M

O

N

A

S

H

FOURTH YEAR ELECTRICAL AND COMPUTER SYSTEMS ENGINEERING
4th YEAR THESIS PROJECT ECE

UAV – Visualization using a Flight Simulator

Supervisor: Professor Greg Egan

Evan Price

DEPARTMENT OF ELECTRICAL AND COMPUTER SYSTEMS ENGINEERING
MONASH UNIVERSITY, CLAYTON, VICTORIA 3168, AUSTRALIA

Abstract

This thesis describes a graphical visualization system for a UAV which was successfully implemented using Microsoft Flight Simulator. The graphical display consists of a 3D simulation of the aircrafts position and orientation as well as a graphical instrument panel which displays important flight parameters.

To send the UAVs flight data to the simulator a Windows application was written using C++. This application reads the flight data from either a log file, or from the computer's serial port in real-time, converts this data into the format required by the simulator, and then pipes it to an instance of Microsoft Flight Simulator.

The instrument panel for the display was designed in consultation with the pilots of the UAV and built using Flight Simulator's XML gauge system; it displays such information as the aircrafts speed, altitude, position, battery voltages, temperature levels, and the current state of its autopilot and avionics systems.

A 3D graphical model of the UAV was constructed using the 3D modeling program *gmax*. The model was converted to a Flight Simulator Aircraft to be displayed in the 3D simulation window.

Satellite images and aerial photos of areas in which the UAV is usually flown were collected and mapped to the terrain's surface to improve the realism of the simulation.

The complete system that was implemented, and detailed in this document, offers a fully functional system capable of providing crucial information about the state of the UAV during its flight to the pilots on the ground.

Declaration

I declare that this thesis is my own work and has not been submitted in any form for another degree or diploma at any university or other institution of tertiary education. Information derived from the published or unpublished work of others has been acknowledged in the text and a list of references is given.

Evan Price

Acknowledgements

I extend my sincere gratitude and appreciation to many people who made this undergraduate thesis possible. Special thanks are due to my supervisor Professor Greg Egan for accepting me to carry out this project and for his support, guidance and availability. I thoroughly enjoyed working and learning from him.

Many thanks go to Ray Cooper and the rest of the Aerobotics Research Group for their ideas and input into the design and layout of the graphical instrument panel.

Evan Price

Contents

List of Tables

List of Figures

Chapter 1

Introduction

An unmanned aerial vehicle (UAV) is a self-descriptive term used to describe the latest generation of pilotless aircraft. The modern UAV originated in the early 1970s and promises to transform both military and civilian aerospace operations. The attractiveness of UAVs is their ability to do the dirty, dangerous and dull jobs autonomously.

Almost all of the UAVs currently in operation are military aircraft. Most of them are used for reconnaissance, targeting and battle damage indication, while some are armed with missiles for active combat. In the near future, UAVs are expected to be used more widely for civilian missions as well. Some of the possible civilian uses include:

- Agricultural crop spraying
- Inspecting pipelines and electrical power lines
- Atmospheric research
- Traffic/security surveillance
- Data-link relay

Director of the association for unmanned vehicle systems international (AUVSI) says, “they're going to be used in commercial markets for things we haven't even thought of” [1].

The Aerobotics© (Aerial Robotics) Research Group at Monash University, is interested in all aspects of the design, construction and application of electrically powered UAVs. The research group’s current aircraft transmit a significant amount of information while in flight. This project aims to display this data within a flight simulator package to obtain a pilots eye view of the aircrafts behavior in real-time to assist with flying the aircraft. This system will be particularly useful when the UAV is flown out of sight, giving the pilots on the ground information of its current position and orientation. Another very useful aspect of the system is the displaying of warning alarms such as low battery voltage and motor temperature too high.

The display was to include both a 3D simulation of the UAV as well as a graphical instrument panel displaying the relevant flight data to the pilot. The design of the instrument panel was conducted in consultation with the pilots of the UAV.

This document provides a detailed description of the design and implementation of the complete visualization system.

Chapter 2

Objectives

The project deliverables were divided into three distinct parts; an application to pipe the collected data into an instance of Microsoft Flight Simulator; a graphical model of the UAV for the 3D simulation, and an instrument panel displaying all the necessary flight information. An optional extension to the project was to create more realistic terrain for the areas in which the UAV is flown. A diagram showing the complete system is shown in Figure 2.1 below.

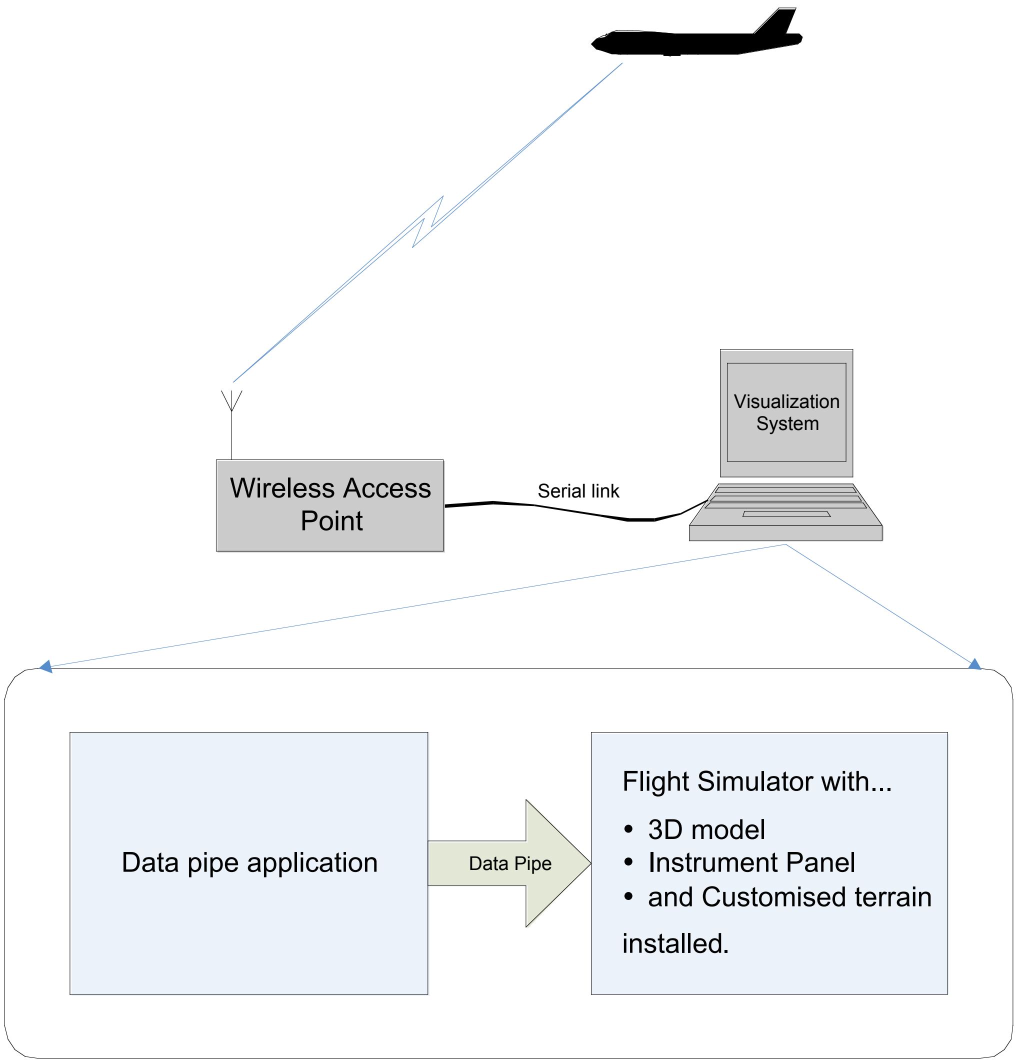


Figure 2.1 Complete system diagram

The three main objectives as well the optional extension were successfully implemented. This chapter provides an overview of each of these objectives.

2.1 Application to pipe flight data to a flight simulator

This application was required to read in the data collected by the UAV, convert this data into the format required by Flight Simulator, and then pipe it to an instance of Flight Simulator. The application was required to read the data from three different sources:

1. A text file with the data stored in the MicroPilot format.
2. A binary file with the data stored in the Monash binary format.
3. The serial port in real-time with the data in the Monash binary format.

Details of the Micropilot format and Monash binary format are described in section 4.1.

The application was also required to implement some smoothing and interpolation of the incoming data as the data is only sent from the UAV once every second; whereas the flight simulator requires about 20 frames per second for smooth simulation.

2.2 Constructing a 3D model of the UAV

A scale model of the UAV was to be made using a graphical modeling package. The model was then to be converted to a new flight simulator aircraft so that when simulating the UAV it would actually appear as the UAV rather than one of the standard aircrafts supplied with the flight simulator.

2.3 Creating a graphical instrument panel

An instrument panel was to be designed that displays all the relevant flight data, such as:

- Position and orientation (latitude, longitude, altitude, heading, pitch, roll, etc)
- Battery voltages and warning alarms
- Temperature levels and warning alarms
- Navigational information

The content and layout of the panel was to be designed in consultation with those who fly the UAV with the aim of designing a panel which displays the most important information in an easy to read format without overloading the pilot with unnecessary information.

2.4 Customizing the terrain

The standard terrain used within Microsoft Flight Simulator contains major roads and landmarks but is mostly just computer generated from knowledge of the population density in a particular area. To provide more realistic terrain, satellite and/or aerial photos were to be collected for the areas in which the UAV is flown and mapped to the terrain surface.

Chapter 3

Choice of flight simulator software

Two possible choices of flight simulator software were considered; FlightGear and Microsoft Flight Simulator. The features of both flight simulators were researched and it was concluded that both simulators provided all the necessary features to meet the objectives of this project. However, it was also concluded that the documentation provided with Microsoft Flight Simulator was more comprehensive and understandable than that provided with FlightGear. Based mainly on this reason Microsoft Flight Simulator was chosen to use for this project. This chapter gives an overview of the two flight simulators, outlining the basis for the choice of simulator. Also included is an overview of all the Microsoft Flight Simulator SDKs which were used in this project.

3.1 FlightGear

The FlightGear flight simulator is an open-source, multi-platform, cooperative flight simulator development project [2]. Source code for the entire project is available free under the GNU General Public License [3]. Because FlightGear's code is open-source the simulator is very flexible, allowing any desired modifications to be made. The downside of using FlgihtGear however is the limited documentation that it provides. When comparing FlightGear to Microsoft Flight Simulator a research group from the University of South Florida concluded, "The scarce documentation makes its modification a time-consuming and error-prone process" [4].

Given the limited timeframe that was available to complete this project, it was felt that it would take too much time to learn how to use FlightGear, and thus Microsoft Flight Simulator was considered the better option for this project.

3.2 Microsoft® Flight Simulator

Microsoft Flight Simulator 2004 is a proprietary flight simulator developed by Microsoft Corporation [5]. Although it is aimed mainly at the gaming market, it must be said that the program is primarily a simulation, not a game. Flight Simulator provides a high degree of realism and indeed for this reason it is widely used by pilots for flight training.

Although Flight Simulator is not an open-source program like FlightGear is does comprise a structure that allows users to modify the game content. Individual aspects that can be edited include panel layout and gauges, aircraft model, aircraft model textures, aircraft flight characteristic models, scenery models and scenery textures. The other imperative feature of Flight Simulator is the ability to drive the simulation from an external data set, in our case from a UAV.

Microsoft provides a number of downloadable Software Development Kits (SDKs) from the Flight Simulator website which document how to customize Flight Simulator and also provides a few tools for doing so.

3.3 Software Development Kits (SDKs)

Contained in this section is an overview of all the SDKs for Microsoft Flight Simulator that were used throughout this project.

3.3.1 Netpips: Data Input/Output

This SDK describes the format in which data must be sent to Flight Simulator. The format is the same as the .fsr file which is produced by the flight video recording feature of Flight Simulator. It is a binary file and is organized around frames of data representing a snapshot of the simulation and environment at a moment in time. Each data-frame is time stamped to allow for synchronization on playback. Full details of the format are described in section 4.3.

The SDK also provides details on how to set-up a pipe to an instance of Flight Simulator to pass external data to the simulator.

3.3.2 Building Panels and Gauges

This SDK provides all the necessary information to create new gauges and panels for aircraft in Flight Simulator. It includes the following information:

- An overview of the Panel system within Flight Simulator.
- A discussion of how to write the necessary code to make the gauges work.
- A programming reference listing the functions and variables available to create panels and gauges.
- A complete list of all the parameters used in flight simulator.
- Tips on creating artwork for panels and gauges.

Microsoft Flight Simulator provides two ways of programming gauges; C code and XML. The SDK provides sample C code for several typical gauges. It does not provide any sample code for XML gauges; however the XML files from the panels supplied with Flight Simulator are stored in .cab files and can be extracted and viewed.

More information about the Panel system within Flight Simulator is given in chapter 5.

3.3.3 Aircraft Container System

This SDK provides details of the aircraft container system's organizational structure. An aircraft "container" is simply an aircraft folder within the FS9\Aircraft directory. The SDK provides details of the file hierarchy within this container and also explains the structure of and parameters in the various configuration files contained in it. The aircraft container includes the following configuration files:

- **aircraft.cfg –** configuration of overall aircraft system
- **panel.cfg –** configuration of the panel and its gauges
- **model.cfg** – configuration of the aircrafts 3D model
- **sound.cfg –** configuration of the aircrafts sound files

3.3.4 Make Model

Contained in this SDK is an application *MakeMDL.exe* which converts 3D models, created using a 3D modeling package, into aircraft or scenery objects that can be used by Microsoft Flight Simulator. Information on using this program is also supplied with the SDK.

3.3.5 gmax Aircraft Creation

Microsoft Flight Simulator supports models created in *gmax*, a 3D modeling tool from Discreet [6]. By using *gmax*, one can create a model of an aircraft, apply textures and animations to that model, and then export the model to a Flight Simulator aircraft using the *MakeMDL* program described above. The documentation provided with this SDK includes a set of best practices to follow when using *gmax* to build aircraft.

3.3.6 Terrain

This SDK describes how to use a Digital Elevation Model (DEM) to customize the terrain in Microsoft Flight Simulator and how to add custom satellite or aerial imagery to the terrain. It also provides an application for each of these processes, to convert the custom data into the required format for Microsoft Flight Simulator.

Chapter 4

Data pipe application

In order to drive Microsoft Flight Simulator with external data from the UAV a Windows application was written using Visual C++. This application reads in the flight data from a selected source (either from a log file or from the serial port), converts the data into the format required by Microsoft Flight Simulator, and sends this data to the simulator via a pipe. This chapter seeks to detail the main functionality of this application. Full listing of the code is provided in Appendix C.

4.1 Input data formats

The application was required to input flight data stored in two different formats. The first format is a text log file, while the second format is a binary data stream which is sent via the serial link in real-time. As well as being able to receive this data stream from the serial port in real-time, the application was also given the functionality to read this same data from a binary file. This allows a flight to be recorded and played back at a later time.

Details of the two input data formats will be outlined in this section.

4.1.1 The MicroPilot log file format

One of the autopilots being used by the Aerobotics Research Group is the MicroPilot MP2028 produced by a Canadian Company [7]. It produces a text log file with the following fields:

Table 4.1 MicroPilot log file format

Column	Data	Unit
1	Time	hh:mm:ss
2	Longitude	Degrees and decimal degrees
3	Latitude	Degrees and decimal degrees
4	Pitch	Radians times 1024
5	Roll	Radians times 1024
6	Airspeed	Feet per second
7	GPS Speed	Feet per second
8	Altitude	Feet times -8
9	Vertical Speed	Feet time -8 per second
10	Heading	Degrees times 100
11	Error field	
12	Status bitfield	

13	Main battery voltage	Volts times 100
14	Servo battery voltage	Volts times 100
15	Throttle position	Percentage times 2.55
16	Status2 bitfield	
17	Empty	
18	Empty	

Each field is separated with a space, except after columns 7, 8 and 9 which are followed by a comma and then a space. A typical line of data is shown below:

```
13:20:39 145.209949E  37.881020S  352 224 68 66, -3192, -210, 26633 0 141 580 576 137 1 0 0
```

The log file contains approximately 10 data records every second. However, all the data is not updated every record, some data fields such as the latitude and longitude are only updated once every second.

4.1.2 The Monash binary data format

The data which is sent by the ground station via a serial link comes in the form of three C data structures. Each C structure is preceded by a '$' character and ends with a checksum. The set of data structures are sent once every second.

$	LevState Struct – 50 bytes	Checksum
$	AirframeState Struct – 13 bytes	Checksum
$	NavState Struct – 70 bytes	Checksum

Figure 4.1 Format of data from ground station

The C file (common.h) where these structures are defined can be found in Appendix C. In order to store the data in these structures the common.h file was included in the application's source code. The incoming byte stream could then be copied directly into these structures and used from there.

A difficulty did arise however as the original structures are stored on an 8-bit microcontroller, and thus the members of the data structures are byte aligned, while the members of the data structures on a 32-bit PC are DWORD aligned. Hence, when copying the data stream into the data structures, careful consideration of the alignment of the structures members had to be taken into account.

4.2 Reading data from the serial port

The UAV ground station transmits the current flight data via the serial link once every second. This section outlines the code that was used to read this data from the serial port.

4.2.1 Opening the serial port

The following code was used to open the serial port, with windows.h included as a header file.

```
HANDLE hSerial;

hSerial = CreateFile("COM1",
                  GENERIC_READ,
                  0,
                  0,
                  OPEN_EXISTING,
                  FILE_ATTRIBUTE_NORMAL,
                  0);

if (hSerial == INVALID_HANDLE_VALUE)
{
      if (GetLastError() == ERROR_FILE_NOT_FOUND)
      {
            //serial port does not exist. Inform user.
      }
      //some other error occurred. Inform user.
}
```

Figure 4.2 Code to open serial port

First a variable of type `HANDLE` is declared and initialized with a call to `CreateFile`. The first argument to `CreateFile` is the name of the file to open, which is usually `COM1`. The next argument tells Windows that we want to read from the serial port rather than write. The 5th argument simply tells windows that the serial port already exists, while argument 6 just tells windows that we don’t want to do anything fancy here.

4.2.2 Setting parameters

Once a handle to the serial port has been made the next step is to set the parameters for it. Windows does this through a data structure called `DCB`.

```
DCB dcbSerialParams = {0};

dcbSerialParams.DCBlength = sizeof(dcbSerialParams);

if (!GetCommState(hSerial, &dcbSerialParams))
{
      //error getting state
}

dcbSerialParams.BaudRate = CBR_4800;
dcbSerialParams.ByteSize = 8;
dcbSerialParams.StopBits = ONESTOPBIT;
dcbSerialParams.Parity = NOPARITY;

if (!SetCommState(hSerial, &dcbSerialParams))
{
      //error setting serial port state
}
```

Figure 4.3 Code to set the serial port parameters

First, a variable of type `DCB` is created with all its fields cleared and its size parameter set. The `GetCommState` function is then used to fill in the parameters for the serial port, namely the baud rate, byte size, number of stop bits and parity. Functionality was later given to change these constants within the user interface.

4.2.3 Reading data

Once the serial port is open and set up with the correct parameters, reading the actual data is straight forward. The following code was used to read one byte at a time from the serial port.

```
const int n = 1;
char szBuff[n + 1] = {0};
DWORD dwBytesRead = 0;

If (!ReadFile(hSerial, szBuff, n, &dwBytesRead, NULL))
{
      //error occurred. Report to user.
}
```

Figure 4.4 Code used to read data from the serial port

`ReadFile` takes in a `HANDLE` to a file (the serial port in our case), a buffer to store the data in, the number of bytes to read, a pointer to an integer that will be set to the number of bytes actually read, and `NULL`.

4.2.4 Setting timeouts

One of the problems with serial port communications is that if there isn't any data coming into the serial port (if the serial port device gets disconnected or turned off, for example) then attempting to read from the port can cause the application to hang while waiting for data to show up. The problem was avoided through the use of timeouts.

```
COMMTIMEOUTS timeouts = {0};

timeouts.ReadIntervalTimeout = MAXDWORD;
timeouts.ReadTotalTimeoutConstant = 0;
timeouts.ReadTotalTimeoutMultiplier = 0;

if(!SetCommTimeouts(hSerial, &timeouts))
{
      //error setting timeouts
}
```

Figure 4.5 Code used to set timeouts for the serial port

The `COMMTIMEOUTS` structure has the following fields:

- `ReadIntervalTimeout` specifies how long (in milliseconds) to wait between receiving characters before timing out.
- `ReadTotalTimeoutConstant` specifies how long to wait (in milliseconds) before returning.
- `ReadTotalTimeoutMultiplier` specifies how much additional time to wait (in milliseconds) before returning for each byte that was requested in the read operation.

By setting `ReadIntervalTimeout` to `MAXDWORD` and the other two fields to zero, any read operations are caused to return immediately with whatever characters are in the buffer. Thus even if the buffer is empty the operation will return immediately.

It was essential that the program execution did not wait at the serial port because Flight Simulator requires a new frame every 50 ms or so (depending on how many frames per second are being sent), and if it doesn't receive the next frame in time the simulation will just hang there until it does. It will then discard the next frame anyway as the time given in its timestamp will already have passed. Hence it is imperative that the reading of data from the serial port and the processing of the data be completed in less time than the time interval between frames.

The code was originally written so that when a packet started to arrive, the application would wait until the complete packet was received. However it was found that this took too long and the simulation would not play smoothly. The serial port operates at 4800 baud and each packet is of about 200 bytes long, which gives a total transmission time of 330 ms, much too long to wait in-between frames. Hence the code was changed so that only the data already in the buffer was read in-between sending frames to Flight Simulator. The alternative approach would have been to have a multithreaded program; one thread to read data from the serial port and one thread to pipe data to Flight Simulator. This however would have added great complexity to the program and was really not necessary.

