Faculdade de Engenharia da Universidade do Porto

Multi-agent Platform for the Development of Cooperation Methodologies Using Heterogeneous Vehicles

”Camada de Comunicação em Simulador de Aviação”

Ricardo Nuno Nóbrega Silva

A dissertation submitted in partial fulfillment of Master’s degree on

Electronics and Computers Engineering

Telecommunications Major

Orientation: Luís Paulo Reis (PhD)
Co-orientation: Daniel Castro Silva (MsC)

July 31, 2008

U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

MIEEC - MESTRADO INTEGRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES 2007/2008

A Dissertação intitulada

"Camada de Comunicação em Simulador de Aviação"

foi aprovada em provas realizadas 18/Julho/2008

o júri

Presidente Professor Doutor António Augusto de Sousa
Professor Associado da Faculdade de Engenharia da Universidade do Porto

Professor Doutor Fernando Augusto Cruz e Silva Mouta
Professor Coordenador do Instituto Superior de Engenharia do Porto

Professor Doutor Luis Paulo Gonçalves dos Reis
Professor Auxiliar da Faculdade de Engenharia da Universidade do Porto

O autor declara que a presente dissertação (ou relatório de projecto) é da sua exclusiva autoria e foi escrita sem qualquer apoio externo não explicitamente autorizado. Os resultados, ideias, parágrafos, ou outros extractos tomados de ou inspirados em trabalhos de outros autores, e demais referências bibliográficas usadas, são correctamente citados.

Autor - Ricardo Nuno Nobrega Silva

Faculdade de Engenharia da Universidade do Porto

Abstract

This document describes the initial stage of development of a multi-agent platform, which will serve as a platform for the development and test of cooperation methodologies using heterogeneous vehicles. A study of possible simulation platforms was made being determined that flight simulators constituted the strongest candidates for this project. For each of the chosen simulators, interfacing external applications were developed, which allow the control of the simulated vehicles and the simulator's internal variable manipulation. From this study emerged Flight Simulator X, which, from other features, includes a mission environment SDK, which can be used for task or mission creation, and SimConnect, a tool that can be used for vehicle control and event processing. Other types of vehicles are available across the web or can be created using the SDK modeling tools. An application was developed as well as a simple mission to demonstrate the possibilities and the performance of the platform. This application also allows the test of the performance of the basic (heading, altitude and speed) and high-level (go to a point, follow a circular or helicoidal path) vehicle control functions. At this early stage, agent autonomy, as far as navigation decisions are concerned, is still somewhat limited, since the main focus was basic vehicle control and interaction. The developed platform offers promising perspectives for the completion of the system, with some issues still to resolve, such as collision detection and mission planning, ground and water vehicle performance and network related limitations, such as the number of agents allowed in the system.

Resumo

Este documento descreve a etapa inicial de desenvolvimento de uma plataforma de multi-agente, que servirá como uma plataforma para o desenvolvimento e teste de metodologias de cooperação usando veículos heterogéneos. Um estudo de possíveis plataformas de simulação foi feito sendo determinado que os simuladores de voo seriam os candidatos mais fortes para este projecto. Para cada um dos simuladores escolhidos, foram desenvolvidas aplicações que através das interfaces disponibilizadas permitem o controle dos veículos simulados e a manipulação das variáveis internas. Deste estudo emergiu o Flight Simulator X, que, entre outras características, inclui um *SDK* que permite o desenvolvimento de missões e que poderá ser usado para a criação de tarefas ou missões, e o SimConnect, uma ferramenta que pode ser usada para controle de veículos e processamento de eventos internos do simulador. Outros tipos de veículos estão disponíveis através da *internet* ou poderão ser criados usando o *SDK* de modelação de veículos. Uma aplicação foi desenvolvida assim como uma missão simples com o objectivo de demonstrar as possibilidades e o desempenho da plataforma. Esta aplicação também permite o teste do desempenho das funções de controle dos veículos: básicas (direcção, altitude e velocidade) e de alto nível (vai para um ponto, segue uma trajectória circular ou helicoidal). Nesta primeira etapa, a autonomia dos agentes, ainda é algo limitada em termos de decisões de navegação, já que o objectivo principal para esta fase era o controle básico do veículo e interação entre os agentes. A plataforma desenvolvida promete boas perspectivas para a conclusão do sistema, com alguns problemas ainda por resolver, como a detecção de colisões e planeamento de missões, desempenho dos veículos terrestres e marítimos e algumas limitações provenientes da arquitectura adoptada, como a performance em redes congestionadas e número máximo de agentes permitidos no sistema.

Acknowledgments

I would like to thank my parents, Vasco and Fátima, brother Bruno, and girlfriend Styliani, for supporting me in my educational pursuits. I would also like to thank the advisory committee, Luis Paulo Reis and Daniel Castro Silva, and the staff and colleagues at LIACC/NIAD&R for the help in the development of this project, the department of electronics engineering DEEC, and all the teachers and colleagues at MIEEC/FEUP.

Contents

List of Figures

List of Tables

Acronyms

List of acronyms

API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
ATC	Air Traffic Control
FAA	Federal Aviation Administration (US)
FIPA	Foundation for Intelligent Physical Agents
GPS	Global Positioning System
IDE	Integrated Development Environment
KQML	Knowledge Query and Manipulation Language
MMOG	Massive Multiplayer Online Game
NASA	National Aeronautics and Space Administration (US)
PID	Proportional-integral-derivative (controller)
SDK	Software Development Kit
TCP	Transmission Control Protocol
UAV	Unmanned Aerial Vehicle
UDP	User Datagram Protocol
XML	Extensible Markup Language

Chapter 1

Introduction

This chapter's main purpose is for the reader to obtain insight about the main subjects focused by this project and to understand the motivation and objectives of this document. In the first section the project context and background will be discussed, the following subsections are introductory texts (recommended for readers that are unfamiliar with such subjects) of four distinct fields: Artificial Intelligence, regarding basic concepts about agents and multi-agent systems, Controllers and autopilots, the general method of vehicle control applied in simulation platforms and its implementation, Great Circle Navigation, formulae used for navigation on earth (assumed spherical), to calculate distances and headings and for agent communication the well established FIPA Contract Net protocol will be used. Finally the last two sections describe the project motivation and objectives.

1.1 Project background

This project is inserted in the context of another project for the development of methodologies for agent cooperation in mission operations using air, ground and sea vehicles. The platform must possess sufficient complexity either in it's world, types of vehicles and the quantity of the internal system and their complexity. A study of possible platforms was made being determined that flight simulators would be the best solution in favor of available game engines (such as MMOG's). The defining reasons include:

- Available interfacing SDK and documentation: in terms of the development language and available examples or samples on available market products
- Simulation data and sensor availability: which simulation data is available (vehicle systems, environment variables, other objects, etc.)
- Vehicle models: are the vehicle models customizable or the model can be driven externally

- Customizable 'world' (weather conditions, traffic, etc.): is it possible to change weather conditions (wind, rain, snow, ice, temperatures, gravitational pull, etc.)
- Failure injection and correction: quantity of systems that allow failures and consequently correction (engine, gear, structure, etc.)
- Interaction with other objects: the platform allows the vehicles to interact with scenery, moving or transportable objects

Flight simulators have emerged steadily since the first PC's games allowing the growth of huge communities of flight enthusiasts, pilots and air traffic controllers. A study of the available simulators followed as the initiating step of this project, as the consequence three flight simulators were selected as good candidates: FlightGear, Flight Simulator X and X-Plane.

1.1.1 Artificial intelligence

There are two major dimensions used as a definition of artificial intelligence nowadays: human vs ideal, in the sense that the agents should model human behavior or should be conceived using an ideal concept, and thought vs action, in that the agents should be more action prone or possess problem solving abilities. As will be seen promptly, a generally accepted definition of intelligence is one that states that an agent is able of executing rational action, that is, an intelligent agent takes the best possible action using the processing of the perceived information [1].

> "An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives"
>
> *[Wooldridge and Jennings].*

There isn't an universally accepted definition for agent, different domains yield different importance for the attributes associated with agency.

An agent (Figure 1.1) can be described as a system that can perceive information from it's 'surrounding environment', process this information on a not necessarily intelligent manner an act on it using it's acting abilities.

Intelligent agents are agents that comply with the following capabilities:

- **Reactivity**: the agent must be able to to interact with the environment, must be able to perceive or act with the environment in order to complete its goal

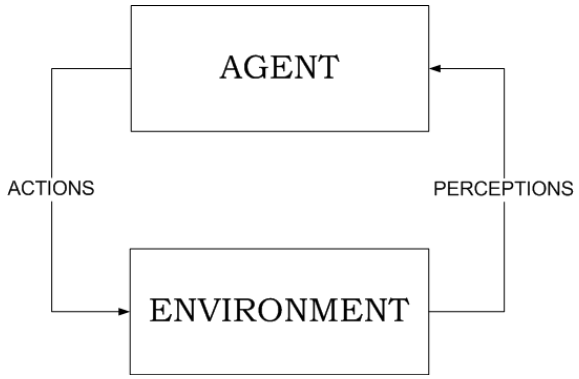

Figure 1.1: Abstract agent definition

- **Pro activeness**: the agent must be able to initiate action using decision making algorithms
- **Social ability**: the agent must be able to interact with other systems or agents

A multi agent system (Figure 1.2) is a a system where two or more computational entities called agents which can interact with one another, but that don't necessarily possess information that other agents possess about the environment they are in.

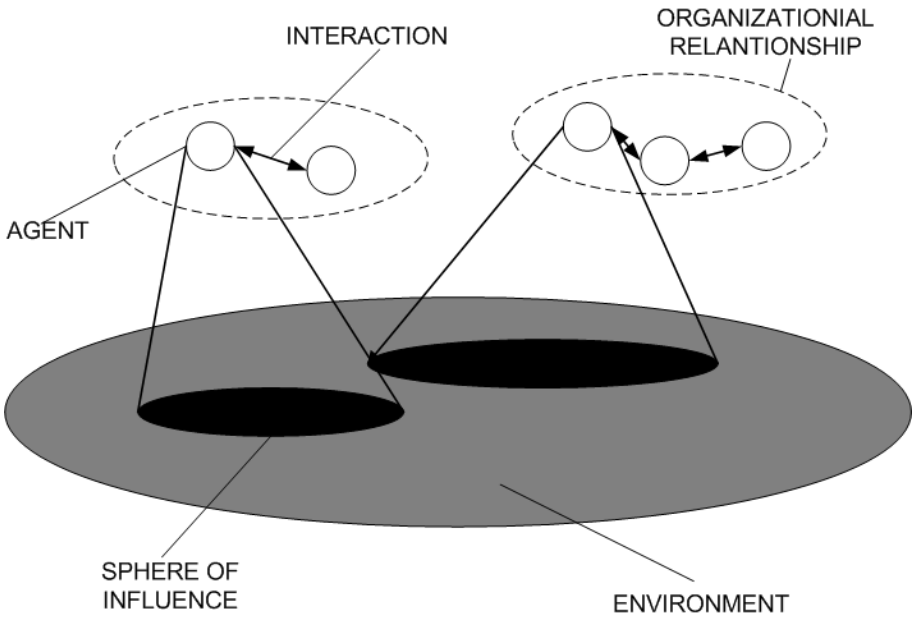

Figure 1.2: Multiagent system structure - *adapted from* [2]

For this project the agents must be able to communicate amongst them selfs in order to coordinate their actions.

1.1.2 Controllers and autopilots

An autopilot is a mechanical, electrical, or hydraulic system used to guide a vehicle autonomously. Most people associate autopilots specifically to aircraft, but there are autopilots for boats, cars and even motorcycles. Modern day flight can be divided into: taxi, take-off, ascent, level flight, descent, approach and landing. All phases except taxiing are or can be controlled via autopilot. Autopilots today are generally control software running on dedicated hardware which obtain data from the Inertial Navigation System (INS), Global Positioning System (GPS) and via radio using a Distance Measuring System (DME) commonly known as transponder.

One of the most powerful tools to emerge from process control early days, and that is still used in almost all control systems, is the proportional-integral-derivative (PID) controller (Figure 1.3).

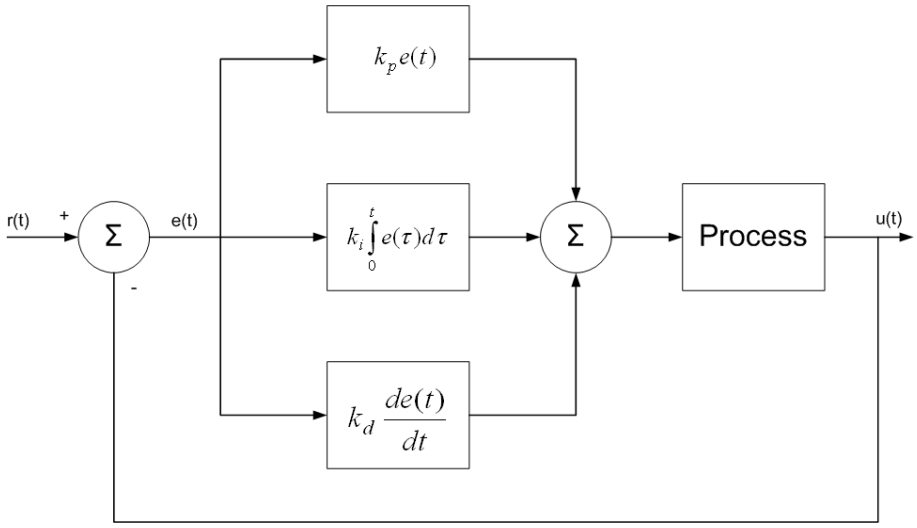

Figure 1.3: Elemental PID loop

It's numeric implementation consists on the evaluation of the equation 1.1

$$u(t) = k_p e(t) + k_i \int e(\tau)d\tau + k_d \frac{de(t)}{dt}, \tag{1.1}$$

where *e(t)* is the difference between the reference value and the feedback measured value and k_p, k_i and k_d are the proportional, integral and derivative gain, respectively. The proportional term adjusts the output signal in direct proportion of the error, the integral term compensates the offset between the reference and the measured value and the derivative term measures the approximate rate of change of the controlled process.

Loop Tunning

Loop tunning methods are used for selecting the adequate values of k_p, k_i and k_d, one of the most used closed loop methods is the Ziegler-Nichols method. The method consists in removing the integral and derivative terms, then create a small disturbance and adjust the proportional gain until the output oscillations have constant amplitude. The period (T_c) of the oscillations and the value of the proportional gain (k_c) can then be used the determine the PID parameters using the table 1.1.

Controller type	k_p	k_i	k_d
P	$0{,}5k_c$	-	-
PI	$0{,}45k_c$	$0{,}83T_c$	-
PID	$0{,}59k_c$	$0{,}5T_c$	$0{,}125T_c$

Table 1.1: Ziegler-Nichols closed loop tunning rules

A method which normally produces unsatisfactory results, as a tunning method by itsefl, is the manual method of tunning. However, this method can be used at a later stage for fine tunning the control loop, using table 1.2.

Parameter	**Rise time**	**Overshoot**	**Setting time**	**Steady state error**
k_p	Decrease	Increase	Small change	Decrease
k_i	Decrease	Increase	Increase	Eliminate
k_d	Small Decrease	Decrease	Decrease	None

Table 1.2: Effects of increasing k_p k_i and k_d *from* [3]

1.1.3 Great Circle Navigation

A great circle [4] is the resulting circumference from the intersection of a sphere with a plane that passes through its center(c). The resulting arc (d) from the intersection of a great circle with two points (A and B) on the surface of the sphere defines the shortest path between those points. Where d can be given by,

$$d = acos(sin(lat1) \times sin(lat2) + cos(lat1) \times cos(lat2) \times cos(lon2 - lon1)), \tag{1.2}$$

and A=(lat1,long1) and B=(lat2,long2) with d,lat1,long1,lat2 and long2 in radians. To obtain d in km, we have to multiply it by the earth's radius, and,

$$d_{km} = 6378 \times d, \tag{1.3}$$

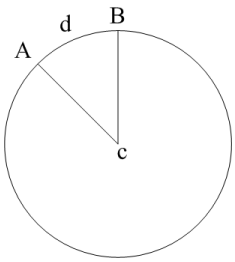

Figure 1.4: Great circle

To obtain the true course(tc) form A to B, we can use, for $sin(lon1 - lon2) < 0$

$$tc = acos((sin(lat2) - sin(lat1) \times cos(d))/(sin(d) \times cos(lat1))), \tag{1.4}$$

and for other values,

$$tc = 2 \times \pi - acos((sin(lat2) - sin(lat1) \times cos(d))/(sin(d) \times cos(lat1))), \tag{1.5}$$

with tc, lat1, lat2 and d in radians. To obtain tc in degrees,

$$tc_{degress} = tc \times 180/\pi. \tag{1.6}$$

1.1.4 FIPA Contract Net Interaction Protocol

FIPA (Foundation for Intelligent Physical Agents) is a non-profit association dedicated to the development of specifications for agent interaction. The FIPA contract net protocol [5] adds rejection and confirmation communicative acts to the original contract net protocol. The protocol 1.5 consists in: the initiator sends a call for proposals, pending a deadline, with some information regarding the task execution focusing on the conditions that have to be met by the participants (commonly designated as the 'price' of the task), the participants must either submit a proposal, presenting their acceptance conditions (namely the 'bid') or refuse the task. After evaluating the proposals, the initiator selects the proposal or proposals which satisfy the initial conditions and sends an acceptance message. The initiator waits for the result of the task execution. All of the messages contain an unique conversation ID that is used for the agent to manage its communication strategies. At any point of the communication the agents may send a not understood message or cancel message. The cancel message must be confirmed by the receiver with a inform or failure message.

