Using Flight Simulation Environments with Agent-Controlled UAVs

Ricardo Gimenes   Daniel Castro Silva   Luís Paulo Reis   Eugénio Oliveira

Abstract— **Developed countries have made significant efforts to integrate UAV operations in controlled aerial space due to a rising interest of using UAVs for civilian and military purposes. This paper focuses on looking for reliable solutions and a way to validate an autonomous multiagent control system using Flight Simulators. This study has two main lines, the first being the use of multiagent systems in UAVs (or other robotic platforms), aiming at a fully autonomous control. The second is a survey about the variety of simulation systems dedicated to aerodynamics and aircrafts, comparing them and their feasibility to validate the multiagent control system. A comparative study of existing simulation environments is presented, both commercial simulation game engines and research simulators. One critical factor is hazard situations, like emergency landing or equipment failure, which should be predicted in an automated system. Most flight simulators are not realistic enough to validate a multiagent algorithm in hazard situations. At the same time, it is impossible to predict every type of failure in real world. The boundaries of simulation should be very well enclosured in order to present results using simulation. A multiagent control system architecture that makes use of flight simulators is also presented.**

I. INTRODUCTION

Ever since the construction of the first vehicles, men have dreamed of automating its operations. This determination is visible in commercial cars, with automatic transmissions, automatic parking capabilities, and many other automated systems; in airplanes, the foremost visible feature that demonstrates this is the autopilot capability; in other vehicles, one or other automated systems are always present (as automated landing in British Heathrow airport under zero visibility, defined as Category IIIc [1]), making it easier to maneuver them. While this autonomy has been restricted to limited functions, it is desirable to elevate it to a full scale, allowing vehicles to be fully autonomous. When full autonomy is reached, vehicles will have to cooperate and coordinate their actions with one another, in order to ensure both security and an optimal use of resources.

The choice of an agent-oriented approach to control autonomous vehicles is intended to save both time and resources. With remotely operated vehicles, an operator needs to operate the vehicles at all times, thus only eliminating the danger factor in missions that take place in hazardous locations. Even with autonomous vehicles, capable of autonomous maneuvering and mission execution, those missions must be planned prior to their execution, thus using resources and time. Sometimes, these missions require real-time adjustments to be made, which also implies the use of human resources during mission time. By using an agent-oriented approach, the vehicles would be able to communicate amongst themselves, and create their own mission planning and real-time decisions based on information gathered during mission execution.

In order to better develop and test such agent-oriented approaches and coordination methods, a simulation environment must be used. Such environment also saves time and resources, since it would be prohibitory to test these systems using real vehicles.

This paper focuses on several existing simulation environments that can be adapted to serve the objective at hand. The main focus is given to aerial and flight simulation environments, since it is the most difficult environment to simulate, due to all constraints involved in simulating a fluid atmosphere, like air or water, and the way vehicles move through it. The cost and accessibility of flight simulation environments are criteria which consider commercial off-the-shelf (COTS) and low cost software solutions.

A. Unmanned Aerial Vehicles

UAV, acronym for Unmanned Aerial Vehicle, is usually associated with a group of aerial elements in constant development that still need better definition. This paper considers UAVs as vehicles able to fly autonomously without any external remote control. The acronym UVS (Unmanned Aircraft System) is becoming commonly used by the scientific community in order to represent more necessary elements instead of just the aircraft and flight aspects.

UAVs may be described and categorized according to many criteria, such as flight altitude, endurance and size, among others. There are efforts currently being made by Standards Development Organizations (SDOs) to create industry-wide standards for the design, manufacture, testing, and operation of UAVs [2].

There are already some examples of civilian UAV usage, either operating or in developmental stage, namely those related to surveillance or in need of access to risky areas, such as nuclear facilities, power transmission line maintenance and others [3]. These do not have significant influence on the air traffic system, since the UAVs fly over clearly defined areas at low altitude and speed.

