) hardware components.

A. Software Costs

All software used in this project falls into one of three categories:

1. Open source software available at no cost (SUSE Linux, FlightGear).
2. Software that has been developed at Penn State (PSU-WOPWOP) or provided to Penn State at no charge for research purposes (GENHEL, CHARM).
3. Commercially available software provided via a site license (Windows 2000, Visual Fortran, Visual Studio, and MATLAB).

Due to the fact that all of the software was available to us at no cost, our initial software costs were zero. However, not everyone will be able to take advantage of site licensing, so software costs have also been calculated in the Table 1 using prices obtained from Nextag,[23] an online price guide.

Since necessary updates are provided at no additional cost for the commercial software used in the simulator, the recurring software costs are negligible.

Table 1 Software Cost Summary.

Software component	Approximate cost
MathWorks MATLAB 7 (Note: Version 6.5 not available)	$1,900
Compaq Visual Fortran 6.6	$485
Microsoft Visual Studio 6.0	$260
Microsoft Windows 2000	$85
SUSE Linux 9.0	$0
FlightGear 0.9.4	$0
GENHEL	$0
CHARM	$0
PSU-WOPWOP	$0
Total Software Cost	$2,730

B. Hardware Costs

All hardware components are commodity components readily available online or in retail stores. Approximate costs of the major hardware components are listed in the Table 2. Some of the components were purchased using educational discounts:

Recurring hardware costs are essentially zero, although a small amount should be budgeted to account for minor repairs that may be needed (i.e. additional fuses for the subwoofers or replacement cables).

The overall initial cost of the MDRSF comes to approximately $37,600, while the initial cost including software is approximately $40,330—both well under the budget limit of $45,000, with no substantial recurring costs. Furthermore, the system is readily expandable with additional funds; for example, the only cost involved in adding a computational node is the price of a rackmount PC.

C. Development Time

Due to labor costs and time restraints, the amount of time required to develop the MDRSF is an important consideration alongside the actual costs of the hardware and software. Since the MDRSF was built and developed at an academic institution by faculty members and graduate students, labor costs were not factored into the budget. The total development time was approximately 200 hours; a breakdown of the development tasks, as well as the time spent on each task, is given in the Table 3.

Table 2 Hardware Cost Summary.

Hardware component	Approximate cost
3 HDTV Plasma Displays and Floor Stands	$18,200
5 Computers	$10,600
Sound System	$2,500
FlightLink Flight Controls	$2,000
4 LCD Monitors	$1,700
Uninterruptible Power Supply, Rackmount Enclosure, Other Accessories	$2,600
Total Hardware Cost	$37,600

Table 3 Development Time Summary.

Development task	Approximate time required
Hardware Assembly/Installation	20 hours
Software Installation/Configuration	30 hours
Modifications to GENHEL	60 hours
Modifications to FlightGear	30 hours
Development of MATLAB Scripts and GUI	60 hours
Total Development Time	200 hours

IV. Multi-Disciplinary Research Applications

The MDRSF is being used for a number of different research projects at the Penn State Rotorcraft Center of Excellence, spanning a number of different disciplines, including flight dynamics, controls, aerodynamics, and acoustics. It is important to recognize that the main application of the MDRSF is as an engineering simulation. The simulator design effort was focused on developing an affordable tool that is suitable for model and flight control development, not on developing a facility suitable for handling qualities analysis or pilot training.

A. Advanced Flight Control Design

An important application of the facility is in the area of advanced flight control design. The GENHEL code has been modified to allow simulation of advanced fly-by-wire (FBW) flight control systems. Individual channels of the existing SAS control laws can be disabled and replaced with user-defined control laws. For example, a model-following control law with attitude command/attitude hold (ACAH) response has been developed and tested in the simulator.[24] With the touch of a button on the cyclic stick, the automatic flight control system can be switched between the basic UH-60A limited-authority SAS to the full-authority model-following controller. This is a powerful method for demonstrating the potential differences in handling characteristics of a FBW controller and for illustrating basic concepts such as ACAH response.