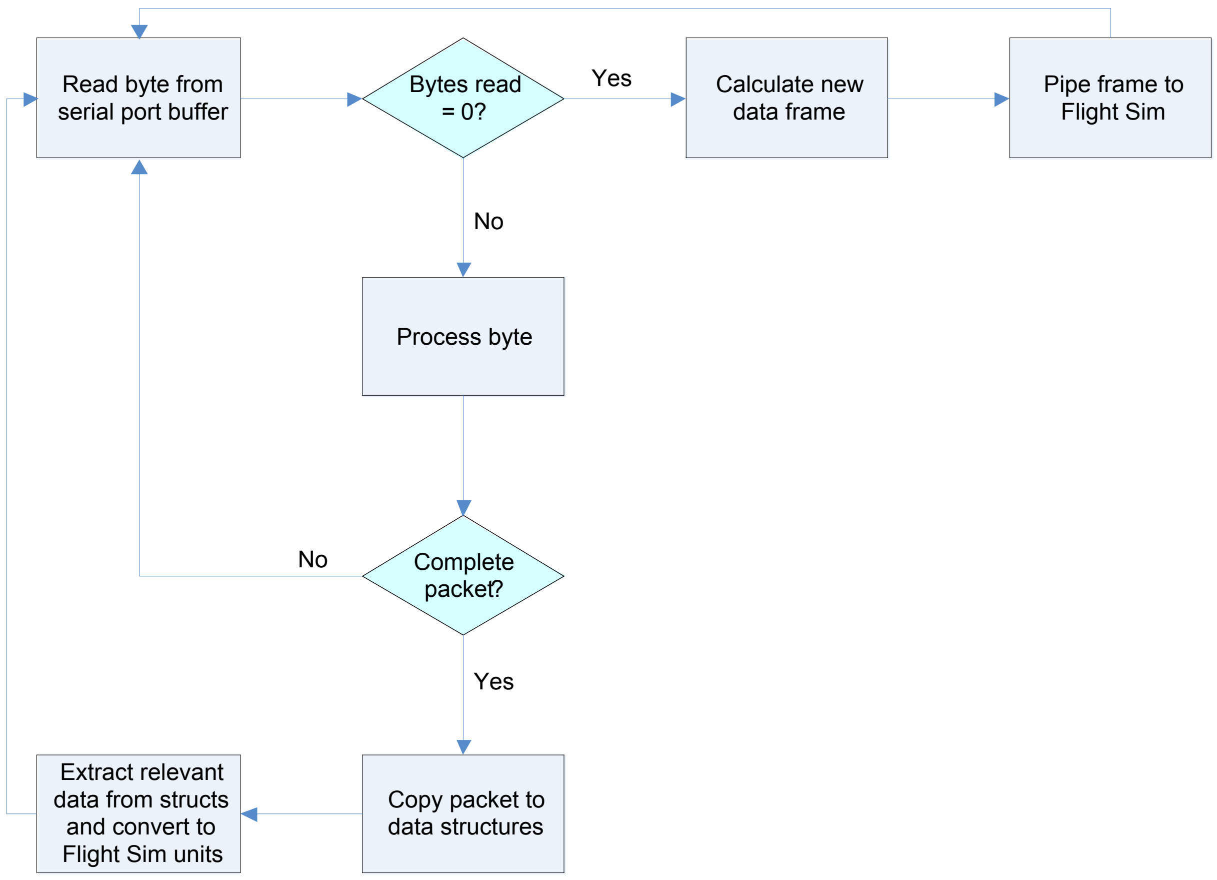

Read byte from serial port buffer
Bytes read = 0?
Yes
Calculate new data frame
Pipe frame to Flight Sim
No
Process byte
No
Complete packet?
Yes
Extract relevant data from structs and convert to Flight Sim units
Copy packet to data structures

Figure 4.6 Algorithm for reading data from the serial port

4.3 The Flight Simulator Recorder .fsr file format

In Microsoft Flight Simulator, the "video" recorder captures data about the simulation and stores it in a file with the extension .fsr. When piping external data to Microsoft Flight

Simulator the same file format must be used. An overview of this file format will be discussed here. For a more detailed description see the *Netpipes: Data Input/Output SDK* available from the Microsoft Flight Simulator website [5].

An .fsr file is organized around frames of data representing a snapshot of the simulation and environment at a moment in time. The data recorded in each data-frame is indexed into a dictionary of properties and objects specified at the top of the data file.

At a high level the format follows this basic structure:

```
<File Header>
<Data Record 1 header>
    <data section header: e.g., object definitions>
    <data section data>
<Data Record 1 footer>
<Data Record 2header>
    <data section header: e.g., property definitions>
    <data section data>
<Data Record 2 footer>
<Data Record 3header>
    <data section header: frame data>
    <frame data section>
    <data section header: frame data>
    …
<Data Record 3 footer>
    …
<Data Record N header>
    <data section header: frame data>
    <frame data section
    …
<Data Record N footer>
<File Trailer>
```

Figure 4.7 Basic structure of Flight Simulator Recorder .fsr file

4.3.1 File Header

The file header is composed of three fields:

- A four-character string. This is always “FSIB” for Flight Simulator 2004.
- A two-character string for the format version. This is always “80” for Flight Simulator 2004.
- A single WORD for the Flight Simulator version. 0x800 for Flight Simulator 2004.

4.3.2 Data Records

Between the file header and the file footer are data records. A data record is composed of a header, followed by some number of data sections, and ending with a footer.

The header has the following form:

- A four character string. This is always “BFIB” in Flight Simulator 2004.
- A DWORD for the total size in bytes, including the header and footer. For this record.

The footer has the following form:

- A reserved DWORD that currently must be set to zero.
- A DWORD for the total size in bytes of the data record. This size should match the size in the header.

4.3.3 Data Section header

The header is of the form:

String – Section ID	DWORD – Section Size

Section ID – A four character string that identifies the section. There are three possible values:

- “ODIP” – Object Definition Section Data
- “PDIP” – Property Dictionary Section Data
- “FRIB” – Frame Section Data

Section Size – A DWORD indicating the size of the section, including the header, in bytes.

4.3.4 Object Definition Section Data

An object definition is a dictionary of objects, where each object is represented as three fields:

WORD – Object ID	WORD – String Length	String – Object Name

Object ID – A WORD used to identify this particular object.

String Length – The length of the object name string, in bytes.

Object Name – Used to map the property values of an instance to a Flight Simulator object. In Flight Simulator 2004 there are two possible values recognized for this field:

- UserAircraft – object representing the user aircraft
- Environment – object which has properties such as time

4.3.5 Property Dictionary Section Data

The property dictionary defines the set of properties for each object in the object definition section:

WORD – Object ID	WORD – Property ID	Enum – Value Type	WORD – Name Size	WORD – Unit Size	String – Property	String – Unit

Object ID – A WORD that must match an Object ID from the object definition section.

Property ID – A WORD that is used in the frame section to identify this property.

Entry – A 2-byte enumeration that indicates what the property type is. Defined values are: (1) Boolean; (2) Number; (3) Ordinal; and (4) String.

Name Size – The length of the property name, in bytes.

Unit Size – The length of the properties unit name, in bytes.

Property – The properties name. A complete list of the property names is given in the *Building Panels and Gauges SDK*.

Unit – The name of the unit that the property is reported in. The list of possible unit names is also given in the *Building Panels and Gauges SDK*.

4.3.6 Frame Section Data

Each frame section begins with a timestamp of the form:

Signed 64-bit Integer – Time Stamp	DWORD – Frame Number	DWORD – Tick Count

Time Stamp – This number is in 1/256th of seconds since some moment in history. It is compared to the previous frames timestamp, and represents the simulated time which has passed since the last frame.

Frame Number – A sequential number of this frame since the start of the file.

Tick Count – An internal Flight Simulator counter. This value can just be set to zero when piping data to Flight Simulator.

Following the timestamp is a list of object property values. Each of the form:

WORD – Object ID	WORD – Property ID	WORD – Value Type	Data Value

Object ID – A WORD that must match a Object ID from the object definition section.

Property ID – The property ID defined in the property dictionary.

Value Type – This is a duplicate of the value type enumeration in the Property Dictionary.

Data Value – The value of the property reported in the units specified in the Property Dictionary. The number of bytes for this data item is dependent on the Value Type field.

4.3.7 File Trailer

The File Trailer is a three part record where the first and third parts are identical. The format for these two parts is:

- A four-character string. This is always "TAIB" for Flight Simulator 2004.
- A DWORD for the size in bytes of the trailer record.

The part in the middle is used to give the simulation a name and description and has the following format:

WORD – Item ID	WORD – Data Length	String - Data

Item ID – This can be either (1) Title, or (2) Description.

Data Length – The length in bytes of the data string.

Data – The title or description of the file.

4.4 Piping data to Flight Simulator

This section details the process of storing the data in the Flight Simulator parameters, constructing the .fsr file, and creating the pipe to Flight Simulator.

4.4.1 Flight Simulator parameters

The names of all the internal Flight Simulator parameters are provided with the *Building Panels and Gauges SDK*. These names must be used when defining the parameters in the properties section of the .fsr file. There is no way to add new parameters.

Some of the UAV flight variables such as battery voltages and temperatures had no equivalent parameters to use in Flight Simulator. To pass such variables to Flight Simulator, and hence be able to display them on the panel, other unimportant Flight Simulator parameters had to be found to represent these variables. Although there are hundreds of internal Flight Simulator parameters, it was found that only a very limited number of them could be modified externally, mostly only those that directly relate to the simulation. For example, airspeed and aircraft position could be modified while next waypoint position could not. This presented a problem as there were a limited number of parameters in which to store all the UAV flight data. To overcome this problem the many Boolean variables were stored bitwise within the one variable.

The complete list of internal Flight Simulator parameters that were used for each UAV flight variable are shown in the function *InitializeFilghtVariables()* defined in the file netpipes.cpp, listed in Appendix C.

4.4.2 Constructing the .fsr file

The data to be piped to Microsoft Flight Simulator must be constructed in the .fsr file format described in section 4.3. To do this a function was written for each section of the file:

```
FileHeader()
ObjectSection()
PropertySection()
FrameSection()
FileTrailer()
```

Each of the above functions constructs its respective section and adds its data to the end of a vector of bytes, named *DataPacket*, and defined as follows:

```
std::vector<__int8>* DataPacket;
DataPacket = new std::vector<__int8>;
```

This vector of bytes is then piped to Microsoft Flight Simulator by a function named *PipeData()*, which pipes the complete vector of bytes to Microsoft Flight Simulator, and then clears the vector to be used again. A flow chart detailing this process is shown below.

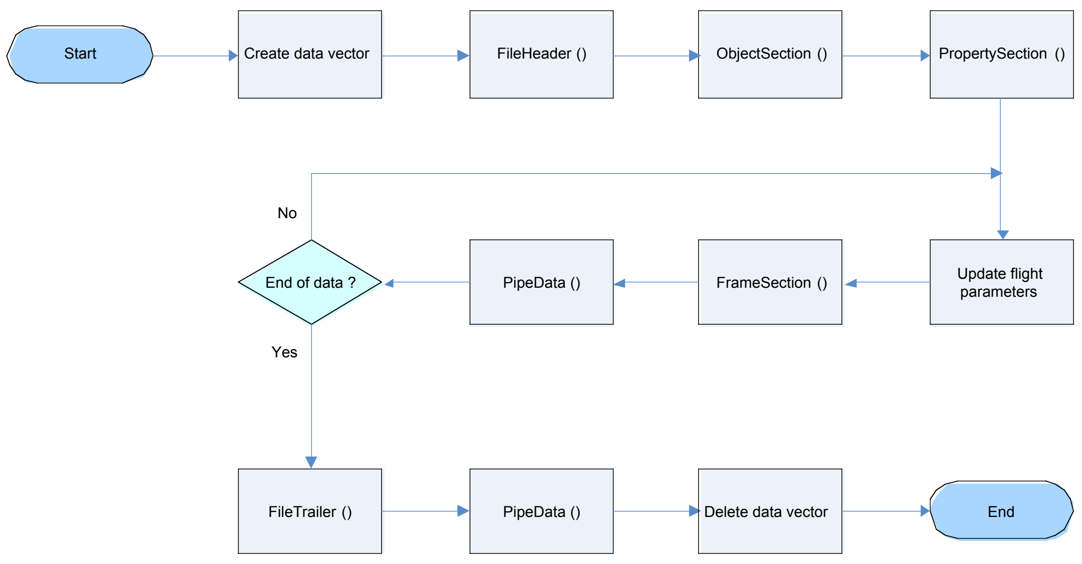


Figure 4.8 Construction of a .fsr file

4.4.3 Creating the pipe

Included with the *Netpipes SDK* is sample code for collecting data from one instance of Flight Simulator and playing it back in real time in another instance of Flight Simulator, to drive views of the same flight on two computers. Thus the code to create a pipe to Flight Simulator was already given and was used by including two C++ files and there corresponding header files from the *Netpipes SDK*. These files are named *AnonymousPipe* and *FlightSimLink* and are contained in Appendix C. The classes contained in these two files were used to create a pipe to Microsoft Flight Simulator and play the simulation. The following code shows how this was implemented:

```
try
{
	// Open pipe to FlightSim
	AnonymousPipe* pipe = new AnonymousPipe();
	pipe->Open (GENERIC_WRITE);

	//Tell FlightSim to start playing back
	CFlightSimLink flightsim;
	flightsim.Tell (CFlightSimLink::FS_START_PLAYBACK, pipe-
>GetTheHandle ());

	try
	{
		// pipe data
		pipe->Write();
	}
	catch (...)
	{
		try
		{
			flightsim.Tell (CFlightSimLink::FS_STOP_PLAYBACK);
		}
		catch (...) {}
		throw;
	}
	flightsim.Tell (CFlightSimLink::FS_STOP_PLAYBACK);
}
catch (const char* msg)
{
	//error occurred. Inform user.
}
```

Figure 4.9 Code to create a pipe to MS Flight Simulator and playback simulation

4.5 Data interpolation and smoothing

New data is sent from the UAV once every second. For smooth simulation in Flight Simulator about 20 frames per second are necessary. Thus interpolation of the data was required and was implemented within the application. The data was also smoothed, resulting in an even flowing simulation.

When the data is read from a file a moving average technique is used, with two data points behind and two data points ahead being stored to calculate the values for each data frame. Between each data point a number of interpolated points are calculated, equal to the number of frames sent to Flight Simulator per second. Then, from these interpolated points, a moving average curve is calculated using a number of points either side.

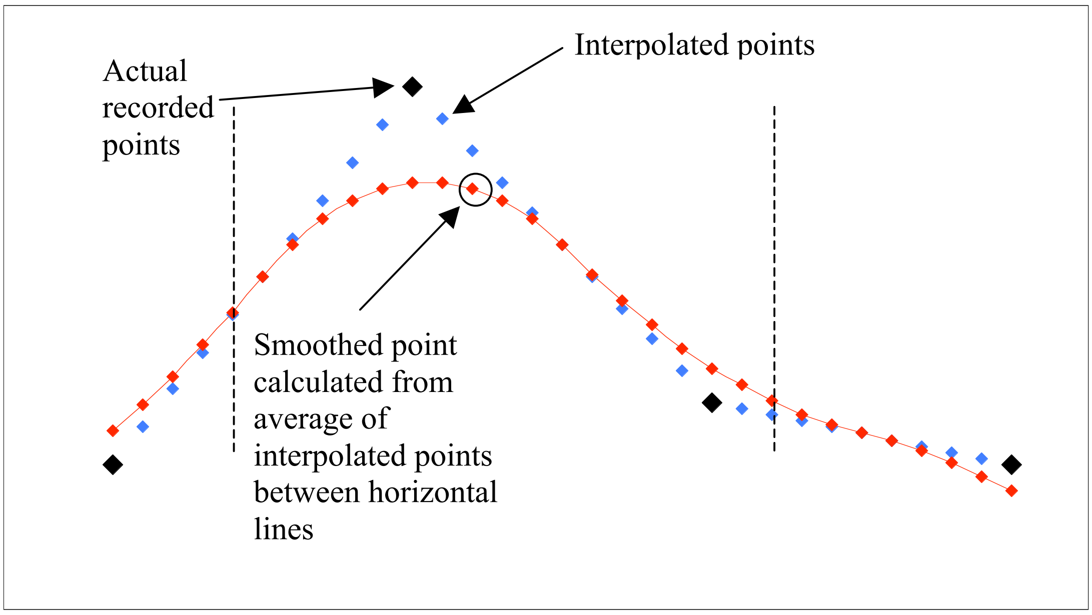

Actual recorded points
Interpolated points
Smoothed point calculated from average of interpolated points between horizontal lines

Figure 4.10 Interpolation and smoothing of the flight data

When the data is read from the serial port in real-time a moving average technique is again used but only those points preceding the current point are used, otherwise an unnecessary delay in the simulation would need to be introduced.

Examples of some smoothed data for both cases are shown in the following two graphs.

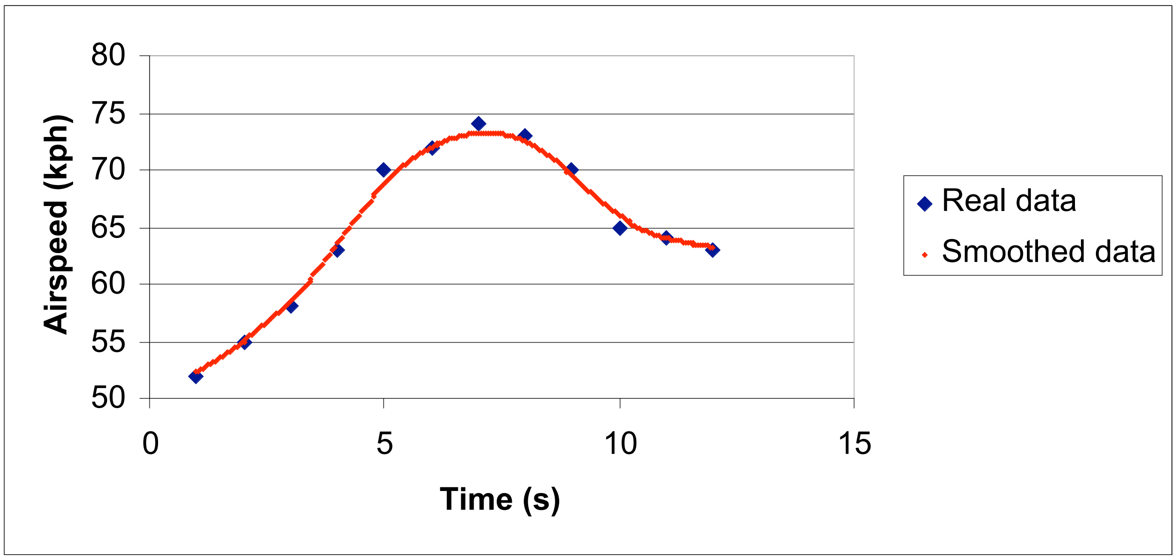


Figure 4.11 Example of smoothed data when reading from a file

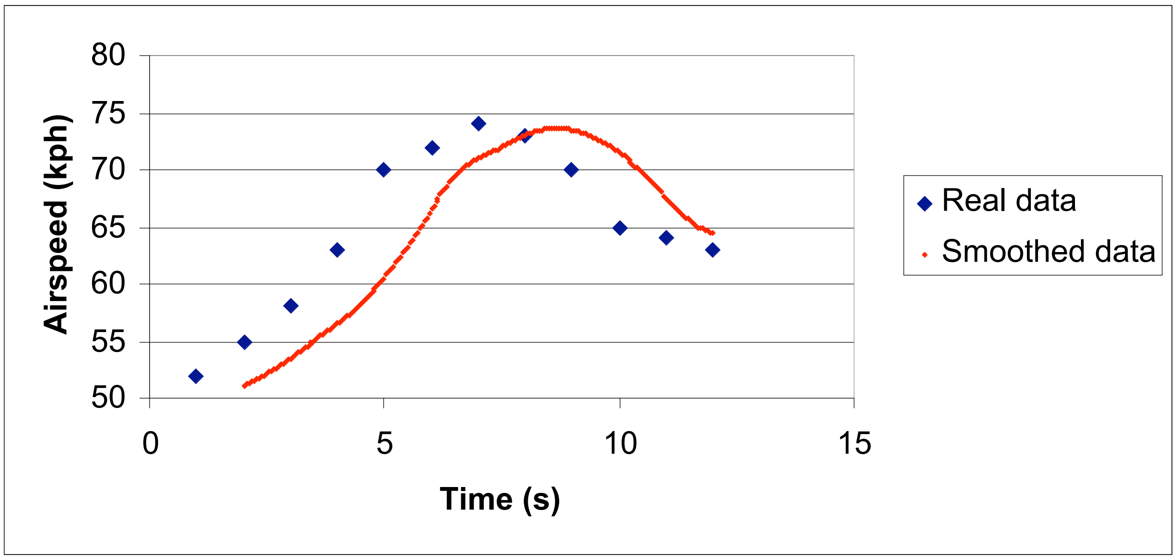


Figure 4.12 Example of smoothed data when reading in real-time

As can be seen in Figure 4.12, when reading the data in real-time a lag of about one second is introduced as new data is only received once a second. Also, the outputted data is not quite as smooth as when reading from a log file, since the moving average is not taken over as many data points.

It is only necessary to smooth the data which is used by Fight Simulator to play the simulation, data such as latitude, longitude, heading, roll, etc. Other data which is only to be displayed on the panel such as battery voltages are not smoothed.

A complication to the smoothing process arose when attempting to smooth the aircraft's heading. This occurred because the aircraft's heading changes from 360° to 0° at a north bearing, the moving average technique described above does not produce correct results if there are points on either side of this bearing, i.e. when the aircraft is passing through a north bearing.

To overcome this problem, when there are points either side of the 0° bearing, 360° is added to the points which are to the east of it. The average is then calculated and if it is greater than 360° than 360° is taken off it to result in a correct value.