FIPA performatives:

Call for proposals (*cfp*) is used to specify the task and conditions that have to be met by the participants

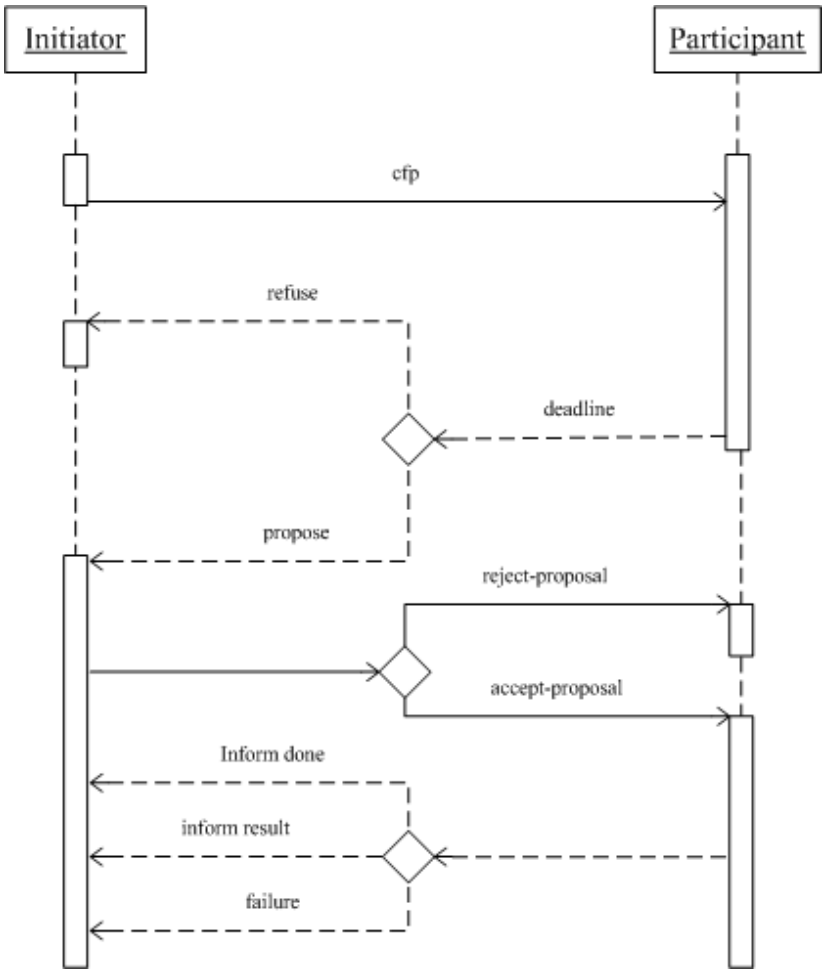

Figure 1.5: FIPA Interaction Protocol

```
(cfp
:sender 'name of the sender'
:receiver 'name of the receiver'
:content 'content'
:ontology 'message context'
:conversation-id 'conversation identifier'
:language 'message language')
```

Propose (*propose*) is used by the participants to indicate their conditions to perform the task

```
(propose
:sender 'name of the sender'
:receiver 'name of the receiver'
:content 'content'
:ontology 'message context'
:in-reply-to 'interaction identifier'
```

```
:conversation-id 'conversation identifier'
:language 'message language')
```

Refuse (*refuse*) is used by the participant to refuse a proposal

```
(refuse
:sender 'name of the sender'
:receiver 'name of the receiver'
:content 'content'
:conversation-id 'conversation identifier'
:language 'message language')
```

Accept proposal (*accept-proposal*) is used to accept the proposals of the participants

```
(accept-proposal
:sender 'name of the sender'
:receiver 'name of the receiver'
:in-reply-to 'interaction identifier'
:content 'content'
:conversation-id 'conversation identifier'
:language 'message language')
```

Refuse proposal (*reject-proposal*) is used to reject the proposals of the participants

```
(reject-proposal
:sender 'name of the sender'
:receiver 'name of the receiver'
:content 'content'
:conversation-id 'conversation identifier'
:in-reply-to 'interaction identifier')
```

Failure (*failure*) indicates that the could not complete the task or performative

```
(failure
:sender 'name of the sender'
:receiver 'name of the receiver'
:content 'content'
:conversation-id 'conversation identifier'
:language 'message language')
```

Inform (*inform*) is used by the participants to inform the initiator that the task was complete and information about the success of the completion of the task

```
(inform
:sender 'name of the sender'
```

```
:receiver 'name of the receiver'
:content 'content'
:conversation-id 'conversation identifier'
:language 'message language')
```

1.2 Motivation

The motivation for this project emerged from the need to have a standardized API that will serve as a platform for the development of vehicle based multi agent systems, using a realistic platform (in term of the complexity of the simulation 'world' and vehicles). This standardized API will then allow the developers to overlook the development of time consuming dedicated visualization systems, providing an easy and transparent way of controlling the vehicles. It can be also be used by UAV developers as a test platform, visualization system or in hardware in the loop solutions.

1.3 Objectives

The first step will be to compare the available simulators, exploring the interfacing protocols determining whether it is possible to control the vehicles from an external application and how, and the ability to control other types of vehicles. The next step will be to implement the basic vehicle control and some standard movement related actions, following the implementation of the communications protocol to allow cooperation between the agents. Finally, a simple application will be developed for testing purposes.

1.4 Problem description

In the development of multi agent interaction methodologies it is important to have a supporting platform that provides a realistic simulation environment. For this project the methodologies involve cooperating vehicles that will be acting in joint missions, from rescue missions to military scenarios. The objective will be to develop or adapt a platform that is able to meet these requirements. A communication layer 1.6 that will allow the control of the vehicles will be developed. The layer consists of a structured API, that will be available from an external application and execute the actions in a transparent fashion.

The first step is to learn how to send and extract information, to and from the platform, regarding vehicle control, autopilot control, environment and vehicle action, this information will then allow the development of the basic functions concerning the vehicle control, such as moving from point to point, taking off, landing, circle movements, vehicle intersection and general acting functions concerning the vehicle and 'world' available sensors and object interaction. Some of the market's

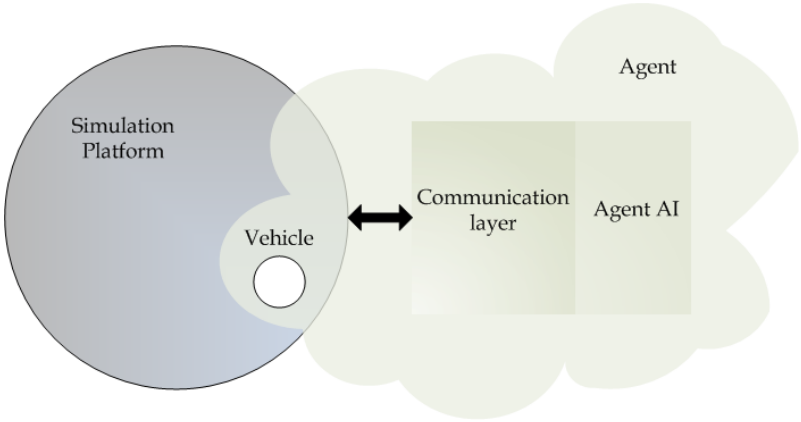

Figure 1.6: Logic communication layer architecture

available flight simulators allow some form of exchanging data with an external application, generally this is used for creating hardware-the-loop simulators [6]. One of the major challenges will be to adapt the platform to other types of vehicles obtaining an acceptable level of performance, for example in the case of terrestrial vehicles will they be able to interact with the scenery, other vehicles and objects with the same level of realism? Will it be possible to have underwater or even shuttle/rocket vehicles? What are the possible actions, besides movement, that the vehicles can perform? How many vehicles (agents) will the platform allow to simulate simultaneously? In the next chapters we will try to answer most of these challenges and demonstrate the capability of the selected platform.

1.5 Document structure

We will start by making a study of the preselected platforms and it's major interfacing features and comparison between them to determine which of the simulators, Flight Gear, Flight Simulator or X-Plane, is best suited for the project. In the project development chapter, the control of the vehicles will be developed and the communication between them. In the test analysis chapter we will present a simple example of cooperation between agents including different vehicle types. On the last chapter we will present the conclusions and possible improvements and future work.

1.6 Conclusion

In this chapter a general introduction of the project's main subjects was presented, the use of this subjects will be made clear in the following chapters. A broader discussion of the project was discussed focusing on the background, motivation objectives and the description of the problem itself. The structure of the document was presented to give the reader a scope of the following chapters.

Chapter 2

Flight simulators

In this chapter will present the study of the interfacing capabilities of the preselected flight simulators: FlightGear v1.0, Flight Simulator X - Deluxe Edition and X-Plane 8.64. In order to demonstrate this capabilities small application examples will be developed, with focus on vehicle control and associated variables.

2.1 X-Plane

X-Plane is a commercial flight simulator developed by Laminar Research. It is available for Windows, Linux and Mac. X-Plane's developer Austin Meyer was a senior simulation programmer for major airlines, producing dedicated simulators for airline pilots training. A version of the game was approved by FAA for initiate pilots training.

X-Plane's comes with a wide variety of aircraft, from classical historical models to the Space Shuttle and even conceptual airplanes. Other types of vehicles can be obtained from third party developers [7], including ground vehicles and boats. Contains world-wide scenery from -60 to 74 degrees latitude with a huge possibility of landing sites including 18.000 airports, aircraft carriers, helipads and oil rigs, including also a flyable and realistic (in terms of gravitational pull, topography, atmosphere pressure, density and temperature [1]) scenery for the planet Mars. These features benefit from the simulator flight model which is based on the 'blade element theory' (in a few words, consists on breaking the plane's geometric shape into small elemental pieces and obtain the elemental actuating forces which will then be used to determine how the aircraft will fly) thus granting the model the ability to handle all kinds of airfoils and plane shapes. It is possible to manually or randomly fail 35 different systems such as instrumentation, flight controls, engines and landing gear.

[1] Data retrieved from numerous NASA websites

The weather in X-Plane is fully controllable and there the possibility to download real-time weather from the internet. You can create your own planes, instrument panels, scenery, airfoil, textures and sound effects.

There are two possible options (refer to appendix A) for interfacing X-Plane: using an UDP socket protocol or using the plugin SDK. From flight control to program menus and object placement, it is possible to build applications that allow you to read or write any variable of the property tree.

An application that uses the UDP socket protocol is X-Plage [8] which enables to create a moving map, or, a display on Goggle Earth of the plane's position in the world, that is then used like a GPS device. Other applications include a tactile controller [9] or flight data recorders. Some of the most important applications of the simulator are the Fidelity Flight Simulation Inc. [10] simulation devices used mainly for pilot training. Another important project is the Cirrus 'The Jet' [11] on which X-Plane was used for the airplane demonstration. Companies that use the simulator include: NASA, Boeing, Lockheed-Martin, Carter Copter and Scaled composites.

2.1.1 Example application

The application was developed using the SDK C++ headers available at [12], consequently the test application was developed using C++ programming language and Visual C++ 6.0 IDE.

Development

To extract/send data from/to X-Plane using the plugin SDK, the exchanged data needs to be attained by using the SDK API and creating a protocol to exchange that data with the outside applications (Fig. 2.1). The SimData example (available on the SDKExamples.zip file) was used as a template to create the plugin.

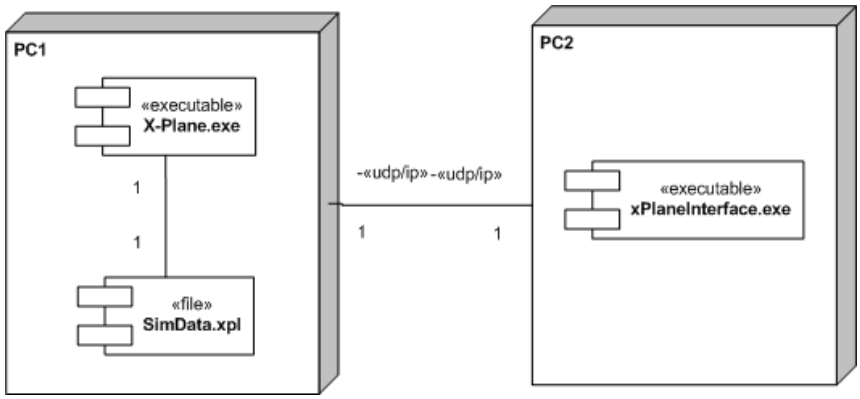

Figure 2.1: X-Plane UML 2.0 deployment diagram

The development consisted of two distinct stages, the first stage was the adaptation of the SimData sample in order to exchange the required variables, namely, throttle, gear, ignition, x position, y position, z position, barometer, wind speed, temperature, yolk roll and yolk pitch. The second stage was the development of the UDP protocol socket communication between the plugin and the the interface application. The test application allows the basic plane control or the visualization

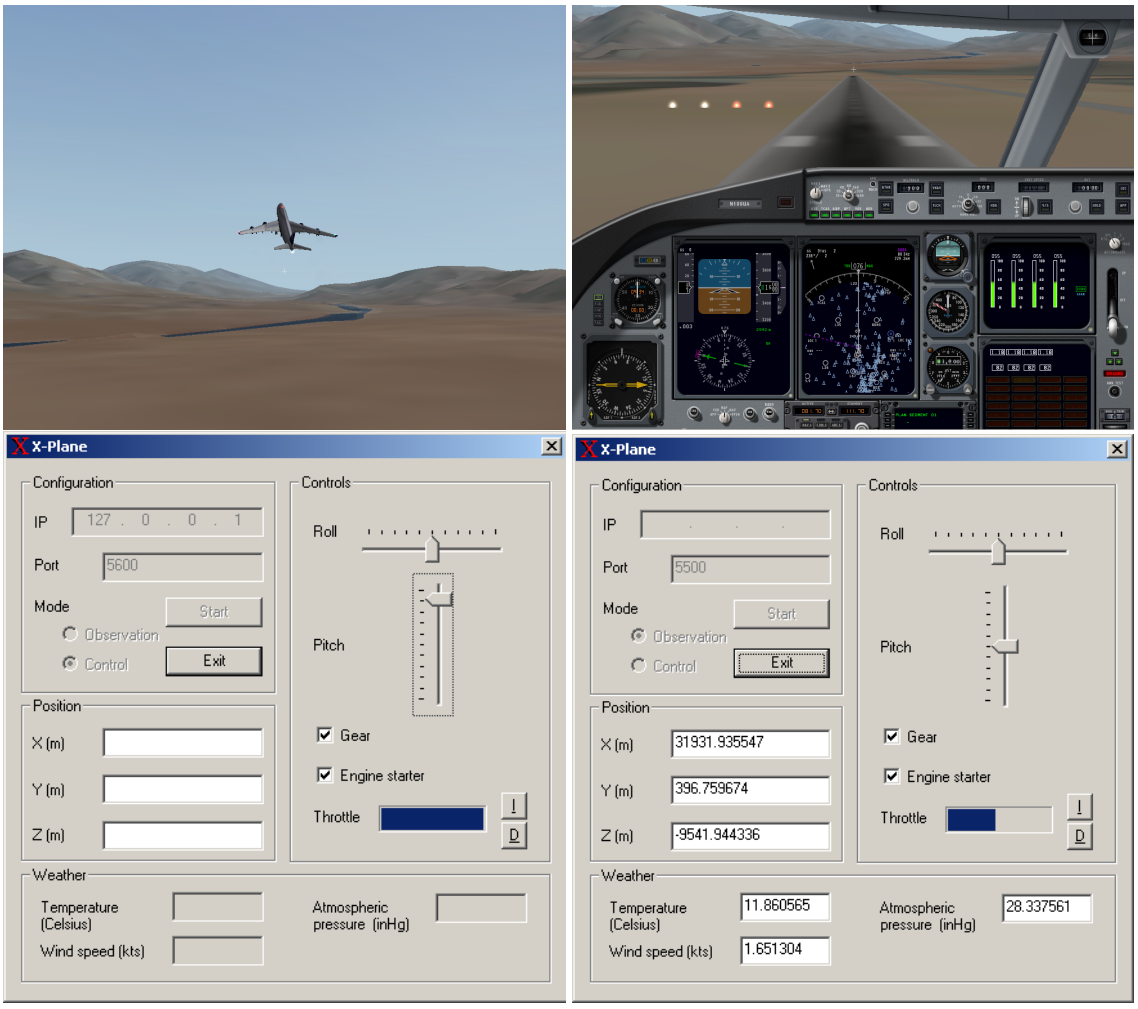

Figure 2.2: X-Plane test application

(Fig. 2.2) of some flight and environment related variables. The use of the UDP protocol allows the application to be used remotely or locally.

Testing

The applications allows the user to control the vehicle, position the vehicle, control the engine and gear. Also allows the reading of the variables mentioned before. The temperature, wind speed and pressure variables cannot be altered because of internal models updating.

2.1.2 Conclusion

The first mode of interfacing X-Plane is very unstable with inaccurate and disperse support documentation (the web page used to support this protocol is no longer active), on the other hand the plugin SDK is very well documented, it is supported with examples and allows the use your own protocol for exchanging data with an external application. Also includes functions for accessing other plane, camera views and AI manipulation.