The current pilot control hardware is limited by its lack of re-centering or trim capability. To help mitigate this problem, a software emulation of force trim release has been implemented. When the pilot presses and holds a certain button on the cyclic stick, the controls are temporarily disabled. The pilot can then re-center the stick and release the button, and the software offsets the stick input so that the neutral position matches the stick position when the button was pressed. Thus, the pilot can trim the aircraft while holding force on the stick and then use the button to relieve the stick force. Furthermore, the hat switch on the cyclic stick is programmed to provide small adjustments to the cyclic stick input so it effectively acts as a software emulated beep trim. In the future, a joystick will be integrated with the system to simulate a sidestick control. Some commercially available joysticks have some limited force feedback capability and may eventually be used for research on carefree maneuvering cueing systems.[25]

Although the simulator was found to be effective at testing and demonstrating advanced control laws, its effectiveness for handling qualities analysis may be somewhat limited. The main limitation is that the spring gradient, friction, and damping characteristics of the controls are not adjustable and the existing feel characteristics are not necessarily representative of a real aircraft. As described previously, the system also lacks force trim release; this could be remedied by purchasing high-fidelity control inceptors and control loaders, but would result in a significant cost increase. The lack of motion cueing would also limit its application for handling qualities evaluation of certain tasks.

B. Simulation of Shipboard Operations

The MDRSF is being used for current research on helicopter shipboard operations. This research investigates the effect of turbulent ship airwakes on the flight dynamics of helicopters operating in the vicinity of the ship. The MDRSF is primarily being used for developing the model of the helicopter/ship interface and for initial evaluation of the impact of ship airwake on handling characteristics. More detailed handling qualities study would probably have to be performed on a simulator with motion cueing.

CFD solutions of the ship airwake from a current U.S. Navy ship have been integrated with GENHEL.[9] The entire CFD database of the time-varying airwake is quite large—approximately 2 GB for a single wind-over-deck (WOD) condition. GENHEL needs to constantly access this data in real-time as the aircraft flies around the ship. It is not practical to store the entire database in internally in RAM, so the simulation must access the data from the hard drive. This was found to result in significant slow down and occasional loss of real-time execution. The current solution is to store only a portion of the CFD data for a particular landing spot and WOD (wind over deck) condition. In future implementations, the networking capability of the facility will be used to include the entire database for multiple WOD conditions. The airwake velocities will be stored and calculated on a separate computer and then relayed to the flight model via network socket communication.

C. Acoustics Simulation of Maneuvering Flight

The acoustic simulation system has been designed to predict and play the noise from of a rotorcraft, initially in a "playback" mode, but eventually in real-time.[15] The "playback" mode starts with a simulated flight, which could be flown by either a pilot or a computer model. During the flight simulation, aircraft position, attitude, blade loading, and blade motion are all recorded. These quantities are provided as inputs to the PSU-WOPWOP noise prediction code to predict the rotor noise during the flight. The acoustic pressure output from PSU-WOPWOP is then converted into an audio file format and played back over a high-fidelity audio subsystem of the flight simulator. This simulation will represent the noise heard by an observer at a fixed position on the ground. The hardware is capable of surround sound, which can be utilized through noise computations for each of the separate speakers.

In the future it is planned to develop a real-time noise prediction capability; therefore, the simulation hardware has been developed for that task. In real-time noise simulation, the pilot will be able to hear and monitor the noise the rotorcraft is radiating to a particular observer location. This information will provide feedback, enabling the pilot to adjust the flight controls to minimize the noise for a noise sensitive area on the ground. Directional acoustic predictions (surround sound) require several noise calculations to provide the directionality; hence, they are not planned for initial real-time noise prediction.

D. Integration of Free Wake Model

Free wake models have proven to be an accurate means for modeling the inflow dynamics of helicopter rotors. Current simulation models primarily use finite-state wake models in order to run in real-time. However, recent advances have demonstrated the potential for using free wake models in real-time simulation.[14] Free wake representation of the main rotor wake has the potential of solving the so-called off-axis coupling discrepancy in current blade element simulations of rotorcraft, and it can also be used to model the complex interactions of the main rotor wake and the empennage.[26]

The CHARM free wake model (provided to Penn State by Continuum Dynamics, Inc.) has been integrated with the GENHEL simulation model. Preliminary results of this model were presented in Reference 27. The wake model can potentially increase the fidelity of the simulation, and provide an accurate model of rotor-tail interaction without empirical correction factors. A visualization tool can be used to animate the rotor wake vortices in real-time as the aircraft performs a maneuver, allowing researchers to better understand the behavior of the rotor wake in maneuvering flight.

The CHARM module has a number of input parameters that can be adjusted to control the level accuracy and speed of execution. The module can be executed in real-time with certain options or run with more fidelity with other options. Currently, a number of different variations in the input parameters are being analyzed to study the trade-off between fidelity and real-time performance. Eventually, the CHARM module will run on a separate node in parallel with the main GENHEL module. This will greatly increase speed of execution and allow real-time operation with a more sophisticated wake.

