Department of Electrical and Electronic Engineering
Part IV Project
2004
3D Interface for the Unmanned Aerial Vehicle

by Brendan Cervin, Chris Mills(Project Partner)
Supervisor: Burkhard Wuensche

September 13, 2004

UAV Project Abstract

The New Zealand Defence Force is commissioning an Unmanned Aerial Vehicle (UAV) as part of their long term strategy. The vehicle is to support units in the field with reconnaissance. The UAV has been developed by Phil Strong at the Defence Technology Agency (DTA) [1].

- The first aim is to integrate the UAV flight data into a flight simulator to create a 2D and 3D view of the flight.
- The second aim is to ensure the scenery within the flight simulator is accurate. The use of high resolution data from JGSF in Geographical Information System (GIS) format will be used to create detailed and accurate scenery. The JGSF data must be converted into formats to be used in a flight simulator. To support a 2D view, the created scenery should also be viewable in bitmap format.
- Integrate the UAV Interface into the Virtual Maritime System Architecture VMSA to simulate Virtual Battleground Scenarios

A 3D environment was created for the UAV by interfacing with the FlightGear flight simulator. TerraGear was used to compile low quality scenery for FlightGear. JGSF data will be used to create high quality scenery, however due to time limitations this was not implemented although research supports that scenery can be successfully created. The VMSA interface to FlightGear was not implemented due to lack of time however there is some support for such an interface from FlightGear.

Acknowledgments

This project has drawn on help from various people. Phil Strong has developed the UAV and the UAV simulator which is the base of much of our work. Noah Coad has written a wrapper class for accessing the Windows serial port libraries using C#. Timothy J. Krell for developing a serial port package that accesses and controls the serial port itself. Wayne Ewington from Microsoft arranged for the free use of Microsoft Flight Simulator 2002. The developers of the FlightGear open source project, related applications for use of FlightGear and the documentation which is an integral part of the project. Special thanks to Burkhard Wuensche for his guidance and advice on the project.

Contents

List of Figures

1 Introduction

1.1 Unmanned Aerial Vehicle (UAV) Background and Development

The New Zealand Defence Force is commissioning an Unmanned Aerial Vehicle (UAV) as part of their long term strategy. The vehicle is to support units in the field with reconnaissance. The UAV has been developed by Phil Strong at the Defence Technology Agency (DTA) [1]. The DTA is the research and development arm of the New Zealand Defence Force.

The UAV is a remotely controlled aircraft with cameras and communication equipment on board, the UAV is flown by autopilot and directed by an operator on the ground. Currently the DTA have 2D map image software which they use to direct the UAV in flight. The map has roads buildings and elevation contour lines.

Its aim is to be used for reconnaissance of units in the field. The UAV has a communication system already developed to provide detailed information about its environment such as exact position, altitude and status of equipment like servo motors.

The DTA require a 3D Interface to the UAV to provide more accurate information about the UAV flight path. They wish to use a flight simulator to provide a 3D environment with which they can clearly locate land features to help operate the UAV.

The Joint Geospatial Support Facility (JGSF, see glossary) [2] provides the New Zealand Defence force with accurate map data. They have extensive map data in various formats that can be used to create scenery for the flight simulator.

The DTA are also involved in development of the Virtual Maritime System Architecture (VMSA). VMSA is a way of describing units. Units are described using federates, a federate can be a communication federate, weapon federate, command federate or counter-measure federate. By combining and reusing federates a unit can be described. VMSA describes each sub-component to create the whole unit. These units are used to for Virtual Battleground Scenarios which are battle simulations conducted on computer networks

VMSA is used to simulate the relation between units. The DTA is creating VMSA aircraft units for simulation. The DTA hope that the 3D interface of the UAV can be used to show VMSA units in a battleground scenario.

1.2 Aims for the UAV project

The DTA would like a more accurate interface to view the UAV in flight. This is to improve the operator's ability to safely control the aircraft. In order to accomplish this wish two aims have been devised for the project:

- The first aim is to integrate the UAV flight data into a flight simulator to create a 2D and 3D view of the flight.

- The second aim is to ensure the scenery within the flight simulator is accurate. The use of high resolution data from JGSF in Geographical Information System (GIS) format will be used to create detailed and accurate scenery. The JGSF data must be converted into formats to be used in a flight simulator. To support a 2D view, the created scenery should also be viewable in bitmap format.

Another interest of the DTA is creating a connection between the above 3D interface and VMSA. This resulted in the final aim of the project:

- Integrate the UAV Interface into the VMSA to simulate Virtual Battleground Scenarios

1.3 Project Scope

Initially the scope of the project was wide and undefined. The aims of the project did not tie down the direction of research into a small area. Any number of tools could have been implemented to complete the aims, this resulted in a great deal of research to find and identify useful tools. The initially undefined nature of the implementation made research difficult many avenues were investigated which lead nowhere. The scope was eventually tied to our chosen flight simulator but often wandered in the hope of finding other useful tools.

1.4 Report Overview

This report will detail all research and work done during the course of this project. The report will give details of the project in the order in which they were conducted. The first aim will be covered in the beginning, describing the research and implementation of the solution. The scenery creation will be covered next with background information about scenery tools and data types followed by tool compilation and scenery generation. The final section will provide a description of research and a possible implementation of the VMSA requirements. A conclusion will describe the accomplishments and difficulties of the project. The glossary, references and appendix will come last.

2 Initial Project Design

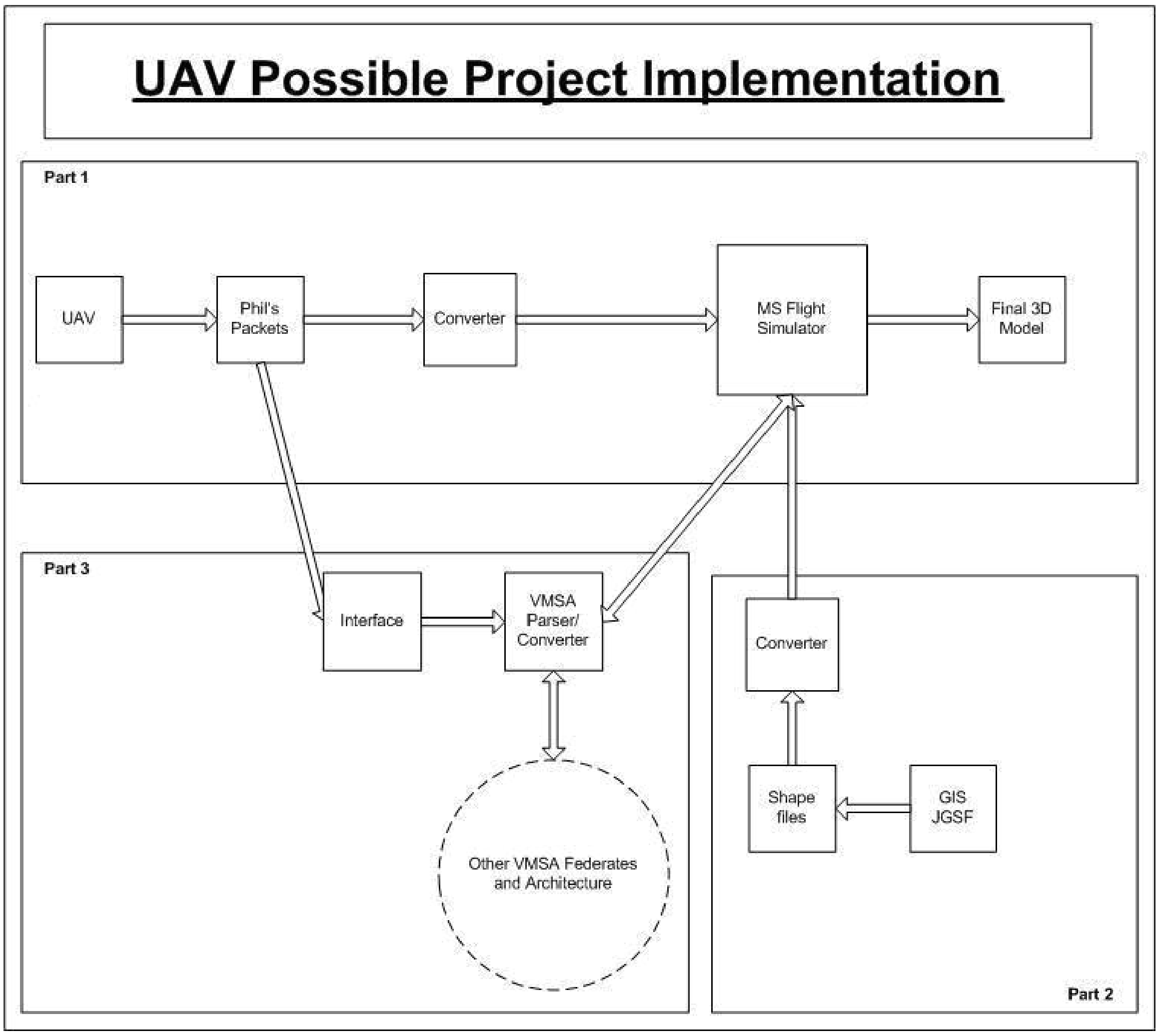
The project was divided into three parts to represent the aims. Figure 1 below describes the initial project implementation. This shows the relation between each part of the project, the key components and flow of data amongst them.