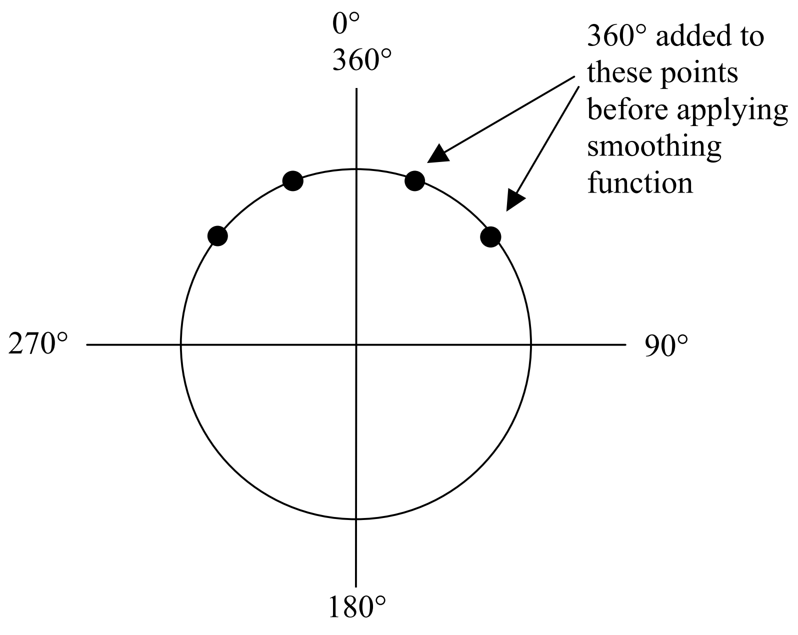
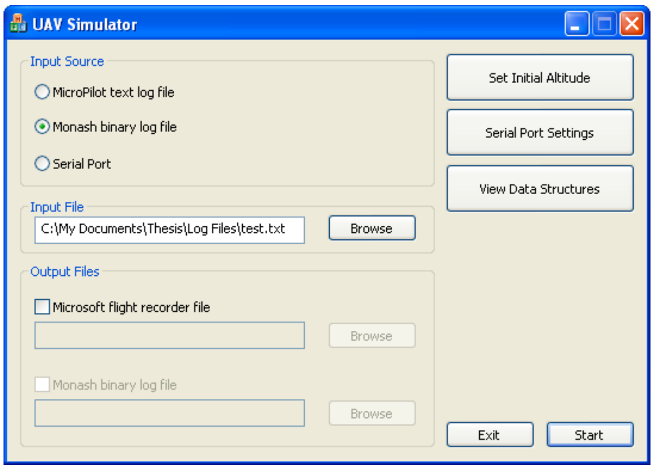

0°
360°
360° added to these points before applying smoothing function
270°
90°
180°

Figure 4.12 Smoothing the aircraft heading data

4.6 Graphical User Interface

A graphical user interface (GUI) was built for the data pipe application by using the Microsoft Foundation Class Library (MFC). While a user interface could be created without using MFC, it would take a tremendous amount of work. The advantage of using MFC is that an interface can be quickly and efficiently produced using classes that Microsoft has already created.

The GUI was built by first creating a basic dialog box, and then adding to it a number of controls; radio buttons, edit controls, check boxes and buttons. The main dialog box is shown below in Figure 4.13.


UAV Simulator

Input Source

MicroPilot text log file [✓ Monash binary log file] Serial Port

Input File: C:\My Documents\Thesis\Log Files\test.txt

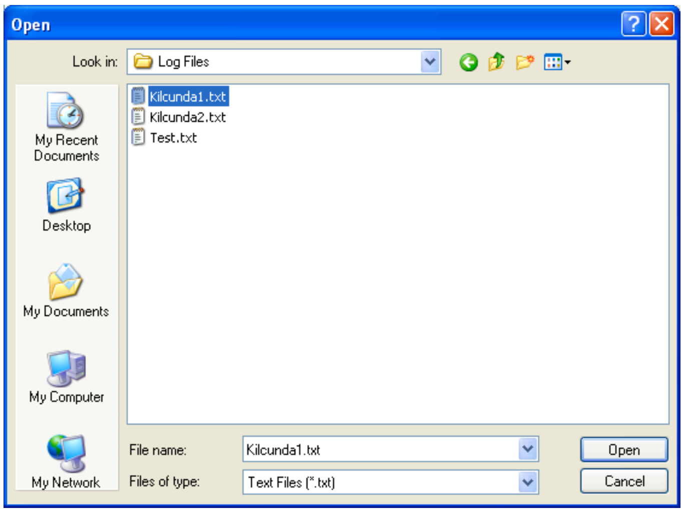
Output Files

☐ Microsoft flight recorder file

☐ Monash binary log file

Figure 4.13 Graphical user interface

The *Browse* buttons are used to select an input file and to select a name and destination for the two possible output files. When the browse button is clicked a new window is displayed using the *CFileDialog* class which creates the standard Windows Open File dialog box as shown below in Figure 4.14.


Open

Look in: Log Files

Kilcunda1.txt
Kilcunda2.txt
Test.txt

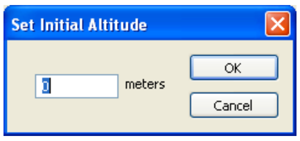
File name: Kilcunda1.txt

Files of type: Text Files (*.txt)

Figure 4.14 Open file dialog box

Three other dialog boxes were created, one to set the initial altitude, one to select the serial port settings and one to view the current values in the data structures from the UAV.

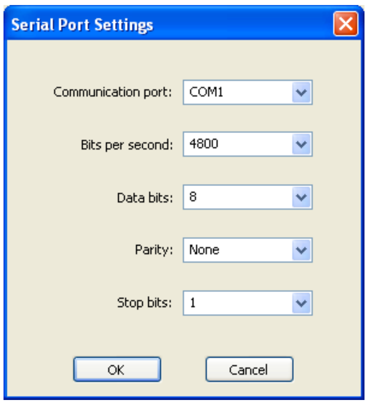
The initial altitude is required because the altitude on the UAV is initialized to zero before takeoff, whilst the Flight Simulator requires an altitude above sea-level. Hence the initial height of the UAV above sea-level must be known and set with this dialog box; otherwise the plane is likely to be placed under the ground's surface.



0 meters

Figure 4.15 Set initial altitude dialog box

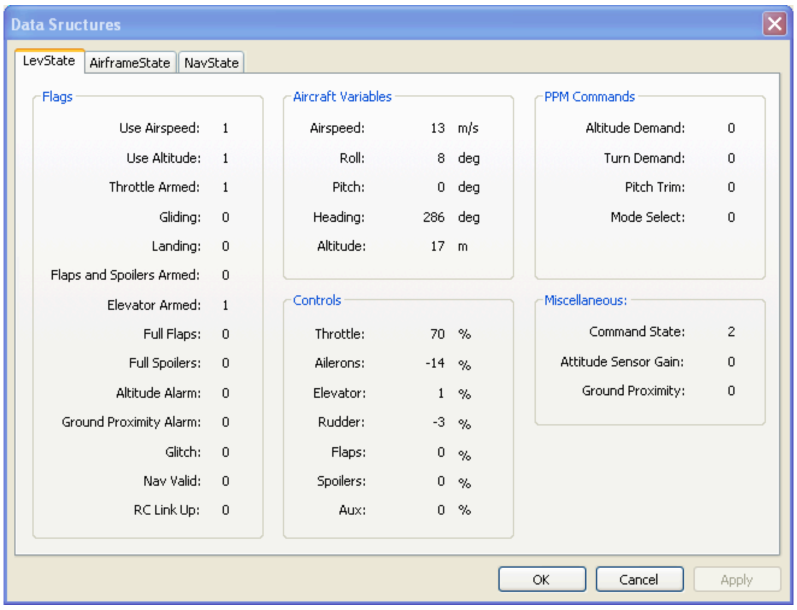
The Serial Port Settings dialog box contains a series of combo boxes containing drop down lists of all the possible settings. The serial port settings are initialized to their default values at startup and thus it is not necessary to select the port settings every time the program is run if the default settings have not changed.



Communication port:	COM1
Bits per second:	4800
Data bits:	8
Parity:	None
Stop bits:	1

Figure 4.16 Serial port settings dialog box

When the View Data Structures button is pressed a property sheet is created using the *CPropertySheet* class.



Flags

Use Airspeed:	1
Use Altitude:	1
Throttle Armed:	1
Gliding:	0
Landing:	0
Flaps and Spoilers Armed:	0
Elevator Armed:	1
Full Flaps:	0
Full Spoilers:	0
Altitude Alarm:	0
Ground Proximity Alarm:	0
Glitch:	0
Nav Valid:	0
RC Link Up:	0

Aircraft Variables

Airspeed:	13	m/s
Roll:	8	deg
Pitch:	0	deg
Heading:	286	deg
Altitude:	17	m

Controls

Throttle:	70	%
Ailerons:	-14	%
Elevator:	1	%
Rudder:	-3	%
Flaps:	0	%
Spoilers:	0	%
Aux:	0	%

PPM Commands

Altitude Demand:	0
Turn Demand:	0
Pitch Trim:	0
Mode Select:	0

Miscellaneous:

Command State:	2
Attitude Sensor Gain:	0
Ground Proximity:	0

Figure 4.17 Data Structures property sheet

The property sheet contains three property pages, one for each of the data structures (as described earlier in section 4.1.2). These pages display all of the fields contained in these data structures and their values are updated when a new packet is received from the UAV. This provides a means of viewing the less important data fields which are not displayed on the graphical instrument panel, and was also very useful during debugging.

The user input collected from the various controls on the main dialog box, and the other child dialog boxes, is stored in a data structure, named *params*, which is passed as a parameter to the main *run()* function in *netpipes.cpp* (see Appendix C), when the start button is pressed. The *run()* function is created on a new thread so that the user still has control of the user interface and is thus able to stop the simulation by pressing the stop button. The start and stop button are actually the same button but the text on the button is made to change to reflect what state it is in. On completion of the simulation, the thread sends a message back to the initial thread to let it know that it is finished and can start another simulation at the user's request.

If an error occurs in the *run()* function and the thread is ended prematurely, an error code is passed back to the main thread via the *params* data structure, and a message box containing the error is displayed. A list of all the possible error's and their codes is shown below.

Table 4.2 List of errors and their codes

Code	**Error**
0	No error
1	"FlightSim is not running"
2	"Please select an input file"
3	"Incorrect input file path"
4	"Please select a name for the .fsr output file"
5	"Please select a name for the binary log file"
6	"Serial port does not exist"
7	"Could not open serial port"
8	"Error getting serial port state"
9	"Error setting serial port state"
10	"Error setting timeouts for serial port"
11	"Error reading from serial port"

Chapter 5

3D model of the UAV

To add to the realism of the simulation a graphical model of the UAV was built and converted to a Microsoft Flight Simulator aircraft. To build the model the software package *gmax* was used. *Gmax* was chosen as it is supported by Microsoft with an SDK which provides an application to convert a model created within *gmax* to a Flight Simulator aircraft, and also provides details on how to design aircraft within *gmax*. Gmax is supplied with the Microsoft Flight Simulator installation disks.

5.1 Constructing the UAV model

The following steps were taken to construct a 3D model of the UAV:

- Measurements of all the components of the aircraft were taken and recorded on diagrams. It was not necessary to draw the diagrams to scale as all the information needed was contained within the measurements.
- Photos of the side and of the top of the UAV were taken and loaded as background images in *gmax*. These were used as guides to the shape of the aircraft components.
- A box with the dimensions of the body was created and placed over the background image. The box was split into multiple length, width and height segments to allow the shape of it to be modified.
- A taper modifier was first used to roughly match the shape of the box to that of the aircraft body.

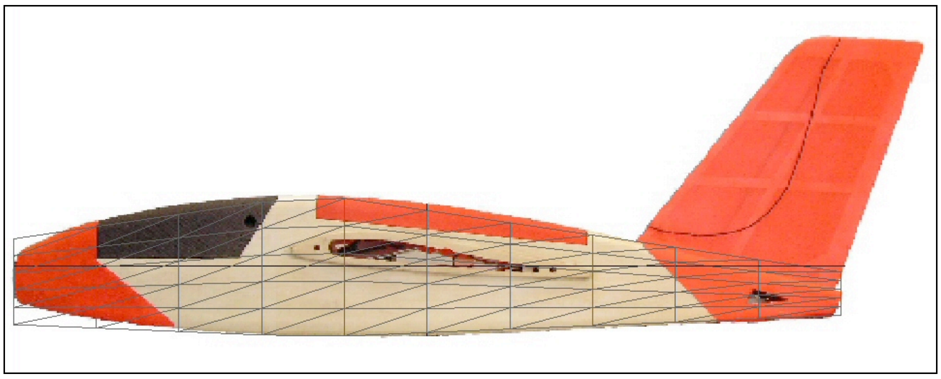


Figure 5.1 Side view of body after applying a taper modifier

- The vertices of each segment were then aligned with the background image to produce the exact shape of the aircraft body. This was done in both the side view plane and the top view plane.

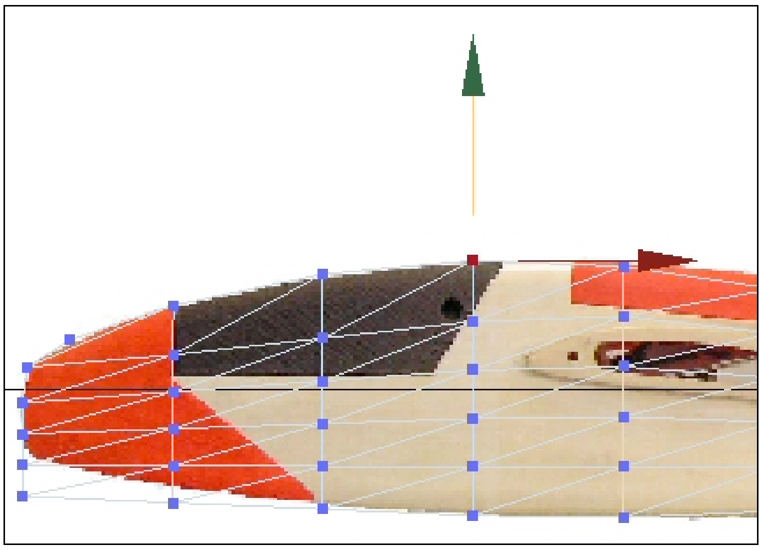


Figure 5.2 Aligning vertices to the background image

- In a similar way objects were created for the wings, the tail, the nose cone, and for the propellers. Plan views of the complete model are showed in Figure 5.3 below.

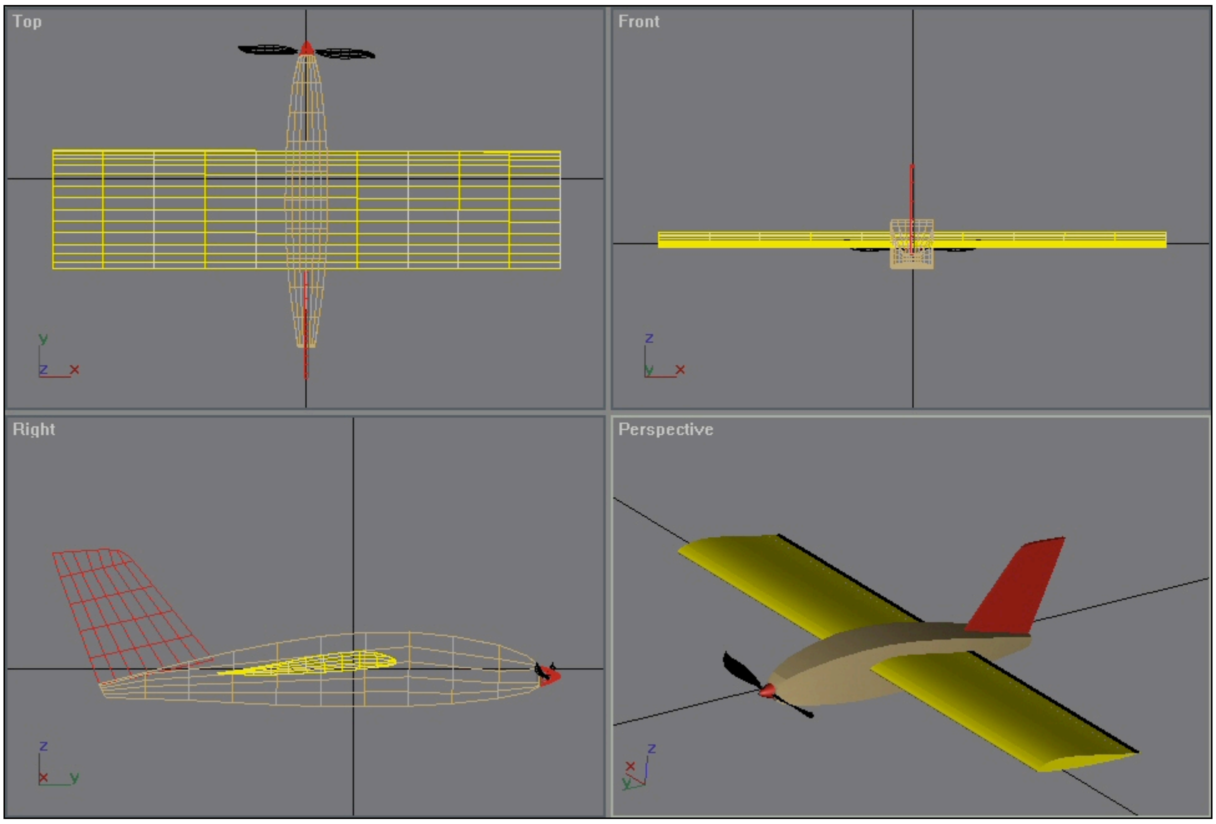


Figure 5.3 Plan views of the complete model

5.2 Adding textures to the model

To improve the realism of the model, photos of every surface were taken and mapped to the model as textures. To map multiple textures to the one object, e.g. the four sides of the body, *sub-object* selections were made, each with a different texture mapped to its surface.

To be used within Flight Simulator the textures were converted into *mipmaps*. A mipmap is a sequence of textures, each of which is a progressively lower resolution representation of the same image. The height and width of each image, or level, in the mipmap is a power of two smaller than the previous level. Mipmaps do not have to be square.

A high-resolution mipmap image is used for the plane when it appears close. Lower-resolution images are used as the plane appears farther away. Mipmapping decreases the time required to render a scene and also improves the scene's realism [8]. However they do require more memory.

The following illustration shows an example of the levels in a mipmap.

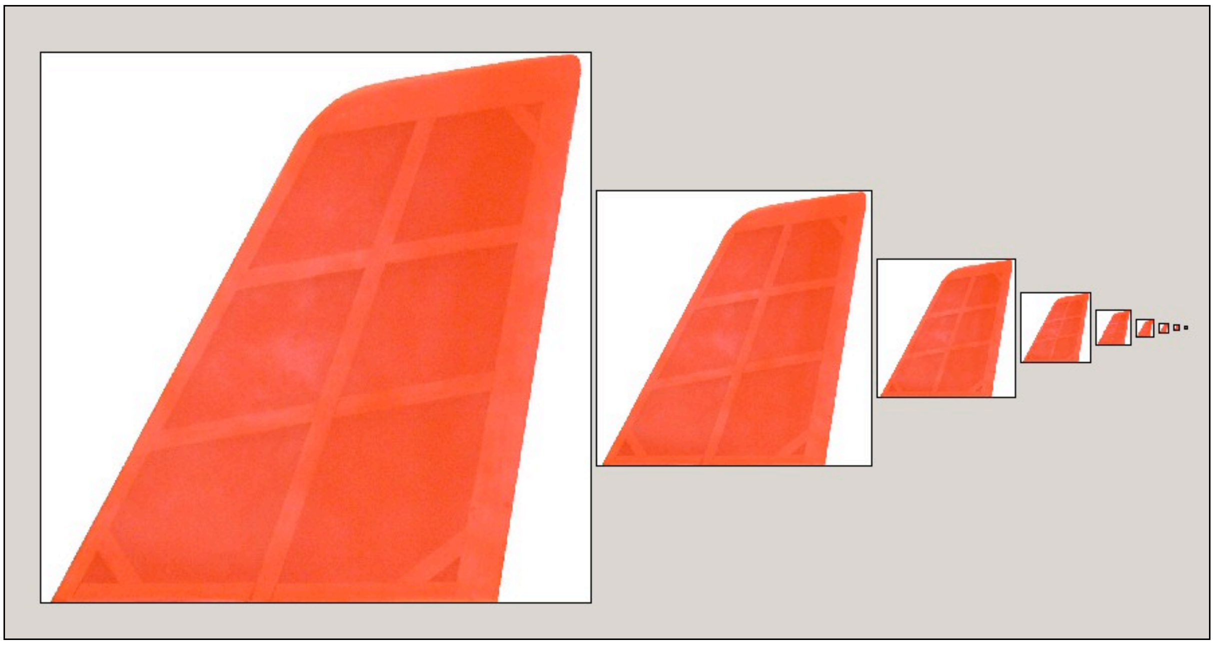


Figure 5.4 Illustration of a mipmap

To create mipmaps for each of the aircrafts textures a program called *Imagetool* (provided within the *gmax Aircraft Creation SDK*) was used. The textures also needed to be given a filename with the suffix _T to be recognized by Flight Simulator.

The final model showing the texture mapping is shown below in Figure 5.5.