2.2 FlightGear

FlightGear [13] is an open-source multiplatform flight simulator designed mainly for academic purposes as a research support platform. It is available for Windows, Linux, Mac OS-X, FreeBSD, Solaris, IRIX. Source code, world wide scenery and third party vehicles are available from the project's webpage. This simulator allows you to communicate with the outside using various media, such as files, serial port, UDP or TCP socket and pipes. It is also possible to do this using different protocols using predefined ASCII messages, network format (or Big-Endian) or even binary strings, XML or using scripted languages like Perl, Nasal.

Four of the most important protocols are the Generic, Native, Telnet and HTTP (refer to appendix B). The Generic protocol was initially developed for an individual access to the simulator's internal variables, it is mostly used for flight recording and playback. Examples of application include the serial port model control [14], where actions taken on the simulator are replicated by the external model. The native protocol was developed to allow the communication between two or more instances of FlightGear. One the numerous applications of this protocol is the Piccolo system [15] is a complete avionics control system for small UAV, this systems uses the native protocol for a hardware-in-the-loop or software-in-the-loop simulation which uses FlightGear as a visualization tool for flight and mission testing. The telnet protocol is also used for individual internal variable access, as is seen on the remote control example [16].

2.2.1 Example applications

Three applications were developed for testing the: Generic, Native and Telnet protocols. The purpose was to test the control of the vehicles and internal properties variable reading from an external application, these applications were developed using C++ and Visual C++ 6.0 IDE.

The applications allow the manual control of the vehicle and the reading (Fig. 2.3) of several internal variables (altitude, latitude, longitude, pitch, roll, wind speed, temperature, etc.). The applications can be executed locally or remotely. There were also created batch files to initiate the simulator with a configuration suited for every protocol: protocol, data configuration and sample rate, ip, port, flight model (on or off).

2.2.2 Conclusion

The first two protocols (Generic and Native) are more adequate for visualization applications where the flight model data is sent into the simulator. Although it is possible to run the FlightGear model JsbSim [17], this is not the best solution for this project.

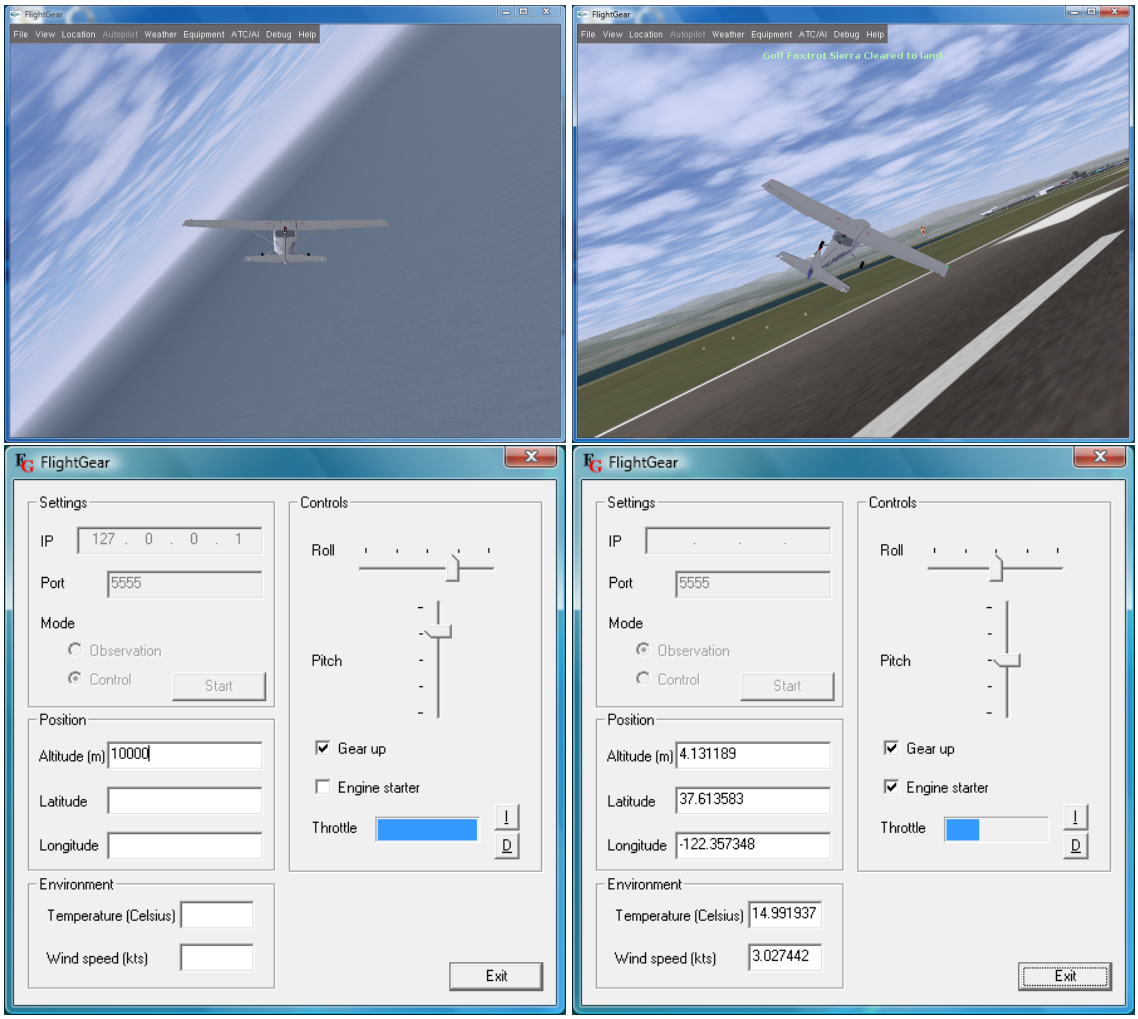

Figure 2.3: FlightGear test application

The Telnet protocol permits the external application to interface FlightGear with its internal flight model active, however, some internal system variables cannot be overridden from the outside (temperature, wind speed,etc...).

2.3 Flight Simulator X

Flight Simulator X [18] is the latest version of the flight simulator series developed by Microsoft. There are two available versions of Flight Simulator X, the Standard Edition and the Deluxe Edition, the latter with a SDK for almost every feature of the simulator. This SDK tools include: the programmers interface SDK (SimConnect), camera configuration, terrain, scenery, modeling, aircraft and boat traffic, special effects, mission, aircraft and other vehicle, panel and gauges creation, auxiliary tools, documentation [19] and samples. Furthermore, there is a feature extending expansion pack - Flight Simulator Acceleration Pack, that, among other features, allows the possibility for multiplayer missions. The evaluated version, in this document, will be Flight Simulator X - Deluxe Edition + Acceleration Pack.

2.3.1 SimConnect SDK

The SimConnect SDK enables programmers to write add-on components using any of the languages present in the .NET platform. There is also available a third party Java implementation of SimConnect client library (jSimConnect [20]). SimConnect uses a client-server architecture (Fig. 2.4) via Named Pipe, TCP/IP WinSock IPv4 and IPv6, in an asynchronous mode. The client sends requests to the server, which then processes them and sends the answer back. It is also possible for clients to communicate between each other. This mode of communication provides more safety in multi-threaded or multi-processor applications, uses process cycling more efficiently and provides greater server stability on the other hand if the client add-on is to work in a closed-loop configuration, such as autopilots, latency as to be taken into consideration.

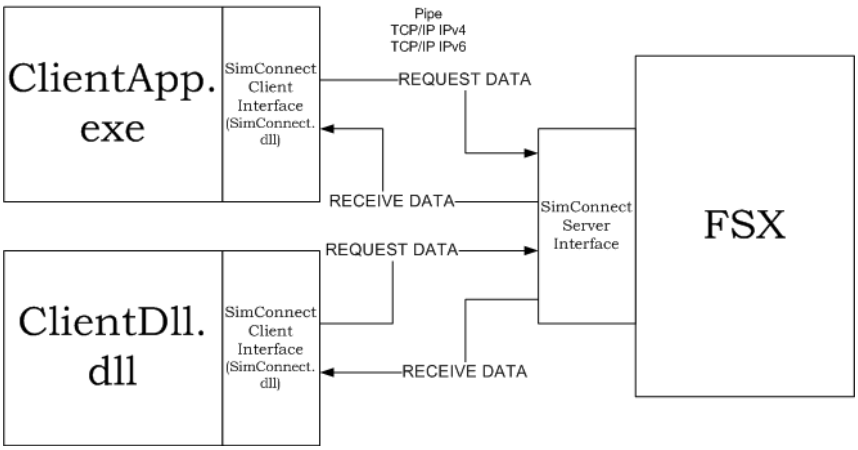

Figure 2.4: SimConnect client-server architecture

The documentation suggests the use of out-of-process add-ons, or applications instead of in-process or .dll. Another interfacing mode is the Gauge system, this system is used for developing

gauges and panels for interactive use in the cockpit and is able to access the same internal variables as the simconnect SDK. In alternative to the data request it is also possible to send or receive events, such as key presses and generally events generated by the simulator.

2.3.2 The mission SDK

Mission creation consists on the editing of XML files or using the Object Placement Tool, there also a third party block diagram tool available [21]. The Object Placement Tool (OPT) is an add-on XML based tool that helps the creation of missions for Flight Simulator X. The missions can be from simple way point following to complex search and rescue operations and pylon racing. Almost every aspect regarding the simulation can altered in the OPT, scenario object placing, AI vehicle control, special effects (fire, smoke, etc...), to help designers obtain the desired performance. A complete description of all the OPT development possibilities is beyond the scope of this document, although, most mission development can be divided, roughly, on two major properties: actions and triggers. Basically triggers initiate actions, and that is how you can control the mission. Some of the most important mission actions include: attaching droppable objects, attaching effects, counters, causing system failures, object activation, random and custom actions. The object activation action enables the activation of a number of objects(triggers, AI vehicles and scenery objects). The custom action allows the communication with SimConnect. The most used triggers are: airport and area landing, counter, proximity, property and timer. The property trigger will fire if a specified condition involving some simulation variables (altitude, vehicle speed, component failure, etc...) is met. Again the SDK documentation is very descriptive and includes examples and sample missions.

2.3.3 Example application

The application will be developed using the managed API for C# using Visual Studio 2005. The first step in developing an application for interfacing SimConnect is adding the references to the SimConnect library Microsoft.FlightSimulator.SimConnect.dll. The first reference allows you to use the managed API methods, and the second allows you to interoperate with unmanaged code, namely, C++.

```
using Microsoft.FlightSimulator.SimConnect;
using System.Runtime.InteropServices;
```

The second step is to create a simconnect object and create the user-defined win32 event which will be used to identify which system message is being sent by SimConnect.

```
SimConnect simconnect;
const int WM_USER_SIMCONNECT = 0x0402;
...
simconnect = new SimConnect("<Program name>", this.Handle, WM_USER_SIM
CONNECT, null, 0);
```

This is basic procedure for making an application able to communicate with Flight Simulator. There are generally two options for exchanging data: data request and events. For data requests the general procedure, is to: define the data, register the data with the simconnect managed wrapper marshaller and catch it using a event handler.

```
...
//define the data
simconnect.AddToDataDefinition('data structure enumeration', "'variable
 name'", 'unit name', 'data type', 0.0f, SimConnect.SIMCONNECT_UNUSED);
...
//register the data
simconnect.RegisterDataDefineStruct<'struct name'>('data structure enum
eration');
...
//request the data
simconnect.RequestDataOnSimObjectType('request enumeration', 'data stru
cture enumeration', 'radius', 'object type');
...
//catch the data
simconnect.OnRecvSimobjectDataBytype += new SimConnect.RecvSimobjectDat
aBytypeEventHandler(simconnect_OnRecvSimobjectDataBytype);
...
void simconnect_OnRecvSimobjectDataBytype(SimConnect sender, SIMCONNECT
_RECV_SIMOBJECT_DATA_BYTYPE data)
{
switch (('request enumeration')data.dwRequestID)
  {
case 'request enumeration'.'request name':
...
```

The procedure to receive data, is somewhat analogous, in that it is necessary to define and register the data.

```
simconnect.SetDataOnSimObject('data structure enumeration', 'object id'
,0, 'data structure');
```

For sending events, it is necessary to map, or, associate a client defined event id to a flight simulator event name, and then transmit the event with the corresponding value.

```
simconnect.MapClientEventToSimEvent('client defined event id enumerati-
on', "'flight simulator event name'");
simconnect.TransmitClientEvent('object ID', 'client defined event id e-
numeration', 'value to be set', 'priority', 0);
```

To receive events, it is necessary to subscribe the event, or, tell the simulator that the client shall receive notification for that event. The procedure to catch the event is similar to the request data handler, except that the handler is specific for the type of event subscribed.

```
//for a system event
simconnect.SubscribeToSystemEvent('event id enumeration', "'name of the
 event'");
```

An application was developed in order to evaluate the SimConnect managed API interface. The application allows the user to control the vehicle from externally as well as to read relevant internal variables, namely: Title (of the controlled vehicle), Plane Latitude, Plane Longitude, Plane Altitude, Ambient Temperature, Ambient Wind Velocity, Ambient Pressure, Elevator Position, Aileron Position, Gear Position and Throttle. The Managed Data Request example, available in the SimConnect documentation, was used as a template. Other applications of SimConnect in-

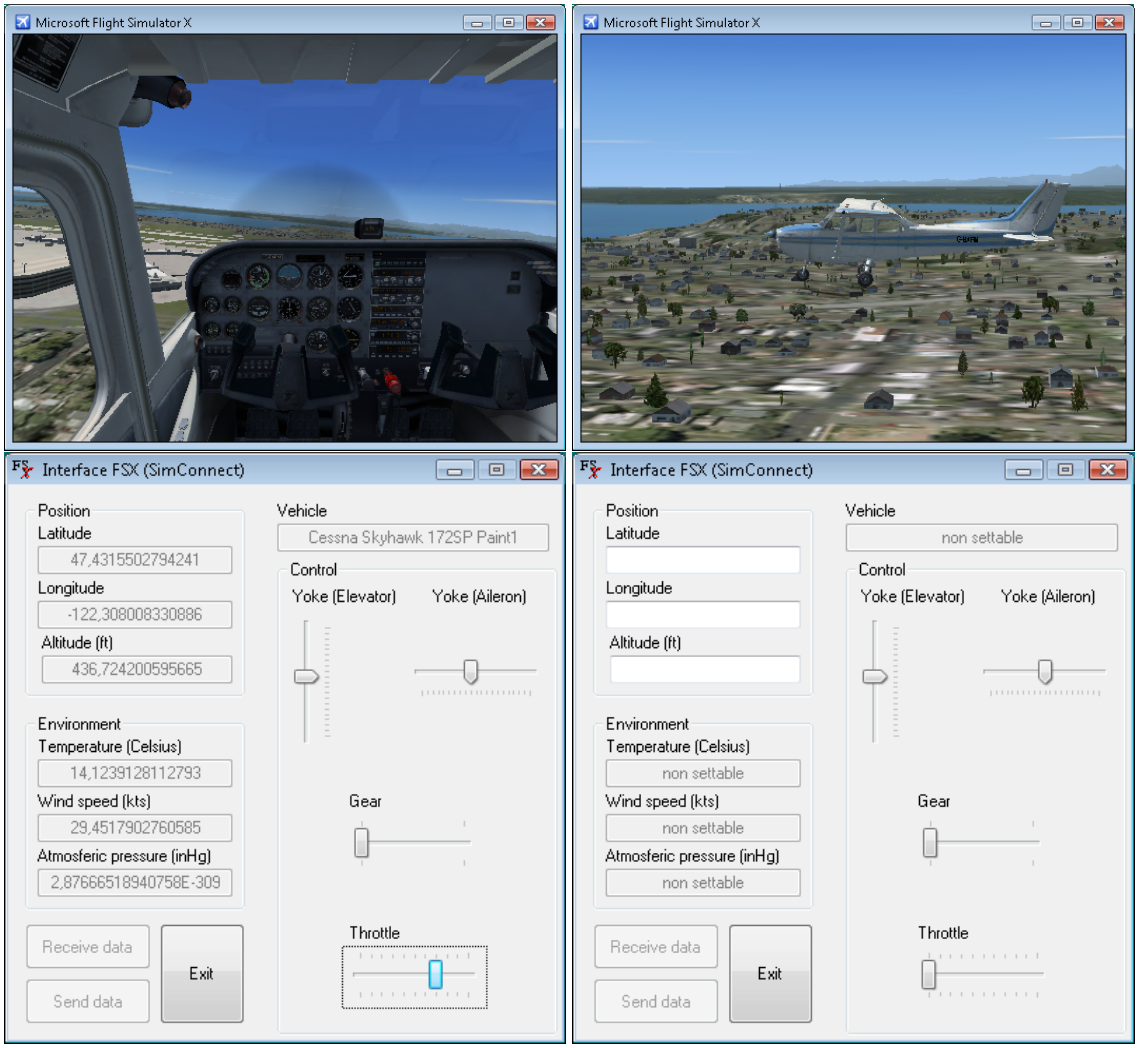

Figure 2.5: Flight Simulator X test application

clude the Google Earth Tracker [22], which is an application that shows the user plane and AI planes on Google Earth [23] along with course prediction and flight plans, most of the addons provide experience enhancement like customized scenarios, panels and gauges, aircraft and other

vehicles, virtual online airliners [24], customized missions, special effects, live traffic [25]. Available addons include other types of user controlled vehicles such as cars, motorcycles, trucks, boats, cruisers, balloons , blimps and conceptual vehicles.

2.3.4 Conclusions

The Flight Simulator SDK is a powerful tool complemented by a complete and structured documentation. The SimConnect allows access to most of the simulator's internal properties, other than the request system the event system allows through key events (keyboard press emulation) complete control of the vehicle as if you were using the simulator itself.