The data flow starts with the UAV which outputs packets that are defined by Phil Strong's Packet Specification [3, 4]. The converter in part one plays the key role of using Phil's packets to provide data to the flight simulator in the appropriate format.

The second part of the project is the creation of scenery for use in the flight simulator. JGSF have a great deal of GIS information on terrain around New Zealand and the world. They can produce accurate shape files for use in applications; these shape files have information on roads, terrains, buildings, elevation and much more. A converter is needed to convert JGSF data into scenery for use by the flight simulator.

The VMSA interface requires the sending of data about multiple units into the flight simulator used to display the UAV. A form of multiplayer support could be used to implement this requirement. Support for as many units as possible is wanted. The 3D interface should allow the Virtual Battleground Scenario to be viewed from the UAV cockpit and as many other angles as possible.

Figure 1: Project Description



3 Flight Simulator Research

In general flight simulators allow users to fly a plane in a 3D environment. Much of the focus of flight simulators goes into realistic modelling of aircraft in flight. We will be primarily interested in the scenery and communication abilities of the flight simulators.

3.1 Requirements

- The flight simulator must show a plane being controlled by UAV in 3D environment
- Accurate scenery is available or can be created
- The flight simulator must be able to incorporate other units into the display for the VMSA aim, this could be provided through some multiplayer functionality

The first requirement means that the flight simulator must have the ability to allow the aircraft to be controlled by external data. The flight simulator aircraft must show as much of the UAV data as possible like location, altitude, pitch, roll, yaw and even fuel status.

For the flight simulator to be useful the scenery data must be accurate. Either scenery must be provided or tools for creating scenery are available.

The flight simulator should provide a form of multiplayer support that allows for many units to be displayed, preferably not just aircraft but other ground units. The units should be represented by appropriate 3D objects.

Initial investigation suggested that Microsoft Flight Simulator [5] and FlightGear [6] were two possible flight simulators.

3.2 Microsoft Flight Simulator 2002

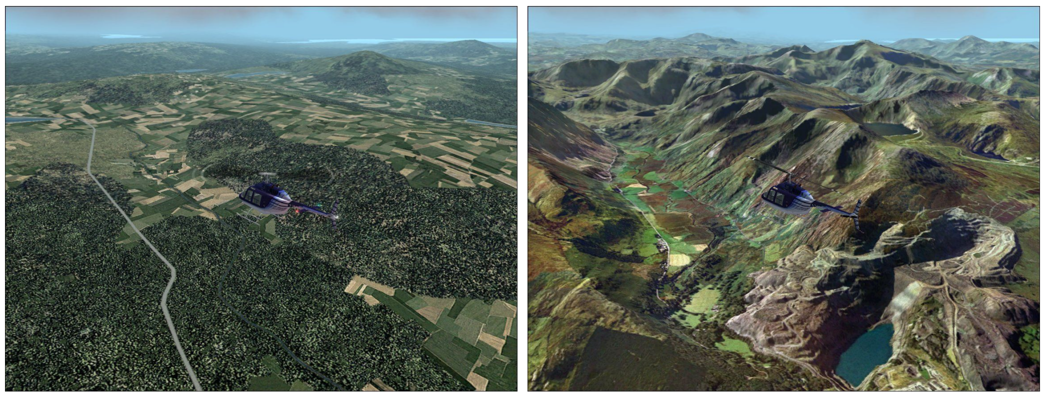
Microsoft provides extensive online documentation for their flight simulator. There is a great deal of information about creating plug-in applications to add functionality to MSFS. Many applications have already been developed and can be purchased online. MSFS has a comprehensive set of aircraft and scenery objects for cities, this can be further enhanced by commercial plug-ins with even more data.

MSFS costs \$50-\$100 depending on the version. It is the most popular flight simulator in the world and enjoys support and documentation from many enthusiasts.

Direct X provides standard libraries to interface with MSFS, Microsoft also provides Software Development Kits (SDK's)[7] for creating aircraft, scenery objects and other more complicated plug-ins. Scenery within MSFS uses the data format BGL. Microsoft provides it's own tools for scenery generation using data from the United States Geological Survey data. Details for scenery creation are available in “Creating Terrain and Land Classification“ [8].

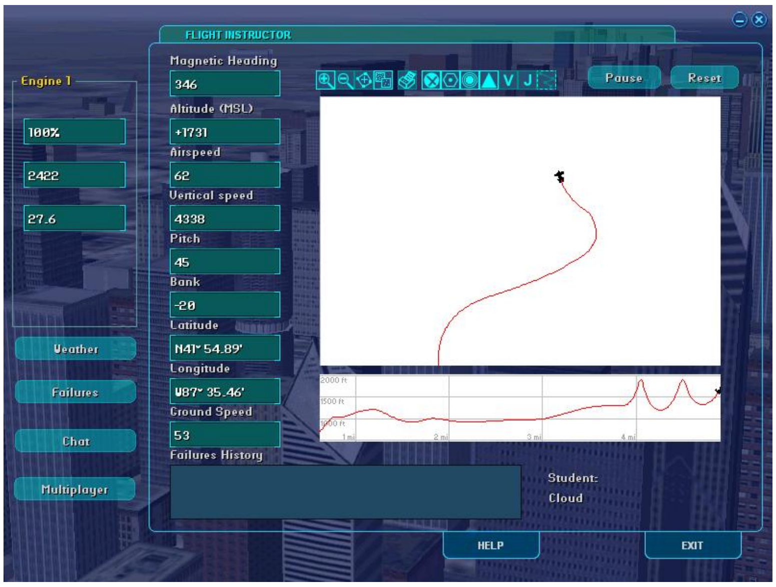
There are support tools for creating scenery available for free online. Some tools are available at SHOF [9] that allow for importing aerial photographs and BGL scenery analysis. Some of the commercial plug-ins available for MSFS are very complex. One such plug-in called VFR [10] provides photographic and terrain scenery to transform standard MSFS scenery see Figure 2. This is only available for the UK though. There are other comparable products on the market but not for New Zealand specifically.

Figure 2: Comparison between standard Microsoft scenery and commercial scenery created by VFR



The DirectX DirectPlay package is used for multi-player connections to share flight information. This interface does not allow for one player to sit in another player's cockpit. This means if an instance of MSFS hosts a connection to the UAV interface it can not show the cockpit view of the UAV. A flight instructor mode allows a user to see another player's plane in a 2D interface as shown in Figure 3, but not within a 3D interface.

Figure 3: 2D Instructor Mode



Engine 1
100%
2422
27.6

Magnetic Heading
346
Altitude (MSL)
+1731
Airspeed
62
Vertical speed
4338
Pitch
45
Bank
-20
Latitude
N41° 54.89'
Longitude
W87° 35.46'
Ground Speed
53
Failures History

Student:
Cloud

While researching the MSFS several spikes (small test programs) were created in C++ to test some capabilities of MSFS. These discovered it wasn't possible to sit in the cockpit of a client's plane (the UAV) from the host.

Microsoft has multiplayer functionality that allows a maximum of sixteen players to fly within the same game. Graham Macferson has created an interface to MSFS for VMSA. This interface allowed Mr Macferson to view 3D objects placed in the scenery which represented VMSA units. The maximum number of players made this implementation limited this interface.

3.3 FlightGear

FlightGear is an open source project with comprehensive documentation provided. There are many open source applications related to FlightGear. FlightGear was developed as an academic project for realistic flight simulation. It has various flight models that can be used to describe the flight characteristics of aircraft. The terrain in FlightGear is constructed using common formats for easy creation and modification. TerraGear [11] is a related open source toolkit that creates FlightGear scenery from publicly available geographical data.