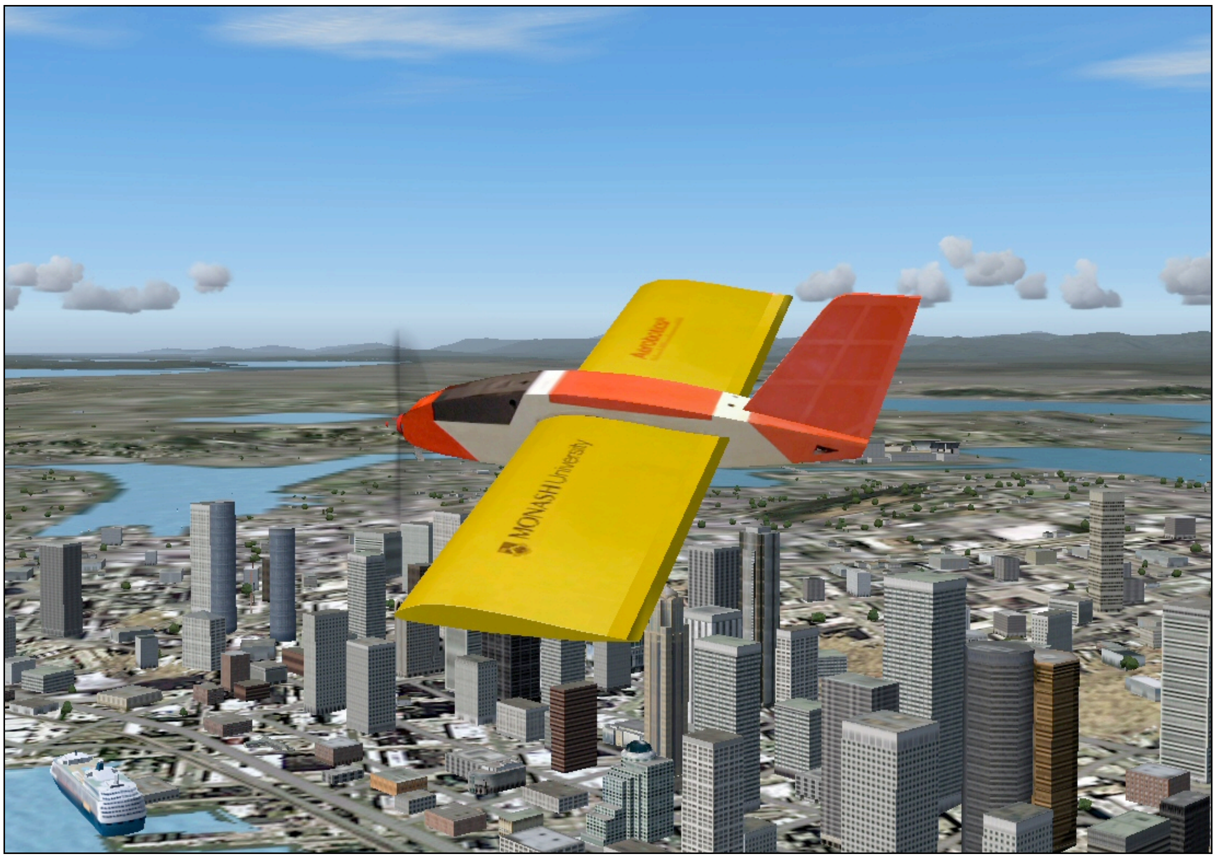


Figure 5.5 Final 3D model in flight

5.3 Adding animations to the model

Two animations were added to the model; a spinning propeller, and moving wing flaps.

For Microsoft Flight simulator to recognize animated parts it uses a naming convention described in the *gmax Aircraft Creation SDK*. The following names are used for the propeller:

prop0_still – visible when the engine is off

prop0_blurred – visible and rotating when the engine is on

To create the effect of a spinning propeller a disk was created with the texture shown below in Figure 5.6 mapped to it. The texture was extracted from another aircraft within Flight Simulator.

The result of this texture mapping on the spinning propeller can be seen in the screen shot shown above in Figure 5.5.

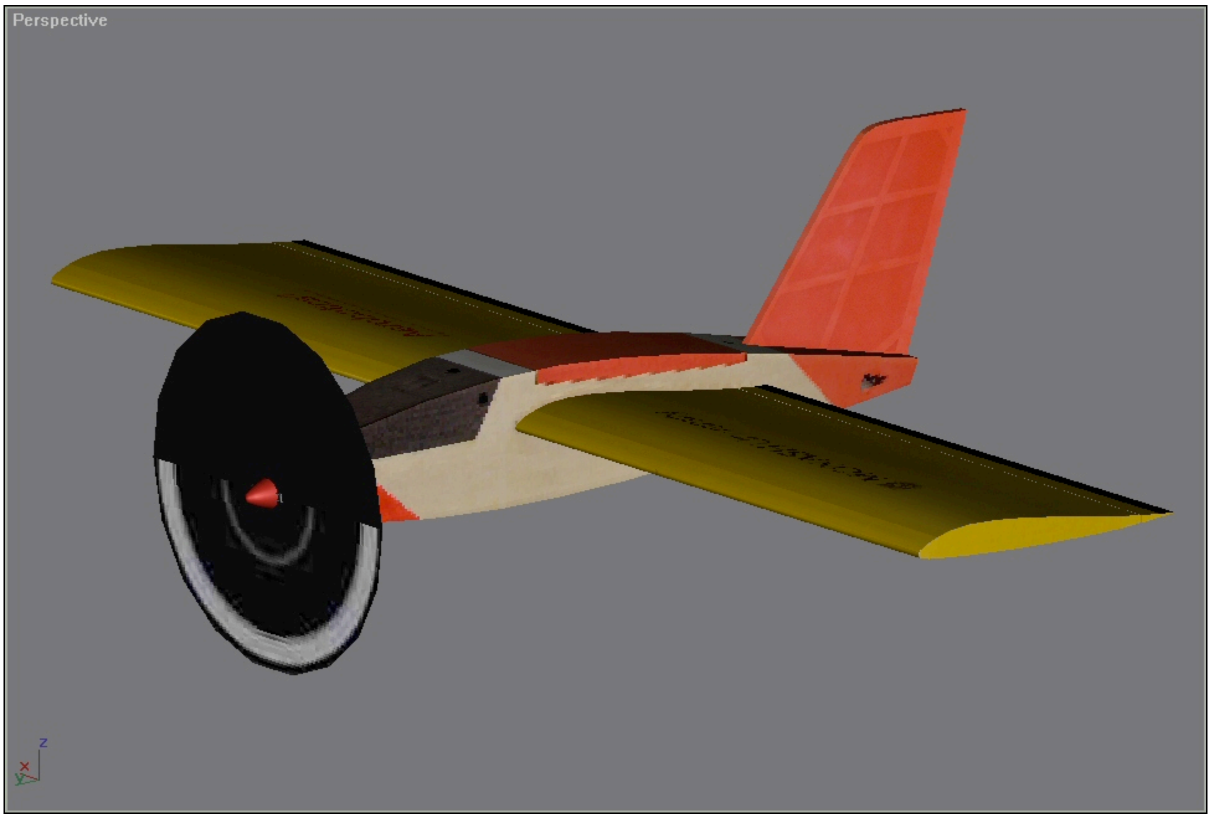


Figure 5.6 Texture used to create spinning propeller effect

The naming conventions required by Microsoft Flight Simulator for the flaps are *L_Flap_Key* and *R_Flap_Key*. To control the movement of the flaps a *keyframe* animation was set-up in *gmax* by specifying the parameters of rotation. The flaps could then be controlled through Flight Simulator.

5.4 Converting the model to a Flight Simulator aircraft

Once the *gmax* model was completed it was converted to a Flight Simulator aircraft using the *MakeMDL.exe* program which is provided with the *Make Model SDK*. The model was then placed in the directory FS9\Aircraft\UAV\model, while the textures were placed in the directory FS9\Aircraft\UAV\texture, as detailed in the *Aircraft Container System SDK*.

It was found that with the model at the correct scale (approx 1.5m wingspan) it would not display in the aircraft preview window. The model was also found to disappear during flight at certain view angles. This is shown in Figure 5.7 below where the right wing and the tail are just starting to disappear. The graphics generator within Flight Simulator is just too crude for models of that small a scale to display correctly.

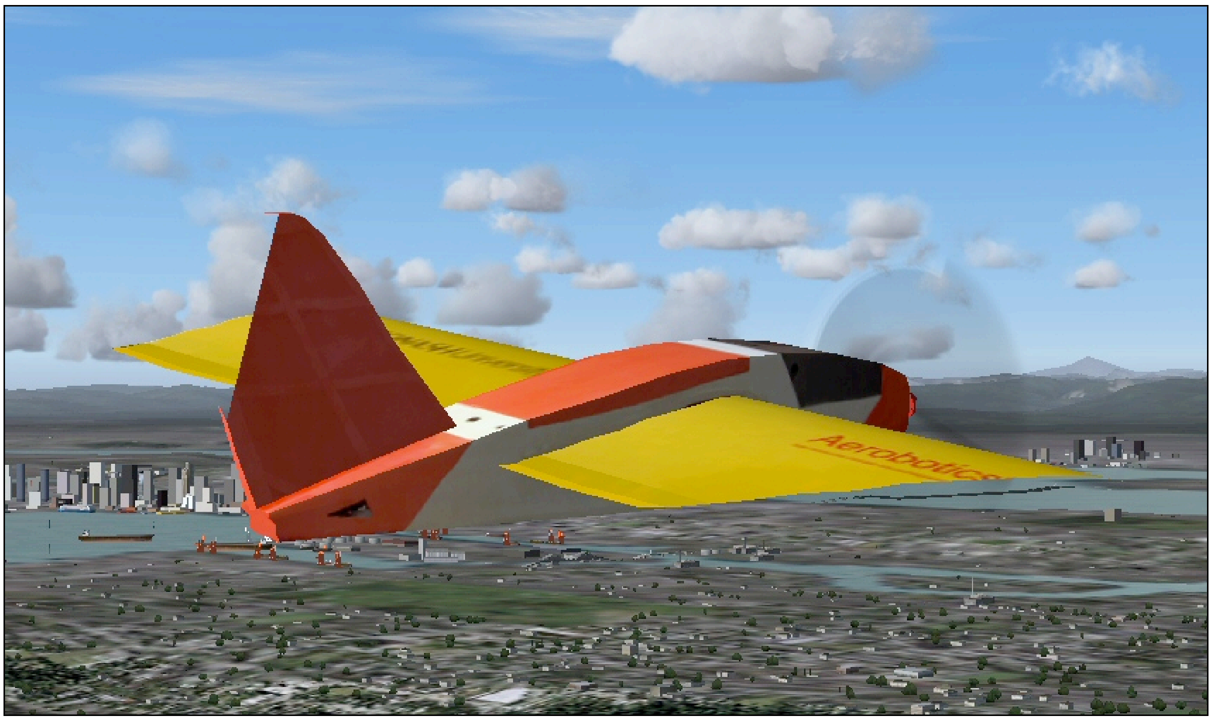


Figure 5.7 Disappearance of the aircraft model while in flight

To overcome this problem of the model disappearing, the scale of the model was increased to have a wing span of about 5 meters. It was found that at this size the model would display inside the aircraft preview model and would also display correctly during flight. It was not important to have the model at the correct scale as it is simply a visual of the aircraft's position and orientation during flight.

Chapter 6

Instrument panel

A large amount of information is captured by the UAV during flight. Some of this data is very useful to the person piloting the aircraft, whilst other information is pretty well irrelevant. The objective of this part of the project was to consult with those people who pilot the UAV to determine which data would be useful to the pilot and how this data would best be displayed. During the initial consultation process the importance of each parameter was ranked by the pilots for each of the flight modes. Recommendations on the format of the display for each parameter were also given. After this initial consultation process a panel was built and then brought back to the pilots for feedback. Changes and improvements were then made until the pilots were satisfied with the content and layout of the panel.

This chapter contains a list of all the flight data which was considered to be useful to the pilots and gives details of each of the gauges that were built. It also provides details of the configuration file that was written to configure the panel from all the individual gauges.

6.1 Flight data displayed on panel

A total of approximately 100 flight parameters are currently recorded by the UAV and stored in the three C data structures; *LevState*, *AirframeState* and *NavState*. The definitions of these data structures are in the file common.h, provided in Appendix C. From these data structures the pilots of the UAV determined which parameters are useful and should be displayed on the panel, and how they should be displayed.

The following tables list all the parameters from the three C data structures that were deemed to be useful.

Table 6.1 LevState parameters displayed on the panel

Type	Parameter name	Gauge type
LevFlags	ThrottleArmed	Light; 0 = off, 1 = green
	FlapsAndSpoilersArmed	Light; 0 = off, 1 = green
	ElevatorArmed	Light; 0 = off, 1 = green
	AltitudeAlarm	Altitude display flashes red when set
	GroundProximityAlarm	Altitude display flashes red when set
	NavValid	Light; 0 = red, 1 = green
	RCLinkup	Light; 0 = red, 1 = green
enum	CommandState	Text window displaying command state

VarRec	AirspeedV	Dial (kilometers per hour)
	RollV	Vertical horizon indicator
	PitchV	Vertical horizon indicator
	HeadingV	Dial
	AltitudeV	Dial (meters)
int	ThrottleC	Dial (percentage) Positioned with ThrottleArmed light
	ElevatorC	Digital display (percentage) Positioned with ElevatorArmed light
	FlapsC	Digital display (percentage) Positioned with FlapsAndSpoilersArmed light
	SpoilersC	Digital display (percentage) Positioned with FlapsAndSpoilersArmed light

Table 6.2 AirframeState parameters displayed on the panel

Type	**Parameter name**	**Gauge type**
AirframeFlags	AvionicsBatteryVoltsAlarm	Display background goes red
	ServoBatteryVoltsAlarm	Display background goes red
	MotorBatteryVoltsAlarm	Display background goes red
	BatteryTemperatureAlarm	Display background goes red
	MotorTemperatureAlarm	Display background goes red
int	AvionicsBatteryVolts	Digital display (volts)
	ServoBatteryVolts	Digital display (volts)
	MotorBatteryVolts	Digital display (volts)
	BatteryTemperature	Digital display (degrees celsius)
	MotorTemperature	Digital display (degrees celsius)

Table 6.3 NavState parameters displayed on the panel

Type	**Parameter name**	**Gauge type**
NavFlags	NavArrivalAlarm	Light; 0 = off, 1 = yellow flashing
	ReturningToOrigin	Light; 0 = off, 1 = yellow
	GPSValid	Light; 0 = red, 1 = green
	WayPointValid	Light; 0 = red, 1 = green
long	MissionTime	Digital clock (hh:mm:ss)
WayState	Latitude	Waypoint marker on GPS display
	Longitude	Waypoint marker on GPS display
	Altitude	Digital display on GPS
GPSState	Latitude	Current position marker on GPS display
	Longitude	Current position marker on GPS display
	RateOfClimb	Dial (meters per second)
long	ClosingRange	Digital display on GPS
int	WayHeading	Digital display on GPS

6.2 Design of panel components

Microsoft Flight Simulator has traditionally used gauges programmed in C. However, in Flight Simulator 2002 a new kind of gauge was introduced, the XML gauge. Currently, in Flight Simulator 2004, both C gauges and XML gauges are supported, although the majority of the standard panels still use C gauges. The shift is however towards XML gauges and thus the panel for the UAV was built using XML gauges to increase the chances that it will be compatible with future versions of Microsoft Flight Simulator.

A downside of using XML gauges was that there is currently very limited documentation on how to build XML gauges in the *Building Panels and Gauges SDK*. However the SDK does provide an XML schema which was used as a resource for all the possible XML fields and types of their values. The XML files from the standard panels which use XML gauges are also available and were used as references.

An XML gauge in Flight Simulator consists of three parts:

1. The first part provides generic information about the gauge: its name, its background image, and its size.
2. The second part of the XML gauge is a list of elements of the gauge (marked by <Element> tags). Each element is generally a bitmap image or a text string.
3. The third part of the XML gauge describes the gauge's mouse rectangles, which can be used to display tool tips or to obtain user input from a mouse click.

The root element of an XML gauge is the <Gauge> element.

The graphics of the panel and its gauges were designed using the drawing tools in Adobe Photoshop to create vector shapes (mathematically defined lines and curves), which could be easily manipulated and resized as desired. Once designed the images were converted to 8-bit bitmap images as required by Microsoft Flight Simulator. When transparency was required, such as for the rounded ends at the top of the panel and for all of the circular dials, the background colour was set to true black (0,0,0 RGB value). True black is made to appear transparent in the simulation.

The final design of the graphical instrument panel for the UAV is shown below in Figure 6.1.

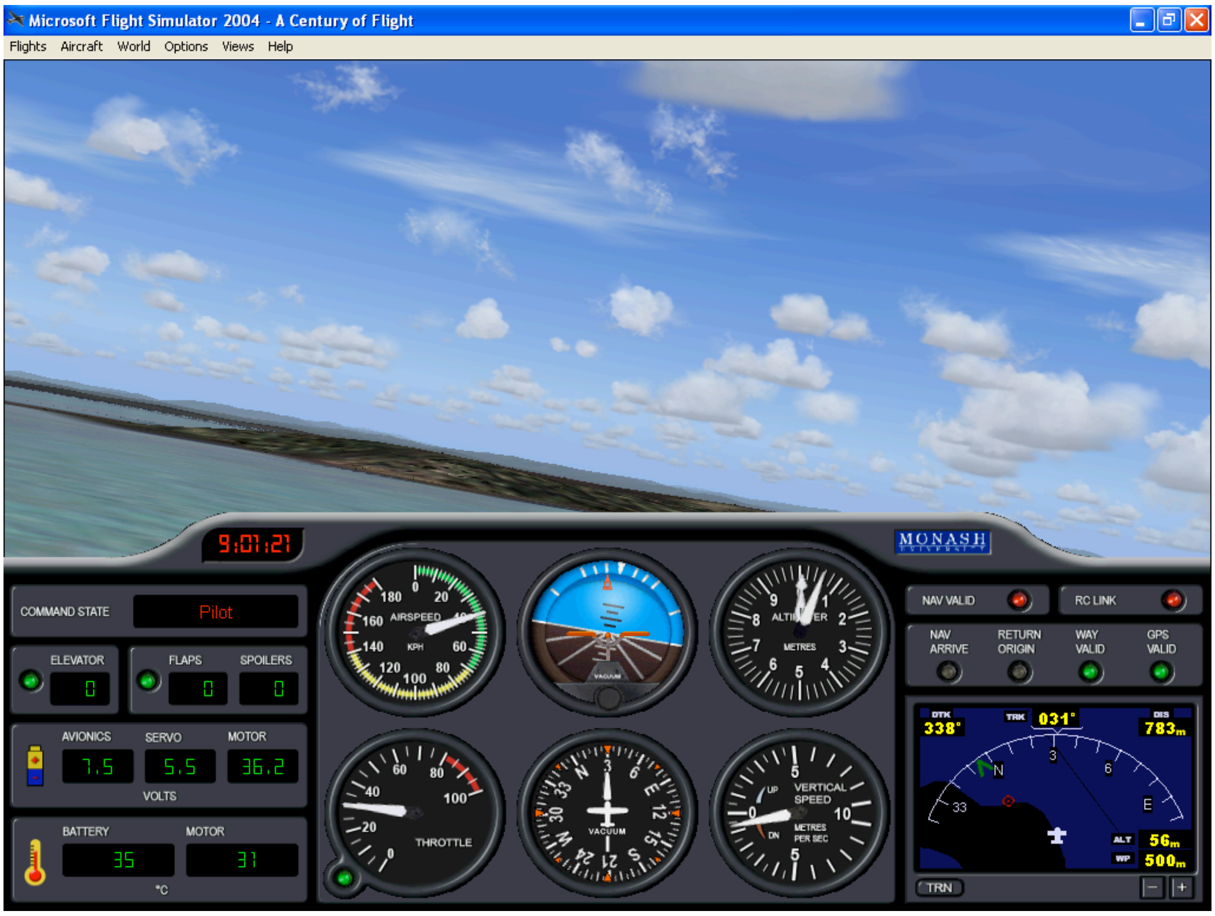


Figure 6.1 UAV instrument panel

The following sections detail all of the components that make up the UAV panel, and shows the major XML elements which were used to build them. A complete list of all the XML files for the panel is contained in Appendix D.

6.2.1 Left Panel

The left component of the panel consists of a text display of the UAV's command state, digital displays of the control surfaces (elevators, flaps, and spoilers), digital displays of the battery voltages and digital displays of the temperature readings.

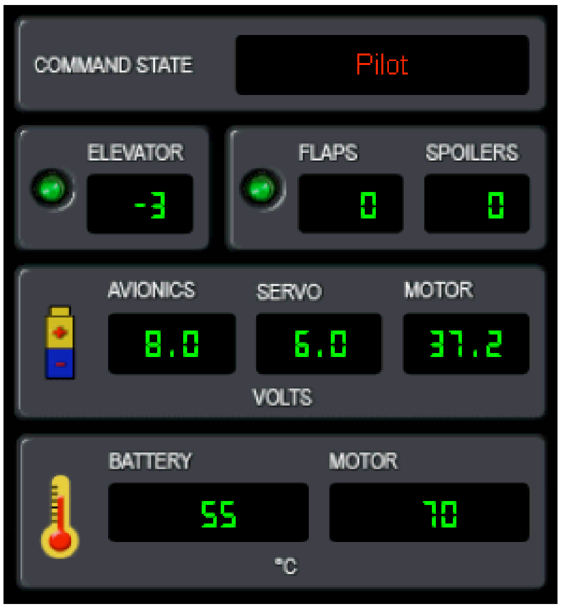


Figure 6.2 Graphics of left panel section

The XML file associated with this section of the panel is named *Left Panel.xml* and is given in Appendix D.

The lights for the *ElevatorArmed* and *FlapsAndSpoilerArmed* flags are displayed using the <Select> element. Figure 6.3 below shows the code used for the *ElevatorArmed* light.

```
<Element>
   <Position X="10" Y="83"/>
   <Select>
      <Value>(L:ElevatorArmed, boolean)</Value>
      <Case Value="0">
         <Image Name="GreenLightOff.bmp"/>
      </Case>
      <Case Value="1">
         <Image Name="GreenLightOn.bmp"/>
      </Case>
   </Select>
</Element>
```

Figure 6.3 Example of a gauge using the <Select> element

The <Position> element specifies the X, Y position of the <Element> relative to the gauge. The <Select> element displays an image depending on the result of the <Value> element. The <Value> element checks if the *ElevatorArmed* flag is set. The <Case> element then displays the *GreenLightOn* image if the flag is set or the *GreenLightOff* image if the flag is not set.