The use of UAVs still involves a few uncertainties and legal limitations as to where and how they can be used. The majority of projects involving UAVs are strictly military. An important example of an UAV in use is the military aircraft GlobalHawk. It is a high-altitude, long-endurance UAV developed by the United States Department of Defense to undertake reconnaissance missions. This aircraft can fly autonomously for 36 hours at altitudes up to 65,000 feet (nearly twice the height used by commercial airliners) [4].

Ricardo Gimenes is with the Polytechnic School - University of São Paulo, Av. Prof. Luciano Gualberto, trav. 3, n. 158, CEP 05508-010, São Paulo, Brazil `ricardo.gimenes@poli.usp.br`

Daniel Castro Silva, Luís Paulo Reis and Eugénio Oliveira are with FEUP - DEI / LIACC, Rua Dr. Roberto Frias, s/n 4200-465, Porto, Portugal `dcs@fe.up.pt, lpreis@fe.up.pt, eco@fe.up.pt`

One can see that categories still cling to a military heritage, for the military are still involved in most major UAV development. The main military feature is avoiding human crews at all, in spite of severe critics against armed military operation, such as what criteria should an algorithm assess in order to open fire [5].

But, in order to the aeronautics industry to integrate civilian UAV usage to the current system, a variety of issues must be dealt with: technical (aircraft control technology), political (border control issues) and legal (liabilities in case of accidents).

Opposing to the military vision, crewless civilian passenger aircraft rise a completely diverse paradigm in terms of reliability, compared to military usage. Because of this, new hurdles come in the way of a future categorization of passenger carrying UAV, as well as of non-passenger carrying UAV sharing airspace with conventional passenger carrying aircrafts.

Thus, operating UAV on international airspace must entail two closely related conditions [6]: the UAV must be safe and reliable enough to fly over densely populated areas and must be safely operated through airspace. Although both requirements are defined generically, they should account for the widest possible range of unsafe situations both for the airplane itself and its surrounding environments.

From such general safety requirements, arise research subjects such as risks on collision to the ground and midair collision. Both relate primordially to integrating UAV operation to the current system, which demand appropriate rules that enforce fulfillment of those requisites. In this paper we assume that UAVs are fit for use in the traffic system, and building on this hypothesis present ground collision and midair collision related safety issues.

B. Multiagent Systems

A multiagent system can be seen as a system where entities are represented by independent agents, which in turn communicate with each other, coordinating their activities. The area has known a good expansion in the last decades, and it is now possible to develop systems that use an agent-oriented approach in several programming languages, using several available libraries and standard protocols.

The UAV operation can surely be considered a multiagent system in which every aircraft is an agent with its own goals (destination, time frame of arrival, service standards, etc) and is independent of the goals of any other aircraft. With this approach, it is possible to apply negotiation techniques that have been developed for such systems. These techniques will allow the aircraft to cooperate in solving airspace conflicts and, by doing so, it will also be able to efficiently solve its own conflicts.

Answers involving artificial intelligence software or multi-agent systems have been researched on, in the search for decision-making systems capable of avoiding aircraft encounters in a given airspace [7].

In section two of this paper, some of the requirements that were taken into account when evaluating the simulation environments are briefly described. Section three presents some of the analyzed simulators, game engines, flight simulators and middleware engines. In the last section, some conclusions are withdrawn and future work is outlined.

II. Simulation Environment Requirements

The choice of a simulation environment is dependent on the objectives of the project. In this section, some key features that may be desirable in the simulation environment are briefly described. These features can be grouped into categories, such as graphical, physical and openness characteristics, among others, for a better analysis.

Regarding the graphical characteristics, one can always wish for the most detailed and visually appealing interface, using the most recent graphical capabilities, such as DirectX10, but one also has to keep in mind that this is not a priority, since it is not a game that is being developed.

The openness of the software is an important aspect. Openness can be defined in terms of the possibility to view and alter the source code, if desirable, as well as expansion capabilities, such as the possibility to develop add-ons, external modules and different agents that can be easily linked to the environment, via defined APIs, interaction models or communication protocols, the accessibility and format of the data that can be input to the environment, and the output data format as well.

Accurate physical simulation is very important, when dealing with robotic mobile agents, such as UAVs. The following subsections explain in more detail the importance of a correct physical modeling and simulation of the aircraft and the environment.