The multi-player engine for FlightGear is currently in alpha development, this is in house testing. While FlightGear doesn't currently support many players at once it has already implemented many forms of communication including reading and writing information from files, over a network using sockets with either User Datagram Packet (UDP) or Transmission Control Protocol (TCP) and even a serial port.

Atlas [12] is a related open source project that can construct bitmap images from FlightGear scenery files. Atlas can display bitmap images while communicating with a running FlightGear program to display 2D flight path information of the aircraft in FlightGear.

FlightGear uses tiles to represent the entire world. There is accurate modelling of stars and planets that change depending on the axis of the world and the data and time of day. FlightGear can connect to the internet to retrieve real-time weather data which can effect weather effects such as visibility.

There are user forums [13] for both FlightGear and TerraGear which contain a great deal of useful information and tips. The forums are used extensively by developers with questions and many of the senior developers and experienced users reply quickly with useful information. FlightGear has been the basis for many university academic research projects and this is supported and encouraged by the FlightGear community.

When FlightGear is configured to receive data it can display a remote control aircraft. This means that it displays exactly what it is told to by the incoming data, when FlightGear hosts a game it can act as a slave to the UAV data. This ability is desirable for the project as it represents the best solution for simulating and displaying the UAV in flight

3.4 MSFS and FlightGear comparison

3.4.1 Microsoft Flight Simulator 2002

- A lot of online documentation and SDK's give detailed information about creating plug-ins
- DirectPlay package provides good communication interface to MSFS
- Many plug-ins are available
- Extra aircrafts and scenery are irrelevant in the decision since the project only flies one aircraft and will create it's own scenery
- Only capable of 2D display of UAV in flight using the Instructor Mode

3.4.2 FlightGear

- Open source and fully documented
- Related open source applications
  - Atlas map creator and 2D displayer looks useful for the second component of the project
  - TerraGear creates scenery from standard GIS data formats
  - Flight Scenery Designer [14] creates scenery from photos
- Simple network communication using UDP or TCP
- Host acts as slave to incoming client flight data to display UAV in 3D environment

FlightGear was chosen primarily because it was the only simulator that could display the UAV in 3D flight. It also provides effective communication methods and tools for scenery creation.

4 UAV Interface

4.1 Research of UAV communication

In the real implementation, communication occurs with the UAV through a transmitter connected to a computer with a serial port. This was unrealistic for testing purposes so a UAV simulator called HawkSim was created by Phil Strong for use during the interface development. During testing HawkSim was connected to one serial port and the UAV Interface was running on another. The two serial ports were connected by a cable; this implementation simulates the real UAV for the UAV Interface.

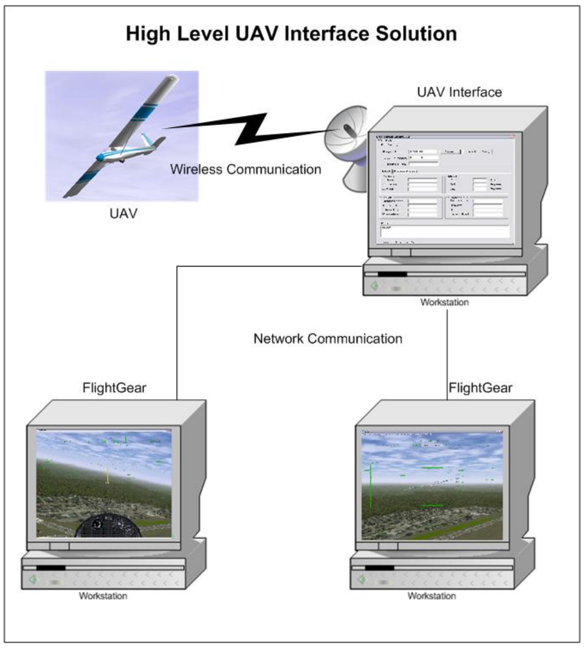
A communication protocol has been specified for the UAV by Phil Strong. This protocol is private and details will be kept vague. The UAV sends data in packets, there are many packets for different information like positional packets, motion packets and environmental packets. Several spikes were written in C++ and C# to test serial port communication.

C++ serial port communication is basic, there is limited high level data handling and event based communication. Inexperience using C++ also created difficulties in programming. C# serial port communication was implemented much faster. The UAV Interface was implemented in C# due to personal preference.

4.2 Proposed Solution

Figure 4 below shows the envisaged overall solution. The UAV will communicate wirelessly with a transmitter; the transmitter will be connected to a workstation running the UAV Interface using a serial port. The UAV Interface will control communication with the UAV by requesting and processing packets; once data is accumulated it will be re-packaged data and send it to one or more computers running FlightGear.

Figure 4: Proposed Solution



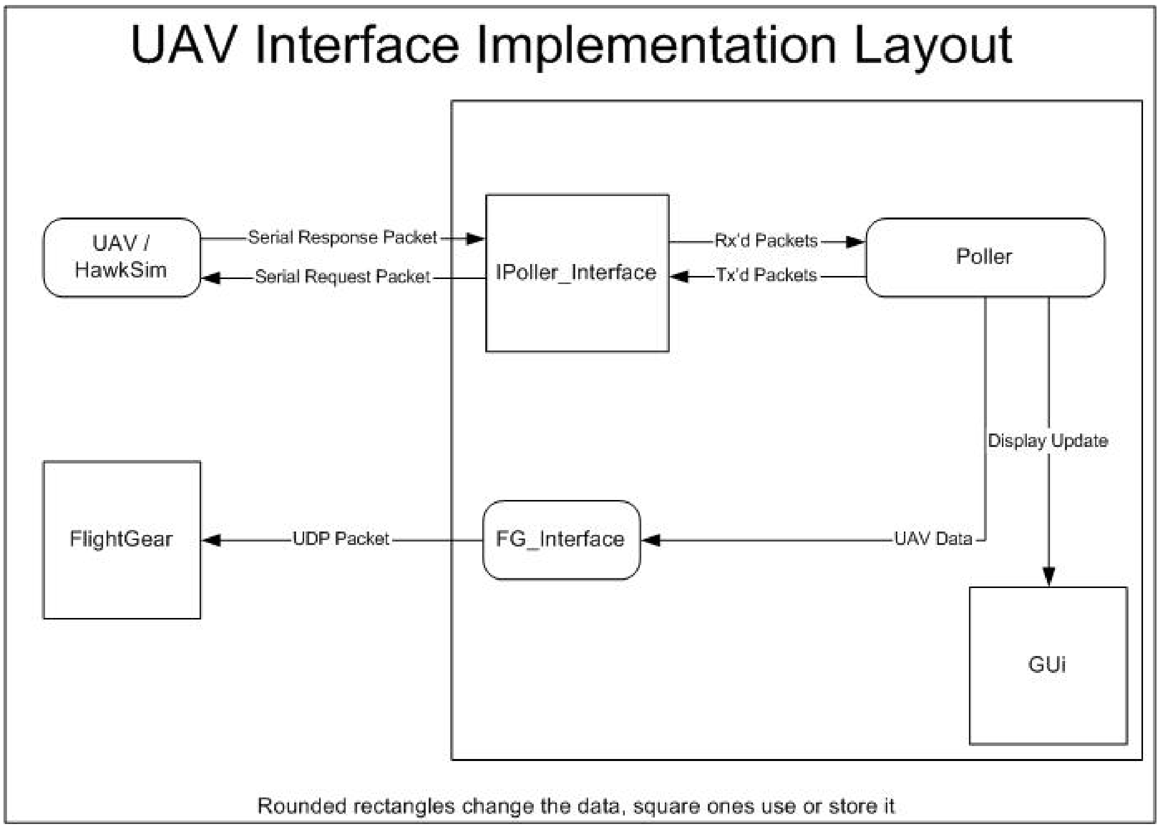
4.3 UAV Interface Design

Two designs for the UAV Interface were implemented. The first design was nearly completely implemented before it was re-factored to improve modularity, installation and good coding practices. There are a few key requirements of the UAV Interface. Firstly it must provide serial communication with the UAV; secondly it must provide communication with FlightGear. The other requirements are good coding practices, a GUI to provide configuration options, current UAV data display and any relevant debug information. The UAV Interface was divided in to four main sections to implement the above design.

- The windows form which provides the GUI, this holds an instance of the UAV_Poller
- The UAV_Poller which was central to communication between the UAV and FlightGear
- Serial port communication
- FlightGear communication interface called FG_Interface

This implementation can be represented by Figure 5

Figure 5: Proposed high level implementation



4.4 Serial Port Communication Changes

The main difference between the first and final design was the serial port communication. Windows provides a communication library called AxMSCommLib; this can be used for serial communication. It is the simplest and fastest way to implement serial communication in C#. To use this library however it must be installed and registered on the computer, the library was bundled with commercial programs like Visual Studio 6 or earlier.