2.4 Simulator comparison

These features were attained through developers documentation, magazines, journals, testing and ultimately through user experience. Generally all of the simulators allow the control of the vehicle from an external application, with X-Plane and Flight Simulator X also providing the possibility to control AI planes. Flight Simulator SDK has the advantages of being able to process simulation events through the SimConnect SDK and the ability to create task or missions through the mission development environment using the Object Placement Tool. FlightGear is the only one who provides access to the source code and manipulate the multiplayer protocol. X-Plane is the only simulator who does not provide interfacing support from the product developer. The following table (Table 2.1) displays some of the most important features of the studied simulators.

	FlightGear	**Flight Simulator X**	**X-Plane**
SDK languages	mainly C++	all .Net	mainly C++
SDK documentation (quantity)	low	very high	low
SDK documentation (quality)	low	good	medium
Number of accessible internal variables	very high	very high	very high
Vehicle types (user controlled)	medium	very high	high
Other vehicle and object control	low	very good	medium
Other customization options	medium	very high	low
Mission creation SDK	not available	good	not available
Failure injection	high	high	high
Product cost (€)	free	115	38
Available samples/examples	very low	high	medium

Table 2.1: Simulator comparison table

The chosen simulator is Flight Simulator X Deluxe Edition + Acceleration Pack, the quality of the SDK documentation exceeds by far that of the other simulators, with a complete description of the API and application examples and samples. Also, the mission development tool allows an interaction vehicle-world that suits perfectly the needs of this project.

2.5 Conclusion

In this chapter a study of the preselected simulators was made. The study consisted on evaluating the interfacing SDK and protocols, as well as some of the major features of each of the simulators. From this study the solution Flight Simulator X Deluxe Edition + Acceleration Pack was selected as the simulation platform to be used in the development of the project.

Chapter 3

Project development

In this chapter we will discuss the project development, starting with the discussion of the possible architectures, vehicle control and communication possibilities.

3.1 Architecture

As seen before we need to develop an agent platform were the agents can interact with each other. There are two possibilities using Flight Simulator and SimConnect: an AI vehicle based architecture or using the game's multiplayer system.

3.1.1 AI plane based architecture

The AI vehicle based architecture which can be seen on Fig 3.1, basically consist of a SimConnect server running on Flight Simulator were the controlled vehicles are AI vehicles and the user vehicle. AI vehicles are non ATC controlled vehicles that have their own flight model and systems, but that are not as complete as the user vehicle. The SimConnect clients can run on the local machine or remotely. Limitations to this architecture include a simulated area that is a 64 Km square, being that any AI vehicle falling off of this area will stop being simulated. Also since SimConnect is asynchronous, in the case of multiple vehicles, the requests can result in a high system degradation through latency. It is possible to use these vehicles in the mission environment, although with some limitations, this as to do with trigger firing, because the mission triggers will fire for AI planes but don't have the ability to distinguish between them. The communication between the agents can be done using SimConnect by creating a client data (See the SimConnect SDK documentation - CreateClientData method) area on the server which can be read and written by the other clients. AI vehicle control is provided by SimConnect since: speed, heading, waypoint following and flight plans.

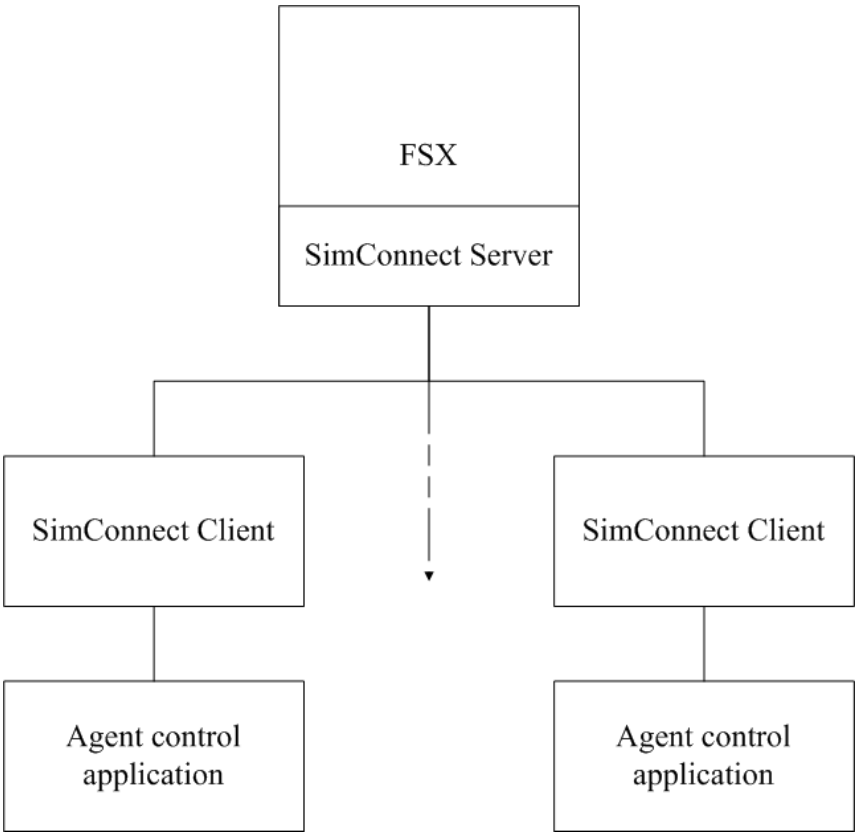

Figure 3.1: AI based architecture

3.1.2 Multiplayer based architecture

The multiplayer architecture Fig. 3.2 uses the game multiplayer system, and each agent application has its own SimConnect client which connects to the respective server, thus eliminating the undesired latency. The multiplayer system has a physical limitation of 99 players, although a recommended value of 32 players exits. This system also provides an easier integration with the mission system. The communication system has to be designed externally (Tcp socket) because the flight simulator multiplayer protocol is not accessible. For the control of the vehicles it is necessary to develop new controllers or use the internal autopilot system. The second architecture provides better possibilities, in term of stability and scalability, and integration with the mission environment. The development of the this architecture will be divided into three phases: vehicle control, mission development and agent communication protocol. And the API functions will be developed in C# under Windows Vista.

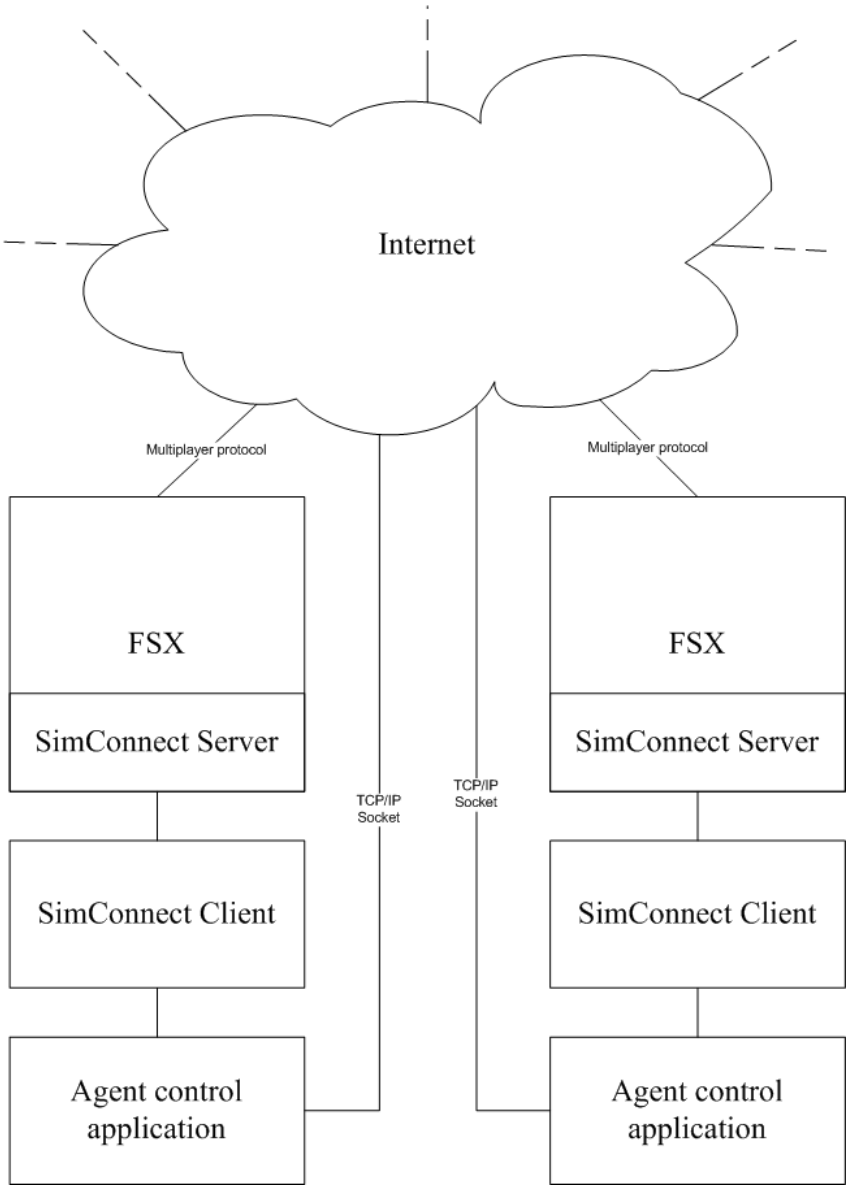

Figure 3.2: Distributed architecture

3.2 Vehicle control

Two types of vehicle control were developed: one based on the simulator's internal autopilot and one PID based external control.

3.2.1 External PID-based control

The following controls were developed as an alternative to the autopilot-based control.

PID heading control A heading controller (Fig. 3.3) was developed using the vehicle aileron as the actuator.

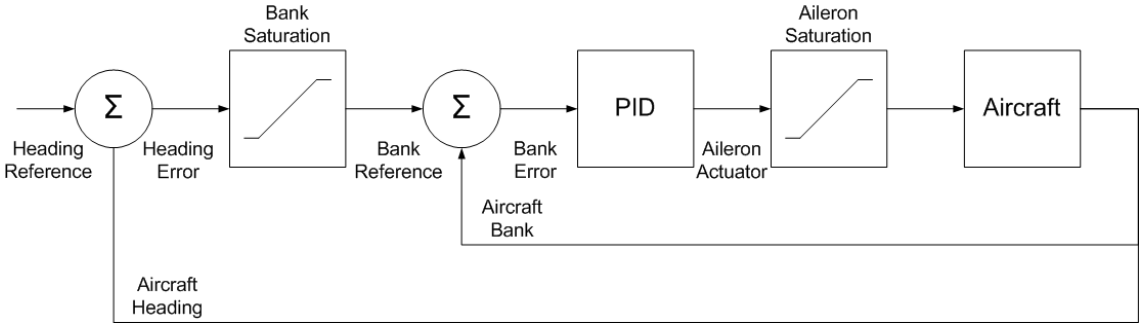

Figure 3.3: PID heading control

The PID, saturation and integral wind-up constants are presented in table 3.1.

Bank saturation			PID					Aileron saturation		
Lower limit	Upper limit	Slope	k	T_i	T_d	Int. lower limit	Int. upper limit	Lower limit	Upper limit	Slope
$-\pi/4$	$\pi/4$	$\pi/4/360$	2,26	17,24	0,7	-100	100	-0,4	0,4	1/100

Table 3.1: Heading controller parameters

PID altitude control A altitude controller (Fig. 3.4) was developed using the vehicle elevator as the actuator.

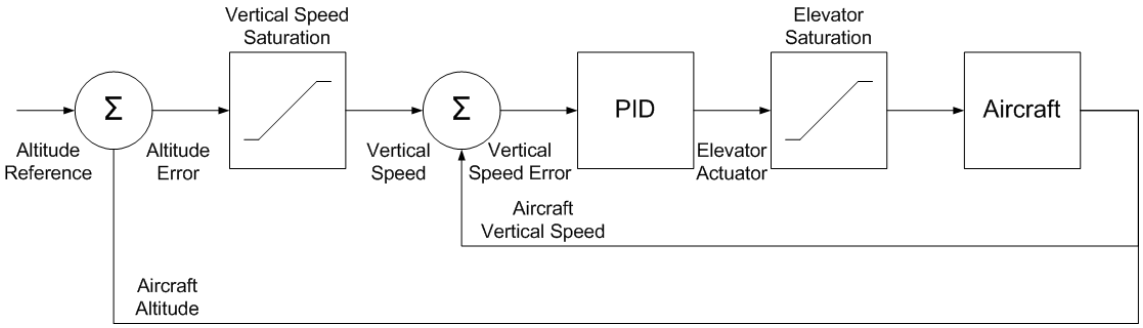

Figure 3.4: PID altitude control

The PID, saturation and integral wind-up constants are presented in table 3.2.

V. speed saturation			PID					Elevator saturation		
Lower limit	Upper limit	Slope	k	T_i	T_d	Int. lower limit	Int. upper limit	Lower limit	Upper limit	Slope
-5	5	$1/800$	2	16	0,7	-100	100	-0,5	0,5	1/1000

Table 3.2: Altitude controller parameters

PID speed control A altitude controller (Fig. 3.5) was developed using the vehicle throttle as the actuator.

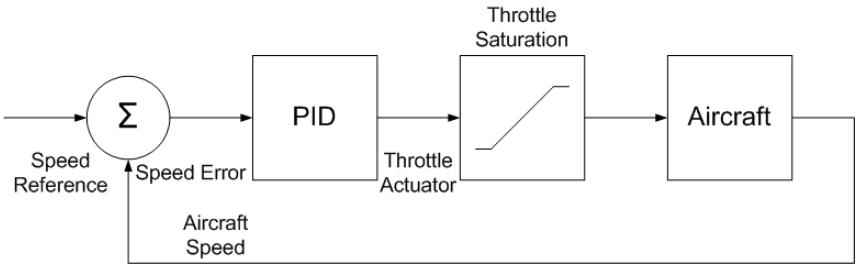

Figure 3.5: PID speed control

The PID, saturation and integral wind-up constants are presented in table 3.3.

PID					Throttle saturation		
k	T_i	T_d	Int. lower limit	Int. upper limit	Lower limit	Upper limit	Slope
2	5	0,7	0	0	0	1	1/10000

Table 3.3: Speed controller parameters

3.2.2 Internal autopilot-based control

Besides the normal functions for heading, altitude and speed control there were also developed functions for taking off, landing, going to a point, flying in a circle, flying an helicoidal path and finding surface types. These functions were developed using the simulator's internal autopilot. The autopilot properties can be configured by editing the vehicles flight model configuration file (.../SimObjects/'Vehicle type'/aircraft.cfg).

```
[autopilot]
autopilot_available= 1
flight_director_available= 0
default_vertical_speed= 0.0
autothrottle_available= 0
pitch_takeoff_ga=8.0
max_pitch=10.0
max_pitch_acceleration=1.0
```

```
max_pitch_velocity_lo_alt=2.0
max_pitch_velocity_hi_alt=1.5
max_pitch_velocity_lo_alt_breakpoint=20000.0
max_pitch_velocity_hi_alt_breakpoint=28000.0
max_bank=90.0
max_bank_acceleration=20
max_bank_velocity=40.00
max_throttle_rate=0.10
nav_proportional_control=9.00
nav_integrator_control=0.25
nav_derivative_control=0.00
nav_integrator_boundary=2.50
nav_derivative_boundary=0.00
gs_proportional_control=9.52
gs_integrator_control=0.26
gs_derivative_control=0.00
gs_integrator_boundary=0.70
gs_derivative_boundary=0.00
yaw_damper_gain = 0.0
```

The navigational (lateral control) and glideslope (vertical control) controllers are in standard PID form. The integrative and derivative terms are bounded, this boundary represents the maximum error input that keeps the term active. The proportional, integrator and derivative gain can be edited to obtain optimal control. It is also possible to edit the controllers actuators for pitch, bank and vertical speed. To control the autopilot from a SimConnect client we need to send Flight Simulator key events. A key event is an event that is generated when you press a key on your keyboard, that event will then be interpreted by the simulator to perform some action (ex: when you press the 'p' key the simulator pauses). To achieve this through SimConnect we need to create a client defined event that will be sent to the simulator, which will interpret it as a key press. First we associate the client event with the key press simulator event using MapClientEventToSimEvent, and then transmit it using TransmitClientEvent to achieve the desired action. This procedure will be used extensively through the development of the vehicle control functions. The following controls are valid for every vehicle type, except helicopters. Although, at an early stage of the project PID controllers were developed to control vehicles (altitude, heading and speed control) these were disregarded in favor of the internal controllers, once it was known how to manipulate them to use them for any vehicle type.

Heading control This is an elemental function that sets the vehicle heading in degrees. It is consist on turning the autopilot and the heading hold module on, and setting the heading bug (refer to appendix - Heading Control).

Altitude control This function sets the vehicle altitude in feet. It is similar to the previous function with the exception of setting the altitude instead of the heading (refer to appendix - Altitude Control).

Speed control [1] This function sets the vehicle air speed in knots. Similar to altitude control (refer to appendix - Air Speed Control)

Take off This function implements a simple take off action, where the sole parameter is the final altitude (altRef). The function disregards taxing and other formal procedures which should be considered by the user (refer to appendix - Take off).

Set on ground This function implements a simple set on ground action, disregarding landing patterns and taxiing operations (refer to appendix - Set on ground).

Find surface This function retrieves the localization of a certain surface type (water,sand, asphalt, etc...) on a certain condition (normal, wet, icy or snow). The parameters are the desired surface type and surface condition (refer to appendix - Find Surface).