A. Specific Aviation-Related Requirements in Simulated Environments

Realistic simulation of robotic sensors and actuators in a complex, unstructured, dynamic environment, such as fluids, constitutes a research challenge. Most simulators consider only the four basic vectors that compose a flight: lift, weight, drag and thrust. Drag and lift only exist when there is movement through a fluid, like air. In order to maintain an aircraft in a straight level flight, the Lift is equal to Weight and Thrust is equal to Drag. [8].

A basic simulation of these four elements is possible for aircrafts like a small radio-controlled helicopter. To simulate its flight, we just have to define how to implement their values and directions. However, in real flight, an aircraft has to deal with numerous factors, not only pertaining to the aircraft itself, but also external, environmental factors. An important element not commonly found in flight simulators is the relation of the rudder with the lift in each wing. For example, if the rudder is forced to its left limit, the right wing will completely loose its lift but the right wing will maintain its lift. This case already happened in a flight accident, where a Boeing 737 lost its control resulting from the movement of the rudder surface to its limit [9].

This way, to choose a flight simulator system, it is very important to define what is more relevant in the research. Each simulator has its characteristics suitable (or not) with a specific kind of research. Considering developing an autonomous multiagent environment as the main goal, the relevant parameters in the flight system can have three major lines: Acceptable flight model, considering the application under investigation; Flexibility to interact with the agent via programming interfaces; Possibility to have the flight model or simulated elements changed by external software. This is an important factor, considering the research many times has specific necessities like a simulated failure of a unique simulated element.

Agent algorithms which have to control an aircraft will need a flight simulator model as real as possible. Flight elements like lift variation over the wings, mass centre, angle of attack are just a few necessary in an acceptable flight model. The weather is another complex element to simulate. It implies that the simulator has to consider common but not well know factors like wind shear, turbulence, microbursts (of wind) and variations of density, pressure and temperature (with severe variations inside clouds). As the number of simulated elements offered by the flight simulator increases, so does its adequacy to serve as a suitable tool for simulated validation of the agent-based UAVs.

B. Fault Injection in Flight Simulation Models

The firm safety requirements related with aviation require the studies of any kind of flight simulation to keep in mind failure considerations. Fault injection is a complex research line to reach a reliable and safe flight model in flight simulation.

A fault injection method is essentially hardware fault injection. In a flight simulation, faults can be injected into different inputs or outputs of the aircraft.

Considering the UAV and aircraft completely controlled by software, the range of failure possibilities determines the way the faults have to be injected.

The quantity of injected faults is a quality parameter of any fault injection method. Many flight simulations are connected with a visual interface, and the time to simulate it is based in real time. This simulation limits the quantity of forced fault injections and the way the flight model is validated. Most fault injection software validates the system under evaluation with a huge quantity of different fault injections.

Basically, the fault injection module is a software module which interrupts the original inputs and outputs of the simulated aircraft. The corrupted inputs our outputs can be generated between the aircraft control (agent) and the flight simulator or the aircraft model and flight simulator, as can be seen in Fig. 1.

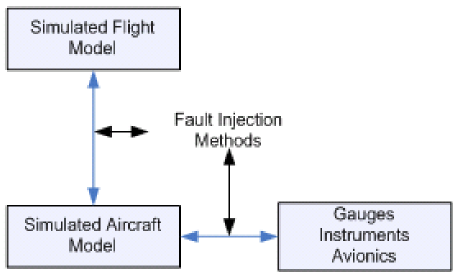

Fig. 1. Fault Injection Interactions.

A typical fault injection validates a small piece of software, simulating corrupted inputs (simulated hardware faults) and automatically evaluating the software output. It is important to notice that fault injection in a flight simulation is a hardware fault injection over a simulated hardware. For example, it is possible to simulate that a fault in the fuel pumps (hardware fault injection) implies an engine failure (hardware failure). The precision of how this engine failure will be simulated determines the quality of the flight simulator, since there is a response time between the simulated physical elements.