To circumvent the registration of AxMSCommLib we used a wrapper class created by Noah Coad [15] which encapsulated a registered version of AxMSCommLib. This wrapper class successfully provided easy event based communication and data handling for the UAV Interface's serial port communication.

The draw back of this implementation was that every computer which ran the UAV Interface had to install the AxMSCommLib file into the windows directory. This was also not a strictly legal application of the AxMSCommLib file. To fix this problem the serial port communication was changed and also improved to be more modular.

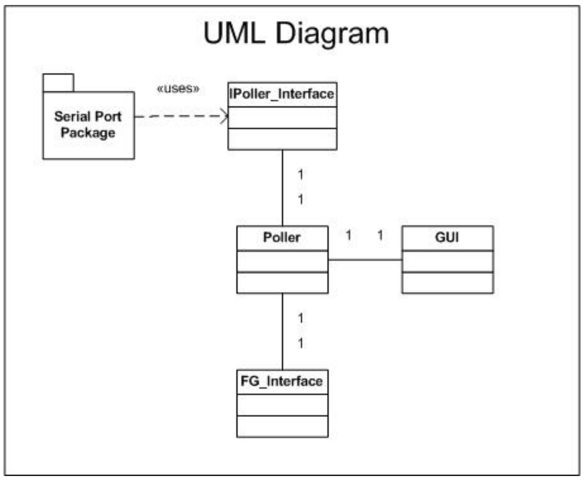
A serial port package developed by Timothy J. Krell [16] was used to replace the AxMSCommLib. The key class within the package was named SerialBase.cs this class was altered to implement an interface class called IPoller_Interface.cs. The serial port communication is now modular. Any class that implements the IPoller_Interface can be used to communicate with the UAV.

4.5 Final Implementation

4.5.1 Implementation Overview

Below is the final design for the working UAV Interface, an UML diagram showing the relation of classes is shown in Figure 6. Below the diagram are descriptions of the key components and reasons for why they were implemented in this manor.

Figure 6: Proposed high level implementation



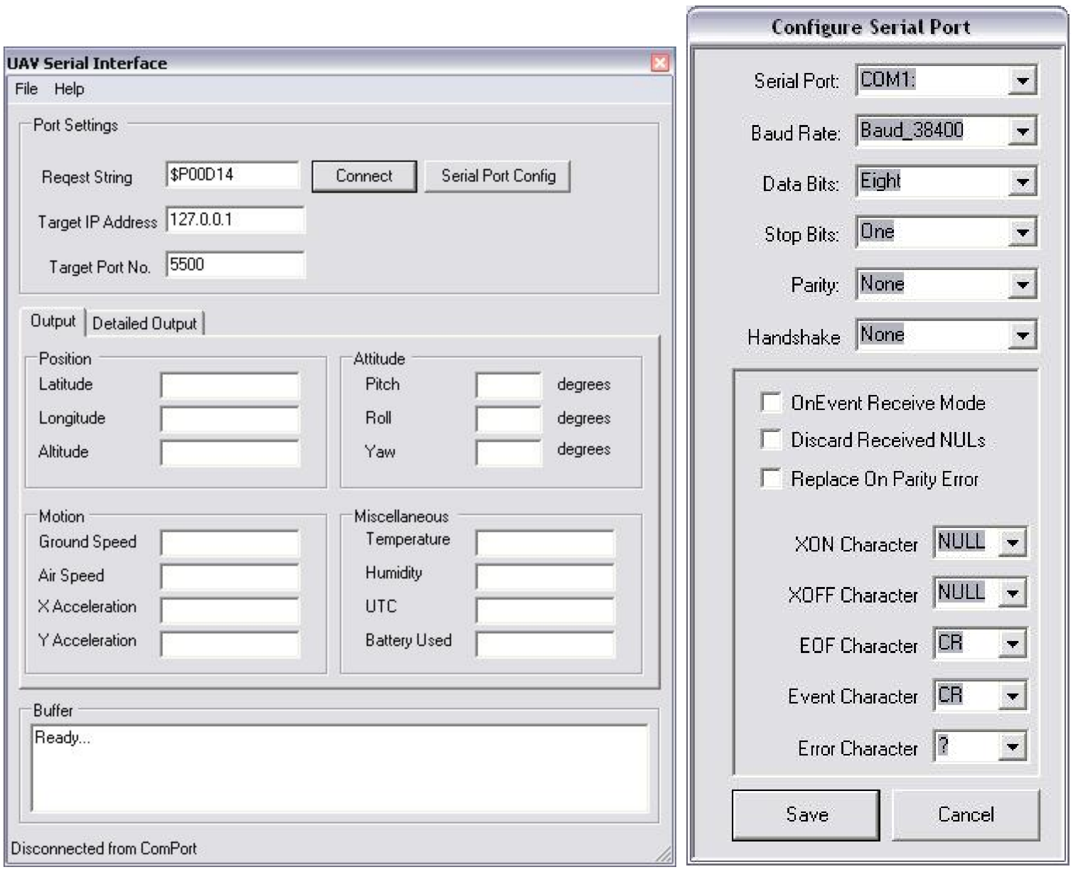
4.5.2 UAV_Interface

This is a GUI class concerned only with the display of data. The UAV_Poller has been created to remove all the functional code from the display, the UAV_Interface holds an instance of the UAV_Poller. The GUI provides configuration options for the communication between the UAV and FlightGear, this configuration is passed to the UAV_Poller to create connections with the UAV and FlightGear. All data which UAV_Poller processes is sent to the UAV_Interface through events, the use of events is a good coding practice as it reduces dependency between the classes.

The user interface is shown below with the serial port configuration panel in Figure 7 . The top of the serial port shows configuration options. The 'Request String' option is the string sent to the UAV requesting data, if there are any errors in the string no data will be returned, it has a default value which requests all relevant data the UAV has for FlightGear. Below it are the options for the connection to the computer running FlightGear, these are the computer's IP address and the port number. The button 'Serial Port Config' allows the user to alter all the options associated with a serial port and save them to a '.cfg' automatically so they are remembered next time the UAV_Interface is executed. All the options are used when the 'Connect' button is pressed; altering the configuration after a connection has been made will have no effect. Alterations will take effect only once a new connection is made.

The middle area of the GUI shows the current values returned by the UAV. This is useful to check that FlightGear is displaying the UAV correctly. The text box at the bottom of the GUI shows debug information. Most importantly when data is received from the UAV if there are any errors in transmission the data is deemed corrupt and disregarded, the debug window provides notification of this event.

Figure 7: UAV Interface and Serial Configuration window


UAV Serial Interface

Port Settings
Reqest String: $P00D14
Target IP Address: 127.0.0.1
Target Port No.: 5500

Buffer
Ready...

Disconnected from ComPort

Configure Serial Port
Serial Port: COM1:
Baud Rate: Baud_38400
Data Bits: Eight
Stop Bits: One
Parity: None
Handshake: None
OnEvent Receive Mode
Discard Received NULs
Replace On Parity Error
XON Character: NULL
XOFF Character: NULL
EOF Character: CR
Event Character: CR
Error Character: ?

4.5.3 UAV_Poller

This is the central class for communication and processing. It holds the connection to both FlightGear and the UAV. The UAV connection is with the IPoller_Interface over a serial port. The FlightGear connection is through the FG_Interface. The UAV_Poller requests and processes data from the UAV, the data is received in hexadecimal strings. Some conversion must be done between float formats and once all the data is processed both the FG_Interface and the UAV_Interface are supplied with the data.

Phil Strong implemented a cyclic redundancy check (CRC) number into his packet specification to check data integrity. The UAV_Poller processes each packet checking the CRC number. This is done by the adding the exclusive or (XOR) values of the returned data to ensure it equals the checksum value. If they are not equal an error is detected and the data is disregarded. The UAV_Poller most also convert the float values sent by the UAV into the standard IEEE float format. The UAV contains microchips manufactured by CCS [17], their float values vary from the C# implementation of the IEEE float. The UAV_Poller converts the returned float values from the CCS float value to the IEEE float value.

4.5.4 IPoller_Interface

This is a simple interface which Timothy J Krell's complex serial port package implements. The advantage of the IPoller_Interface is that if the serial port communication is swapped for another medium as long as the connection implements the IPoller_Interface this solution will continue working.