The text displays for the command state, for the flaps, spoilers and elevator positions, for the battery voltages, and for the temperature readings, are implemented with the <Text> element. Figure 6.4 below shows the code used for the *ElevatorC* display.

```
<Element>
   <Visible>(L:ElevatorArmed, boolean)</Visible>
   <Position X="45" Y="90"/>
   <Text X="40" Y="20" Length="3" Fixed="No" Font="Quartz"
FontHeight="0" FontWeight="400" Charset="Default" Attributes="Normal"
Adjust="Center" VerticalAdjust="Center" Multiline="No" Color="#00FF00"
BackgroundColor="transparent" HilightColor="white" Bright="Yes"
UseTransparency="Yes">
      <String>%((A:Elevator position,percent))%!3d!%</String>
   </Text>
</Element>
```

Figure 6.4 Example of a gauge using the <Text> element

The <Text> element contains a number of attributes for the text such as the position, length, font color etc. The <String> element then defines the string to display. The <Visible> element is used to display the elevator position only if the *ElevatorArmed* flag is set; otherwise the text string is not displayed.

The other text displays are implemented in a similar way. For the battery voltage and temperature reading displays, the background colour and the text colour are changed to red and white respectively when the corresponding alarm flag is set.

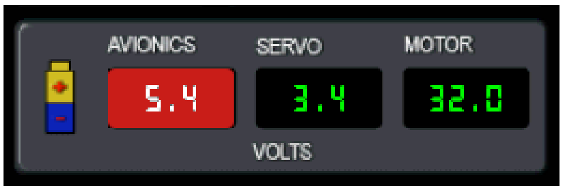


Figure 6.5 Battery voltage display with avionics alarm set

The battery voltage and temperature gauges were originally designed as analog displays but it was decided by the pilots during the consultation process that it would be better for them to be digital displays. The main reason for this is that a number of different batteries are used with different voltage levels, making it difficult to design a scale for an analog dial. The added benefit is that they use up less space as digital dials, resulting in a smaller panel and thus more room for the 3D simulation.

6.2.2 Central dials

The central component of the panel contains six analog dials displaying airspeed, attitude, altitude, throttle, heading and vertical speed.

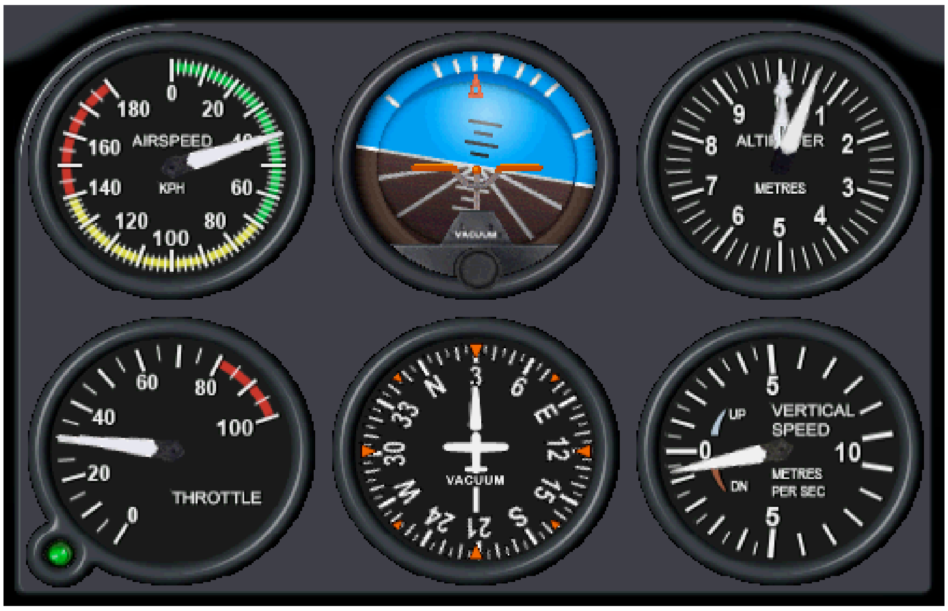


Figure 6.6 Graphics of central dials

The following XML files are associated with each of the dials and are given in Appendix D.

Airspeed Indicator.xml
Attitude Indicator.xml
Altimeter.xml
Throttle Indicator.xml
Heading Indicator.xml
Vertical Speed Indicator.xml

Each of these gauges uses the <Rotate> element to rotate either a needle or a background image in accordance with a particular parameter. Figure 6.7 below shows how the rotate element was used for the Airspeed Indicator.

```
<Element>
   <Position X="79" Y="79"/>
   <Image Name="Airspeed_Needle.bmp" PointsTo="East">
      <Axis X="7" Y="6"/>
   </Image>
   <Rotate>
      <Value Minimum="0" Maximum="180">(A:Airspeed indicated, kilometer
per hour)</Value>
      <Nonlinearity>
         <Item Value="0" X="79" Y="18"/>
         <Item Value="180" X="43" Y="30"/>
      </Nonlinearity>
      <Delay DegreesPerSecond="20"/>
   </Rotate>
</Element>
```

Figure 6.7 Example of a gauge using the <Rotate> element

The *Airspeed Indicated* parameter to control the rotation of the needle is selected with the <Value> element; the Minimum and Maximum attributes are set to specify the allowable range. The <Nonlinearity> element specifies the X, Y coordinates that the needle is to point towards for a value of “0” and for a value of “180”. The needle is then rotated linearly between these two points. The element is given the name <Nonlinearity> because it can be used to rotate the needle in a nonlinear manner by specifying more than two <Item> elements. The <Delay> element is used to control the speed at which the needle can move. This is done by setting the maximum rotation of the needle over a one second period.

Each of these gauges also uses the <Mouse> element to display a ToolTip as the mouse enters the area defined by that gauge. A complete list of all the possible ToolTip IDs is provided with the *Panels SDK*.

```
<Mouse>
   <Tooltip ID="TOOLTIPTEXT_AIRSPEED_KILOS"/>
</Mouse>
```

Figure 6.8 Example of a gauge using the <Mouse> element

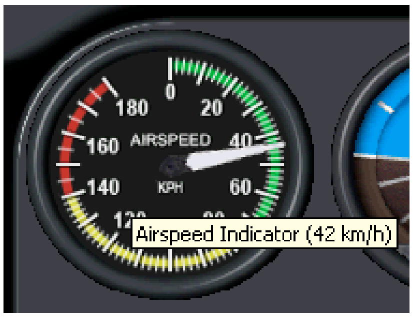


Figure 6.9 Airspeed gauge showing its ToolTip

When either the *AltitudeAlarm* (the UAV is flying to high) or the *GroundProximityAlarm* (the UAV is too close to the ground) flags are set the altitude dial flashes red. This was implemented by using the time parameter and alternating the background colour every second.

6.2.3 Navigational lights

This section of the panel contains six lights that display information discerning the navigational state of the UAV.

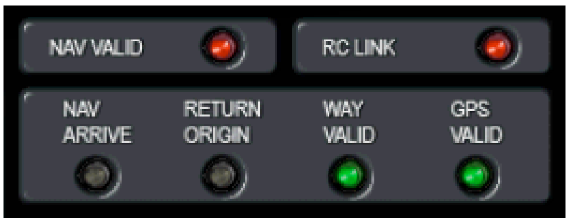


Figure 6.10 Graphics of navigational lights

The four lights at the bottom of this panel display the state of the four flags from the *NavState* data structure. The two lights at the top actually display two of the flags belonging to the *LevState* data structure; however it was decided by the pilots that these flags would be best displayed together with the navigational flags as they relate to the navigational state of the UAV.

The XML file associated with this section of the panel is named *Nav Lights.xml* and is given in Appendix D. The XML elements used in this section of the panel are much the same as those used to display the *ElevatorArmed* and *FlapsAndSpoilersArmed* lights on the left panel, described earlier in Section 6.2.1.

6.2.4 GPS display

Microsoft Flight Simulator comes with a standard GPS display which opens in a separate window. As most of the functionality of this GPS was not needed, a new GPS was built and incorporated into the main panel as seen earlier in Figure 6.1. The XML file for the standard GPS was used as a reference to build the new GPS. Much of the functionality of the standard GPS is contained in a DLL file within the FS9\Modules directory and unfortunately no documentation on these functions is provided. However, by studying the standard GPS XML file enough knowledge of these functions was gained to be able to use them.

The main function which was used was the <CustomDraw> element named fs9gps:map which displays the map. This element accepts a number of attributes and child elements which set parameters for the map such as its dimensions, global position and heading, as seen in Figure 6.11 below.

```
<CustomDraw Name="fs9gps:map" X="230" Y="145" LayerTerrain="Yes"
LayerBorders="Yes" LayerFlightPlan="Yes" FlightPlanLineWidth="2.5"
ActiveColorLayerFlightPlan="0x7010B0" astColorLayerFlightPlan="0xF0F0F0"
ColorLayerFlightPlan="0xF0F0F0" Bright="Yes" CenterX="115">
      <CenterY>115</CenterY>
      <Zoom>(@g:map_ZoomFactor) 1852.0 *</Zoom>
      <Latitude>(A:GPS POSITION LAT, Radians)</Latitude>
      <Longitude>(A:GPS POSITION LON, Radians)</Longitude>
      <Heading>(A:GPS GROUND TRUE TRACK, Radians)</Heading>
      <DetailLayerTerrain>(@g:setup_ColorTerrain) 0 == if{ 1 } els{ 2
}</DetailLayerTerrain>
</CustomDraw>
```

Figure 6.11 The custom draw element used to display the GPS map

The <DetailLayerTerrain> element sets the map to either show with the terrain OFF as seen earlier in Figure 6.1, or to show with the terrain ON as seen in Figure 6.12 below, depending on the value of the variable *setup_ColorTerrain*. This variable is changed with the TRN button. Likewise the *map_ZoomFactor* variable used with the <Zoom> element is changed with the + and - buttons.

The fs9gps:map function provides the functionality to display waypoints on the map. However, only airports can be set as waypoints and thus this functionality could not be used and instead the waypoint marker was placed on the map manually. The point at which to display the marker was calculated using the distance to the waypoint, together with the map zoom factor, and the desired heading.

The heading arc seen on the GPS display is also a <CustomDraw> element, named fs9gps:rose.

A number of digital displays were included on the GPS to display the heading, desired heading, distance to waypoint, current altitude and altitude of waypoint, as seen in Figure 6.12 below.

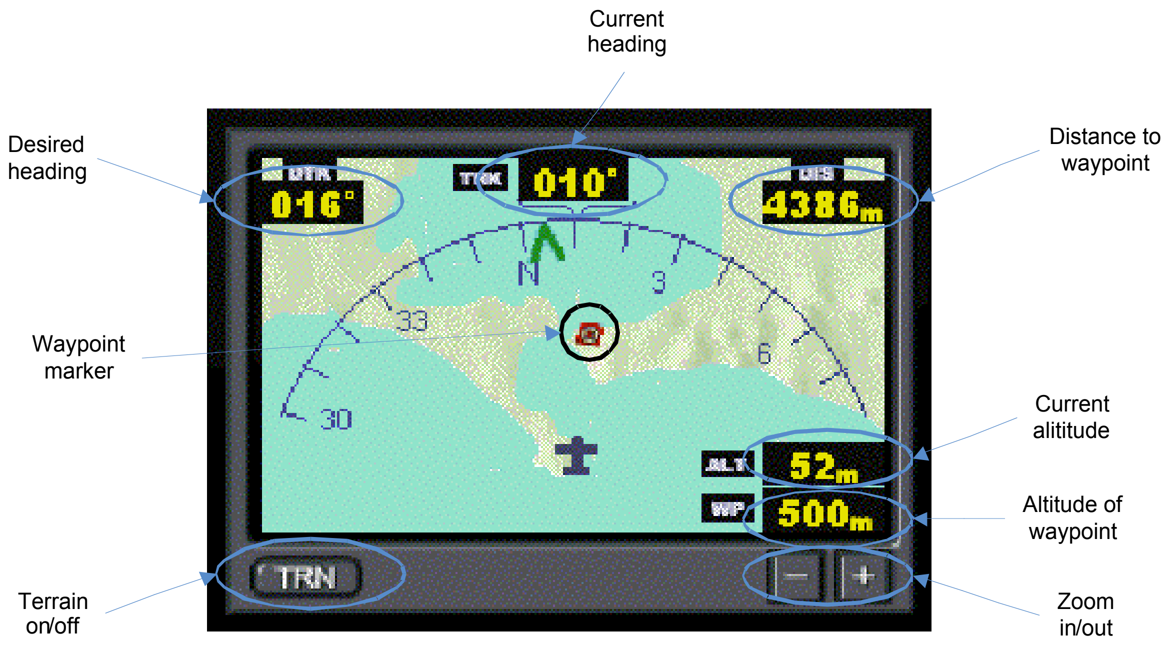


Figure 6.12 Graphics of GPS display with terrain on

The XML file associated with the GPS is named *GPS.xml* and is given in Appendix D.

6.2.5 Clock

When the input file is in the MicroPilot format the digital clock displays the time that the flight was recorded. When the input is of the Monash binary format, however, the actual time is not available and so the clock displays the time since the start of the flight, using the *MissionTime* field in the *NavState* data structure.

Figure 6.13 Graphics of clock

The XML file associated with the clock is named C*lock.xml* and is given in Appendix D. It uses the <Text> element, described earlier, to display the time.

6.3 Panel configuration file

When Microsoft Flight Simulator loads an aircraft, the panel is configured from the *panel.cfg* file which is located in the aircraft's *Panel* folder. It defines the characteristics of the aircraft's cockpit, including window settings, view settings, and gauges. This section details the structure of the *panel.cfg* file.

A panel configuration file contains five possible sections:

[Windows Titles]
[Views]
[WindowXX]
[VCockpitXX]
[Default View]

The [Views] section and the [VCockpitXX] section were not used and thus will not be described here. Full detail of all five sections is provided in the *Building Panels and Gauges SDK*.

The configuration file written for the UAV's panel is shown below in Figure 6.14.

```
[Window Titles]
window00=Main Panel

[Window00]
file_1024=UAV_Panel_Background.bmp
size_mm=1024
position=7
ident=MAIN_PANEL
gauge00=UAV!Clock,175,17,70,24
gauge01=UAV!Left Panel,0,60,265,300
gauge02=UAV!Airspeed Indicator,270,30,158,158
gauge03=UAV!Attitude Indicator,433,30,158,158
gauge04=UAV!Altimeter,596,30,158,158
gauge05=UAV!Throttle,270,193,158,158
gauge06=UAV!Heading Indicator,433,193,158,158
gauge07=UAV!Vertical Speed Indicator,596,193,158,158
gauge08=UAV!Nav Lights,759,60,265,105
gauge09=UAV!GPS,759,160,265,200

[Default View]
X=0
Y=0
SIZE_X=8192
SIZE_Y=6144
```

Figure 6.14 Panel configuration file

Following is an explanation of each of the sections and the variables which were used:

[Window Titles] section: Creating Panel Windows

The panel system creates the aircraft's panel window by using the `[Window Titles]` section. For each `WindowXX` variable in this section, the panel system creates a panel window. The string assigned to each `WindowXX` (e.g. `Window00=Main Panel`) is the title displayed within Flight Simulator.

[WindowXX] section: Describing panel windows

This section describes a panel window (i.e., an individual panel): its shape, position, properties, the background image, and what gauges belong to the panel. Each of the variables which were used in this section is described in the following table.

Table 6.4 Variables in the [WindowXX] section of the panel configuration file

Variable	Description
file	Specifies the bitmap file to load and use for the panel window background. (Flight Simulator 2004 uses a background designed for 1024 by 768 resolution.)
size_mm	Specifies the width of the panel window, in millimeters. To keep things simple this value was set to the width of the panel in pixels. This allows pixels and millimeters to be used interchangeably for all references within the Panel.cfg file.
position	Specifies the relative position of the panel window to the main window. There are nine possible values for this field. A value of 7 was used which places the panel window centered at the bottom of the main window.
ident	Specifies a unique identifier to define the panel window.
gauge00-gaugeXX	Specifies which gauge file to load and the position of the gauge. The basic format is: gauge## = gaugefolder!gaugename, X, Y, W, H Where: gauge## – indicates the order in which the gauge is loaded. gaugefolder! – indicates the folder in which the gauge is found. X, Y – indicates the X and Y position of the gauge in millimeters, relative to the panel background. W, H – indicates the width and height of the gauge in millimeters.

[Default View] section: Setting the view

This section sets the overall view for the default 3-D simulation window.

Table 6.5 Variables in the [Default View] section of the panel configuration file

Variable	Description
X	Specifies the X position of the default 3D window.
Y	Specifies the Y position of the default 3D window.
SIZE_X	Specifies the width of the default 3D window. This was set to the maximum width of 8192.
SIZE_	Specifies the height of the default 3D window. This was set to the maximum width of 6144.

Chapter 7

Customizing the terrain

To improve the realism of the terrain in Microsoft Flight Simulator, satellite images from NASA were collected for Victoria, as well as one aerial photo of an area of east of Melbourne where the UAV is often flown, and mapped to the terrain using the *Terrain SDK*. This chapter gives details of the steps that were taken to produce these terrain maps.

7.1 Image requirements

The raw image must be either a 24-bit per pixel Windows .bmp or a 32-bit per pixel Targa .tga file. The latitude and longitude of the northwest corner of the image must be known as well as the spacing (in decimal degrees) between the image pixels. The spacing between pixels however can be calculated from the latitude/longitude values of the image corners and the image pixel size:

$$\text{North/South pixel spacing} = \frac{\text{lat1 - lat2}}{\text{image pixel height}}$$

$$\text{East/West pixel spacing} = \frac{\text{long2 - long1}}{\text{image pixel width}}$$

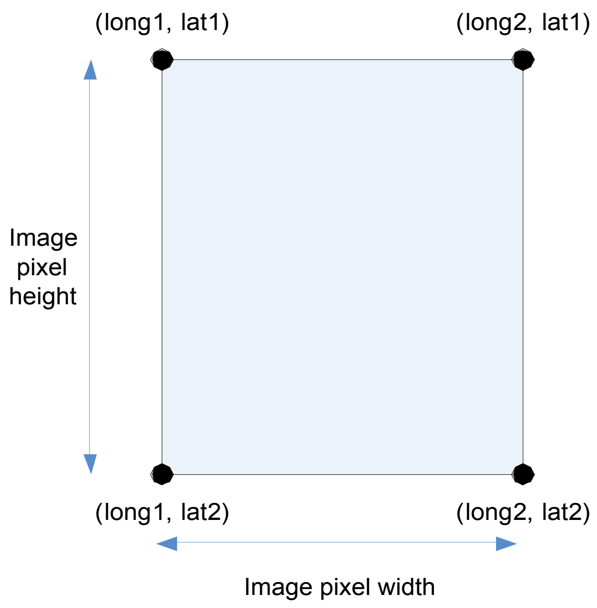


Figure 7.1 Terrain image coordinates

The extent of each terrain texture pixel is 4.75 meters in Flight Simulator. The re-sampling process will filter the raw image to the right size.

- If the raw image is more than 4.75 meters per pixel, the final result will appear blurry.
- If the raw image is less than 4.75 meters per pixel, some detail will be lost.

The terrain textures in Flight Simulator are stored as terrain cells, with each cell falling on a multiple of 360/32768 degrees latitude and 480/32768 degrees longitude. If the input image does not completely cover a terrain cell the uncovered area will be mapped as water. Thus to avoid this happening the borders of an image should be cropped to fall exactly on a cell boundary.

The picture below illustrates the mapping that results when an image is not defined to fall exactly on a cell boundary. Unfortunately this aerial photo is too small and doesn't completely cover a terrain cell. Terrain mapping is better done with larger images which can be cropped to align with a cell boundary.

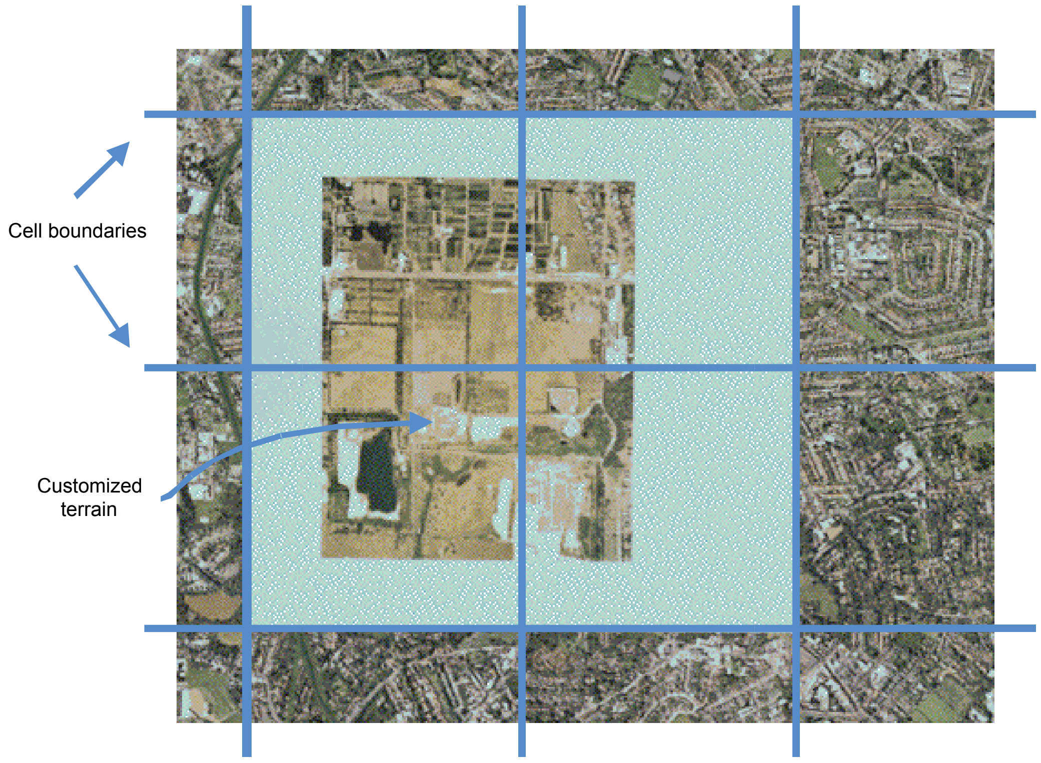


Figure 7.2 Result of terrain mapping when image doesn't fall on a cell boundary.