Go to point This is actually a thread that can be initiated when you want your vehicle to go to a certain point (lat='destination latitude',long='destination longitude'). It calculates the distance (d_{km}) and the heading(tc), using the great circle navigation formulae, in a loop until the vehicle reaches the vicinity of the destination point (refer to appendix - Go to point). The thread requires that the vehicle actual latitude and longitude be requested (Refer to chapter 2 point 3.3.3) at an adequate rate.

Circle control This function makes the vehicle go in circles at a predefined altitude appendix - Circle Control). It is basically an application of the previous function (Go to point), with the difference that when the vehicle reaches the vicinity of one point, there is a switch.

```
LOOP
  x=NOT(x)
  WHILE d_km < Vicinity
    Go to point(x)
  END WHILE
END LOOP
```

This function can be used to follow a landing pattern, as a wait or secure a perimeter function.

[1] To use this function the autothrottle variable in the vehicle configuration file must be set to 1

Helicoidal control This function makes the vehicle go in helicoidal path, specifying the number of laps, until it reaches the final altitude appendix - Helicoidal Control). The algorithm is similar to that of the previous function.

```
ALTITUDE STEP=(INITIAL ALTITUDE-DESTINATION ALTITUDE)/(NUMBER OF LAPS
 * 2)
LOOP
  x=NOT(x)
  WHILE d_km < Vicinity
    Go to point(x)
  END WHILE
  SET ALTITUDE(INITIAL ALTITUDE-ALTITUDE STEP)
END LOOP
```

This function can be used to follow a landing pattern,

3.3 Agent communication protocol

The communication between the agents will use the FIPA contract net protocol. Using an auction metaphor, one of the agents is responsible for initiating (the initiator) the communication, making a 'bid', to which the other agents will respond with a proposal. The 'bider' will then choose the proposal which is closest to the initial 'bid'. The bider has knowledge of the number of participants before the issuing of the call for proposals. The physical layer of the communications is done using TCP/IP sockets, an alternative to socket would be to create a client data area, in the initiator SimConnect client, that is available to the SimConnect clients via TCP/IP of the contractors, as said before SimConnect can be affected by latency in consequence of its asynchronous nature, with no real benefit from this solution, the first option was implemented. The implementation of the protocol is pretty straight forward, the format of the exchanged information can be for example:

```
For the accept-proposal performative
accept-proposal:'sender agent name':'receiver agent name':'bid number':
'content':'language'
```

3.4 Test Application

In this section the development of the test application will be discussed, focusing on major features and application usage. The first screen of the application is the settings screen (Fig. 3.6), here it is possible to configure the rate of exchange of data between the application and SimConnect, and define the limit of the area (with the center on the user plane) from which SimConnect can retrieve information about the vehicles. It is also possible to configure the agent type and communication settings. For an initiator, or, the agent the requests the tasks, the settings are the number of contractor agents, the TCP/IP protocol is initiated using a TCP listener. For the contractor, or, the agent that executes the tasks, the settings are the IP address and the communication port number. There four modes of operation available: the manual control receive mode - used to visualize vehicle data in the manual control screen, the manual control send mode - used to control the vehicle using the manual control screen, the controls test mode - used for the testing of the developed controls and the mission control mode - used to take a role as an agent in a selected mission.

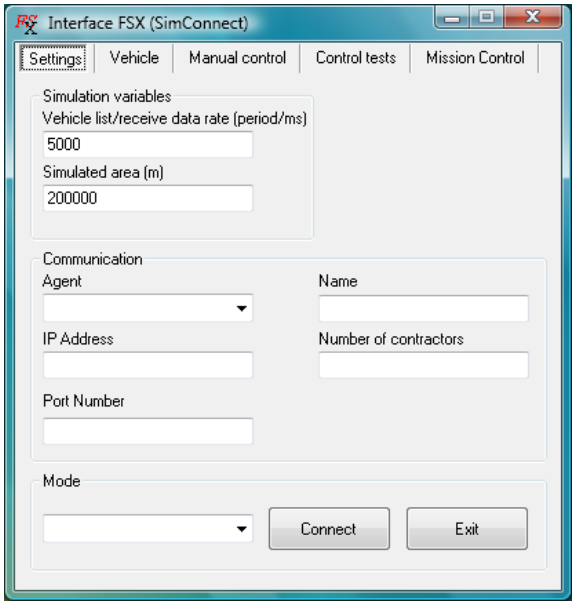

Figure 3.6: Application settings tab

The vehicle list screen (Fig. 3.7) is used to select the vehicle to be controlled. This screen lists all the vehicle present in the area specified in the settings screen, which include airplanes, boats, ground vehicles and helicopters. This functionality can be used in future work to develop a collision detection system. The list has to be regularly updated because the vehicles that fall outside of the specified area can no longer be controlled by the application.

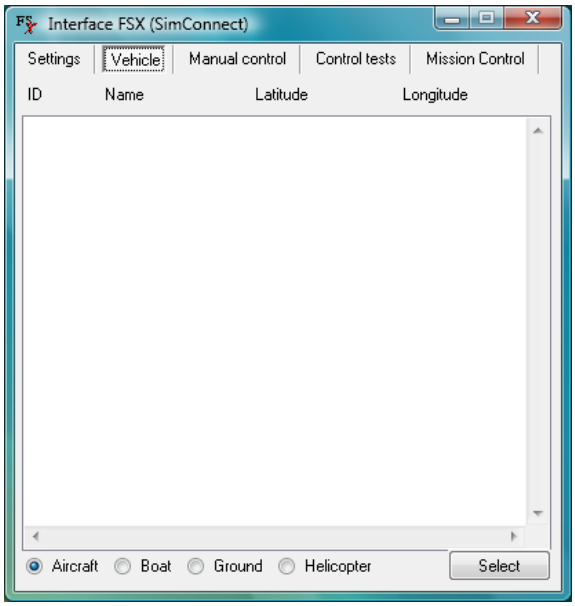

Figure 3.7: Application vehicle selection tab

The manual control screen (Fig. 3.8) is used to control the vehicle or visualize the flight variables.

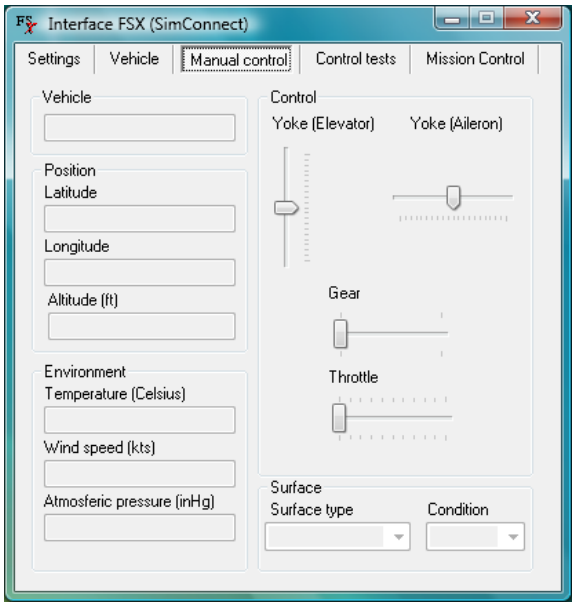

Figure 3.8: Application manual control tab

The control tests screen (Fig. 3.9) is used to test the implemented controls. At an early stage of the project PID controllers were developed and tested, but were set aside in detriment of the internal autopilot controllers.

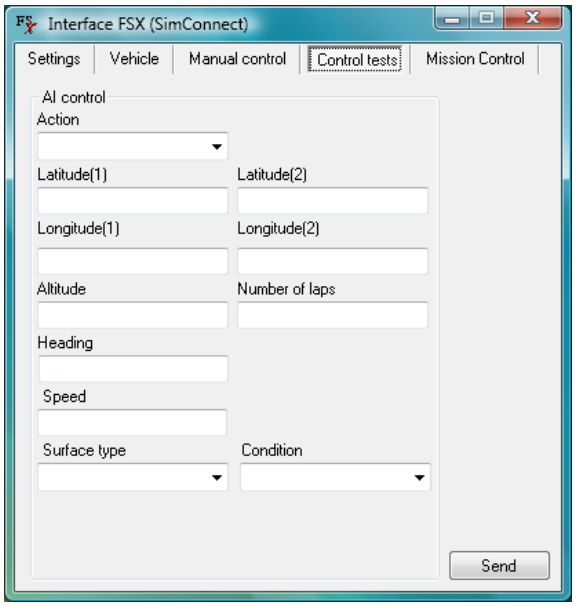

Figure 3.9: Application control tests tab

The mission control screen (Fig. 3.10) is used to select the mission and the agent role. Allows the user to visualize the interaction between the agents in the communications box. An example of this mode of operation will be presented on the next chapter.

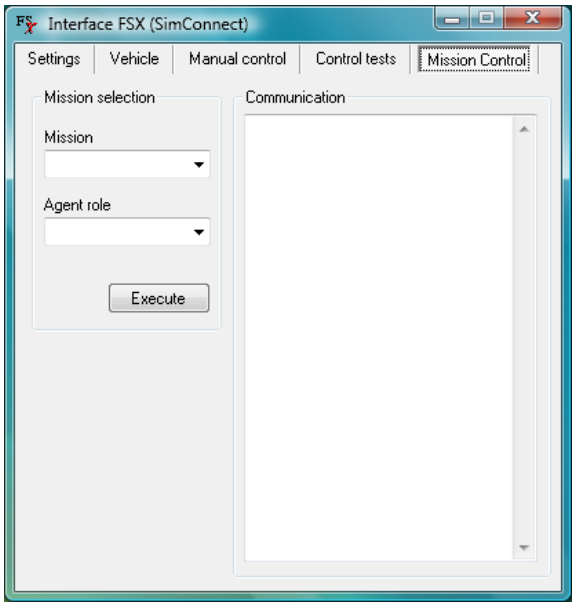

Figure 3.10: Application mission control tab

3.5 Conclusion

In this chapter the development of the project was presented. From two possible solutions the architecture based on Flight Simulator X multiplayer system was chosen. The vehicle control development was presented including the high-level control functions: go to point, land, take off, follow a circular or helicoidal path. For agent interaction the well established FIPA Contract Net protocol was chosen, allowing many interaction possibilities. Finally a presentation of the testing interface was made, focusing on the mission control mode.

Chapter 4

Results and tests analysis

In this chapter, we will first show some test for the vehicle control and then a complete test using the developed mission. An interface was developed for testing.

4.1 Vehicle control tests

The following tests were made by sampling the vehicles variables to demonstrate the performance of the control functions and using the developed application tab for vehicle control testing. We can see in Fig. 4.1 that the controller follows, with an insignificant steady state error (less than 0,1%) using for reference values 90 and 180 degrees.

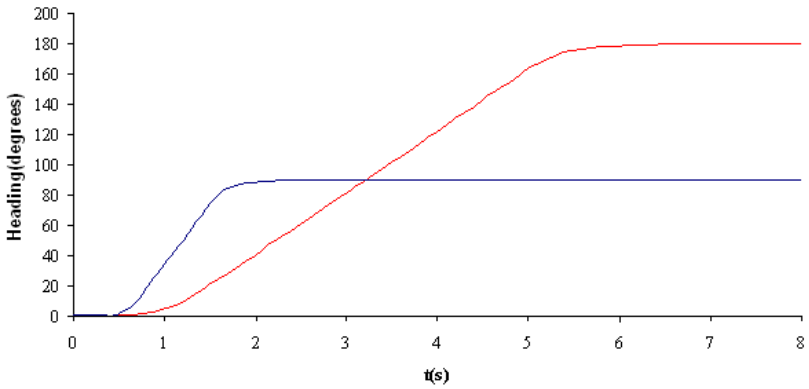

Figure 4.1: Heading control performance

We can see the circle function performance in Fig. 4.2, through the (47,42140675,-122,3042361) and (47,43472455,-122,305) points at an altitude of 1000 feet.

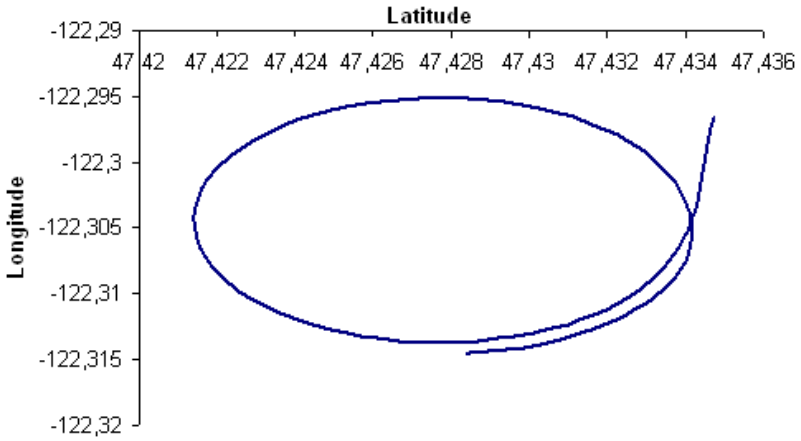

Figure 4.2: Circle control performance

For the helicoidal function Fig. 4.3, with an altitude of 1800 going down to 1000 feet in two laps through the (47,42431118,-122,1276413) and (47,44540565,-122,116448) points.

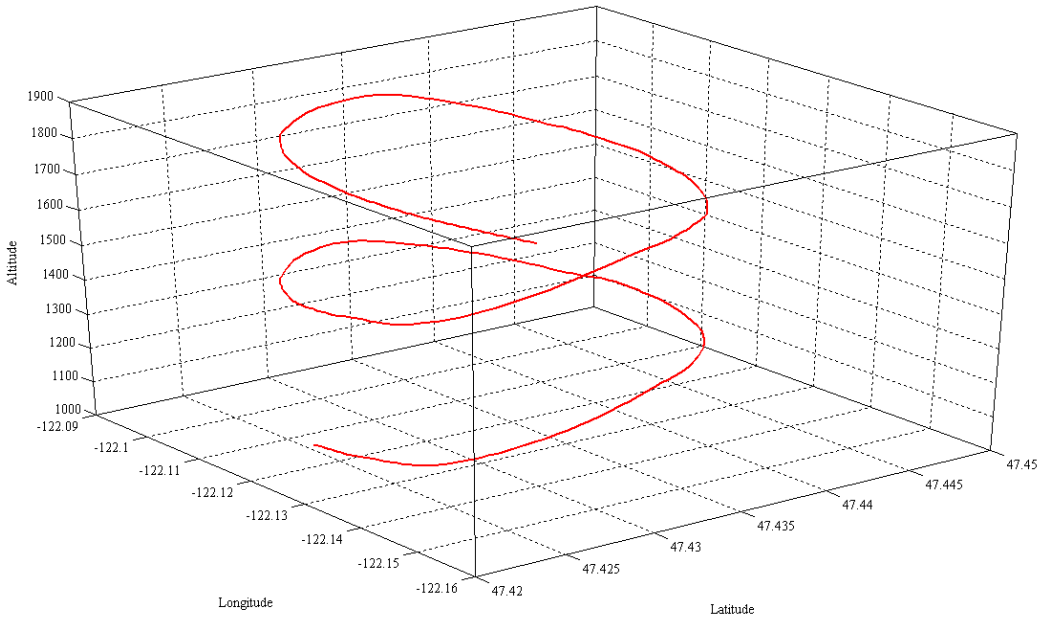

Figure 4.3: Helicoidal control performance

4.2 Mission example

A simple mission was developed for testing of the vehicle interaction.

Mission scope The mission (Fig. 4.4) consists on extinguishing a crashed airplane fire, consequence of an emergency landing. The emergency procedures are then activated with the deploying of two vehicles: a fire truck (AI vehicle) and the airport emergency coordinator (Player 2). The coordinator vehicle, once reached the accident location, will make an assessment of the situation and initiate the requisition of the available vehicles (Player 1, 3, 4 or 5). The mission is assumed complete once the fire is extinguished.

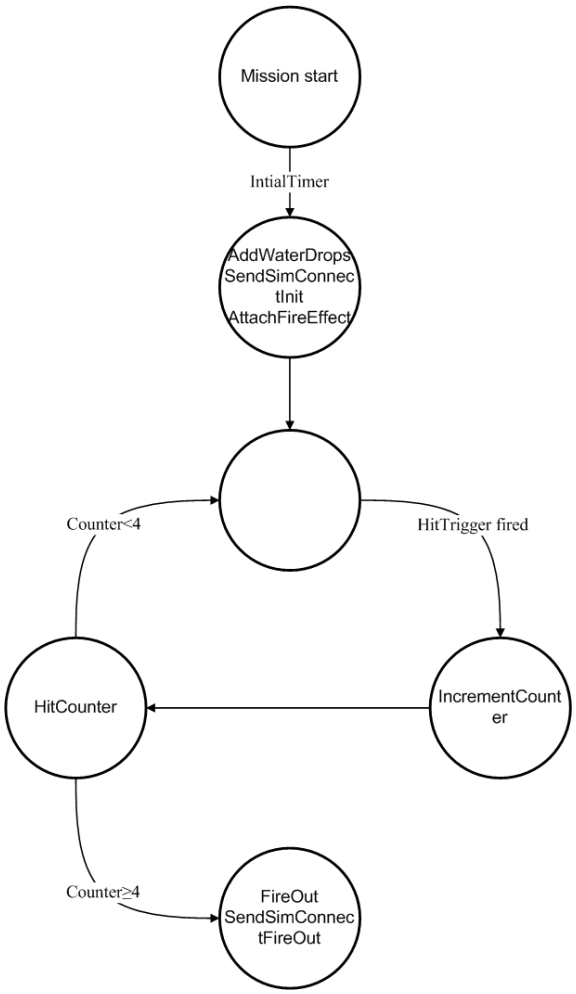

Figure 4.4: Mission state diagram

4.3 Mission test

To test the mission we need to configure an agent as the Initiator (Fig. 4.5), choosing the number of contractors.