4.5.5 FG_Interface

The FlightGear Interface utilised the FGNetFDM class. This class was made primarily for network communication between FlightGear instances. The class was written to describe an aircraft's Flight Dynamic Model (FDM) this is how FlightGear describes the properties of an aircraft such as it's location, speed, heading, pitch, yaw, roll and much more data that wasn't relevant to the UAV.

To communicate with FlightGear this class needs to be instantiated and filled with data about the UAV. Once an instance of the class is full of UAV data it is then sent across the network to FlightGear who uses the data to display the UAV. The class is written in C++ and the UAV Interface has been written in C#, to allow C# to use the FDM class it was re-written in managed C++ code. This meant adding support to the C++ class for managed memory (garbage collection), once C++ manages its own memory it can be run along side C# code. This is a key ability of the .NET framework that Microsoft has created.

The FlightGear Interface is written to include the FDM and also socket communication for sending the FDM across the network. It is placed in a Dynamic Linked Library for use by the C# UAV Interface.

FlightGear documentation discusses geodetic and geocentric coordinate systems [18]. Geodetic specifies latitude and longitude for a perfectly round world while geocentric recognises the ellipse shape of the world and there are slightly different latitude and longitude values. Taking this into account it was important the correct UAV position was given to FlightGear.

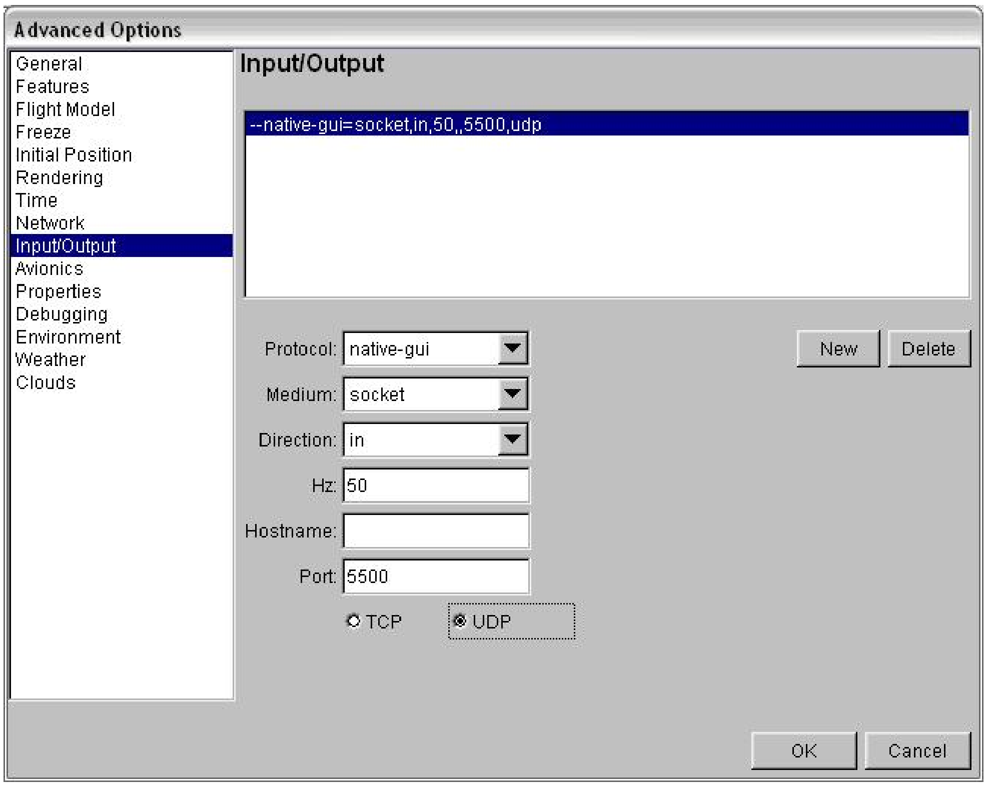
Traditionally, maps and GPS systems have provided geodetic data and this is the case with the UAV GPS system. While FlightGear uses geocentric data internally it recognises the common use of the geodetic system and the FGNetFDM supports geodetic data. FlightGear converts the geodetic values itself for display of the UAV.

4.6 FlightGear Configuration

FlightGear needs to be configured to display a remote controlled aircraft. This required two advanced options to be set. Firstly the Flight Model needed to be set to 'external', this tells FlightGear it is not to compute the movement of the plane this will be provided by an 'external' source.

The next option is to tell FlightGear where the external model is coming from. This requires setting up input / output options. Figure 8 below shows the configuration needed. The communication is over the network socket 5500, using UDP at a frequency of 50 Hz. This is the same frequency which the UAV Interface is requesting and sending packets to the UAV and FlightGear.

Figure 8: FlightGear Configuration for communication with UAV



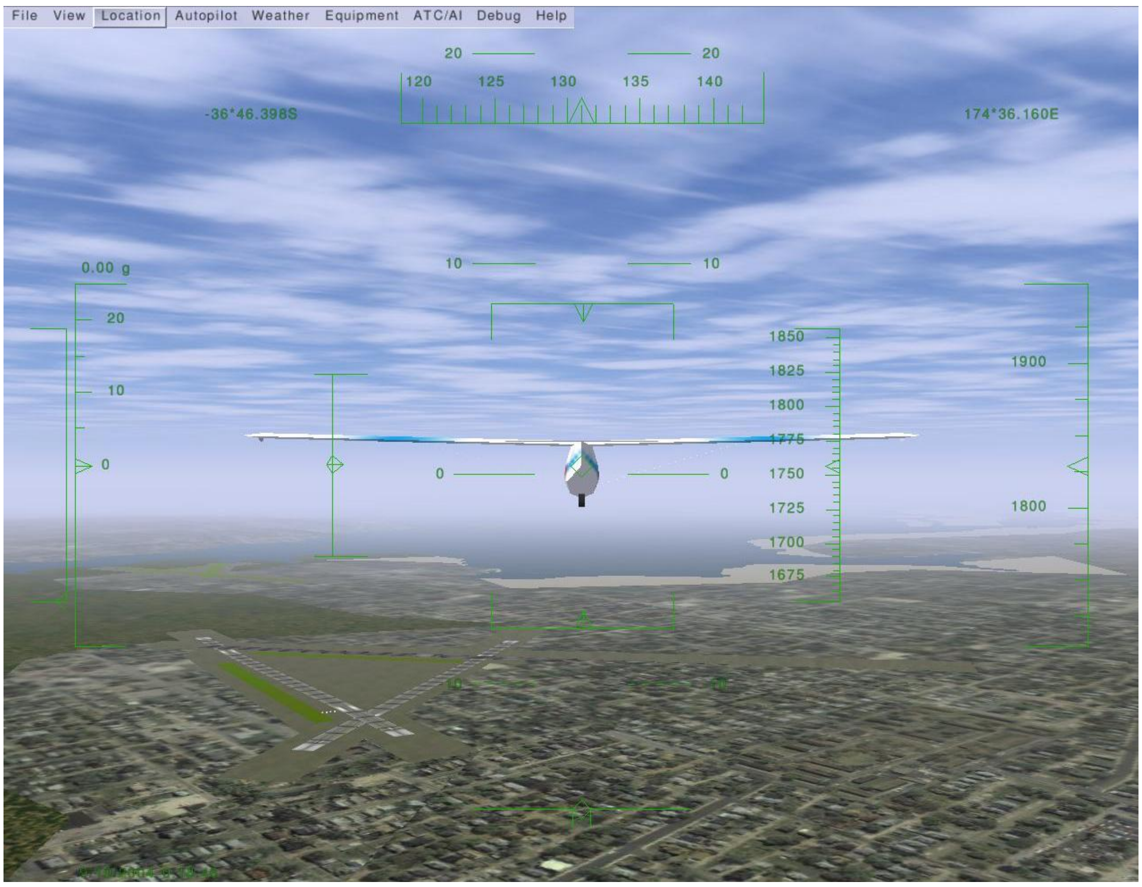
```
--native-gui=socket,in,50,,5500,udp
```

Protocol: native-gui
Medium: socket
Direction: in
Hz: 50
Hostname:
Port: 5500
TCP [✓ UDP]

4.7 Results

This implementation was tested on two computers over a network. One computer ran HawkSim and the UAV Interface; the second computer ran FlightGear as described above. Data was successfully communicated from the UAV to FlightGear which displayed the aircraft moving accurately. When the UAV altitude decreased the pitch of the UAV reflected this, likewise the roll and yaw reflected the movement of the UAV. Below are screen shots of FlightGear controlled by the UAV Interface. Figure 9 shows the UAV above Whenuapai Airbase with Auckland in the background.