7.2 Converting a raw image to a terrain texture

To convert a raw satellite image or aerial photo into a terrain texture an application named *resampler.exe* is provided with the *Terrain SDK*. To use this application an .inf file must be written which provides the re-sampler with the information necessary to create a custom texture area.

There are two distinct areas of the .inf file:

- The [Destination] section, which defines the destination directory, the filename, and the desired geographical bounds of the resulting files.
- The [Source] section, which defines the geographical area covered by the source image and where to find the source file.

The tables below list the sets of directives required for each section.

Table 7.1 Destination directives

Directive	Description
DestDir	Specifies the directory where the files created by the re-sampler will be placed.
DestBaseFileName	Specifies the name given to the .bgl file created by the re-sampler.
BuildSeasons	Tells the re-sampler to generate seasonal texture variations: 0 – Assumes a single source image for all seasons. 1 – Creates a set of seasonal textures.
UseSourceDimensions	Specifies the dimensions of the destination image: 0 – Explicitly defines the dimensions of the resulting image. Used if the source image is much larger than the desired result. 1 – Uses the latitude and longitude specified in the [Source] section as the limits of the bounding area.
NorthLat	Specifies the northern latitude of the destination image (only needed if UseSourceDimensions = 0)
SouthLat	Specifies the southern latitude of the destination image (only needed if UseSourceDimensions = 0)
EastLon	Specifies the eastern longitude of the destination image (only needed if UseSourceDimensions = 0)
WestLon	Specifies the western longitude of the destination image (only needed if UseSourceDimensions = 0)

Table 7.2 Source directives

Directive	Description
Type	Specifies the type value. This must be Custom.
SourceDir	Specifies the directory where the source image file is stored.
SourceFile	Specifies the name of the source image file.
Lat	Specifies the latitude of the Northwest corner of the image in decimal degrees.

Lon	Specifies the longitude of the Northwest corner of the image in decimal degrees.
NumOfCellsPerLine	Specifies the width of the image in pixels.
NumOfLines	Specifies the height of the image in pixels.
CellXdimensionsDeg	Specifies the distance between each pixel in degrees longitude.
CellYdimensionsDeg	Specifies the distance between each pixel in degrees latitude.

For all of the custom texture areas that were created, **BuildSeasons** was set to 0 as satellite images for each season were not available, neither would it have been necessary to have a different terrain texture for each season.

An example of one of the .inf files that was written is shown below.

```
[Destination]
      DestDir                  = "."
      DestBaseFileName         = "dandenong_valley"
      BuildSeasons             = 0
      UseSourceDimensions      = 1

[Source]
      Type                     = Custom
      SourceDir                = "."
      SourceFile               = "dandenong_valley_parklands_12Feb04.bmp"
      Lat                      = -37.8726667
      Lon                      = 145.1992667
      NumOfCellsPerLine        = 4084
      NumOfLines               = 5057
      CellXdimensionDeg        = 4.0931929480901077E-006
      CellYdimensionDeg        = 3.2067747676488036E-006
```

Figure 7.3 Example of an .inf file used to create a custom texture area

After creating an .inf file for the texture data, the re-sampler was run, which produces a .bgl file (which contains the terrain mesh) and a set of .tga files (one for each terrain cell).

The .tga files must then be converted into mipmaps, and this was done with the application *Imagetool*, which is provided with the *Terrain SDK*.

Figure 7.4 below shows the UAV flying over an area of customized terrain east of Melbourne which was produced from an aerial photo with a pixel resolution less than 5 meters. Roads, buildings and other major landmarks are clearly visible at this resolution.

The only satellite images that could be obtained free of charge had a pixel resolution of 15 meters. At this resolution major roads and landmarks are just visible but minor roads and buildings become a blur. Figure 7.5 shows the UAV flying over a satellite image of Philip Island, with the race track just visible in the foreground.

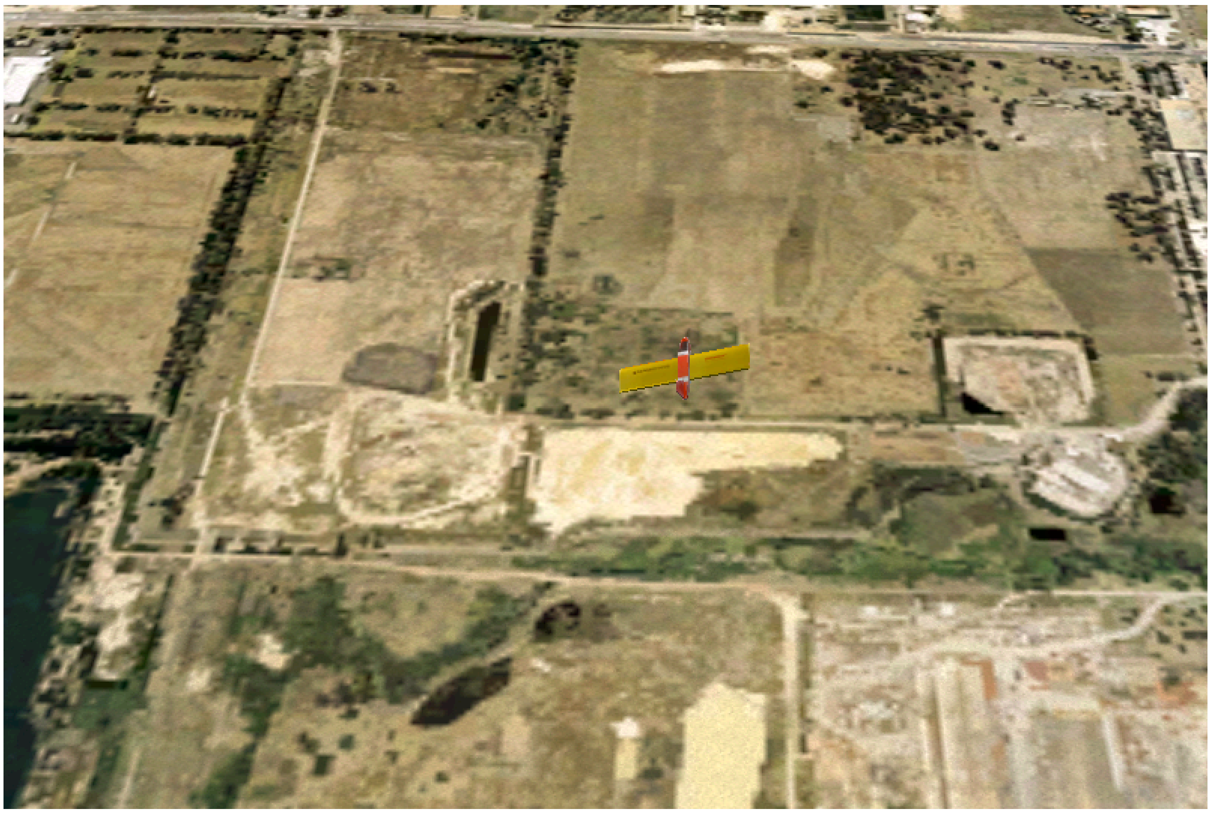


Figure 7.4   UAV flying over some customized terrain east of Melbourne (aerial photo).

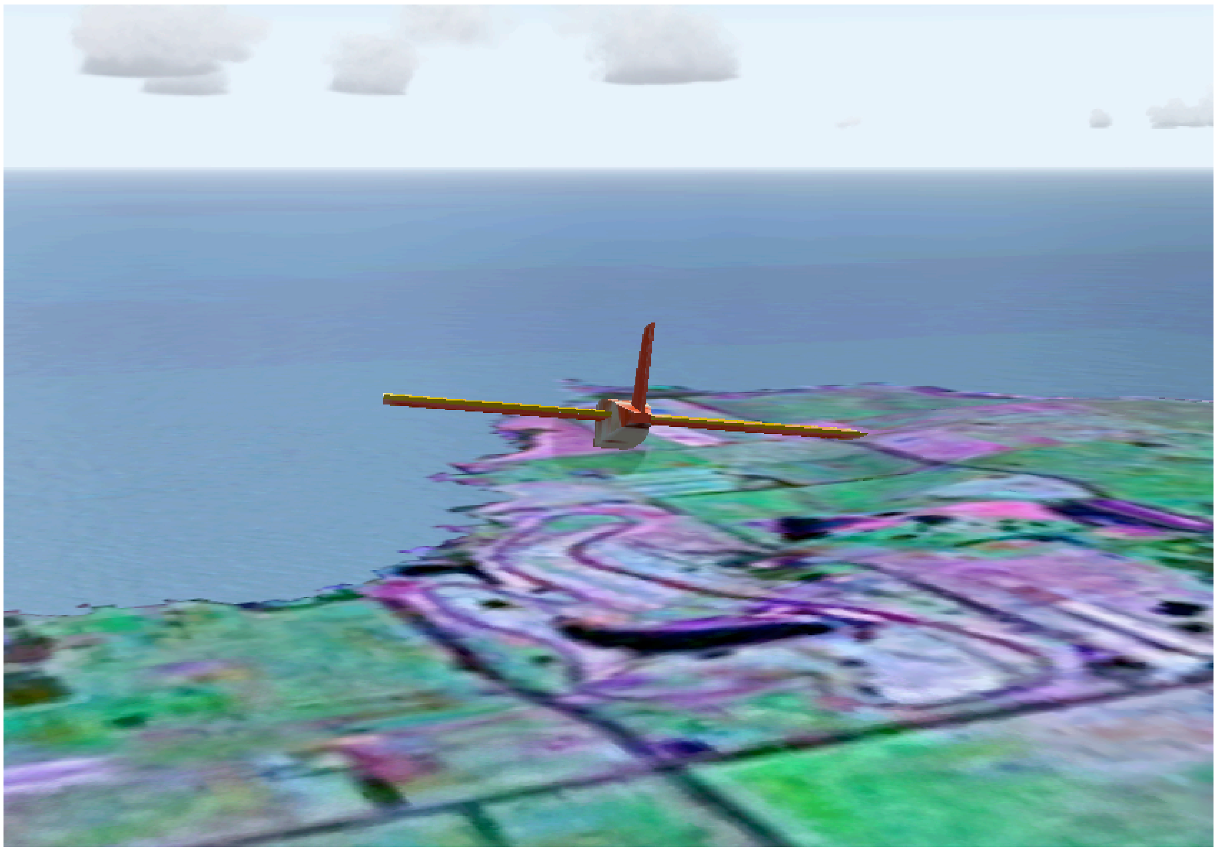


Figure 7.5   UAV flying over some customized terrain of Philip Island (satellite image).

Chapter 8

Recommendations for further work

Although a visualization system for a UAV has successfully been implemented using Microsoft Flight Simulator, there are many possible areas for further improvements. These include:

- Collecting Digital Elevation Models (DEMs) and mapping these to the terrain to produce a more accurate terrain mesh. Currently, 9 second DEMs (data points approximately every 250m) are available as a free download from the Australian government's geoscience website [9]. More accurate 3 second models are available with the payment of a licensing fee.
- Collecting more aerial photos for the areas in which the UAVs are most often flown. These could be either captured with cameras onboard the UAV or could be purchased.
- Creating 3D models of the other UAVs that are used by the Aerobotics© Research Group.

Chapter 9

Conclusion

A visualization system for a UAV has successfully been designed and implemented using Microsoft Flight Simulator. Microsoft Flight Simulator was chosen over the open-source flight simulator FlightGear due to the comprehensiveness of its documentation, although it was concluded that either flight simulator would have been suitable for this project.

A Windows application capable of reading the UAV’s flight data from a log file, or from a serial port in real-time, and piping this data to an instance of Flight Simulator was written. Algorithms to smooth the flight data were also added which resulted in a sufficiently smooth simulation of the UAV. A user interface was written for the application providing an easy method of setting user options.

A 3D graphical model of the UAV was constructed using the 3D modeling program *gmax*. At the correct scale the model was found to not display correctly in the simulator. To overcome this problem the size of the model had to be increased to have a wing span of about 5 meters. It was not considered important to have the model at the correct scale as it is simply a visual of the aircraft’s position and orientation during flight.

In consultation with the pilots of the UAV an instrument panel was designed that displayed all of the UAVs flight data which the pilots considered to be relevant. Changes and improvements to the design were made until the pilots were satisfied with the content and layout of the panel.

As an optional extension to the original project requirements, satellite images and aerial photos were collected for the areas in which the UAV is flown and were mapped to the surface of the terrain to improve the realism of the 3D simulation. For best results images with a pixel resolution of 5 meters or less should be used.

Together these components provide a fully functional system capable of providing crucial information about the state of the UAV during a flight to the pilots on the ground.

References

[1] *Wired News*, "Human Pilots: Who Needs 'Em?", [online], Available: http://www.wired.com/news/technology, November 2003, (Accessed September 2005).

[2] *FlightGear*, [online], Available: http://www.flightgear.org, September 2005, (Accessed September 2005).

[3] *GNU General Public License*, [online], Available: http://www.gnu.org/copyleft/gpl.html, July 2005, (Accessed September 2005).

[4] M. Castillo-Effen, W. Alvis, C. Castillo, K. Valavanis and W. Moreno, "Modeling and Visualization of Multiple Autonomous Heterogeneous Vehicles", *University of South Florida* [online], Available: http://crasar.csee.usf.edu/research/Publications/CRASAR-TR2005-6.pdf, (Accessed September 2005).

[5] *Microsoft Corporation*, "Flight Simulator 2004", [online], Available: http://www.microsoft.com/games/flightsimulator/, 2005 (Accessed April 2005).

[6] *Autodesk*, [online], Available: http://www.discreet.com, 2005, (Accessed May 2005).

[7] *MicroPilot*, [online], Available: http://www.micropilot.com, 2005, (Accessed April 2005).

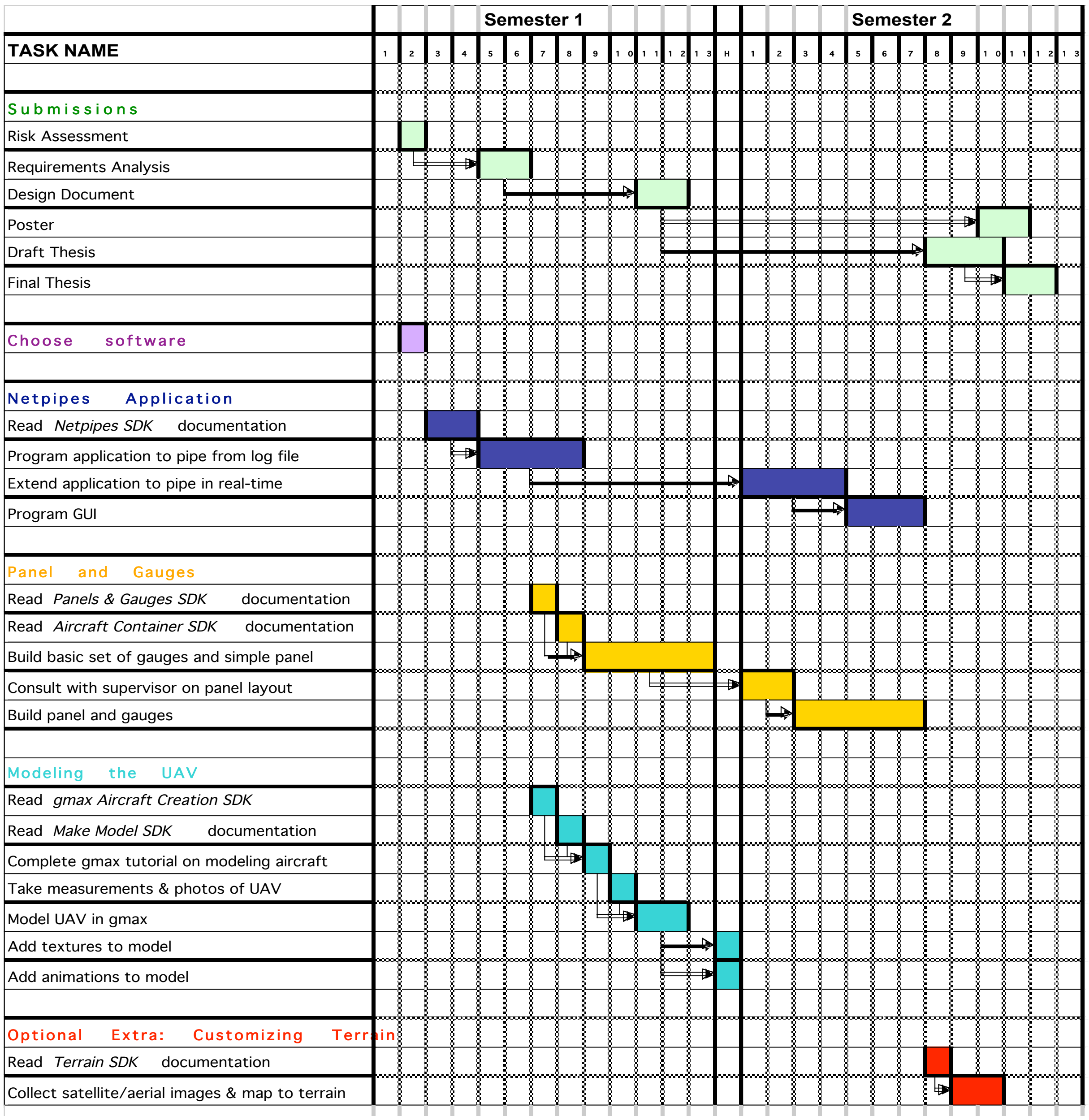
[8] *Microsoft Corporation*, "Texture Filtering with Mipmaps", [online], Available: http://msdn.microsoft.com, 2003, (Accessed June 2005).

[9] *Australian Government Geoscience Australia*, [online], Available: http://www.ga.gov.au, July 2005, (Accessed September 2005).

Appendix A

Timeline

At the beginning of the project a proposed timeline was produced and is shown below.



This timeline was adhered to closely with the only major change being an overlap in the writing of the thesis report and the completion of the practical part of the project. The thesis report was started much early than proposed (week 4 of semester 2), while the data pipe application and the instrument panel were completed later than proposed (week 11).

Appendix B

Contents of CD ROM

The accompanying CD ROM contains the following files and directories:

Aircraft\UAV\	Flight Simulator Aircraft Container. This folder must be copied into the *Aircraft* directory of Flight Simulator. It contains the complete UAV aircraft including model, panel and sound files.
Addon Scenery\	Customized terrain. The contents of this folder should be copied to the *Addon Scenery* directory of Flight Simulator. To install the terrain open the *Scenery Library* from the *Settings* tab within Flight Simulator and click the *Addon Scenery* check box.
Netpipes Project\	Visual Studio project for the data pipe application, including all source files and resources.
Gmax Model\	Gmax model of the UAV.
Gauge Graphics\	Photoshop drawing files of all the gauges which were created.
Log Files\	A collection of log files for testing the project.
netpipes.exe	The executable file for the data pipe application.
Conference Paper.doc	A 5 page IEEE conference paper detailing the outcomes of the project.

Appendix C

Source code

All the source code that was written for the data pipe application is included in this appendix. Also included are the four files from the *Netpipes SDK* which were used in the project and the file *common.h* which contains the data structures sent from the UAV.

Files written for this project: Page

netpipes.cpp netpipes.h	Contains the main functional code for the application.	55 73
NetPipesGui.cpp NetPipesGui.h	Defines the class behaviors for the GUI application.	76 77
NetPipesGuiDlg.cpp NetPipesGuiDlg.h	Implementation and header files for the main dialog box.	78 84
SetAltDlg.cpp SetAltDlg.h	Implementation and header files for the *Set Initial Altitude* dialog box.	85 86
SerialPortDlg.cpp SerialPortDlg.h	Implementation and header files for the *Serial Port Settings* dialog box.	87 88
DataStructPropSheet.cpp DataStructPropSheet.h	Implementation and header files for the *Data Structures* property sheet.	89 90
LevPage.cpp LevPage.h	Implementation and header files for the *LevState* page of the *Data Structures* property sheet.	91 92
AirframePage.cpp AirframePage.h	Implementation and header files for the *AirframeState* page of the *Data Structures* property sheet.	93 94
NavPage.cpp NavPage.h	Implementation and header files for the *NavState* page of the *Data Structures* property sheet.	95 96

Files included from the *Netpipes SDK*: Page

AnonymousPipe.cpp AnonymousPipe.h	Implementation and header files for the *AnonymousPipe* class.	97 98
FlightSimLink.cpp FlightSimLink.h	Implementation and header files for the *CFlightSimLink* class.	99 100

File included from the UAV code: Page

Common.h	File containing the data structures sent from the UAV.	101

netpipes.cpp

netpipes.cpp

netpipes.cpp

netpipes.cpp

netpipes.cpp

netpipes.cpp

netpipes.cpp

netpipes.cpp

netpipes.cpp

netpipes.cpp

netpipes.cpp

netpipes.cpp

netpipes.cpp

netpipes.h

netpipes.h

netpipes.h

NetpipesGui.cpp

NetpipesGui.h

NetPipesGuiDlg.cpp

NetPipesGuiDlg.cpp

NetPipesGuiDlg.cpp

NetPipesGuiDlg.cpp

NetPipesGuiDlg.cpp

NetPipesGuiDlg.cpp

NetPipesGuiDlg.h

SetAltDlg.cpp

SetAltDlg.h

SerialPortDlg.cpp

SerialPortDlg.h

DataStructuresPropertySheet.cpp

DataStructuresPropertySheet.h

LevPage.cpp

LevPage.h

AirframePage.cpp

AirframePage.h

NavPage.cpp

NavPage.h

NavPage.h

AirframePage.cpp

AirframePage.h

AnonymousPipe.cpp

AnonymousPipe.h

FlightSimLink.cpp

FlightSimLink.h

Common.h

Common.h

Common.h

Common.h

Appendix D

XML gauge files

This Appendix contains all the XML files written for the graphical instrument panel.