V. Conclusions

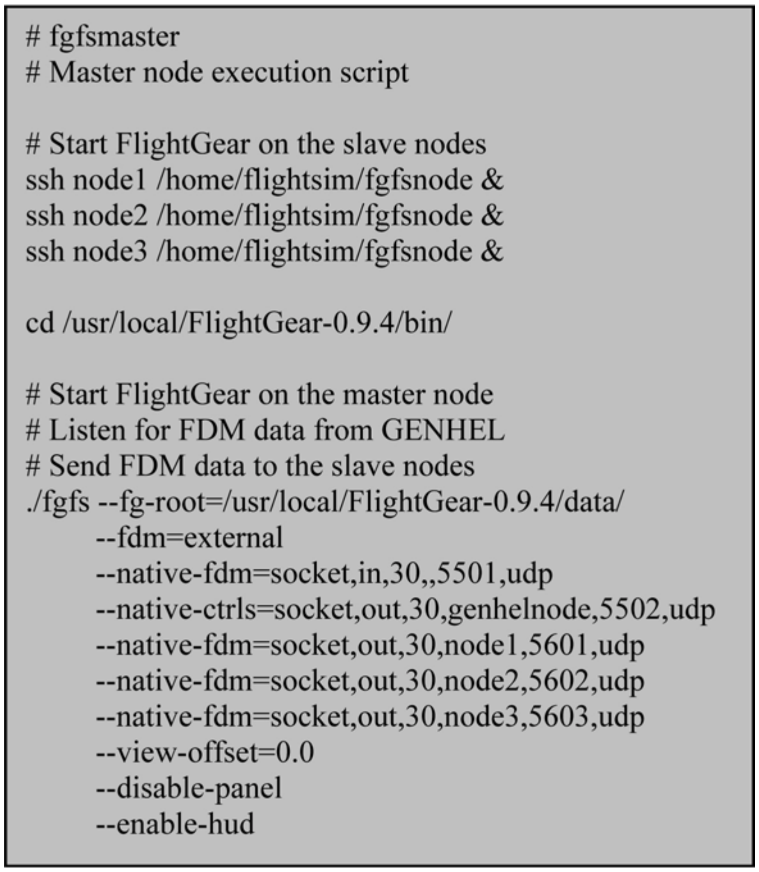
The objective was to develop a rotorcraft simulator for multi-disciplinary research and education in a university environment within relatively restrictive cost constraints. The MDRSF meets this prescribed objective. The simulator has proven to be useful for flight path visualization, coupled flight dynamics/acoustics simulation, flight control design and analysis, and flight dynamics model development. The greatest potential for this facility lies in its expandability. New functionality can be achieved by tying different nodes into the cluster. For example, adding a free wake node or ship airwake node will expand the modeling capability of the system while maintaining real-time performance.

Currently, the simulator has some limitations, primarily with control inceptor feel characteristics and lack of motion cueing, which reduce the simulator's effectiveness for handling qualities analysis. These limitations can be addressed with hardware upgrades, but it is estimated that this would lead to at least a 100% increase in cost. This estimate is based on available quotes for low-end motion bases and control loading systems. At present there seems to be a significant cost increment to upgrade from a basic engineering simulation to one suitable for handling qualities analysis of rotorcraft.

The availability of open source software was essential to this project. Inexpensive commercially available software was found to be too restrictive for research-oriented applications. On the other hand, more “professional grade” rotorcraft simulation software was prohibitively expensive. The use of a government supplied simulation code, GENHEL, provided a good combination of flexibility (in that we were able to modify and add to the source code) and model fidelity. FlightGear is open source software, available at no cost, developed by a community of flight simulator enthusiasts, but it is nonetheless a powerful high-end simulation tool. The main difficulty is the lack of current, complete documentation for advanced features of the software. But the fact that the code is open source maximizes its flexibility, making FlightGear an effective tool if experienced programmers are available. This flexibility, coupled with the fact that it is available at no cost, makes it attractive for universities or small businesses involved in research and development.

Appendix

As described above, FlightGear is started on multiple display nodes in a master/slave format through the use of shell scripts. The scripts are provided in this appendix in order to assist others who may want to build a similar system. Figure 10 presents the script that used on the master node, while Fig. 11 presents a sample script from one of the slave display nodes.