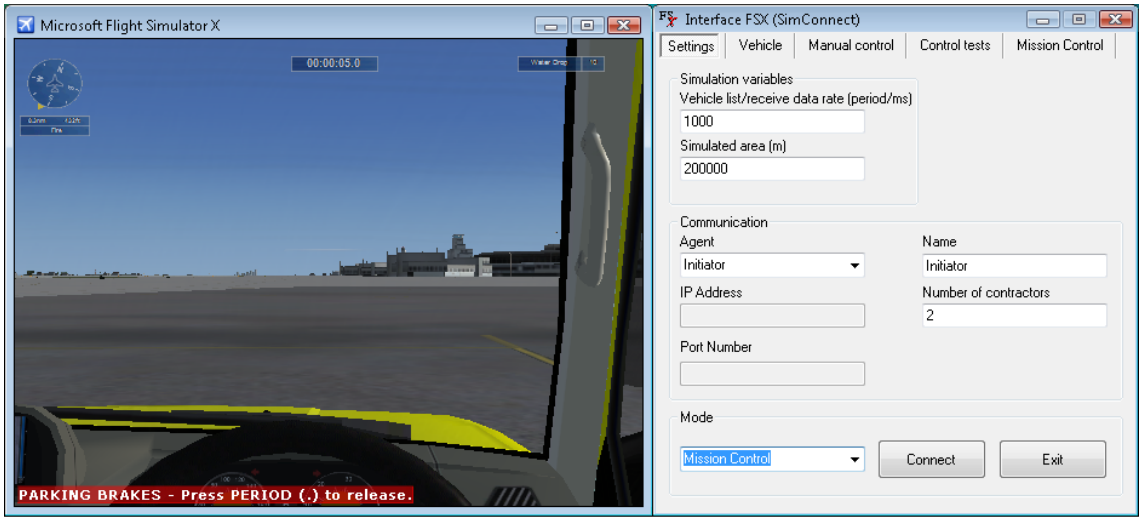

Figure 4.5: Initiator configuration

In the Vehicle selection tab choose the vehicle (Fig. 4.6) that will play a part in the mission.

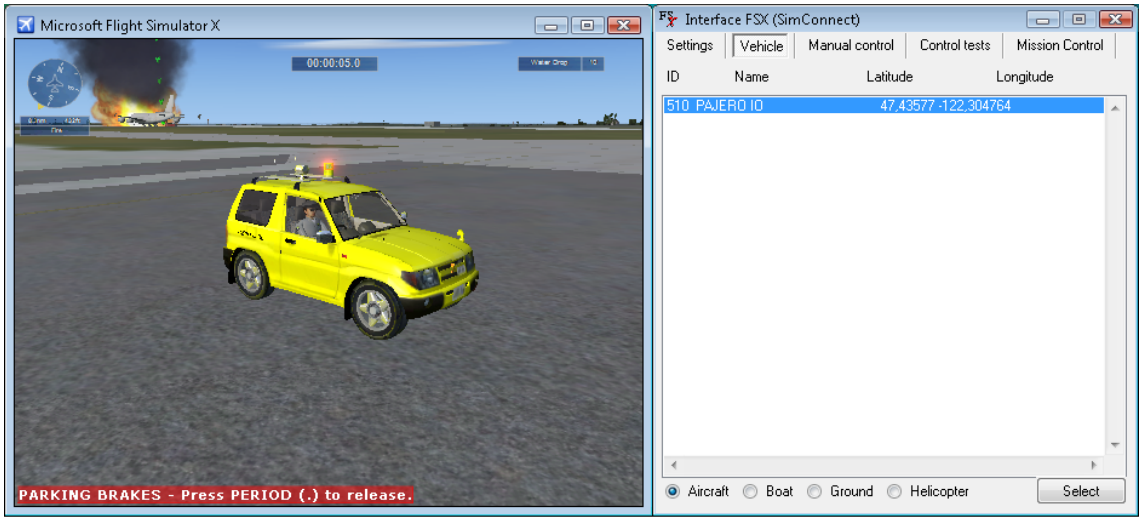

Figure 4.6: Initiator vehicle selection list

The vehicle will then move to the accident site, and will initiate the dialog (Fig. 4.7) with the other two agents (Contractor#1 and Contractor#2). The action is to drop water on the accident site, the proposed value 20 was obtained with the weighted sum of a random number and a temperature

and wind speed factor. The contractors then use the weighted sum of their distance to the accident site and the number of water drops available on their payload. We can see that Contractor#1 refused to propose, this happens because the requested value is too high. Contractor#2 proposes with a value of 30 and has his proposal accepted by the Initiator.

Figure 4.7: Initiator biding dialog

The contractor proceeds to the accident site and executes the proposed action.

Figure 4.8: Contractor#2 biding dialog

The mission is considered over when the vehicle hits the accident area four with four water drops (Fig. 4.9). The situations for timeout, when none of the contractors respond in the previously

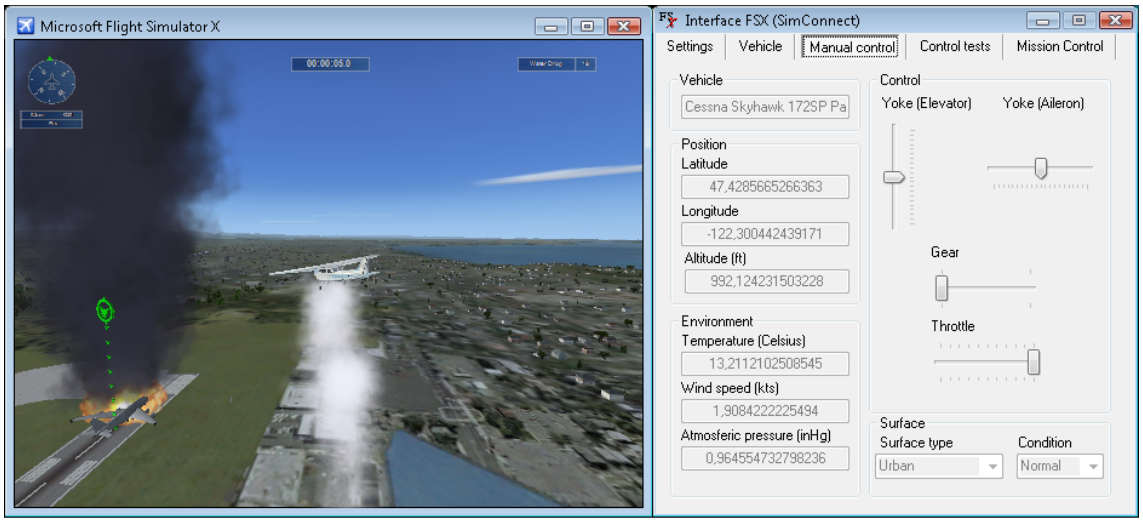

Figure 4.9: Contractor#2 executing proposed task

established time (20 s), and the case that both the contractors submit an equal counter bid where also tested successfully (Table. 4.1).

Test	Initiator#1	Contractor#1	Contractor#2	Result
1	Sent call for proposals			Timeout
2	Sent call for proposals	Sent proposal	Refused	Contractor#1 got the task
3	Sent call for proposals	Refused	Sent proposal	Contractor#2 got the task
4	Sent call for proposals	Refused	Refused	Task not executed
5	Sent call for proposals	Sent proposal	Sent proposal	Best bidder got the task

Table 4.1: Agent interaction tests

4.4 Conclusion

The high-level vehicle control functions were tested with results being presented for *follow a circular path* and *follow a circular path*. The developed mission tests results were also presented as well as tests for interacting possibilities using two agents.

Chapter 5

Conclusions and future prospects

Multi-agent system are an ever developing field that benefits from the growth and evolution, in its distributed form, of the Internet.

From the early start of the project it was clear that Flight Simulator X is the simulator which guarantees future and backward compatibility in terms of the interfacing protocols, namely SimConnect. The huge number of vehicle addons, scenery and software developers along with the quality of the SDK documentation state that the simulator is bound to persevere in favor of the competitors FlightGear and X-Plane. The capability of agent interaction was demonstrated through the FSX mission environment, which still has some limitations, like the fact that it is not possible to assign objects to more than one vehicle, the addon vehicles performance can be poor, for some vehicles but it will still be a steadily growing tool in the future. The mission editing tool is not very intuitive or easy to use, but compensates in the ability to develop realistic missions in terms of weather conditions, vehicle failures and the interaction between simulated objects, such as AI vehicles.

As for future developments, one has to stress the need to improve the agents autonomy. To achieve this a lot can be learned from UAV development. Also for the sake of autonomy it is also necessary to develop a system to avoid collisions with other vehicles and facilities. In the terms of agent communication, a possible exchange to protocols such as FIPA ACL or KQML can lead to richer and more interesting cooperation between the agents.

Bibliography

[1] Russel, S., Norvig, P., Artificial Intelligence - a modern approach, Prentice Hall, 1995, 3 - 30

[2] Wooldridge, M., An introduction to Multiagent Systems, John Wiley & Sons Ltd, 2002, 16 - 25

[3] PID controller, en.wikipedia.org/wiki/PID_controller, Last visited on: 7/06/2008

[4] Aviation formulary, williams.best.vwh.net/avform.htm, Last visited on: 7/07/2008

[5] FIPA Contract Net Interaction Protocol, www.fipa.org/specs/fipa00029/SC00029H.html, Last visited on: 20/06/2008

[6] FlightGear 747 Project, www.flightgear.org/Projects/747-JW/, Last visited on: 12/06/2008

[7] X-Plane developers, xplane.org, Last visited on: 15/05/2008

[8] Google moving map: Displaying X-Plane's flight path on Google Earth, Last visited on: 8/04/2008

[9] X-Plane projects, www.xpluginsdk.org/projects.htm, Last visited on: 7/06/2008

[10] Fidelity flight simulation, flightmotion.com/newwho.htm, Last visited on: 12/06/2008

[11] Cirrus "The Jet" project, www.the-jet.com, Last visited on: 12/06/2008

[12] X-Plane Plugin SDK, www.xsquawkbox.net/xpsdk/phpwiki/index.php, Last visited on: 7/04/2008

[13] FlightGear, www.flightgear.org, Last visited on: 2/03/2008

[14] FlightGear vs C#, linkslink.wordpress.com/takeoff/, Last visited on: 4/03/2008

[15] Piccolo System, www.cloudcaptech.com/piccolo_system.shtm, Last visited on: 2/03/2008

[16] FlightGear remote control (Telnet protocol), cvs.flightgear.org/cgi-bin/, Last visited on: 9/03/2008

[17] FDM JsbSim, jsbsim.sourceforge.net/, Last visited on: 10/03/2008

[18] Flight Simulator X, www.microsoft.com/games/pc/flightsimulatorx.aspx, Last visited on: 29/04/2008

[19] SimConnect Documentation, msdn.microsoft.com/en-us/library/cc526983.aspx, Last visited on: 30/06/2008

[20] jSimConnect, lc0277.nerim.net/jsimconnect/, Last visited on: 26/04/2008

[21] Flight Simulator X - Mission Editor, fsaddon.eu/wpfsaddon/?page_id=26, Last visited on: 20/06/2008

[22] FSX Google Eartf Tracker, www.codeplex.com/fsxget, Last visited on: 17/06/2008

[23] Google Earth, earth.google.com, Last visited on: 17/06/2008

[24] VATSIM, www.vatsim.net, Last visited on: 17/06/2008

[25] FS Live Traffic, www.vatsim.net, Last visited on: 17/06/2008

[26] UDP reference guide, /Program Files/XPlane/X-Plane 8.64/Instructions/Sending Data to X-Plane.html (X-Plane's folder)

[27] X-Plane SDK Documentation, www.xsquawkbox.net/xpsdk/phpwiki/index.php?Documentation, Last visited on: 3/04/2008

[28] X-Plane Data Reference, www.xsquawkbox.net/xpsdk/docs/DataRefs.html, Last visited on: 5/04/2008

[29] SimGear, www.simgear.org/, Last visited on: 10/03/2008

Appendix A

X-Plane interfacing protocols

A.1 Interfacing modes

In this appendix the two options of interfacing X-Plane version 8.64 will be introduced.

A.1.1 The UDP protocol

X-Plane's UDP interfacing consists on sending messages using predefined structures which contain information about the internal variable you want to change or read [26]. This can be achieved by using different structure types as will be seen promptly. This procedure as been considered deprecated (Variable list and data structure constantly changing) by X-Plane's developers therefore only a brief explanation will be presented for the sake of broadness.

X-Plane's internal variable types:

XCHR (character, in MACINTOSH byte-order (biggest byte first))
XINT (4-byte int, in MACINTOSH byte-order (biggest byte first))
XFLT (4-byte ints and floats, in MACINTOSH byte-order (biggest byte first))
XDOB (double-precision float, in MACINTOSH byte-order (biggest byte first))
strDIM is 500
vehDIM is 10

The UDP communication protocol message consists mainly of two parts:

1. Message prologue: used to indicate the type of message

2. Data structure: containing the message data that you want to send or receive

The message prologue consists of the first five **XCHR** array values of the message, where the first 4 chars are the type of data and the fifth char is for X-Plane's internal use (typically 0).

```
xchr data_send[net_SIZE_buff];
data_send[0]='D';
data_send[1]='A';
data_send[2]='T';
data_send[3]='A'];
data_send[4]=0;
```

Types of data:

- DATA: generic data
- DSEL/USEL/DCOC/UCOC: select which data to send/receive
- BOAT: used to steer boats on the simulator
- MOUS: used to simulate mouse clicks
- CHAR: used to simulate key press
- MENU: select menu
- FAIL: fail a system
- RECO: recover a system
- PAPT: place the aircraft at an airport
- VEHN: load an airplane
- VEH1/VEHA: drive X-Plane with an external flight model
- OBJN: place an object
- OBJL: place an object on ground
- SOUN: play a sound
- GSET: set graphics options
- ISET: multi player protocol

The data structure is defined as follows:

```
struct data_struct
{
int index;
float data[8];
};
```

where the integer "index" is the index of the internal variable that can be found on the list displayed in X-Plane's Settings / Data Input & Output... menu, and the data array is the associated data (some list elements may have more than one instance, like gear, engine, etc. hence the eight positions of the array)[1].

So depending on which data you select from the list you're message will look like:

[Message prologue] [Data Structure] [Data Structure] (...)
[Message prologue] [Data Structure] [Data Structure] (...)
(...)

A.1.2 The Plugin SDK

A plugin is a modular extension to a program with the purpose of adding or extending a functionality. More specifically a plugin is a DLL or a set of DLL's which provide a program an in-process shared library. Some of plugin's functionalities include: changing data within a program, create user interface and menus, inter-plugin communication. An open-source X-Plane's SDK is available at xsquawkbox.net. Some of the most relevant routines and functions will be presented along with an application of this mode of interfacing.

SDK reference headers:

XPLMPlugin - Plugin management, simulator messages and interplugin communication.
XPLMDataAccess - Simulator data access, plugin data sharing and publishing.
XPLMProcessing - Running tasks while the simulator runs.
XPLMDisplay - Creating windows, hot keys, key sniffers, and drawing callbacks.
XPLMMenus - Creating and managing menus.
XPLMGraphics - Drawing in 2d or 3d.
XPLMUtilities - Controlling the simulator, miscellaneous functions.
XPLMCamera - Controlling the X-Plane camera.
XPLMPlanes - Accessing other aircraft.
XPLMNavigation - Reading the navigation databases and programming the FMC.
XPWidgetDefs - Basic widget message systems.

[1] If you don't want to change some variable, when sending the message, set it to -999

XPWidgets - API reference for manipulating widgets.
XPWidgetUtils - Utility functions for managing widgets and building your own widgets.
XPStandardWidgets - Definitions of useful common widgets like buttons and text fields.
XPUIGraphics - Low level graphics primitives to draw X-Plane style user interface.
XPLMScenery - Interact with the scenery system (2.0 only)

The three API reference headers used to read, write data and control aircraft on X-Plane will presented next, respectively, XPLMDataAccess, XPLMUtilities and XPLMPlanes [27].

XPLMDataAccess

This API enables us to read and write data [28] from X-Plane and other plugin's. It uses opaque data references, which allows you to gain access to data by looking it up first. Here's an example on how to read and write data:

```
// This creates the reference handler
XPLMDataRef gDataRef=null;

// Look up and assign data reference
gDataRef = XPLMFindDataRef("sim/flightmodel/engine/ENGN_thro");

// Set the correspondent data reference with integer <value>
XPLMSetDatai(gDataRef, <value> );

// Get the correspondent data reference to integer <variable>
XPLMGetDatai(gDataRef, <variable> );
```

XPLMUtilities

This API simulates user interaction through the use of a keyboard or joystick and it is the recommended API for controlling the aircraft from an external application. Example:

```
// Simulate gear up/down "g" key press
XPLMCommandKeyStroke(xplm_key_gear);

// Simulate throttle up button
XPLMCommandButtonPress(xplm_joy_throt_up);
```

XPLMPlanes

This API allows you to control X-Plane's user and non user aircraft, also including simple AI commands. Example:

```
// Sets the user aircraft
XPLMSetUsersAircraft("\X-Plane 8.64\Aircraft\General Aviation\
Cessna 172SP\Cessna 172SP.acf");

// Places user plane at KBOS airport
XPLMPlaceUserAtAirport("KBOS");

// Disables AI on airplane with index=6
XPLMDisableAIForPlane(6);
```

Appendix B

FlightGear interfacing protocols

B.1 Interfacing modes

This section will describe some of the more important modes of interfacing FlightGear. We will first describe the Generic, Native and Telnet interfacing protocols, and then we will illustrate some exemple applications. The version of the platform used for the development and tests of the application will be FlightGear v1.0 and the IDE Microsoft Visual C++ 6.0.