Airspeed Indicator.xml

```
<Gauge Name="Airspeed Indicator" Version="1.0">
   <Image Name="Airspeed_Background.bmp"/>
   <Element>
      <Position X="79" Y="79"/>
      <Image Name="Airspeed_Needle.bmp" PointsTo="East">
         <Axis X="7" Y="6"/>
      </Image>
      <Rotate>
         <Value Minimum="0" Maximum="180">(A:Airspeed indicated, kilometer per hour)</Value>
         <Nonlinearity>
            <Item Value="0" X="79" Y="18"/>
            <Item Value="180" X="43" Y="30"/>
         </Nonlinearity>
         <Delay DegreesPerSecond="20"/>
      </Rotate>
   </Element>
   <Mouse>
      <Help ID="HELPID_GAUGE_AIRSPEED"/>
      <Tooltip ID="TOOLTIPTEXT_AIRSPEED_KILOS"/>
   </Mouse>
</Gauge>
```

Attitude Indicator.xml

```xml
<Gauge Name="Attitude Indicator" Version="1.0">
   <Image Name="Attitude_Background.bmp"/>
   <Element>
      <Position X="0" Y="0"/>
      <MaskImage Name="Attitude_Card_Outside_Mask.bmp">
         <Axis X="79" Y="79"/>
      </MaskImage>
      <Image Name="Attitude_Card_Outside.bmp">
         <Axis X="79" Y="79"/>
      </Image>
      <Rotate>
         <Value>(A:Plane bank degrees,radians)</Value>
      </Rotate>
   </Element>
   <Element>
      <Position X="0" Y="0"/>
      <MaskImage Name="Attitude_Card_Inside_Mask.bmp">
         <Axis X="79" Y="79"/>
      </MaskImage>
      <Image Name="Attitude_Card_Inside.bmp">
         <Axis X="79" Y="79"/>
      </Image>
      <Shift>
         <Value Minimum="-25" Maximum="25">(A:Plane pitch degrees,degrees) /-/</Value>
         <Scale Y="1.25"/>
      </Shift>
      <Rotate>
         <Value>(A:Plane bank degrees,radians)</Value>
      </Rotate>
   </Element>
   <Element>
      <Position X="79" Y="26"/>
      <Image Name="Attitude_Arrow_Icon.bmp">
         <Axis X="5" Y="0"/>
      </Image>
   </Element>
   <Element>
      <MaskImage Name="Attitude_Card_Outside_Mask.bmp">
         <Axis X="79" Y="79"/>
      </MaskImage>
      <Image Name="Attitude_Bars.bmp">
         <Axis X="36" Y="2"/>
      </Image>
      <Shift>
         <Value>(A:Attitude bars position, position)</Value>
         <Scale Y="7"/>
      </Shift>
   </Element>
   <Element>
      <Position X="80" Y="138"/>
      <Image Name="Attitude_Knob.bmp">
         <Axis X="13" Y="13"/>
      </Image>
      <Rotate>
         <Value>(A:Attitude bars position, position) pi *</Value>
      </Rotate>
   </Element>
   <Mouse>
      <Tooltip ID="TOOLTIPTEXT_ATTITUDE_INDICATOR_BANK_PITCH"/>
      <Area Left="67" Right="93" Top="125" Bottom="151">
         <Area Right="13">
            <Cursor Type="DownArrow"/>
            <Click Event="ATTITUDE_BARS_POSITION_DOWN" Repeat="Yes"/>
         </Area>
         <Area Left="13">
            <Cursor Type="UpArrow"/>
            <Click Event="ATTITUDE_BARS_POSITION_UP" Repeat="Yes"/>
         </Area>
      </Area>
   </Mouse>
</Gauge>
```

Altimeter.xml

```
<Gauge Name="Altimeter" Version="1.0">
   <Image Name="Altimeter_Background.bmp"/>
   <Element>
      <Position X="0" Y="0"/>
      <Select>
         <Value>(P:Local time,seconds) flr 2 % if{ (A:Rudder position,percent) near 8192 % 4095 > }
els{ 0 }</Value>
         <Case Value="0"></Case>
         <Case Value="1">
            <Image Name="Altimeter_Warning_Background.bmp"/>
         </Case>
      </Select>
   </Element>
   <Element>
      <Position X="0" Y="0"/>
      <Select>
         <Value>(P:Local time,seconds) flr 2 % if{ (A:Rudder position,percent) near 16384 % 8191 > }
els{ 0 }</Value>
         <Case Value="0"></Case>
         <Case Value="1">
            <Image Name="Altimeter_Warning_Background.bmp"/>
         </Case>
      </Select>
   </Element>
   <Element>
      <Position X="78" Y="79"/>
      <Image Name="Altimeter_Needle_1000.bmp">
         <Axis X="5" Y="5" PointsTo="East"/>
      </Image>
      <Rotate>
         <Value>(A:Indicated Altitude, meters) 1000 / </Value>
         <Nonlinearity>
            <Item Value="0" X="78" Y="15"/>
            <Item Value="5" X="78" Y="138"/>
         </Nonlinearity>
      </Rotate>
   </Element>
   <Element>
      <Position X="78" Y="79"/>
      <Image Name="Altimeter_Needle_100.bmp">
         <Axis X="5" Y="5" PointsTo="East"/>
      </Image>
      <Rotate>
         <Value>(A:Indicated Altitude, meters) 100 /</Value>
         <Nonlinearity>
            <Item Value="0" X="78" Y="15"/>
            <Item Value="5" X="78" Y="138"/>
         </Nonlinearity>
      </Rotate>
   </Element>
   <Mouse>
      <Help ID="HELPID_GAUGE_ALTIMETER"/>
      <Tooltip ID="TOOLTIPTEXT_ALTIMETER_METERS"/>
   </Mouse>
</Gauge>
```

Throttle Inicator.xml

```
<Gauge Name="Throttle Indicator" Version="1.0">
   <Image Name="Throttle_Background.bmp"/>
   <Element>
      <Position X="79" Y="79"/>
      <Image Name="Throttle_Needle.bmp" PointsTo="East">
         <Axis X="7" Y="6"/>
      </Image>
      <Rotate>
         <Value Minimum="0" Maximum="100">(A:General eng throttle lever position:1, percent)</Value>
         <Nonlinearity>
            <Item Value="0" X="49" Y="131"/>
            <Item Value="100" X="137" Y="57"/>
         </Nonlinearity>
         <Delay DegreesPerSecond="100"/>
      </Rotate>
   </Element>
   <Element>
      <Position X="6" Y="128"/>
      <Select>
         <Value>(A:Rudder position,percent) near 1024 % 511 ></Value>
        <Case Value="0">
            <Image Name="GreenLightOff.bmp"/>
         </Case>
         <Case Value="1">
            <Image Name="GreenLightOn.bmp"/>
         </Case>
      </Select>
   </Element>
   <Mouse>
      <Help ID="HELPID_GUAGE_THROTTLE_THROTTLE"/>
      <Tooltip ID="TOOLTIPTEXT_THROTTLE_THROTTLE_PERCENT"/>
   </Mouse>
</Gauge>
```

Heading Inicator.xml

```
<Gauge Name="Heading Indicator" Version="1.0">
   <Image Name="Heading_Background.bmp"/>
   <Element>
      <Position X="0" Y="0"/>
      <MaskImage Name="Heading_Mask.bmp">
         <Axis X="79" Y="79"/>
      </MaskImage>
      <Image Name="Heading_Card.bmp">
         <Axis X="79" Y="79"/>
      </Image>
      <Rotate>
         <Value>(A:Plane heading degrees true,radians) /-/</Value>
      </Rotate>
   </Element>
   <Mouse>
      <Help ID="HELPID_GAUGE_HEADING_INDICATOR"/>
      <Tooltip ID="TOOLTIPTEXT_HEADING_INDICATOR_HEADING"/>
   </Mouse>
</Gauge>
```

Vertical Speed Inicator.xml

```
<Gauge Name="Vertical Speed Indicator" Version="1.0">
   <Image Name="Vertical_Speed_Background.bmp"/>
   <Element>
      <Position X="80" Y="78"/>
      <Image Name="Vertical_Speed_Needle.bmp">
         <Axis X="6" Y="6" PointsTo="East"/>
      </Image>
      <Rotate>
         <Value Minimum="-10" Maximum="10">(A:Vertical speed, meter/second)</Value>
         <Nonlinearity>
            <Item Value="-10" X="139" Y="90"/>
            <Item Value="-5" X="74" Y="140"/>
            <Item Value="0" X="16" Y="78"/>
            <Item Value="5" X="74" Y="19"/>
            <Item Value="10" X="139" Y="69"/>
         </Nonlinearity>
         <Delay DegreesPerSecond="25"/>
      </Rotate>
   </Element>
   <Mouse>
      <Tooltip ID="TOOLTIPTEXT_VSI_METERS_PER_SEC"/>
   </Mouse>
</Gauge>
```

```
<Gauge Name="Left Panel" Version="1.0">
   <Image Name="Left_Panel_Background.bmp"/>
   <Update Hidden="No">
      <!-- AvionicsBatteryVoltsAlarm -->
      (A:Rudder position,percent) flr 2 % 1 == (>L:AvionicsBatteryVoltsAlarm, boolean)
      <!-- ServoBatteryVoltsAlarm -->
      (A:Rudder position,percent) flr 4 % 1 > (>L:ServoBatteryVoltsAlarm, boolean)
      <!-- MotorBatteryVoltsAlarm -->
      (A:Rudder position,percent) near 8 % 3 > (>L:MotorBatteryVoltsAlarm, boolean)
      <!-- BatteryTemperatureAlarm -->
      (A:Rudder position,percent) near 16 % 7 > (>L:BatteryTemperatureAlarm, boolean)
      <!-- MotorTemperatureAlarm -->
      (A:Rudder position,percent) near 32 % 15 > (>L:MotorTemperatureAlarm, boolean)
      <!-- FlapsAndSpoilersrArmed -->
      (A:Rudder position,percent) near 2048 % 1023 > (>L:FlapsAndSpoilersrArmed, boolean)
      <!-- ElevatorArmed -->
      (A:Rudder position,percent) near 4096 % 2047 > (>L:ElevatorArmed, boolean)
   </Update>
   <!-- Command State -->
   <Element>
      <Visible>(A:GENERAL ENG PROPELLER LEVER POSITION:3,percent) 0 ==</Visible>
      <Position X="115" Y="20"/>
      <Text X="130" Y="20" Length="5" Fixed="No" Font="Arial" FontHeight="15" FontWeight="400"
Charset="Default" Attributes="Normal" Adjust="Center" VerticalAdjust="Center" Multiline="No"
Color="#FF0000" BackgroundColor="transparent" HilightColor="white" Bright="Yes"
UseTransparency="Yes">
      <String>Pilot</String>
      </Text>
   </Element>
   <Element>
      <Visible>(A:GENERAL ENG PROPELLER LEVER POSITION:3,percent) 1 ==</Visible>
      <Position X="115" Y="20"/>
      <Text X="130" Y="20" Length="10" Fixed="No" Font="Arial" FontHeight="15" FontWeight="400"
Charset="Default" Attributes="Normal" Adjust="Center" VerticalAdjust="Center" Multiline="No"
Color="#FF0000" BackgroundColor="transparent" HilightColor="white" Bright="Yes"
UseTransparency="Yes">
      <String>Bus Driver</String>
      </Text>
   </Element>
   <Element>
      <Visible>(A:GENERAL ENG PROPELLER LEVER POSITION:3,percent) 2 ==</Visible>
      <Position X="115" Y="20"/>
      <Text X="130" Y="20" Length="10" Fixed="No" Font="Arial" FontHeight="15" FontWeight="400"
Charset="Default" Attributes="Normal" Adjust="Center" VerticalAdjust="Center" Multiline="No"
Color="#FF0000" BackgroundColor="transparent" HilightColor="white" Bright="Yes"
UseTransparency="Yes">
      <String>Autonomous</String>
      </Text>
   </Element>
   <Element>
      <Visible>(A:GENERAL ENG PROPELLER LEVER POSITION:3,percent) 3 ==</Visible>
      <Position X="115" Y="20"/>
      <Text X="130" Y="20" Length="14" Fixed="No" Font="Arial" FontHeight="15" FontWeight="400"
Charset="Default" Attributes="Normal" Adjust="Center" VerticalAdjust="Center" Multiline="No"
Color="#FF0000" BackgroundColor="transparent" HilightColor="white" Bright="Yes"
UseTransparency="Yes">
      <String>Starting Abort</String>
      </Text>
   </Element>
   <Element>
      <Visible>(A:GENERAL ENG PROPELLER LEVER POSITION:3,percent) 4 ==</Visible>
      <Position X="115" Y="20"/>
      <Text X="130" Y="20" Length="8" Fixed="No" Font="Arial" FontHeight="15" FontWeight="400"
Charset="Default" Attributes="Normal" Adjust="Center" VerticalAdjust="Center" Multiline="No"
Color="#FF0000" BackgroundColor="transparent" HilightColor="white" Bright="Yes"
UseTransparency="Yes">
      <String>Aborting</String>
      </Text>
   </Element>
   <Element>
      <Visible>(A:GENERAL ENG PROPELLER LEVER POSITION:3,percent) 5 ==</Visible>
      <Position X="115" Y="20"/>
      <Text X="130" Y="20" Length="18" Fixed="No" Font="Arial" FontHeight="15" fontWeight="400"
Charset="Default" Attributes="Normal" Adjust="Center" VerticalAdjust="Center" Multiline="No"
Color="#FF0000" BackgroundColor="transparent" HilightColor="white" Bright="Yes"
UseTransparency="Yes">
      <String>Parachute Deployed</String>
      </Text>
   </Element>
   <!-- Elevator Light -->
   <Element>
      <Position X="10" Y="83"/>
      <Select>
         <Value>(L:ElevatorArmed, boolean)</Value>
        <Case Value="0">
            <Image Name="GreenLightOff.bmp"/>
         </Case>
         <Case Value="1">
            <Image Name="GreenLightOn.bmp"/>
         </Case>
      </Select>
   </Element>
   <!-- Elevator Display -->
   <Element>
      <Visible>(L:ElevatorArmed, boolean)</Visible>
      <Position X="45" Y="90"/>
      <Text X="40" Y="20" Length="3" Fixed="No" Font="Quartz" FontHeight="0" FontWeight="400"
Charset="Default" Attributes="Normal" Adjust="Center" VerticalAdjust="Center" Multiline="No"
Color="#00FF00" BackgroundColor="transparent" HilightColor="white" Bright="Yes"
UseTransparency="Yes">
         <String>%((A:Elevator position,percent))%!3d!%</String>
      </Text>
   </Element>
   <!-- Flaps and Spoilers Light -->
   <Element>
      <Position X="109" Y="83"/>
      <Select>
         <Value>(L:FlapsAndSpoilersrArmed, boolean)</Value>
        <Case Value="0">
            <Image Name="GreenLightOff.bmp"/>
         </Case>
         <Case Value="1">
            <Image Name="GreenLightOn.bmp"/>
         </Case>
      </Select>
   </Element>
   <!-- Flaps Display -->
   <Element>
      <Visible>(L:FlapsAndSpoilersrArmed, boolean)</Visible>
      <Position X="145" Y="90"/>
      <Text X="40" Y="20" Length="3" Fixed="No" Font="Quartz" FontHeight="0" FontWeight="400"
Charset="Default" Attributes="Normal" Adjust="Center" VerticalAdjust="Center" Multiline="No"
Color="#00FF00" BackgroundColor="transparent" HilightColor="white" Bright="Yes"
UseTransparency="Yes">
         <String>%((A:GENERAL ENG THROTTLE LEVER POSITION:3,percent))%!3d!%</String>
      </Text>
   </Element>
   <!-- Spoilers Display -->
   <Element>
      <Visible>(L:FlapsAndSpoilersrArmed, boolean)</Visible>
      <Position X="205" Y="90"/>
      <Text X="40" Y="20" Length="3" Fixed="No" Font="Quartz" FontHeight="0" FontWeight="400"
Charset="Default" Attributes="Normal" Adjust="Center" VerticalAdjust="Center" Multiline="No"
Color="#00FF00" BackgroundColor="transparent" HilightColor="white" Bright="Yes"
UseTransparency="Yes">
         <String>%((A:GENERAL ENG MIXTURE LEVER POSITION:3,percent))%!3d!%%</String>
      </Text>
   </Element>
   <!-- Avionics Battery Voltage -->
   <Element>
      <Visible>(L:AvionicsBatteryVoltsAlarm, boolean)</Visible>
      <Position X="50" Y="155"/>
      <Image Name="Battery_Warning_Background.bmp"/>
   </Element>
   <Element>
```

```
      <Visible>(L:AvionicsBatteryVoltsAlarm, boolean)</Visible>
      <Position X="55" Y="160"/>
      <Text X="50" Y="20" Length="4" Fixed="No" Font="Quartz" FontHeight="0" FontWeight="100"
Charset="Default" Attributes="Normal" Adjust="Center" VerticalAdjust="Center" Multiline="No"
Color="#FFFFFF" BackgroundColor="transparent" HilightColor="white" Bright="Yes"
UseTransparency="Yes">
      <String>%((A:GENERAL ENG THROTTLE LEVER POSITION:2,percent))%!2.1f!%</String>
      </Text>
   </Element>
   <Element>
      <Visible>(L:AvionicsBatteryVoltsAlarm, boolean) 0 ==</Visible>
      <Position X="55" Y="160"/>
      <Text X="50" Y="20" Length="4" Fixed="No" Font="Quartz" FontHeight="0" FontWeight="100"
Charset="Default" Attributes="Normal" Adjust="Center" VerticalAdjust="Center" Multiline="No"
Color="#00FF00" BackgroundColor="transparent" HilightColor="white" Bright="Yes"
UseTransparency="Yes">
      <String>%((A:GENERAL ENG THROTTLE LEVER POSITION:2,percent))%!2.1f!%</String>
      </Text>
   </Element>
   <!-- Servo Battery Voltage -->
   <Element>
      <Visible>(L:ServoBatteryVoltsAlarm, boolean)</Visible>
      <Position X="120" Y="155"/>
      <Image Name="Battery_Warning_Background.bmp"/>
   </Element>
   <Element>
      <Visible>(L:ServoBatteryVoltsAlarm, boolean)</Visible>
      <Position X="125" Y="160"/>
      <Text X="50" Y="20" Length="4" Fixed="No" Font="Quartz" FontHeight="0" FontWeight="100"
Charset="Default" Attributes="Normal" Adjust="Center" VerticalAdjust="Center" Multiline="No"
Color="#FFFFFF" BackgroundColor="transparent" HilightColor="white" Bright="Yes"
UseTransparency="Yes">
      <String>%((A:GENERAL ENG MIXTURE LEVER POSITION:2,percent))%!2.1f!%</String>
      </Text>
   </Element>
   <Element>
      <Visible>(L:ServoBatteryVoltsAlarm, boolean) 0 ==</Visible>
      <Position X="125" Y="160"/>
      <Text X="50" Y="20" Length="4" Fixed="No" Font="Quartz" FontHeight="0" FontWeight="100"
Charset="Default" Attributes="Normal" Adjust="Center" VerticalAdjust="Center" Multiline="No"
Color="#00FF00" BackgroundColor="transparent" HilightColor="white" Bright="Yes"
UseTransparency="Yes">
      <String>%((A:GENERAL ENG MIXTURE LEVER POSITION:2,percent))%!2.1f!%</String>
      </Text>
   </Element>
   <!-- Motor Battery Voltage -->
   <Element>
      <Visible>(L:MotorBatteryVoltsAlarm, boolean)</Visible>
      <Position X="190" Y="155"/>
      <Image Name="Battery_Warning_Background.bmp"/>
   </Element>
   <Element>
      <Visible>(L:MotorBatteryVoltsAlarm, boolean)</Visible>
      <Position X="195" Y="160"/>
      <Text X="50" Y="20" Length="4" Fixed="No" Font="Quartz" FontHeight="0" FontWeight="100"
Charset="Default" Attributes="Normal" Adjust="Center" VerticalAdjust="Center" Multiline="No"
Color="#FFFFFF" BackgroundColor="transparent" HilightColor="white" Bright="Yes"
UseTransparency="Yes">
      <String>%((A:GENERAL ENG PROPELLER LEVER POSITION:2,percent))%!01.1f!%</String>
      </Text>
   </Element>
   <Element>
      <Visible>(L:MotorBatteryVoltsAlarm, boolean) 0 ==</Visible>
      <Position X="195" Y="160"/>
      <Text X="50" Y="20" Length="4" Fixed="No" Font="Quartz" FontHeight="0" FontWeight="100"
Charset="Default" Attributes="Normal" Adjust="Center" VerticalAdjust="Center" Multiline="No"
Color="#00FF00" BackgroundColor="transparent" HilightColor="white" Bright="Yes"
UseTransparency="Yes">
      <String>%((A:GENERAL ENG PROPELLER LEVER POSITION:2,percent))%!01.1f!%</String>
      </Text>
   </Element>
   <!-- Battery Temperature -->
   <Element>
```

```
      <Visible>(L:BatteryTemperatureAlarm, boolean)</Visible>
      <Position X="50" Y="240"/>
      <Image Name="Temperature_Warning_Background.bmp"/>
   </Element>
   <Element>
      <Visible>(L:BatteryTemperatureAlarm, boolean)</Visible>
      <Position X="55" Y="245"/>
      <Text X="85" Y="20" Length="3" Fixed="No" Font="Quartz" FontHeight="0" FontWeight="100"
Charset="Default" Attributes="Normal" Adjust="Center" VerticalAdjust="Center" Multiline="No"
Color="#FFFFFF" BackgroundColor="transparent" HilightColor="white" Bright="Yes"
UseTransparency="Yes">
      <String>%((A:GENERAL ENG MIXTURE LEVER POSITION:1, percent))%!3d!%</String>
      </Text>
   </Element>
   <Element>
      <Visible>(L:BatteryTemperatureAlarm, boolean) 0 ==</Visible>
      <Position X="55" Y="245"/>
      <Text X="85" Y="20" Length="3" Fixed="No" Font="Quartz" FontHeight="0" FontWeight="100"
Charset="Default" Attributes="Normal" Adjust="Center" VerticalAdjust="Center" Multiline="No"
Color="#00FF00" BackgroundColor="transparent" HilightColor="white" Bright="Yes"
UseTransparency="Yes">
      <String>%((A:GENERAL ENG MIXTURE LEVER POSITION:1, percent))%!3d!%</String>
      </Text>
   </Element>
   <!-- Motor Temperature -->
   <Element>
      <Visible>(L:MotorTemperatureAlarm, boolean)</Visible>
      <Position X="155" Y="240"/>
      <Image Name="Temperature_Warning_Background.bmp"/>
   </Element>
   <Element>
      <Visible>(L:MotorTemperatureAlarm, boolean)</Visible>
      <Position X="160" Y="245"/>
      <Text X="85" Y="20" Length="3" Fixed="No" Font="Quartz" FontHeight="0" FontWeight="100"
Charset="Default" Attributes="Normal" Adjust="Center" VerticalAdjust="Center" Multiline="No"
Color="#FFFFFF" BackgroundColor="transparent" HilightColor="white" Bright="Yes"
UseTransparency="Yes">
      <String>%((A:GENERAL ENG PROPELLER LEVER POSITION:1, percent))%!3d!%</String>
      </Text>
   </Element>
   <Element>
      <Visible>(L:MotorTemperatureAlarm, boolean) 0 ==</Visible>
      <Position X="160" Y="245"/>
      <Text X="85" Y="20" Length="3" Fixed="No" Font="Quartz" FontHeight="0" FontWeight="100"
Charset="Default" Attributes="Normal" Adjust="Center" VerticalAdjust="Center" Multiline="No"
Color="#00FF00" BackgroundColor="transparent" HilightColor="white" Bright="Yes"
UseTransparency="Yes">
      <String>%((A:GENERAL ENG PROPELLER LEVER POSITION:1, percent))%!3d!%</String>
      </Text>
   </Element>
</Gauge>
```

```
<Gauge Name="Nav Lights" Version="1.0">
   <Image Name="Nav_Lights_Background.bmp"/>
   <Update Hidden="No">
      <!-- NavValid -->
      (A:Rudder position,percent) near 32768 % 16383 > (>L:NavValid, boolean)
      <!-- RCLinkUp -->
      (A:Rudder position,percent) near 65536 % 32767 > (>L:RCLinkUp, boolean)
      <!-- NavArrivalAlarm -->
      (A:Rudder position,percent) near 64 % 31 > (>L:NavArrivalAlarm, boolean)
      <!-- ReturningToOrigin -->
      (A:Rudder position,percent) near 128 % 63 > (>L:ReturningToOrigin, boolean)
      <!-- WayPointValid -->
      (A:Rudder position,percent) near 256 % 127 > (>L:WayPointValid, boolean)
      <!-- GPSValid -->
      (A:Rudder position,percent) near 512 % 255 > (>L:GPSValid, boolean)
</Update>
   <!-- Nav Valid-->
   <Element>
      <Position X="89" Y="10"/>
      <Select>
         <Value>(L:NavValid, boolean)</Value>
        <Case Value="0">
            <Image Name="RedLightOn.bmp"/>
         </Case>
         <Case Value="1">
            <Image Name="GreenLightOn.bmp"/>
         </Case>
      </Select>
   </Element>
   <!-- RC Link-->
   <Element>
      <Position X="220" Y="10"/>
      <Select>
         <Value>(L:RCLinkUp, boolean)</Value>
        <Case Value="0">
            <Image Name="RedLightOn.bmp"/>
         </Case>
         <Case Value="1">
            <Image Name="GreenLightOn.bmp"/>
         </Case>
      </Select>
   </Element>
   <!-- Nav Arrival Alarm-->
   <Element>
      <Position X="29" Y="75"/>
      <Select>
         <Value>(L:NavArrivalAlarm, boolean)</Value>
        <Case Value="0">
            <Image Name="YellowLightOff.bmp"/>
         </Case>
         <Case Value="1">
            <Image Name="YellowLightOn.bmp"/>
         </Case>
      </Select>
   </Element>
   <!-- Returning to Origin -->
   <Element>
      <Position X="89" Y="75"/>
      <Select>
         <Value>(L:ReturningToOrigin, boolean)</Value>
        <Case Value="0">
            <Image Name="YellowLightOff.bmp"/>
         </Case>
         <Case Value="1">
            <Image Name="YellowLightOn.bmp"/>
         </Case>
      </Select>
   </Element>
   <!-- Way Point Valid -->
   <Element>
      <Position X="149" Y="75"/>
      <Select>
         <Value>(L:WayPointValid, boolean)</Value>
        <Case Value="0">
            <Image Name="RedLightOn.bmp"/>
         </Case>
         <Case Value="1">
            <Image Name="GreenLightOn.bmp"/>
         </Case>
      </Select>
   </Element>
   <!-- GPS Valid -->
   <Element>
      <Position X="209" Y="75"/>
      <Select>
         <Value>(L:GPSValid, boolean)</Value>
        <Case Value="0">
            <Image Name="RedLightOn.bmp"/>
         </Case>
         <Case Value="1">
            <Image Name="GreenLightOn.bmp"/>
         </Case>
      </Select>
   </Element>
</Gauge>
```