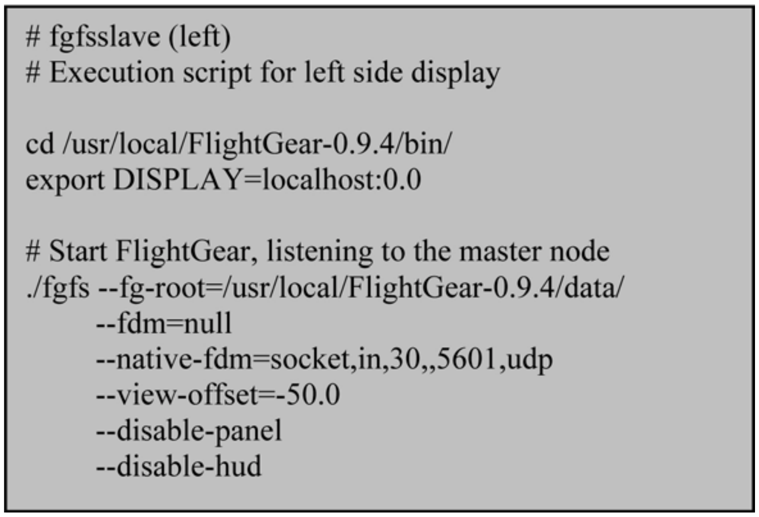
```
# fgfsmaster
# Master node execution script

# Start FlightGear on the slave nodes
ssh node1 /home/flightsim/fgfsnode &
ssh node2 /home/flightsim/fgfsnode &
ssh node3 /home/flightsim/fgfsnode &

cd /usr/local/FlightGear-0.9.4/bin/

# Start FlightGear on the master node
# Listen for FDM data from GENHEL
# Send FDM data to the slave nodes
./fgfs --fg-root=/usr/local/FlightGear-0.9.4/data/
        --fdm=external
        --native-fdm=socket,in,30,,5501,udp
        --native-ctrls=socket,out,30,genhelnode,5502,udp
        --native-fdm=socket,out,30,node1,5601,udp
        --native-fdm=socket,out,30,node2,5602,udp
        --native-fdm=socket,out,30,node3,5603,udp
        --view-offset=0.0
        --disable-panel
        --enable-hud
```

Fig. 10 Master Display Node Execute Script.



```
# fgfsslave (left)
# Execution script for left side display

cd /usr/local/FlightGear-0.9.4/bin/
export DISPLAY=localhost:0.0

# Start FlightGear, listening to the master node
./fgfs --fg-root=/usr/local/FlightGear-0.9.4/data/
        --fdm=null
        --native-fdm=socket,in,30,,5601,udp
        --view-offset=-50.0
        --disable-panel
        --disable-hud
```

Fig. 11 Slave Display Node Execution Script.

References

[1]Padfield, G. D., and White, M. D., "Flight Simulation in Academia: HELIFLIGHT in Its First Year of Operation," The Challenge of Realistic Rotorcraft Simulation RAeS Conference, London, United Kingdom, November 2001.

[2]Edvani, S. K., van Zuylen, E., and Bettendorf, S., "SIMONA - A Reconfigurable and Versatile Research Facility," AIAA Modeling and Simulation Technologies Conference and Exhibit, New Orleans, LA, August 11–13, 1997.

[3]Mansur, M. H., Frye, M., Mettler, B., and Montegut, M., "Rapid Prototyping and Evaluation of Control Systems Designs for Manned and Unmanned Applications," American Helicopter Society 56th Annual Forum, Virginia Beach, VA, May 2000.

[4]Jeram, G. J., "Open Design for Helicopter Active Systems," American Helicopter Society 58th Annual Forum, Montreal, Canada, June 2002.

[5]Redkoles, P., "Distributed Flight Simulation Architectures Using a PC/Linux Cluster," American Helicopter Society 59th Annual Forum, Phoenix, AZ, May 2003.

[6]Perry, A. R., and Olson, C. "FlightGear from History to Future," LinuxTag 2001, Stuttgart, Germany, July 2001.

[7]Deters, R. W., Dimlock, G. A., and Selig, M. S., "Icing Encounter Flight Simulator with an Integrated Smart Icing System," AIAA Modeling and Simulation Technologies Conference and Exhibit, Monterey, CA, August 5–8, 2002.