The fault injection to evaluate the automatic flight control is a hardware fault injection over software and the reaction time from the software is less relevant. An example is an altimeter fault and the flight control has to react with fault tolerant methodologies to maintain the aircraft under safety conditions.

A critical problem for fault injection using COTS flight simulators is how they are opened to interact with their simulated hardware of an aircraft. Thus, open source flight simulators or open API from proprietary software should be analyzed as decisive factors in a flight environments choice.

C. UAV Flight Simulation

The way the flight simulation is developed determines its possibilities and, consequentially, the possibilities and limitations the research has to deal with.

Real-time visual simulation is not necessarily a relevant parameter. Simulations without visual interface offer the possibility to produce a quantity of different situations quicker than the visual simulation does. If the log file of any simulated flight is consistently produced, it is possible to visualize it after the simulation.

The simulation of physics, kinematics and aerodynamics is fundamental in a reliable flight simulation. In contrast, structural forces are not relevant in UAV flight model studies.

III. COMMON SIMULATION ENVIRONMENTS

There are two main simulator categories: Game Engines and Flight Simulators.

In game engines, the most important aspect is an appealing visualization. However, this can be a problem when, in a scientific research, the reliability of the simulation is the main goal. For example, in UTEngine2 there is no drag variation control to force different situations in a flight [10].

Flight Simulators have a different approach. The main focus is the flight simulation itself. There is some focus on the visual aspect as well, but the main efforts are dedicated to aerodynamics and flight factors present in real world. The main concern is to simulate a realistic flight, which is not necessarily as entertaining as a game.

There are Air Traffic Control (ATC) Simulators too. These simulate ATC in order to coordinate different flights at the same time. The use of this kind of simulation is a good approach in order to simulate multi-agent controlled UAVs under a controlled airspace.

A. Game Engines

First-person games offer a possibility to researchers play, test, and evaluate their robots with a different range of sensors. The 3D game engine development explores the best in computer hardware and software for itself. Then, an accessible 3D environment with physics and kinematics opens new robotic research perspectives. Conversely, when aerodynamics is the main focus, game engines normally do not attend the requirements. Precise behaviour of control systems, as found in flight simulators, is not usually part of the worlds simulated by game engines [11].

Although there are many different commercial game engines, the high-concept graphics of Everquest's LithTech engine, Quake III Arena and Unreal Tournament provide the most developed, flexible, and usable engines for research purposes [11].

Research in robotic coordination requires common environments for simulating sensor data and movement responses. The RoboCup offers this researchers integration with the urban search and rescue Simulation (USARSim) robot competition. It is a RoboCup league to increase awareness of the challenges involved in search and rescue applications with a high fidelity robot simulator built over a state of the art commercial game engine [12]. However, the focus of USARSim and the game engines differs from flight simulation and, therefore, are not suitable to validate or test flying mutiagent systems in a convincing simulation.

B. *COTS Flight Simulators*

There is a great variety of COTS Flight Simulators, which implies a large number of possibilities to select the best choice from. The classification of COTS includes open source versions, even though their complexity sometimes implies the impracticability to understand how they work. The most common Flight Simulators present in scientific researches are Flight Gear, X-Plane and Microsoft Flight Simulator. Another important but least known is Piccolo Flight Simulator (and automatic flight control).

The graphics in modern flight simulators are very realistic. However scientific research has to worry about the flight dynamics of an aircraft. Therefore, an evaluation on a few low cost COTS flight simulators is presented below.

1) Microsoft Flight Simulator: In graphical terms, the current version of Microsoft Flight Simulator (Flight Simulator X, or FSX for short) is probably the most realistic on the market, in part thanks to the use of the new DirectX 10 technology.

The Microsoft Flight Simulator flight model is based on a set of tables, which determine how the aircraft will behave under certain conditions. This kind of simulation is defined as behavioral simulation or parametric approach. The flight model is supplied by a set of parameters, and is independent of the visual model [13][14].

The failure-modeling includes equipment failure, but without variation of the manner in which the equipment will fail.

FSX presents the SimConnect API, an FSUIPC-like access to functions and variables, which allows developers to create new add-ons to add or replace functionalities, as well as monitor activities [15].