GPS.xml

```
<Gauge Name="GPS" Version="1.0">
       <Image Name="GPS_Background.bmp"/>
       <Macro Name="c">C:fs9gps</Macro>
       <Macro Name="g">C:fs9gps</Macro>
       <Macro Name="CalculateZoomFactor">2000 1500 1000 500 350 200 150 100 50 35 20 15 10 5 3.5
2.0 1.5 1.0 0.576037 0.329164 0.246873 0.164582 0.082290 0 24 @1 case</Macro>
       <Update Hidden="No">
              <!-- calculate zoom factor range -->
              (@g:map_ZoomStep0) 0 == if{ 6 (>@g:map_ZoomStep0) }
              (@g:map_ZoomStep0) (>@g:map_ZoomStep)
              @CalculateZoomFactor((@g:map_ZoomStep)) (>@g:map_ZoomFactor)
              (A:GENERAL ENG PROPELLER LEVER POSITION:4, degrees) 57.3 / (A:Plane heading degrees
true, radians) >= if{ A:GENERAL ENG PROPELLER LEVER POSITION:4, degrees) 57.3 / 3.1416 - (A:Plane
heading degrees true, radians) >= if{ (A:GENERAL ENG PROPELLER LEVER POSITION:4, degrees) 57.3 /
(A:Plane heading degrees true, radians) 6.2832 + - } els{ (A:GENERAL ENG PROPELLER LEVER POSITION:4,
degrees) 57.3 / (A:Plane heading degrees true, radians) - } } els{ (A:Plane heading degrees true,
radians) 3.1416 - (A:GENERAL ENG PROPELLER LEVER POSITION:4, degrees) 57.3 / >= if{ (A:GENERAL ENG
PROPELLER LEVER POSITION:4, degrees) 57.3 / 6.2832 (A:Plane heading degrees true, radians) - + }
els{ (A:GENERAL ENG PROPELLER LEVER POSITION:4, degrees) 57.3 / (A:Plane heading degrees true,
radians) - } } (>L:Waypoint tracking error, radians)
       </Update>
       <Element Name="ViewWindow" Container="Yes">
              <Position X="17.5" Y="17.5"/>
              <RelClip Left="0" Top="0" Width="230" Height="145"/>
              <Element Name="MapView Pages">
                     <Position X="0" Y="0"/>
                     <CustomDraw Name="fs9gps:map" X="230" Y="145" LayerTerrain="Yes"
LayerBorders="Yes" LayerFlightPlan="Yes" FlightPlanLineWidth="2.5"
ActiveColorLayerFlightPlan="0x7010B0" PastColorLayerFlightPlan="0xF0F0F0"
ColorLayerFlightPlan="0xF0F0F0" Bright="Yes" CenterX="115">
                            <CenterY>115</CenterY>
                            <Zoom>(@g:map_ZoomFactor) 1852.0 *</Zoom>
                            <Latitude>(A:GPS POSITION LAT, Radians)</Latitude>
                            <Longitude>(A:GPS POSITION LON, Radians)</Longitude>
                            <Heading>(A:GPS GROUND TRUE TRACK, Radians)</Heading>
                            <DetailLayerTerrain>(@g:setup_ColorTerrain) 0 == if{ 1 } els{ 2
}</DetailLayerTerrain>
                     </CustomDraw>
                     <!-- Aircraft symbol -->
                     <Element>
                            <Position X="115" Y="115"/>
                            <Polygon FillColor="%((@g:setup_ColorTerrain) if{ 0x503030 } els{
0xF0D0D0 })" Bright="Yes">
                                   <Point X=" 0.5" Y="-7.5"/>
                                   <Point X="   2" Y="-6.5"/>
                                   <Point X="   2" Y="  -4"/>
                                   <Point X=" 7.5" Y="  -4"/>
                                   <Point X=" 7.5" Y="-1.5"/>
                                   <Point X="   2" Y="-1.5"/>
                                   <Point X="   2" Y=" 4.5"/>
                                   <Point X=" 4.5" Y=" 4.5"/>
                                   <Point X=" 4.5" Y=" 6.5"/>
                                   <Point X="-4.5" Y=" 6.5"/>
                                   <Point X="-4.5" Y=" 4.5"/>
                                   <Point X="  -2" Y=" 4.5"/>
                                   <Point X="  -2" Y="-1.5"/>
                                   <Point X="-7.5" Y="-1.5"/>
                                   <Point X="-7.5" Y="  -4"/>
                                   <Point X="  -2" Y="  -4"/>
                                   <Point X="  -2" Y="-6.5"/>
                                   <Point X="-0.5" Y="-7.5"/>
                            </Polygon>
                     </Element>
                     <!-- Way Point -->
                     <Element>
                            <Position X="115" Y="115"/>
                            <Image Name="Way_Point.bmp">
                                   <Axis X="4.5" Y="4.5"/>
                            </Image>
                            <Shift>
                                   <Value>(A:GENERAL ENG MIXTURE LEVER POSITION:4, meter)
(@g:map_ZoomFactor) / 70 * 1852 /</Value>
                                   <Scale Y="-1"/>
```

```
                </Shift>
                <Rotate>
                        <Value>(L:Waypoint tracking error, radians)</Value>
                </Rotate>
                <Clip Top="17.5" Left="17.5" Width="230" Height="145"/>
        </Element>
        <!-- Heading arc -->
        <Element>
                <Position X="115" Y="138"/>
                <CustomDraw Name="fs9gps:rose" X="216" Y="108" CenterX="108"
CenterY="115" Radius="115" LineWidth="1" Font="Arial" FontSize="12" BigFontSize="13" Bright="Yes">
                        <Color>(@g:setup_ColorTerrain) if{ 0x703030 } els{ 0xF0D0D0
}</Color>
                        <BackgroundColor>(@g:setup_ColorTerrain) if{ 0 } els{ 0x080808
}</BackgroundColor>
                        <Heading>(A:Plane heading degrees true, Radians)</Heading>
                        <Pivot X="108" Y="115"/>
                </CustomDraw>
        </Element>
        <Element>
                <Element>
                        <Position X="115" Y="138"/>
                        <Visible>(L:Waypoint tracking error, degrees) abs 60
&lt;</Visible>
                        <Polyline Color="0x208020" LineWidth="3">
                                <Point X="-5" Y="12"/>
                                <Point X=" 0" Y="0"/>
                                <Point X=" 5" Y="12"/>
                                <Pivot X="0" Y="113"/>
                        </Polyline>
                        <Rotate>
                                <Value>(L:Waypoint tracking error, radians)</Value>
                        </Rotate>
                </Element>
                <Element>
                        <Position X="115" Y="138"/>
                        <Visible>(L:Waypoint tracking error, degrees) 60 ></Visible>
                        <Polyline Color="0x208020" LineWidth="3">
                                <Point X="-2" Y="0"/>
                                <Point X=" 2" Y="6"/>
                                <Point X="-2" Y="12"/>
                                <Pivot X="0" Y="113"/>
                        </Polyline>
                        <Rotate>
                                <Value>60 dgrd</Value>
                        </Rotate>
                </Element>
                <Element>
                        <Position X="115" Y="138"/>
                        <Visible>(L:Waypoint tracking error, degrees) -60 &lt;</Visible>
                        <Polyline Color="0x208020" LineWidth="3">
                                <Point X=" 2" Y="0"/>
                                <Point X="-2" Y="6"/>
                                <Point X=" 2" Y="12"/>
                                <Pivot X="0" Y="113"/>
                        </Polyline>
                        <Rotate>
                                <Value>-60 dgrd</Value>
                        </Rotate>
                </Element>
        </Element>
        <!-- Heading pointer -->
        <Element>
                <Position X="115" Y="22"/>
                <Polyline LineWidth="1" Color="%((@g:setup_ColorTerrain) if{ 0x703030 }
els{ 0xF0D0D0 })" Bright="Yes">
                        <Point X="-22" Y="-6"/>
                        <Point X="-20" Y="-4"/>
                        <Point X="-4" Y="-4"/>
                        <Point X="0" Y="0"/>
                        <Point X="4" Y="-4"/>
                        <Point X="20" Y="-4"/>
                        <Point X="22" Y="-6"/>
```

```
                            </Polyline>
                     </Element>
                     <!-- Heading display -->
                     <Element>
                            <Position X="115" Y="0"/>
                            <FormattedText X="40" Y="17" Font="Arial Black" FontSize="14"
Adjust="Center" Color="Yellow" BackgroundColor="#080808" Bright="Yes">
                                   <String>%((A:Plane heading degrees true, degrees)
d360)%!03d!\{dplo= }</String>
                                   <Pivot X="20" Y="0"/>
                            </FormattedText>
                     </Element>
                     <Element>
                            <Position X="71" Y="2"/>
                            <Text X="20" Y="11" Length="3" Font="Arial Black" FontSize="8"
Adjust="Center" VerticalAdjust="Bottom" Multiline="No" Color="0xF0D0D0" BackgroundColor="#080808"
Bright="Yes">
                                   <String>TRK</String>
                            </Text>
                     </Element>
                     <!--Desired heading display -->
                     <Element>
                            <Position X="18" Y="23"/>
                            <FormattedText X="40" Y="17" Font="Arial Black" FontSize="14"
Adjust="Center" Color="Yellow" BackgroundColor="#080808" Bright="Yes">
                                   <String>%((A:Rudder position,percent) near 256 % 127
>)%{if}%((A:GENERAL ENG PROPELLER LEVER POSITION:4, degrees) d360)%!03d!%{else}_ _ _%{end}\{dplo=
}</String>
                                   <Pivot X="20" Y="14"/>
                            </FormattedText>
                     </Element>
                     <Element>
                            <Position X="8" Y="0"/>
                            <Text X="20" Y="11" Length="3" Font="Arial Black" FontSize="8"
Adjust="Center" VerticalAdjust="Bottom" Multiline="No" Color="0xF0D0D0" BackgroundColor="#080808"
Bright="Yes">
                                   <String>DTK</String>
                            </Text>
                     </Element>
                     <!-- Distance to waypoint -->
                     <Element>
                            <Position X="205" Y="23"/>
                            <FormattedText X="45" Y="17" Font="Arial Black" FontSize="14"
Adjust="Center" Color="Yellow" BackgroundColor="#080808" Bright="Yes">
                                   <String>%((A:Rudder position,percent) near 256 % 127
>)%{if}%((A:GENERAL ENG MIXTURE LEVER POSITION:4, meter) near)%!d!%{else}_ _ _%{end}\{dpl=
m}</String>
                                   <Pivot X="20" Y="14"/>
                            </FormattedText>
                     </Element>
                     <Element>
                            <Position X="195" Y="0"/>
                            <Text X="20" Y="11" Length="3" Font="Arial Black" FontSize="8"
Adjust="Center" VerticalAdjust="Bottom" Multiline="No" Color="0xF0D0D0" BackgroundColor="#080808"
Bright="Yes">
                                   <String>DIS</String>
                            </Text>
                     </Element>
                     <!-- Altitude -->
                     <Element>
                            <Position X="205" Y="124"/>
                            <FormattedText X="45" Y="17" Font="Arial Black" FontSize="14"
Adjust="Center" Color="Yellow" BackgroundColor="#080808" Bright="Yes">
                                   <String>%((A:INDICATED ALTITUDE, meter) near)%!d!%\{dpl=
m}</String>
                                   <Pivot X="20" Y="14"/>
                            </FormattedText>
                     </Element>
                     <Element>
                            <Position X="162" Y="113"/>
                            <Text X="20" Y="11" Length="3" Font="Arial Black" FontSize="8"
Adjust="Center" VerticalAdjust="Bottom" Multiline="No" Color="0xF0D0D0" BackgroundColor="#080808"
Bright="Yes">
```

```
                                    <String>ALT</String>
                              </Text>
                        </Element>
                        <!-- Altitude of waypoint -->
                        <Element>
                              <Position X="205" Y="142"/>
                              <FormattedText X="45" Y="17" Font="Arial Black" FontSize="14"
Adjust="Center" Color="Yellow" BackgroundColor="#080808" Bright="Yes">
                                    <String>%((A:Rudder position,percent) near 256 % 127
>)%{if}%((A:GENERAL ENG THROTTLE LEVER POSITION:4, meter) near)%!d!%{else}_ _ _%{end}\{dpl=
m}</String>
                                    <Pivot X="20" Y="14"/>
                              </FormattedText>
                        </Element>
                        <Element>
                              <Position X="162" Y="131"/>
                              <Text X="20" Y="11" Length="2" Font="Arial Black" FontSize="8"
Adjust="Center" VerticalAdjust="Bottom" Multiline="No" Color="0xF0D0D0" BackgroundColor="#080808"
Bright="Yes">
                                    <String>WP</String>
                              </Text>
                        </Element>
                  </Element>
            </Element>
            <Mouse Name="MouseRoutines">
                  <!-- TERR Button -->
                  <Macro Name="TerrainButton">(@g:setup_ColorTerrain) !
(>@g:setup_ColorTerrain)</Macro>
                  <Area Name="TERR Button" Left="15" Top="172" Width="40" Height="15">
                        <Click Kind="LeftSingle+Leave">
                              (M:Event) 'LeftSingle' scmp 0 == (@g:terrButtonDown) 0 == and if{ 1
(>@g:terrButtonDown) @TerrainButton }
                              (M:Event) 'Leave' scmp 0 == (@g:terrButtonDown) 1 == and if{ 0
(>@g:terrButtonDown) }
                        </Click>
                        <Cursor Type="Hand"/>
                        <Tooltip ID="TOOLTIPTEXT_GPS_TERR_BUTTON"/>
                  </Area>
                  <!-- Zoom Button -->
                  <Macro Name="ZOOMINButton">
                        (@g:map_ZoomStep0) -- 1 max (>@g:map_ZoomStep0)
                  </Macro>
                  <Macro Name="ZOOMOUTButton">
                        (@g:map_ZoomStep0) ++ 20 min (>@g:map_ZoomStep0)
                  </Macro>
                  <Area Left="230" Top="170" Width="20" Height="20">
                        <Click Kind="LeftSingle+Leave">
                              (M:Event) 'LeftSingle' scmp 0 == (@g:zoominButtonDown) 0 == and if{ 1
(>@g:zoominButtonDown) @ZOOMINButton }
                              (M:Event) 'Leave' scmp 0 == (@g:zoominButtonDown) 1 == and if{ 0
(>@g:zoominButtonDown) }
                        </Click>
                        <Cursor Type="UpArrow"/>
                        <Tooltip ID="TOOLTIPTEXT_GPS_RANGE_UP"/>
                  </Area>
                  <Area Left="205" Top="170" Width="20" Height="20">
                        <Click Kind="LeftSingle+Leave">
                              (M:Event) 'LeftSingle' scmp 0 == (@g:zoomoutButtonDown) 0 == and if{ 1
(>@g:zoomoutButtonDown) @ZOOMOUTButton }
                              (M:Event) 'Leave' scmp 0 == (@g:zoomoutButtonDown) 1 == and if{ 0
(>@g:zoomoutButtonDown) }
                        </Click>
                        <Cursor Type="DownArrow"/>
                        <Tooltip ID="TOOLTIPTEXT_GPS_RANGE_DOWN"/>
                  </Area>
            </Mouse>
            <Keys>
                  <On Event="GPS_TERRAIN_BUTTON">@TerrainButton</On>
                  <On Event="GPS_ZOOMIN_BUTTON">@ZOOMINButton</On>
                  <On Event="GPS_ZOOMOUT_BUTTON">@ZOOMOUTButton</On>
            </Keys>
</Gauge>
```

```
<Gauge Name="Clock" Version="1.0">
   <Element>
      <Position X="0" Y="0"/>
      <Text X="70" Y="20" Length="8" Fixed="No" Font="Quartz" FontHeight="0" FontWeight="400"
Charset="Default" Attributes="Normal" Adjust="Center" VerticalAdjust="Center" Multiline="No"
Color="#FF0000" BackgroundColor="transparent" HilightColor="white" Bright="Yes"
UseTransparency="Yes">
         <String>%((P:Local time,hours) flr 12 % 3 +)%!2d!:%((P:Local time,minutes) flr 60
%)%!02d!:%((P:Local time,seconds) flr 60 %)%!02d!</String>
         <Hilight>
            <SELECT_CLOCK_HOURS Start="1" End="2"/>
            <SELECT_CLOCK_MINUTES Start="4" End="5"/>
            <SELECT_CLOCK_SECONDS Start="7" End="8"/>
         </Hilight>
      </Text>
   </Element>
   <Mouse>
      <Tooltip ID="TOOLTIPTEXT_CLOCK"/>
   </Mouse>
</Gauge>
```

Appendix E

IEEE Conference Paper