[8]Summers, P., Barnes, D. P., and Shaw, A., "Determination of Planetary Meteorology from Aerobot Flight Sensors," Seventh ESA Workshop on Advanced Space Technologies for Robotics and Automation, ASTRA 2002, ESTEC Noordwijk, The Netherlands, November 19–21, 2002.

[9]Howlett, J., "UH-60A BLACK HAWK Engineering Simulation Program: Volume I—Mathematical Model," NASA CR-177542, USAAVSCOM TR 89-A-001, September 1989.

[10]"InMotion Simulation," URL: http://www.inmotionsimulation.com [cited 25 March 2005].

[11]Schroder, J. A., "Helicopter Flight Simulation Motion Platform Requirements," NASA TP-1999-208766, July 1999.

[12]Sterling, T., Salmon, J., Becker, D. J., and Savarese, D. F. *How to Build a Beowulf*, The MIT Press, Cambridge, MA, 1999.

[13]Lee, D., Horn, J. F., Sezer-Uzol, N., and Long, L. N., "Simulation of Pilot Control Activity During Helicopter Shipboard Operations," AIAA Atmospheric Flight Mechanics Conference and Exhibit, Austin, TX, August 11–14, 2003.

[14]Wachspress, D. A., Quackenbush, T. R., and Boschitsch, A. H., "First-Principles Free-Vortex Wake Analysis for Helicopters and Tiltrotors," American Helicopter Society 59th Annual Forum, Phoenix, AZ, May 2003.

[15]Brentner, K. S., Lopes, L. V., Chen, H., and Horn, J. F., "Near Real-Time Simulation of Rotorcraft Acoustics and Flight Dynamics," American Helicopter Society 59th Annual Forum, Phoenix, AZ, May 2003.

[16]Brentner, K. S., Brès, G. A., Perez, G., and Jones, H. E., "Maneuvering Rotorcraft Noise Prediction: A New Code for a New Problem," American Helicopter Society Aerodynamics, Acoustics, and Test & Evaluation Technical Specialists Meeting, San Francisco, CA, January 23–25, 2002.

[17]Brentner, K. S., Perez, G., Brès, G., and Jones, H. E., "Toward a Better Understanding of Maneuvering Rotorcraft Noise," American Helicopter Society 58th Annual Forum, Montreal, Canada, June 2002.

[18]Chen, H., Brentner, K. S., Lopes, L. V., and Horn, J. F., “A Study of Rotorcraft Noise Prediction in Maneuvering Flight,” AIAA 42nd Annual Aerospace Meeting and Exhibit, Reno, NV, January 5–8, 2004.

[19]Anon., *Building GUIs with MATLAB Version 5*, The MathWorks, Inc., June 1997.

[20]Brentner, K. S., “Prediction of Helicopter Discrete Frequency Rotor Noise—A Computer Program Incorporating Realistic Blade Motions and Advanced Formulation,” NASA TM 87721, October 1986.

[21]Farrasat, F. and Succi, G. P., “The Prediction of Helicopter Discrete Frequency Noise,” *Vertica*, Vol. 8, No. 4, 1983, pp. 309–320.

[22]Ffowcs Williams, J. E., and Hawkings, D. L., “Sound Generation by Turbulence and Surfaces in Arbitrary Motion,” *Philosophical Transactions of the Royal Society of London*, Series A, Vol. 264, No. 1151, May 1969, pp. 321–342.

[23]“NexTag—Compare Prices Before You Buy,” URL: http://www.nextag.com [cited 7 September 2004].

[24]Sahani, N. A., and Horn, J. F., “Adaptive Model Inversion Control of a Helicopter with Structural Load Limiting,” AIAA Guidance, Navigation, and Control Conference and Exhibit, Providence, RI, August 16–19, 2004.

[25]Sahani, N. A., Horn, J. F., Jeram, G., and Prasad, J. V. R., “Hub Moment Limit Protection System Using Neural Network Prediction,” American Helicopter Society 60th Annual Forum, Baltimore, Maryland, June 2004.

[26]Spoldi, S., and Ruckel, P., “High Fidelity Helicopter Simulation Using Free Wake, Lifting Line Tail, and Blade Element Tail Rotor Models,” American Helicopter Society 59th Annual Forum, Phoenix, AZ, May 2003.

[27]Kothmann, B. D., Lu, Y., DeBrun, E., and Horn, J. F., “Prospective on Rotorcraft Aerodynamic Modeling for Flight Dynamics Applications,” American Helicopter Society 4th Decennial Specialist’s Conference on Aeromechanics, San Francisco, CA, January 21–23, 2004.