Figure 9: FlightGear displaying the UAV in flight about Whenuapai Airbase


File View Location Autopilot Weather Equipment ATC/AI Debug Help
-36°46.398S
174°36.160E
0.00 g

5 Scenery Creation

5.1 Research

FlightGear uses publicly available terrain data to generate scenery. For this reason they support the commonly used data formats Geographic Information System (GIS). The quality of the terrain data varies greatly around the world, in North America data is available down to 30 metre resolution.

However data for New Zealand is not as good, the data suggested by FlightGear has a 1 kilometre resolution. David Megginson a head developer for FlightGear said "If you want scenery of places out in the wilderness such as New Zealand, then you're out of luck" [19]. This requires the project to use data provided by the Joint Geospatial Support Facility which is as low as 20 metre resolution.

5.2 Data Types

Terragear uses GIS data to create FlightGear scenery. There are two distinct forms of data, raster and vector. Vector data describes points or areas, an area might be a land mass, lake or even something intangible like a city or country boundaries. It can also be used to describe points like roads, buildings, rivers or pipes. Terragear supports reading vector data from the Vector Product Format (VPF, see glossay) which is the same format as the vector data the JGSF will provide.

The other data type is raster. This is a grid like series of values such that no location data is stored because the position of values within the raster file infers the location. Data is sampled every few metres the suggested data available for New Zealand is sampled every 1 kilometre [19]. The JGSF data is down to 20 metres. This format is used to specify elevation data, it is important we have high resolution data so that hills and valleys are accurately displayed in the scenery.

The raster data that our project dealt in primarily was Digital Elevation Maps (DEM, see glossary) and Shuttle Radar Tomography Mission (SRTM, see glossary). This was supported widely and TerraGear documentation used this data in its tutorial. The FlightGear Scenery Tutorial [19] was created by David Megginson. This tutorial describes the process used to create basic scenery for FlightGear.

5.3 Atlas

Atlas is a tool that provides two functions. Firstly it can create bitmap images from the scenery used by FlightGear. This is a function the DTA have expressed interest in; they have other applications that would benefit from 2D maps of the terrain. The second function of Atlas is to connect to FlightGear while it's running and track the movement of the FlightGear aircraft on the 2D bitmap image Atlas has created.

This application is fairly simple and will meet part of the second requirement. It will provide the 2D interface for the UAV in flight. However it will need to be compiled in Windows so that the DTA can use it.

5.4 FlightGear Scenery Designer

This project supplies functionality to import photos into FlightGear to act as scenery [14]. The photos can be placed over elevation data. There are tools for adjusting photos to fit neatly into other FlightGear scenery. The other side of the scenery designer is creation and placement of 3D objects into FlightGear such as buildings, towers or bridges.

5.5 TerraGear

TerraGear is an open source scenery generation toolkit developed for FlightGear. It contains many tools that can operate on several data formats to create FlightGear scenery. The toolkit source is available for download, it was developed on Linux. There is support for compilation and use of Terragear on Linux but very little for other systems.

The DTA prefers the solution to be usable on windows since that is there default operating system. For this reason several attempts were made to compile Terragear for use on Windows.

5.6 Terragear Compilation

5.6.1 CygWin

Due to the DTA preference for Windows we first tried to compile Terragear in CygWin which is a Linux emulation for Windows. CygWin can compile Linux specific code for use in Windows. CygWin was used first because the Terragear Mailing List [13] said Terragear had been compiled successfully on it. The entire CygWin package was installed to ensure all required components were installed.

However after many attempts Terragear would not compile, instead throwing errors stating required packages weren't installed.

This issue could not be overcome; the required packages were installed and located in the correct place. Work compiling Terragear on CygWin was ended due to these frustrating errors.

5.6.2 Windows

It was thought Terragear could be compiled successfully on Windows. As progress was made compiling parts of Terragear more complicated problems arose. Code was placed into Terragear files that would run dependant on the operating system; this was so in future Terragear could be compiled on both Windows and Linux using the same source. The first and most evident issue was the file system. Windows and Linux use different file separators so wherever there was access to the file system code was inserted so if the current operating system was Windows the backslashes would be replaced with forward slashes.

The next issue that was discovered was within a program called DemChop. This program divided all the elevation data into small files that could be processed more easily. It read and wrote data to files in a very precise format. Strings were read in from the file using 'substring' which returns the exact location with a specified string by indexing into the string. DemChop expected the data to be in a precise location. However the Windows compiler was slightly different to the Linux GCC compiler.

When Windows writes a double value to a file it uses three digits to specify the exponent, under Linux only two digits are used. So when DemChop came to read the double value back using the substring command it did not correctly find the data. This bug was found and corrected so that data was correctly read in. The occurrence of this bug made aware the possibility of other insidious differences in the compiled Terragear toolkit that may lead to a failure.

There were a number of native Linux commands used by Terragear; these consisted of fork, mkdir, and also socket communication. As a result solutions were made wherever these native commands were used. The mkdir command was used by Terragear with an optional command '-p' which told Linux to recursively create folders. The '-p' command would not work under Windows and had to be changed.

The fork command is used by the fgfs-tools-server to create multithreading behaviour. This was replaced under Windows using the Thread class. Socket communication also needed to be altered to work under Windows as well.

TerraGear was successfully compiled using Windows .NET Compiler Version 7. Scenery was created without error by the TerraGear toolkit. The scenery however would not display under FlightGear. No relevant debug information was found which could be used to debug the scenery. With the due date of the project approaching the focus became producing working scenery rather than a Windows compilation. So Windows was abandoned and compilation on Linux begun.

5.6.3 Linux

Work compiling TerraGear was initially done on RedHat Enterprise Edition. The same errors encountered during the CygWin compilation appeared and so another flavour of Linux was attempted. RedHat 8 became the next test bed for TerraGear compilation. The compilation was successful and test elevation data was downloaded from the United States Geological Survey in DEM format. The vector data was downloaded from the National Geospatial-Intelligence Agency in VPF format known as VMap0 which is accurate between 2040 meters and 4,270 depending on the source [20].

Scenery was successfully created using the data from above. This scenery could be displayed in FlightGear running under both Windows and Linux. It was noticed that the scenery created under Linux was several times larger in disk space than the Windows created scenery. The goal was now creation of scenery from the JGSF data.

5.7 Joint Geospatial Support Facility Data

Once we had correctly created scenery JGSF supplied us with data to create accurate New Zealand scenery. We requested to data formats; the first was raster data in DEM format and the second was vector data in VPF format. We received three sets of data from JGSF

- Shape files (NZ Topographic Data, see glossary) which are contour maps containing both raster elevation data and vector data of roads, rivers, buildings and much more
- Digital Terrain Elevation Maps (DTED, see glossary) which is a format close to DEM
- VPF files in Vector Map Level 1 (VMap1) format

The VMap1 data is compatible with TerraGear and is not publicly available. There are some problems with these data formats. Firstly shape files are not supported directly by FlightGear, secondly DTED files are not supported. Research began in order to convert JGSF data to formats that TerraGear supported. There are various conversion tools such as Geographic Resources Analysis Support System (GRASS) [21] which have been investigated and do show potential.

After contacting JGSF it was discovered they can provide the data in the formats we require, due to time constraints however this has not been tested any further. It is still believed that TerraGear can be used to create scenery from the JGSF data.

5.8 Results

TerraGear was compiled on both Linux and Windows. Scenery was only created successfully on Linux, this scenery can be used on Windows by FlightGear. JGSF have not supplied the required formats for scenery creation. Research shows that once the data formats are available scenery can be successfully created.

6 VMSA Interface

The third requirement was to allow for many units to be viewed within the 3D environment of the UAV. While there is only a testing version of the multiplayer engine for FlightGear which has been called buggy and unreliable, there is still potential. The multiplayer is believed to support the FGNetFDM which was discussed earlier in the FG_Interface.

The use of FGNetFDM in multiplayer would allow the reuse of the FG_Interface to send data about units into FlightGear. There is also the possibility of displaying ground and sea units within FlightGear as well. The FDM contains a variable to specify an objects height above ground; this could be set to zero to keep ground units on the ground permanently.

7 Conclusion

The aims of the project were:

- Integrate the UAV flight data into a flight simulator to create a 2D and 3D view of the flight.
- Ensure the scenery within the flight simulator is accurate. The use of high resolution data from JGSF in Geographical Information System (GIS) format will be used to create detailed and accurate scenery. The JGSF data must be converted into formats to be used in a flight simulator. To support a 2D view, the created scenery should also be viewable in bitmap format.
- Integrate the UAV Interface into the VMSA to simulate Virtual Battleground Scenarios.