2) X-Plane: X-Plane uses the geometric approach (aircraft structure and other engineering parameters) and its model is defined as structural. It determines the flight dynamics directly from the visual model. An engineering process called ”blade element theory” calculates the flight model, which involves breaking the aircraft down into many small elements and then finding the forces on each little element many times per second. These forces are then converted into accelerations which are then integrated to velocities and positions [16][14][17].

X-Plane also has failure-modeling, with 35 systems that can be forced to fail. Fails can be injected into instruments, engines, flight controls, and landing gear at any moment, but as in Microsoft Flight Simulator, it is not allowed to vary the way the equipments fail [18].

X-Plane allows the use of one's own flight model. This flexibility offers the possibility to develop a dedicated flight model due to match particularities. So, in this case, it is just used as a visual representation of the external simulated flight.

A factor of creditability about X-Plane is its professional use and approval by the FAA for training towards airline transport certificate. In contrast, studies of real flight behavior in comparison of flight simulators have shown concept flaws in X-Plane as in Microsoft Flight Simulator. In [14], Stock compared then according to the following aspects: the elevator, the ailerons, the rudder, the throttle, flaps, stall, and spin. Its results show how the low cost COTS flight simulators have serious impressions in their models which can compromise the use of simulator to substitute or even to validate models in the real world.

3) FlightGear: The FlightGear flight simulator project is a free, open-source, multi-platform, cooperative flight simulator development project. Source code for the entire project is available and licensed under the GNU General Public License. The goal of the FlightGear project is to create a sophisticated flight simulator framework for use in research or academic environments [19].

FlightGear allows the users to choose between three primary Flight Dynamics Models. More than this, it is possible to add new dynamics models or even interface to external flight dynamics models source, such as from Matlab. The three available models are briefly described below.

JSBSim is a generic flight dynamics model for simulating the motion of flight vehicles based in lookup tables, as FSX does. It is written in C++, using XML configuration files [20]. JSBSim can be run in a standalone mode for batch runs, or it can be the driver for a larger simulation program that includes a visual subsystem (such as FlightGear).

YASim is an integrated part of FlightGear and simulates the effect of the airflow on the different parts of an aircraft. As a similar approach of X-Plane, it is possible to perform the simulation based on geometry and mass information, combined with more commonly available performance numbers for an aircraft.

UIUC is lookup table modeling based on LaRCsim originally written by NASA [21]. UIUC extends the code by allowing aircraft configuration files instead and by adding code for simulation of aircraft under icing conditions.

The use of FlightGear in research becomes common in the scientific community once it allows users and aircraft designers access to a very large number of internal state variables via numerous internal and external access mechanisms. It allows to remotely control FlightGear from an external script and to use an external flight dynamics module (including hardware autopilot) [22].

4) Piccolo: Piccolo is a known auto-pilot system for small aircraft, by CloudCap Technology [23]. The main product is commercialized as hardware components that can be mounted on a small aircraft, allowing autonomous flight. There is also a software release of a simulation environment, for software-in-the-loop tests. The graphical component of flight simulation is minimal, only presenting the user with a previously loaded aerial picture and aircraft positional indicators. However, in terms of flight simulation it is very realistic, simulating many of the forces involved in a flight. It also presents an application interface which allows input and output interaction. By default, Piccolo is able to output its data to Microsoft Flight Simulator and Fight Gear, two of the previously analyzed flight simulators, which act as a visualization software.

C. Dedicated Middleware Engines Related with Flight Simulation

There are hundreds of middleware engines which could be used in scientific research. But, considering the UAV scenario controlled by multi-agents algorithms, two are highlighted in this paper.

1) OpenFlight: OpenFlight, MultiGen-Paradigm's native 3D content, is the leading visual database standard in the world and has become the standard format in the visual simulation industry. It is a very common format to store sceneries in, for professional flight simulators [24]. OpenFlight has a logical, hierarchical scene description file format which informs the realtime image generator what, when, and how to render, resulting in realtime 3D scenes with unmatched precision and reliability. Developers can explore its flexibility, open connectivity and easy interoperability [25].