To start FlightGear using the console using any of the interfacing protocols, execute the following command:

```
fgfs --protocol=medium,direction,hz,medium_options,...
```

Where:

```
protocol = { generic, native, nmea, garmin, fgfs, rul, pve,
ray }
medium = { serial, socket, file, etc. }
direction = { in, out, bi }
hz = read/write frequency in Hertz
medium_options=tcp,udp,port#,path
```

It is possible to invoke various protocols simultaneously.

```
fgfs --native_fdm=socket,in,30,,5555,udp
--generic=file,out,30,log.txt,playback
```

This command enables FlightGear to receive information via native_fdm protocol at a 30 Hz rate using an UDP socket (from any IP address) on port 5555 and, simultaneously, allows it to send information to a file named log at 30 Hz rate using the format defined on the playback.xml file.

The following protocols will be discussed on the following sections: Generic, Native, Telnet e HTTP.

B.1.1 Generic Protocol

This protocol allows to predefine the data that will be exchanged in a XML file (`flightgear/data/protocol`). The data can be in a ASCII or binary format and can be exchanged via file, socket or serial port.

Parameter options:

- Serial port: `fgfs --generic=serial,dir,hz,device,baud,protocol`
- Socket: `fgfs --generic=socket,dir,hz,machine,port,style,protocol`
- File: `fgfs --generic=file,dir,hz,filename,protocol`

Where:

```
dir = direction (in,out,bi)
device = COM1, COM2, etc.
baud = Baud Rate
protocol = XML configuration data file
machine = IP destination address
port = serial port number
style = tcp or udp
filename = file path
```

For example, it is possible to record a flight using this protocol and an adequate data predefined in `playback.xml`:

```
fgfs --generic=file,out,30,log.txt,playback
```

and play it back, using:

```
fgfs --generic=file,in,30,log.txt,playback
```

In this mode it is possible to predefine which data from the property tree you want to send or receive using a simple XML protocol configuration file, the disadvantages are that at each read/write cycle you will have to receive/send all the data that is predefined in the XML file which may not be desirable in remote applications or when you just need some part of the data. If your building an application, it will be necessary to parse the data every reading cycle.

B.1.2 Native protocol

The Native protocol was created to allow the communication between two or more instances of FlightGear. This protocol allows the communication beetween different operating systems using the fixed lenght datagrams in network format or big-endian. This protocol is divided into the subprotocols `native_fdm`, `native_ctrls` e `native_gui`. It basically consists on the filling of the structures defined in `net_ctrls.hxx`, `net_fdm.hxx` e `net_gui.hxx` [1] which contain the variables for flight control, and flight model, respectively.

Parameter options:

- Serial port: `fgfs --native=serial,dir,hz,device,baud`
- Socket: `fgfs --native=socket,dir,hz,machine,port,style`
- File: `fgfs --native=file,dir,hz,filename`

It is possible to drive a FlightGear instance from another, using for the 'driver' instance:

```
fgfs --native=socket,out,30,127.0.0.1,5555,udp
```

and for the 'driven' instance:

```
fgfs --native=socket,in,30,,5555,udp
```

The greatest advantage is that this protocol allows transparent communication between different operating systems, on the other hand this protocol provides access to fewer property tree variables (although, there is a reserved space for user defined variables) and has compatibility problems with previous versions of the platform.

B.1.3 Telnet protocol

This mode launches FlightGear as Telnet server.

Parameter options:

```
fgfs --telnet=port#

Example:
1) Launch FlightGear, using: fgfs --telnet=5500
2) Connect to FlightGear using a telnet client: telnet
```

[1] These header files are available in SimGear [29]

```
 localhost:5500
3) Send: 'set /sim/master/freeze true' to stop the simulation
```

This is the most reliable method for interfacing FlightGear, it is possible to access all of the property tree variables individually. The disadvantage is that for reading you have to send the request and wait for the answer.

B.1.4 HTTP

This is an HTTP daemon which is useful to consult the internal properties. Supposedly it should be able to send data into FlightGear but this option does not appear to be functioning.

- Usage: `fgfs --httpd=<port#>`

Ex: `fgfs--httpd=5500 and on a web browser:`
`http://127.0.0.1:5500/`

Appendix C

Vehicle control functions - autopilot-based

The following enumeration is a requisite for all the functions present in this section. The parameter objectID represents the identification number of the user aircraft. The 'current' variables are being requested (RequestDataOnSimObject) and updated every second by a (OnRecvSimobjectData) handler.

```
enum KEY_EVENTS
{
        KEY_AUTOPILOT_ON,
                        //Turns autopilot on
        KEY_AP_HDG_HOLD_ON,             //Turns heading hold
            mode on
        KEY_AP_HDG_HOLD_OFF,
                //Turns heading hold mode off
        KEY_AP_ALT_HOLD_ON,             //Turns altitude hold
            mode on
        KEY_AP_ALT_HOLD_OFF,
                //Turns altitude hold mode off
        KEY_HEADING_BUG_SET,            //Set heading hold
            reference bug (degrees)
        KEY_AP_ALT_VAR_SET_ENGLISH,     //Sets reference
            altitude in feet
        KEY_AP_SPD_VAR_SET,             //Sets airspeed
            reference in knots
        KEY_AP_AIRSPEED_ON,
                        //Turns air speed hold mode on
```

```
        KEY_AP_AIRSPEED_OFF,
                    //Turns air speed hold mode off
        KEY_PARKING_BRAKES,
                            //Toggles parking brake on/off
        KEY_RELEASE_DROPPABLE_OBJECTS,  //Release droppable
            objects
        KEY_BRAKES,
                                                    //Increment
            brake pressure
};
```

C.1 Heading control

```
...
//Associate the KEY_EVENTS.KEY_AUTOPILOT_ON event ID with the
    autopilot_on event
simconnect.MapClientEventToSimEvent(KEY_EVENTS.KEY_AUTOPILOT_ON,
    "autopilot_on");
//Associate the KEY_EVENTS.KEY_HEADING_BUG_SET event ID with the
    HEADING_BUG_SET event
simconnect.MapClientEventToSimEvent(KEY_EVENTS.
    KEY_HEADING_BUG_SET, "HEADING_BUG_SET");
//Associate the KEY_EVENTS.KEY_AP_HDG_HOLD_ON event ID with the
    AP_HDG_HOLD_ON event
simconnect.MapClientEventToSimEvent(KEY_EVENTS.
    KEY_AP_HDG_HOLD_ON, "AP_HDG_HOLD_ON");
...
void setHeading(uint hdgRef)
{
        //Turn autopilot on
        simconnect.TransmitClientEvent(objectID, KEY_EVENTS.
            KEY_AUTOPILOT_ON, 0, GROUP.ID_PRIORITY_STANDARD,
            SIMCONNECT_EVENT_FLAG.GROUPID_IS_PRIORITY);
        //Turn heading hold module on
        simconnect.TransmitClientEvent(objectID, KEY_EVENTS.
            KEY_AP_HDG_HOLD_ON, 0, GROUP.ID_PRIORITY_STANDARD,
            SIMCONNECT_EVENT_FLAG.GROUPID_IS_PRIORITY);
        //Set heading bug
```

```
            simconnect.TransmitClientEvent(objectID, KEY_EVENTS.
                KEY_HEADING_BUG_SET, hdgRef, GROUP.
                ID_PRIORITY_STANDARD, SIMCONNECT_EVENT_FLAG.
                GROUPID_IS_PRIORITY);
}
```

C.2 Altitude control

```
...
//Associate the KEY_EVENTS.KEY_AUTOPILOT_ON event ID with the
    autopilot_on event
simconnect.MapClientEventToSimEvent(KEY_EVENTS.KEY_AUTOPILOT_ON,
    "autopilot_on");
//Associate the KEY_EVENTS.KEY_AP_ALT_VAR_SET_ENGLISH event ID
    with the AP_ALT_VAR_SET_ENGLISH event
simconnect.MapClientEventToSimEvent(KEY_EVENTS.
    KEY_AP_ALT_VAR_SET_ENGLISH, "AP_ALT_VAR_SET_ENGLISH");
//Associate the KEY_EVENTS.KEY_AP_ALT_HOLD_ON event ID with the
    AP_ALT_HOLD_ON event
simconnect.MapClientEventToSimEvent(KEY_EVENTS.
    KEY_AP_ALT_HOLD_ON, "AP_ALT_HOLD_ON");
...
void setAltitude(uint altRef)
{
//Turn autopilot on
simconnect.TransmitClientEvent(objectID, KEY_EVENTS.
    KEY_AUTOPILOT_ON, 0, GROUP.ID_PRIORITY_STANDARD,
    SIMCONNECT_EVENT_FLAG.GROUPID_IS_PRIORITY);
//Turn altitude hold module on
simconnect.TransmitClientEvent(objectID, KEY_EVENTS.
    KEY_AP_ALT_HOLD_ON, 0, GROUP.ID_PRIORITY_STANDARD,
    SIMCONNECT_EVENT_FLAG.GROUPID_IS_PRIORITY);
//Set altitude
simconnect.TransmitClientEvent(objectID, KEY_EVENTS.
    KEY_AP_ALT_VAR_SET_ENGLISH, altRef, GROUP.
    ID_PRIORITY_STANDARD, SIMCONNECT_EVENT_FLAG.
    GROUPID_IS_PRIORITY);
```

C.3 Air Speed control

```
...
//Associate the KEY_EVENTS.KEY_AUTOPILOT_ON event ID with the
    autopilot_on event
simconnect.MapClientEventToSimEvent(KEY_EVENTS.KEY_AUTOPILOT_ON,
     "autopilot_on");
//Associate the KEY_EVENTS.KEY_AP_SPD_VAR_SET event ID with the
    KEY_AP_SPD_VAR_SET event
simconnect.MapClientEventToSimEvent(KEY_EVENTS.
    KEY_AP_SPD_VAR_SET, "KEY_AP_SPD_VAR_SET");
//Associate the KEY_EVENTS.KEY_AP_AIRSPEED_ON event ID with the
    AP_AIRSPEED_ON event
simconnect.MapClientEventToSimEvent(KEY_EVENTS.
    KEY_AP_AIRSPEED_ON, "AP_AIRSPEED_ON");
...
void setAirSpeed(uint spdRef)
{
//Turn autopilot on
simconnect.TransmitClientEvent(objectID, KEY_EVENTS.
    KEY_AUTOPILOT_ON, 0, GROUP.ID_PRIORITY_STANDARD,
    SIMCONNECT_EVENT_FLAG.GROUPID_IS_PRIORITY);
//Turn air speed hold module on
simconnect.TransmitClientEvent(objectID, KEY_EVENTS.
    KEY_AP_AIRSPEED_ON, 0, GROUP.ID_PRIORITY_STANDARD,
    SIMCONNECT_EVENT_FLAG.GROUPID_IS_PRIORITY);
//Set air speed
simconnect.TransmitClientEvent(objectID, KEY_EVENTS.
    KEY_AP_SPD_VAR_SET, spdRef, GROUP.ID_PRIORITY_STANDARD,
    SIMCONNECT_EVENT_FLAG.GROUPID_IS_PRIORITY);
```

C.4 Go to point

```
...
simconnect.MapClientEventToSimEvent(KEY_EVENTS.KEY_AUTOPILOT_ON,
     "autopilot_on");
simconnect.MapClientEventToSimEvent(KEY_EVENTS.
    KEY_HEADING_BUG_SET, "HEADING_BUG_SET");
```

```
simconnect.MapClientEventToSimEvent(KEY_EVENTS.
    KEY_AP_HDG_HOLD_ON, "AP_HDG_HOLD_ON");
...
private void goToPoint(object hdgRef)
{
        double d, tc, headingDeg, distanceKm=1000,ilat,ilong,
            dlat,dlong;

        //Latitude and longitude have to be converted into
            radians
        dlat = 'destination latitude' * Math.PI / 180.0;
        dlong = 'destination longitude' * Math.PI / 180.0;

        //Turn autopilot on
        simconnect.TransmitClientEvent(objectID, KEY_EVENTS.
            KEY_AUTOPILOT_ON, 0, GROUP.ID_PRIORITY_STANDARD,
            SIMCONNECT_EVENT_FLAG.GROUPID_IS_PRIORITY);
        //Turn heading hold module on
        simconnect.TransmitClientEvent(objectID, KEY_EVENTS.
            KEY_AP_HDG_HOLD_ON, 0, GROUP.ID_PRIORITY_STANDARD,
            SIMCONNECT_EVENT_FLAG.GROUPID_IS_PRIORITY);

        //While the vehicle doesn't reach at least 500 m from
            the destination
        while (distanceKm > 0.5)
        {
                ilat = 'current vehicle latitude' * Math.PI /
                    180.0;
                ilong = 'current vehicle latitude' * Math.PI /
                    180.0;

                //Calculate the distance between the two points
                d = Math.Acos(Math.Sin(ilat) * Math.Sin(dlat) +
                    Math.Cos(ilat) * Math.Cos(dlat) * Math.Cos(
                    dlong - ilong));
                //Calculate the heading from i to d
                if (Math.Sin(ilong - dlong) < 0)
                {
```

```
            tc = Math.Acos((Math.Sin(dlat) - Math.
                Sin(ilat) * Math.Cos(d)) / (Math.Sin(
                d) * Math.Cos(ilat)));
    }
    else
    {
            tc = 2.0 * Math.PI - Math.Acos((Math.Sin
                (dlat) - Math.Sin(ilat) * Math.Cos(d)
                ) / (Math.Sin(d) * Math.Cos(ilat)));
    }

    //In case the starting point is a pole
    if (Math.Cos(ilat) < 0.00000001)
    {
            if (ilat > 0)
            {
                    tc = Math.PI;
            }
            else
            {
                    tc = 2 * Math.PI;
            }
    }

    //Convert true course into degrees
    headingDeg = (tc * 180.0 / Math.PI);
    if (headingDeg < 0)
    {
            headingDeg += 360.0;
    }
    if (headingDeg > 360)
    {
            headingDeg = headingDeg - 360;
    }
    //Convert the distance into kilometers
    distanceKm = 6378.0 * d;

    //Set heading
```

```
                simconnect.TransmitClientEvent(objectID,
                    KEY_EVENTS.KEY_HEADING_BUG_SET, Convert.
                    ToUInt32(headingDeg), GROUP.
                    ID_PRIORITY_STANDARD, SIMCONNECT_EVENT_FLAG.
                    GROUPID_IS_PRIORITY);

                //The iteration rate
                Thread.Sleep(rate);
        }
}
```

C.5 Take Off

```
...
[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Ansi,
    Pack = 1)]
struct Structgear
{
        public double gear;
};

[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Ansi,
    Pack = 1)]
struct Structthro
{
        public double throttle;
};

[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Ansi,
    Pack = 1)]
struct Structthro1
{
        public double throttle1;
};
...
enum DEFINITIONS
{
Structgear,
Structthro,
```

```
Structthro1,
};
...
simconnect.MapClientEventToSimEvent(KEY_EVENTS.KEY_AUTOPILOT_ON,
    "autopilot_on");
simconnect.MapClientEventToSimEvent(KEY_EVENTS.
    KEY_AP_ALT_VAR_SET_ENGLISH, "AP_ALT_VAR_SET_ENGLISH");
simconnect.MapClientEventToSimEvent(KEY_EVENTS.
    KEY_AP_ALT_HOLD_ON, "AP_ALT_HOLD_ON");
simconnect.AddToDataDefinition(DEFINITIONS.Structthro, "GENERAL␣
    ENG␣THROTTLE␣LEVER␣POSITION:1", null, SIMCONNECT_DATATYPE.
    FLOAT64, 0.0f, SimConnect.SIMCONNECT_UNUSED);
simconnect.RegisterDataDefineStruct<Structthro>(DEFINITIONS.
    Structthro);
simconnect.AddToDataDefinition(DEFINITIONS.Structthro1, "GENERAL
    ␣ENG␣THROTTLE␣LEVER␣POSITION:2", null, SIMCONNECT_DATATYPE.
    FLOAT64, 0.0f, SimConnect.SIMCONNECT_UNUSED);
simconnect.RegisterDataDefineStruct<Structthro1>(DEFINITIONS.
    Structthro1);
simconnect.AddToDataDefinition(DEFINITIONS.Structgear, "GEAR␣
    POSITION:0", null, SIMCONNECT_DATATYPE.FLOAT64, 0.0f,
    SimConnect.SIMCONNECT_UNUSED);
simconnect.RegisterDataDefineStruct<Structgear>(DEFINITIONS.
    Structgear);
...
void takeOff(uint altRef)
{
//Initialize the parameter structures
Structthro t1 = new Structthro();
Structthro1 t2 = new Structthro1();
Structgear sgear = new Structgear();
//Set throttle on engine 1
t1.throttle = 1;
simconnect.SetDataOnSimObject(DEFINITIONS.Structthro, objectID,
    SIMCONNECT_DATA_SET_FLAG.DEFAULT, t1);
//Set throttle on engine 2
t2.throttle1 = 1;
```

```
simconnect.SetDataOnSimObject(DEFINITIONS.Structthro1, objectID,
    SIMCONNECT_DATA_SET_FLAG.DEFAULT, t2);

//Turn autopilot on
simconnect.TransmitClientEvent(objectID, KEY_EVENTS.
   KEY_AUTOPILOT_ON, 0, GROUP.ID_PRIORITY_STANDARD,
   SIMCONNECT_EVENT_FLAG.GROUPID_IS_PRIORITY);
//Set desired altitude
simconnect.TransmitClientEvent(objectID, KEY_EVENTS.
   KEY_AP_ALT_VAR_SET_ENGLISH, altRef, GROUP.
   ID_PRIORITY_STANDARD, SIMCONNECT_EVENT_FLAG.
   GROUPID_IS_PRIORITY);

//Turn altitude hold mode on
simconnect.TransmitClientEvent(objectID, KEY_EVENTS.
   KEY_AP_ALT_HOLD_ON, 0, GROUP.ID_PRIORITY_STANDARD,
   SIMCONNECT_EVENT_FLAG.GROUPID_IS_PRIORITY);
sgear.gear = 1;
Thread.Sleep(40000);
//Gear up
simconnect.SetDataOnSimObject(DEFINITIONS.Structgear, objectID,
   SIMCONNECT_DATA_SET_FLAG.DEFAULT, sgear);
}
```