Both Microsoft Flight Simulator and FlightGear were extensively researched for use in the solution. FlightGear was chosen for its ability to display the UAV in 3D as well as its communication support and expected multiplayer engine.

The solution to the 3D Interface was

- Implement communication across the serial port with the UAV.
- The UAV data is processed by the UAV_Poller then passed to the GUI and the FG_Interface.
- The GUI displays current UAV data and debug information.
- The FG_Interface wraps the UAV data into the FlightGear Flight Dynamic Model and then sends it to FlightGear across the network using socket communication.

FlightGear effectively displayed real-time data from the UAV in a 3D environment.

The second aim of the project was the creation of accurate scenery for use in the 3D environment using high resolution data from JGSF. TerraGear was selected to create the scenery; it was successfully compiled for Linux and used to create scenery using basic public data. JGSF data was not in a compatible format and due to time constraints could not be used to create accurate scenery. Research shows that JGSF data can be used to successfully create FlightGear scenery.

The third requirement of the project involved the creation of an interface for the Virtual Maritime Systems Architecture. Research suggests that the FG_Interface can be reused to send data about other units into FlightGear. Due to time constraints and a limited multiplayer engine this aim was not implemented.

8 Glossary

JGSF The primary military role of JGSF is to acquire, collate and maintain high integrity databases for production and dissemination of Geospatial products and services to static and deployed NZDF HQs and Force Elements. JGSF is also the NZDF central provider of maps and charts, digital databases and feature foundation data and undertakes commercial hydrographic services on behalf of the RNZN. [2]

DTED DTED2 is the basic high resolution elevation data source for all military activities and systems that require landform, slope, elevation, and/or terrain roughness in a digital format. DTED 2 is a uniform gridded matrix of terrain elevation values with post spacing of one arc second (approximately 30 meters). [22]

VPF The Vector Product Format (VPF) is a standard format, structure, and organization for large geographic databases that are based on a georelational data model and are intended for direct use. [23]

Shape File stores nontopological geometry and attribute information for the spatial features in a data set. The geometry for a feature is stored as a shape comprising a set of vector coordinates. Shapefiles handle single features that overlap or that are noncontiguous. Shapefiles can support point, line and area features. [24]

DEM Digital Elevation Model : A model of terrain relief in the form of a MATRIX. Easch element of the DEM is regared as a node of an imaginary grid. The gird is defined by identifying one of its corner (lower left usually), the distance between nodes in both the X and Y directions, the number of nodes in both the X and Y directions and the gird orientation. [25]

SRTM The Shuttle Radar Topography Mission (SRTM) obtained elevation data on a near-global scale to generate the most complete high-resolution digital topographic database of Earth. SRTM consisted of a specially modified radar system that flew onboard the Space Shuttle Endeavour during an 11-day mission in February of 2000. [26]

References

[1] "Defence Technology Agency," September 2004. http://www.nzdf.mil.nz/corporate/hqnzdf.html#Defence

[2] "Joint Geospatial Support Facility," September 2004. http://www.nzdf.mil.nz/jgsf/role.html.

[3] P. Strong, "UAV Uplink Specification." This report is awaiting security vetting and as such is restricted.

[4] P. Strong, "UAV Downlink Specification." This report is awaiting security vetting and as such is restricted.

[5] "Microsoft Flight Simulator 2002 Home Page," September 2004. http://www.microsoft.com/games/pc/fs2002.aspx.

[6] "FlightGear Project," September 2004. http://www.flightgear.org/.

[7] "Microsoft Flight Simulator 2002 Software Development Kits," September 2004. http://www.microsoft.com/games/flightsimulator/fs2002_downloads_sdk.asp.

[8] Microsoft, *Creating Terrain and Land Classification*.

[9] "BGL Tools," September 2004. http://www.scenery.org/design_utilities_a.htm.

[10] "VFR Scenery of England and Wales," September 2004. http://www.horizonsimulation.com/index.html.

[11] "TerraGear Project," September 2004. http://www.terragear.org/.

[12] "Atlas Home Page," September 2004. http://atlas.sourceforge.net/.

[13] "FlightGear User Mailing List," September 2004. http://www.mail-archive.com/flightgear-users@lists.sourceforge.net/maillist.html.

[14] "FlightGear Scenery Designer," September 2004. http://fgsd.sourceforge.net.

[15] ".NET CoadTools," April 2004. http://www.coad.net/noah.

[16] T. J. Krell, "Serial Communications : The .NET Way," September 2004. http://www.codeproject.com/dotnet/DotNetComPorts.asp.

[17] " CCS Microchips," September 2004. http://www.ccsinfo.com/picc.shtml.

[18] " Flight Gear Internal Scenery Coordinate Systems and Representations," September 2004. http://www.flightgear.org/Docs/Scenery/CoordinateSystem/CoordinateSystem.html.

[19] D. Megginson, "FlightGear Scenery Tutorial," September 2004. http://glennm.orcon.net.nz/flightgear/fg-scenery-tutorial2.html.

[20] "VMAP0 Accuracy," September 2004. http://www.mapability.com/index1.html?http&&&

[21] "Geographic Resources Analysis Support System," September 2004. http://grass.itc.it/.

[22] "DTED2 - Digital Terrain Elevation Data Level 2," September 2004. http://gcmd.nasa.gov/records/GCMD_Canada_DND_DTED2.html.

[23] "Vector Product Format Overview," September 2004. http://WWW.NIMA.MIL.

[24] "ShapeFile," September 2004. http://pcsa.hrsa.gov/datamaps/glossaries/gis_glossary.htm.

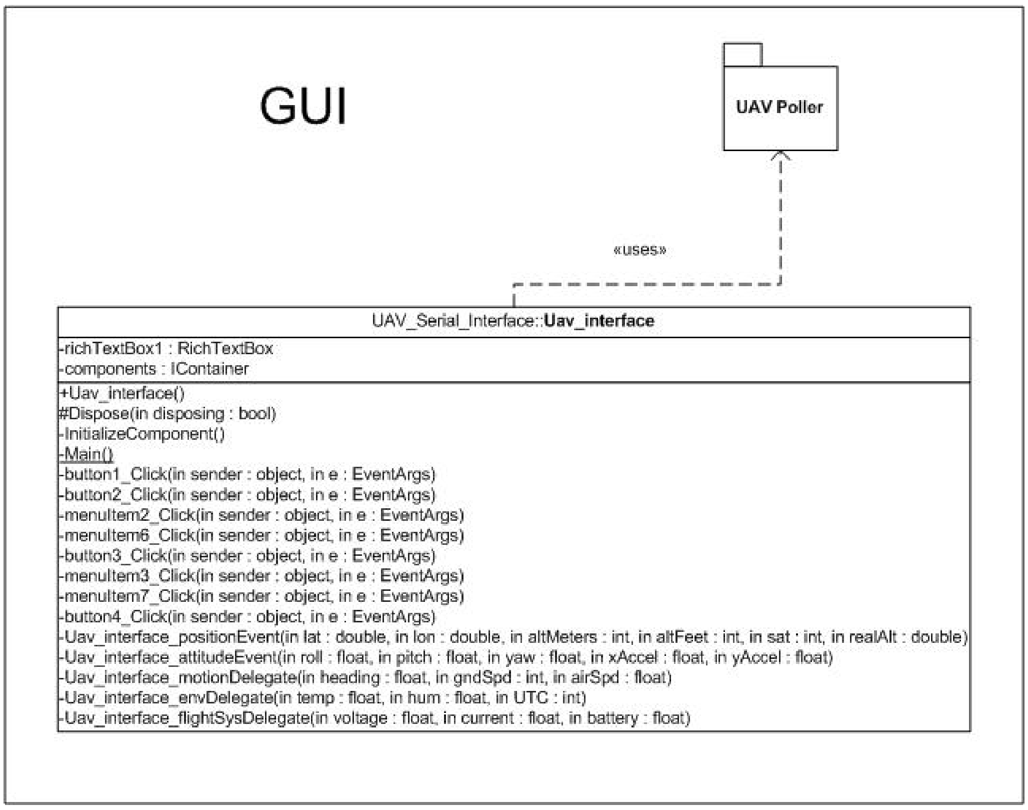
[25] "Digital Elevation Maps, Glossary," September 2004. http://www.geocities.com/capecanaveral/9727/gist.html.