2) AeroSim: AeroSim is a Matlab block library which provides components for development of nonlinear (with six degrees of freedom) aircraft dynamic models. As most flight simulators, it includes environment models such as standard atmosphere, background wind, turbulence, and Earth Models. These models are implemented in such a manner that they can easily be replaced with models based upon analyses of the sensors to be used on the actual aircraft. It is a low cost solution, considering it is free of charge for academic and non-commercial users [26] [18]. The graphical flight visualisation can be made through a flight simulator like FlightGear or X-Plane. It offers more flexibility in flight modelling and fault injection [18]. A general purpose graphical interface to Aerosim is the OpenSceneGraph. The OpenSceneGraph is an OpenSource platform for the development of high performance graphics applications. It is a 3D graphics library for C++ programmers [27].

3) MATRIXx: MATRIXx is a National Instruments Design and Development Tool to build accurate, high-fidelity models and perform interactive simulations for control design applications in aerospace and defense. It allows to analyze system models and build robust control algorithms, generate readable, traceable, highly optimized C or Ada code and target the code to a real-time platform for control prototyping and hardware-in-the-loop test. The MATRIXx suite of software includes four main modules: SystemBuild - a graphical environment for rapid model development and simulation. Xmath - is a mathematical analysis, visualization, and scripting software environment featuring a unique, high-level, object-oriented programming language called MathScript. DocumentIt - provides a customizable method to generate formatted documentation of the SystemBuild models. AutoCode - automatic embedded code generation for C and Ada AutoCode is an customizable automatic code generation tool. It can generate code using AutoCode Add-On Modules to generate C or Ada for fixed-point processors or multiprocessor target systems.

The FlightGear Flight Simulator, when utilized in conjunction with National Instruments MATRIXx allows for direct visual feedback of aircraft flight dynamic model simulation results in real-time. This connectivity is provided via a user code block (UCB) in MATRIXx.

This FlightGear user code block (UCB) supports an interface via a unidirectional transmission from MATRIXx to FlightGear using FlightGear's published binary data exchange specification. Data is transmitted via UDP to a running instance of FlightGear.

IV. CONCLUSIONS AND FUTURE WORK

Since the choice of an appropriate environment depends on specific needs, no single environment is elected as the best, but the choices of a suitable environment have been defined concerning specific characteristics chosen according to the objective in sight.

Some conclusions can be withdrawn from the analysis made to the simulating environments in the previous section, where characteristics like simulation of graphical presentation, kinematics, physical interpretation of the object, weather simulation and simulation cycle method were considered. In terms of software openness, each presents its advantages, the new version of the Microsoft Flight Simulator series presenting for the first time an open API, and other products with their long-time known interfaces. One can also see that the visual module of the environment does not need to be the same as the simulation modules, since data can be exported from one to another, which makes the selection of the most appropriate one easier, since one does not need to limit oneself to only one environment. Also, and when a robust architecture is used, one can also have the option of using a different environment for specific purposes, thus making applications more flexible. This flexibility allows researchers to choose the combination of simulator modules that best suits their interests, thus being able to focus on the development or improvement of either the graphical content, the airplanes' model, or, in this particular case, the creation of automated aircraft able to communicate with one another and coordinate actions.

A variety of software able to simulate dynamics and graphical interfaces challenges the researchers in order to choose the best solution to their investigations.

V. ACKNOWLEDGMENTS

We wish to acknowledge our institution, LIACC, for providing us with all the working conditions necessary for the execution of our work.

References

[1] Helios, "Assessment of the technical, regulatory and socioeconomic constraints and feasibility of the implementation of more spectrally efficient radiocommunications techniques and technology within the aeronautical and maritime communities," Tech. Rep., June 2004, available online at http://www.helios-tech.co.uk.

[2] "Uavforum. uav standards." 2007, available online at http://www.uavforum.com/.

[3] "Civilian applications: the challenges facing the uav industry," *Air & Space Europe*, vol. 1, no. 5, pp. 63–66(4), September 1999. [Online]. Available: http://www.ingentaconnect.com/content/els/12900958/1999/00000001/00000005/art88873

[4] J. Hafesjee, Poster, March 2002, available online at http://www.nist.gov/public_affairs/Posters/ globalhawk.htm.