C.6 Set on ground

```
...
[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Ansi,
   Pack = 1)]
struct Structgear
{
        public double gear;
};

[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Ansi,
   Pack = 1)]
struct Structthro
{
        public double throttle;
```

```
};

[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Ansi,
    Pack = 1)]
struct Structthro1
{
        public double throttle1;
};
...
enum DEFINITIONS
{
Structgear,
Structthro,
Structthro1,
};
...
simconnect.MapClientEventToSimEvent(KEY_EVENTS.KEY_AUTOPILOT_ON,
     "autopilot_on");
simconnect.MapClientEventToSimEvent(KEY_EVENTS.
    KEY_AP_ALT_VAR_SET_ENGLISH, "AP_ALT_VAR_SET_ENGLISH");
simconnect.MapClientEventToSimEvent(KEY_EVENTS.
    KEY_AP_ALT_HOLD_ON, "AP_ALT_HOLD_ON");
simconnect.AddToDataDefinition(DEFINITIONS.Structthro, "GENERAL␣
    ENG␣THROTTLE␣LEVER␣POSITION:1", null, SIMCONNECT_DATATYPE.
    FLOAT64, 0.0f, SimConnect.SIMCONNECT_UNUSED);
simconnect.RegisterDataDefineStruct<Structthro>(DEFINITIONS.
    Structthro);
simconnect.AddToDataDefinition(DEFINITIONS.Structthro1, "GENERAL
    ␣ENG␣THROTTLE␣LEVER␣POSITION:2", null, SIMCONNECT_DATATYPE.
    FLOAT64, 0.0f, SimConnect.SIMCONNECT_UNUSED);
simconnect.RegisterDataDefineStruct<Structthro1>(DEFINITIONS.
    Structthro1);
simconnect.AddToDataDefinition(DEFINITIONS.Structgear, "GEAR␣
    POSITION:0", null, SIMCONNECT_DATATYPE.FLOAT64, 0.0f,
    SimConnect.SIMCONNECT_UNUSED);
simconnect.RegisterDataDefineStruct<Structgear>(DEFINITIONS.
    Structgear);
...
```

```
void setOnGround()
{
        Structthro t1 = new Structthro();
        Structthro1 t2 = new Structthro1();
        Structgear sgear = new Structgear();
        //Reduce the throttle
        t1.throttle = 0.5;
        simconnect.SetDataOnSimObject(DEFINITIONS.Structthro,
            objectID, SIMCONNECT_DATA_SET_FLAG.DEFAULT, t1);
        t2.throttle1 = 0.5;
        simconnect.SetDataOnSimObject(DEFINITIONS.Structthro1,
            objectID, SIMCONNECT_DATA_SET_FLAG.DEFAULT, t2);
        //Turn autopilot on
        simconnect.TransmitClientEvent(objectID, KEY_EVENTS.
            KEY_AUTOPILOT_ON, 0, GROUP.ID_PRIORITY_STANDARD,
            SIMCONNECT_EVENT_FLAG.GROUPID_IS_PRIORITY);
        //Set altitude 0
        simconnect.TransmitClientEvent(objectID, KEY_EVENTS.
            KEY_AP_ALT_VAR_SET_ENGLISH, 0, GROUP.
            ID_PRIORITY_STANDARD, SIMCONNECT_EVENT_FLAG.
            GROUPID_IS_PRIORITY);
        //Turn altitude hold mode on
        simconnect.TransmitClientEvent(objectID, KEY_EVENTS.
            KEY_AP_ALT_HOLD_ON, 0, GROUP.ID_PRIORITY_STANDARD,
            SIMCONNECT_EVENT_FLAG.GROUPID_IS_PRIORITY);
        //Gear down
        sgear.gear = 2;
        simconnect.SetDataOnSimObject(DEFINITIONS.Structgear,
            objectID, SIMCONNECT_DATA_SET_FLAG.DEFAULT, sgear);
}
```

C.7 Find Surface

```
...
surfType='current surface type';
sCondition='current surface condition';
...
private void findSurface(object param)
{
```

```
        int exit = 0;
        while(exit==0)
        {
                if(surfType.CompareTo('chosen surface type')==0
                   && surfCondition.CompareTo('chosen surface
                    condition')==0)
                {
                        MessageBox.Show("Found at lat: "+ '
                           current latitude'+" long: "+'current
                           longitude');
                        exit = 1;
                }
                Thread.Sleep(rate);
        }
}
```

C.8 Circle control

```
...
simconnect.MapClientEventToSimEvent(KEY_EVENTS.KEY_AUTOPILOT_ON,
    "autopilot_on");
simconnect.MapClientEventToSimEvent(KEY_EVENTS.
   KEY_HEADING_BUG_SET, "HEADING_BUG_SET");
simconnect.MapClientEventToSimEvent(KEY_EVENTS.
   KEY_AP_HDG_HOLD_ON, "AP_HDG_HOLD_ON");
simconnect.MapClientEventToSimEvent(KEY_EVENTS.
   KEY_AP_ALT_VAR_SET_ENGLISH, "AP_ALT_VAR_SET_ENGLISH");
...
private void circleControl(object hdgRef)
{
        double d, tc, headingDeg, distanceKm = 1000, ilat, ilong
           , dlat, dlong;
        bool comp=true;
        dlat = 'second point latitude' * Math.PI / 180.0;
        dlong = 'second point longitude' * Math.PI / 180.0;

        //Turn autopilot on
        simconnect.TransmitClientEvent(objectID, KEY_EVENTS.
           KEY_AUTOPILOT_ON, 0, GROUP.ID_PRIORITY_STANDARD,
```

```
    SIMCONNECT_EVENT_FLAG.GROUPID_IS_PRIORITY);
//Turn heading hold module on
simconnect.TransmitClientEvent(objectID, KEY_EVENTS.
    KEY_AP_HDG_HOLD_ON, 0, GROUP.ID_PRIORITY_STANDARD,
    SIMCONNECT_EVENT_FLAG.GROUPID_IS_PRIORITY);
//Set altitude
simconnect.TransmitClientEvent(objectID, KEY_EVENTS.
    KEY_AP_ALT_VAR_SET_ENGLISH, 'desired altitude', GROUP
    .ID_PRIORITY_STANDARD, SIMCONNECT_EVENT_FLAG.
    GROUPID_IS_PRIORITY);
while (endCondition==0)
{
        while (distanceKm > 0.5)
        {
                ilat = 'current latitude' * Math.PI /
                    180.0;
                ilong = 'current longitude' * Math.PI /
                    180.0;

                //true course
                d = Math.Acos(Math.Sin(ilat) * Math.Sin(
                    dlat) + Math.Cos(ilat) * Math.Cos(
                    dlat) * Math.Cos(dlong - ilong));
                if (Math.Sin(ilong - dlong) < 0)
                {
                        tc = Math.Acos((Math.Sin(dlat) -
                             Math.Sin(ilat) * Math.Cos(d)
                            ) / (Math.Sin(d) * Math.Cos(
                            ilat)));
                }
                else
                {
                        tc = 2.0 * Math.PI - Math.Acos((
                            Math.Sin(dlat) - Math.Sin(
                            ilat) * Math.Cos(d)) / (Math.
                            Sin(d) * Math.Cos(ilat)));
                }
                if (Math.Cos(ilat) < 0.00000001)
```

```
                {
                        if (ilat > 0)
                        {
                                tc = Math.PI;
                        }
                        else
                        {
                                tc = 2 * Math.PI;
                        }
                }
                headingDeg = (tc * 180.0 / Math.PI);
                if (headingDeg < 0)
                {
                        headingDeg += 360.0;
                }
                if (headingDeg > 360)
                {
                        headingDeg = headingDeg - 360;
                }
                distanceKm = 6378.0 * d;
                //Set heading
                simconnect.TransmitClientEvent(objectID,
                     KEY_EVENTS.KEY_HEADING_BUG_SET,
                    Convert.ToUInt32(headingDeg), GROUP.
                    ID_PRIORITY_STANDARD,
                    SIMCONNECT_EVENT_FLAG.
                    GROUPID_IS_PRIORITY);
                Thread.Sleep(rate);
        }
                //Switch the points
                comp=!comp;

                if (comp == true)
                {
                dlat = 'first point latitude' * Math.PI
                    / 180.0;
                dlong = 'first point longitude' * Math.
                    PI / 180.0;
```

```
                        distanceKm = 100;

                        }
                        if (comp == false)
                        {
                        dlat = 'second point latitude' * Math.PI
                            / 180.0;
                        dlong = 'second point longitude' * Math.
                           PI / 180.0;
                        distanceKm = 100;
                        }
        }
}
```

C.9 Helicoidal control

```
...
simconnect.MapClientEventToSimEvent(KEY_EVENTS.KEY_AUTOPILOT_ON,
    "autopilot_on");
simconnect.MapClientEventToSimEvent(KEY_EVENTS.
   KEY_HEADING_BUG_SET, "HEADING_BUG_SET");
simconnect.MapClientEventToSimEvent(KEY_EVENTS.
   KEY_AP_HDG_HOLD_ON, "AP_HDG_HOLD_ON");
simconnect.MapClientEventToSimEvent(KEY_EVENTS.
   KEY_AP_ALT_VAR_SET_ENGLISH, "AP_ALT_VAR_SET_ENGLISH");
...
private void heliControl(object hdgRef)
{
        double d, tc, headingDeg, distanceKm = 1000, ilat, ilong
           , dlat, dlong,lap;
        bool comp = true;
        dlat = 'first point latitude' * Math.PI / 180.0;
        dlong = 'second point latitude' * Math.PI / 180.0;

        //Turn autopilot on
        simconnect.TransmitClientEvent(objectID, KEY_EVENTS.
           KEY_AUTOPILOT_ON, 0, GROUP.ID_PRIORITY_STANDARD,
           SIMCONNECT_EVENT_FLAG.GROUPID_IS_PRIORITY);
```

```
simconnect.TransmitClientEvent(objectID, KEY_EVENTS.
    KEY_AP_HDG_HOLD_ON, 0, GROUP.ID_PRIORITY_STANDARD,
    SIMCONNECT_EVENT_FLAG.GROUPID_IS_PRIORITY);
lap = Math.Abs('current altitude' - 'final altitude')
    /(2.0*'number of laps');
while (true)
{
        while (distanceKm > 0.5)
        {
                ilat = 'current latitude' * Math.PI /
                    180.0;
                ilong = 'current longitude' * Math.PI /
                    180.0;
                d = Math.Acos(Math.Sin(ilat) * Math.Sin(
                    dlat) + Math.Cos(ilat) * Math.Cos(
                    dlat) * Math.Cos(dlong - ilong));
                if (Math.Sin(ilong - dlong) < 0)
                {
                        tc = Math.Acos((Math.Sin(dlat) -
                             Math.Sin(ilat) * Math.Cos(d)
                            ) / (Math.Sin(d) * Math.Cos(
                            ilat)));
                }
                else
                {
                        tc = 2.0 * Math.PI - Math.Acos((
                            Math.Sin(dlat) - Math.Sin(
                            ilat) * Math.Cos(d)) / (Math.
                            Sin(d) * Math.Cos(ilat)));
                }
                if (Math.Cos(ilat) < 0.00000001)
                {
                        if (ilat > 0)
                        {
                                tc = Math.PI;
                        }
                        else
                        {
```

```
                        tc = 2 * Math.PI;
                }
        }
        headingDeg = (tc * 180.0 / Math.PI);
        if (headingDeg < 0)
        {
                headingDeg += 360.0;
        }
        if (headingDeg > 360)
        {
                headingDeg = headingDeg - 360;
        }
        distanceKm = 6378.0 * d;
        //set heading
        simconnect.TransmitClientEvent(objectID,
             KEY_EVENTS.KEY_HEADING_BUG_SET,
            Convert.ToUInt32(headingDeg), GROUP.
            ID_PRIORITY_STANDARD,
            SIMCONNECT_EVENT_FLAG.
            GROUPID_IS_PRIORITY);
        Thread.Sleep(rate);
        }
simconnect.TransmitClientEvent(objectID,
    KEY_EVENTS.KEY_AP_ALT_VAR_SET_ENGLISH,
    Convert.ToUInt32('current altitude' - lap),
    GROUP.ID_PRIORITY_STANDARD,
    SIMCONNECT_EVENT_FLAG.GROUPID_IS_PRIORITY);
if ('current altitude' <= 'final altitude')
{
        simconnect.TransmitClientEvent(objectID,
             KEY_EVENTS.
            KEY_AP_ALT_VAR_SET_ENGLISH, Convert.
            ToUInt32('final altitude'), GROUP.
            ID_PRIORITY_STANDARD,
            SIMCONNECT_EVENT_FLAG.
            GROUPID_IS_PRIORITY);
        break;
}
```

```
                comp = !comp;
                if (comp == true)
                {
                        dlat = 'first point latitude' * Math.PI
                           / 180.0;
                        dlong = 'first point longitude' * Math.
                           PI / 180.0;
                        distanceKm = 100;
                }
                if (comp == false)
                {
                        dlat = 'second point latitude' * Math.PI
                            / 180.0;
                        dlong = 'second point longitude' * Math.
                           PI / 180.0;
                        distanceKm = 100;
                }
        }
}
```

Appendix D

Mission development

Object Placement Tool Commands:

Scenery>GEN_SceneryEffect GEN_SceneryEffect
Create a generic scenery object. A generic scenery object is a void, invisible object normally used as an effect place holder.
Referenced by:
AttachFireEffect
FireOut
References:
-

Action>AttachEffectAction AttachFireEffect
This will attach an effect to previously created object.
Parameters: Effect name: fx_forestfire.fx
AttachPointName: attachpt_GenericEffect
Referenced by:
InitialTimer
References:
GEN_SceneryEffect

Trigger>Timer trigger InitialTimer
This is used to initiate the fire effect, add water to the vehicle and alert SimConnect that the mission as started.
Referenced by:
-
References:

AddWaterDrops
SendSimConnectInit
AttachFireEffect

AreaDefinition>RectangleArea FireArea
This is used to define the area of the fire. This are is used to delimit the area that delimits the effective fire area. The objects (water drops) dropped into this area will be used to fire a trigger(HitTrigger).
Referenced by:
HitTrigger
References:
-

Hit trigger Trigger>Proximity trigger HitTrigger
This trigger fires when a certain object enters a determined area (FireArea).
Parameters:
ObjectFilter: Anything
OneShot: False
Area: FireArea
Referenced by:
-
References:
IncrementCounter
FireArea

Action>CountAction IncrementCounter
This will count 1 unit every time the HitTrigger fires.
Count: 1
Referenced by:
HitTrigger References:
HitCounter

Counter trigger Trigger>CounterTrigger HitCounter
This receives counting steps from hitcounter until it reaches a specified number, then fires a trigger that indicates SimConnect that the fire is extinguished and also deactivates the fire.
Stop count: 4
Actions: SendSimConnectFireOut
SendSimConnectFireOut

Referenced by:
HitCounter
References:
FireOutMessage
SendSimConnectFireOut
FireOut

Add Water Action>AttachDroppablePayloadAction AddWaterDrop
This is used to attach one or more droppable objects to a vehicle.
PayloadName: Water Drop
PayloadCount: 10
ObjectReference: The_Player
Referenced by:
InitialTimer
References:
-
Note: Although not referenced in the SDK documentation, the functionality for adding objects to multiple vehicles is not yet available for this version of Flight Simulator.

Action>ObjectActivationAction FireOut
This is used to activate or deactivate object, in this case it is used to deactivate the fire.
ObjectState: False
ObjectReference: GEN_SceneryEffect
TargetPlayer: All Players
Referenced by:
HitCounter
References:
GEN_SceneryEffect

Action>CustomAction This is used to send a message to SimConnect.
SendSimConnectInit
PayloadString: Mission started!
Referenced by:
InitialTimer
References:
-
SendSimConnectFireOut
PayloadString: Fire out
Referenced by:

HitCounter
References:
-

Action>DialogAction FireOutMessage
This is used to display text on screen. Text: Well done!
TargetPlayer: All players

Fire truck MobileScenery>Veh_Truck_Fire_new_1 Fire truck
Is an AI vehicle.

Accident Plane Scenery>GEN_Plane_A380 Plane
Is the accident plane.