[26] "Shuttle Radar Topography Mission," September 2004. http://www2.jpl.nasa.gov/srtm/.

A Appendices

A.1 UAV Interface

Figure 10: UML diagram of UAV Interface GUI


GUI
UAV Poller
«uses»
UAV_Serial_Interface::Uav_interface
-richTextBox1 : RichTextBox
-components : IContainer
+Uav_interface()
#Dispose(in disposing : bool)
-InitializeComponent()
-Main()
-button1_Click(in sender : object, in e : EventArgs)
-button2_Click(in sender : object, in e : EventArgs)
-menuItem2_Click(in sender : object, in e : EventArgs)
-menuItem6_Click(in sender : object, in e : EventArgs)
-button3_Click(in sender : object, in e : EventArgs)
-menuItem3_Click(in sender : object, in e : EventArgs)
-menuItem7_Click(in sender : object, in e : EventArgs)
-button4_Click(in sender : object, in e : EventArgs)
-Uav_interface_positionEvent(in lat : double, in lon : double, in altMeters : int, in altFeet : int, in sat : int, in realAlt : double)
-Uav_interface_attitudeEvent(in roll : float, in pitch : float, in yaw : float, in xAccel : float, in yAccel : float)
-Uav_interface_motionDelegate(in heading : float, in gndSpd : int, in airSpd : float)
-Uav_interface_envDelegate(in temp : float, in hum : float, in UTC : int)
-Uav_interface_flightSysDelegate(in voltage : float, in current : float, in battery : float)

A.2 UAV Poller

A.3 FG Interface

Figure 11: UML diagram of UAV Poller

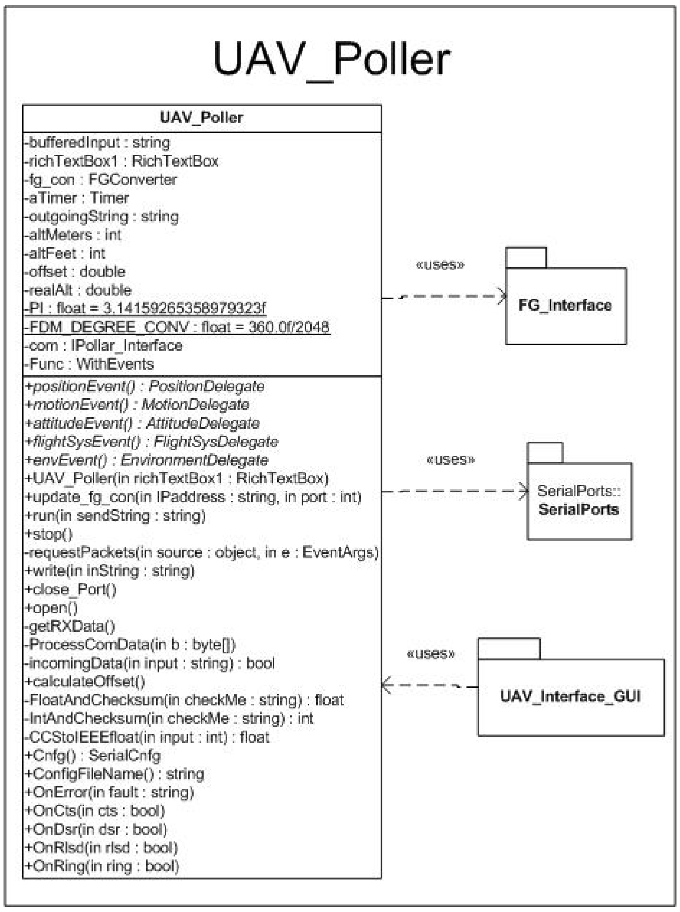
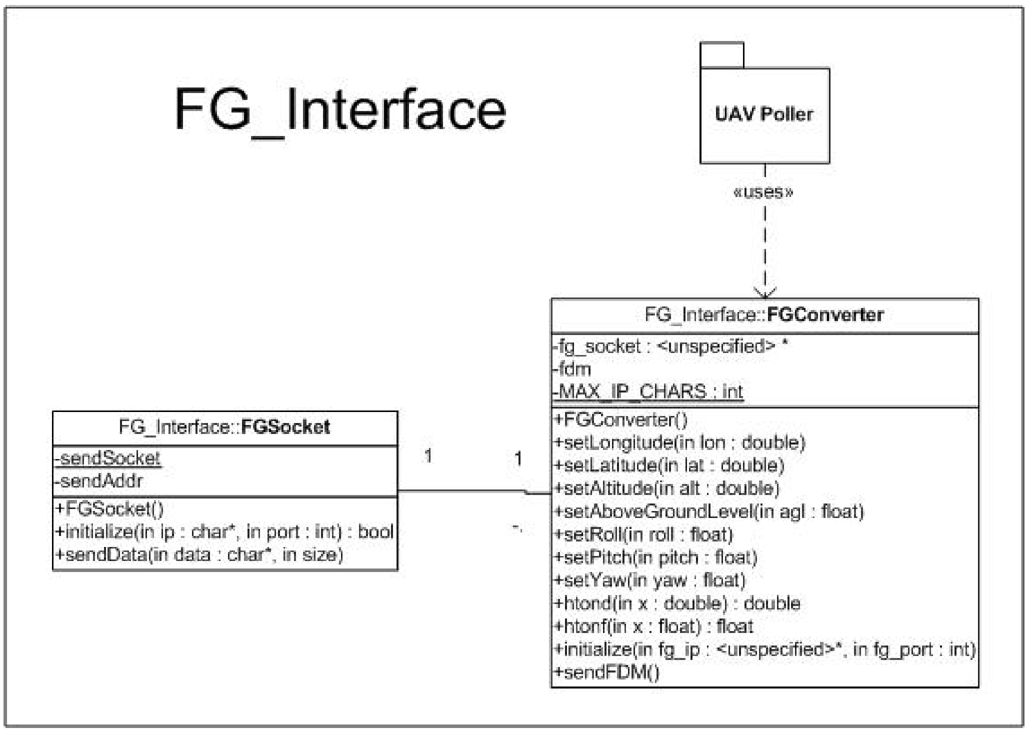

UAV_Poller
UAV_Poller
-bufferedInput : string
-richTextBox1 : RichTextBox
-fg_con : FGConverter
-aTimer : Timer
-outgoingString : string
-altMeters : int
-altFeet : int
-offset : double
-realAlt : double
-PI : float = 3.14159265358979323f
-FDM_DEGREE_CONV : float = 360.0f/2048
-com : IPollar_Interface
-Func : WithEvents
+positionEvent() : PositionDelegate
+motionEvent() : MotionDelegate
+attitudeEvent() : AttitudeDelegate
+flightSysEvent() : FlightSysDelegate
+envEvent() : EnvironmentDelegate
+UAV_Poller(in richTextBox1 : RichTextBox)
+update_fg_con(in IPaddress : string, in port : int)
+run(in sendString : string)
+stop()
-requestPackets(in source : object, in e : EventArgs)
+write(in inString : string)
+close_Port()
+open()
-getRXData()
-ProcessComData(in b : byte[])
-incomingData(in input : string) : bool
+calculateOffset()
-FloatAndChecksum(in checkMe : string) : float
-IntAndChecksum(in checkMe : string) : int
-CCStoIEEEfloat(in input : int) : float
+Cnfg() : SerialCnfg
+ConfigFileName() : string
+OnError(in fault : string)
+OnCts(in cts : bool)
+OnDsr(in dsr : bool)
+OnRlsd(in rlsd : bool)
+OnRing(in ring : bool)
«uses»
FG_Interface
«uses»
SerialPorts::
SerialPorts
«uses»
UAV_Interface_GUI

Figure 12: UML diagram of FG Interface


FG_Interface
UAV Poller
«uses»
FG_Interface::FGConverter
-fg_socket : <unspecified> *
-fdm
-MAX_IP_CHARS : int
+FGConverter()
+setLongitude(in lon : double)
+setLatitude(in lat : double)
+setAltitude(in alt : double)
+setAboveGroundLevel(in agl : float)
+setRoll(in roll : float)
+setPitch(in pitch : float)
+setYaw(in yaw : float)
+htond(in x : double) : double
+htonf(in x : float) : float
+initialize(in fg_ip : <unspecified>*, in fg_port : int)
+sendFDM()
FG_Interface::FGSocket
-sendSocket
-sendAddr
+FGSocket()
+initialize(in ip : char*, in port : int) : bool
+sendData(in data : char*, in size)
1
1