[5] J. Sullivan, "Revolution or evolution? the rise of the uavs," *Technology and Society, 2005. Weapons and Wires: Prevention and Safety in a Time of Fear. ISTAS 2005. Proceedings. 2005 International Symposium on*, pp. 94–101, 8-10 June 2005.

[6] M. Allouche, "The integration of uavs in airspace," *Air & Space Europe - Operations and Safety*, vol. 2, pp. 101–104(4), January 2000. [Online]. Available: http://www.ingentaconnect.com/content/els/12900958/2000/00000002/00000001/art80019

[7] Z. Lian and A. Deshmukh, "Performance prediction of an unmanned airborne vehicle multi-agent system," *European Journal of Operational Research*, vol. 127, no. 2, pp. 680–695, July 2006, available online at http://ideas.repec.org/a/eee/ejores/v172y2006i2p680-695.html.

[8] B. Schiff, *Flying, A Golden Science Guide*, 1978.

[9] "Uncontrolled descent and collision with terrain," National Transportation Safety Board, Aircraft Accident Report NTSB AAR-99/01, September 1999, available online at http://www.ntsb.gov/publictn/1999/AAR9901.htm.

[10] "Physx physics, gameplay, and the physics processing unit," White Paper, March 2005.

[11] M. Lewis and J. Jacobson, "Game engines in scientific research - introduction," *Communications of the ACM*, vol. 45, no. 1, pp. 27–31, 2002. [Online]. Available: citeseer.ist.psu.edu/lewis02game.html

[12] "Robocuprescue simulation league. competition overview," Web Page, 2007, available online at http://wiki.cc.gatech.edu/robocup/index.php/ RoboCupRescue_Simulation_League.

[13] T. Goodrick, "Flight dynamics for microsoft flight simulator," 2000, available online at http://www.flightsimdownloads.com/pub/FlightDynamics.pdf.

[14] C. Stock, "Flight dynamics: Fsx and x-plane battle it out," Internet-based Magazine, 2007, available online at http://www.simpilotnet.com/index.php?option=com_content&task=view&id=20&Itemid=9.

[15] "Simconnect sdk reference," Web Page, 2006, available online at http://www.fs-seine-75.com/SDK/Core%20Utilities%20Kit/SimConnect%20SDK/ SimConnect.htm.

[16] "X-plane flight simulator," Web Page, 2007, available online at http://www.x-plane.com.

[17] M. Zoccola, "Dr. morgan reflects on propeller history," July 1998, available online at http://www.dt.navy.mil/pao/excerpts

[18] I. A. McManus, D. G. Greer, and R. A. Walker, "Uav avionics 'hardware in the loop' simulator," in *10th Australian International Aerospace Congress, Proceedings of, Brisbane, Queensland, Australia*, 2003.

[19] "Flightgear official site," Web Page, 2007, available online at http://www.flightgear.org/.

[20] "Jsbsim, the open source flight dynamics model in c++," Web Page, 2007, available online at http://jsbsim.sourceforge.net/.

[21] E. B. Jackson, "Manual for a workstation-based generic flight simulation program (larcsim), version 1.4," NASA, Tech. Rep. NASA-95-tm110164, 1995.

[22] "Pcs in flight simulation research the lca (navy) experience," May 2001.

[23] "Piccolo system software," Web Page, 2007, available online at http://www.cloudcaptech.com/resources_autopilots.shtm.

[24] "Openflight official site," Web Page, 2007, available online at http://www.multigen-paradigm.com/products/standards/openflight/index.shtml.

[25] A. Gerretsen, "Openflight - flight simulator mvp & scenery designer," Web Page, December 2005, available online at http://msmvps.com/blogs/arnogerretsen/archive/2005/12/13/ openflight.aspx.

[26] "Aerosim blockset," 2007, available online at http://www.u-dynamics.com/aerosim/default.htm.

[27] "Openscenegraph official site," Web Page, 2007, available online at www.openscenegraph.org.