Game Engine Driven, Real-Time Visualisation of Commercial UAV Operations with Varied Data Needs

Lindsey Mackie

BSC(Hons) Computer Game Technology,2020

School of Design and Informatics

Abertay University

Table of Contents

List of Figures

List of Tables

Acknowledgements

Dr David King for all the advice and support to get this project completed, and pushing me to get the writing done early enough to actually be able to finish it in time.

Blair Nichols and Cynthia MacLeod at Cloud3D for all the technical support, hardware and generally dealing with my complete and persistent absence of knowledge on everything UAV.

Chris, thanks for dealing with me while I stressed out and drank dangerous amounts of coffee to get this finished, I couldn't have done it without your support.

Abstract

Context:

Game Engines are being used with increasing frequency beyond the games industry. With powerful real-time rendering and support for a wide range of platforms; game engines are an appealing option for creating highly visual experiences and tools. For the Unmanned Arial Vehicle (UAV) industry, visualisations for remote systems can be extremely useful for understanding system behaviours and maintaining control. Game Engines have potential to drive solutions in this field, but to maximise their usefulness they must support the varied nature of UAV hardware and data requirements in commercial UAV operations.

Aim:

To prototype a networked real-time UAV flight visualisation system using a popular game engine, capable of adaptable parsing of data streams to support varied operational hardware and data streams.

Method:

Implementing a real-time visualisation in Unreal Engine 4.23 (UE4), as a series of components which interact and form the complete solution. Using the plugin system of UE4 to develop the parsing solution, and developing middleware to handle direct communication with UAV hardware and provide the required data stream.

Results:

It was seen that the parsing solution implemented; allowed an adaptable and mostly accurate system to be maintained - there was some accuracy loss seen with floats and doubles which needs to be addressed - where it was adaptable only when the application was not currently running. While the visualisation produced showed promise as a system to support safe operation of UAV systems, although there were limitations on testing this under operational conditions.

Conclusion:

Overall the project served as a demonstration of a potential use of games technology within the UAV industry, although both the parser and the final visualisation both have scope to improve. The parser could be expanded to support a wider range of data types and include data conversions. While the visualisation could benefit from further flexibility to be included in the UI, and still requires field testing to confirm its complete functionality.

Abbreviations, Symbols and Notation

VR	-	Virtual Reality
UAV	-	Unmanned Arial Vehicle
UE4	-	Unreal Engine 4
IP	-	Intellectual Property
FAA	-	Federal Aviation Administration
CAA	-	Civil Aviation Authority
DRL	-	Drone Racing League
MSP	-	MultiWii Serial Protocol
MSPv.2	-	MultiWii Serial Protocol Version 2
UI	-	User Interface

1 Introduction

Recent years have seen considerable expansion of games technology beyond the games industry and into other fields. Such as Architecture, Automotive and Film to name a few industries now making use of games technologies in a variety of ways. This shift in industry has not gone unnoticed by major Game Engine creators, who are now creating and maintaining specialised services and functionalities to support other industries now using their products.

This increase in uptake of Game Engine based solutions, has been in part due to growing recognition of Game Engines' capacity to handle high-fidelity real-time renders. Where presenting high quality visuals is an important but often time-consuming task with previous workflows, Game Engines are becoming an increasingly practical solution, helping to speed up processes. In many cases also improving end results and introducing possibilities for interactivity that were not practical or possible before.

A great example of the expansion of Game Engine uses is the automotive industry; with major manufacturers finding a variety of ways Game Engine driven solutions can benefit their businesses. BMW and Mini have found that by using Unreal Engine (UE4), they have been able to create a mixed reality design tool; combining custom hardware with Virtual Reality (VR) *(Sloan, 2018)*. This has provided designers with the ability to perform more experimentations, to make key decisions earlier and increase the time available for iteration and refinement of designs for deadlines and finalisation.

Another factor driving this shift towards using Game Engine based solutions, is the desire for companies to maintain control over their Intellectual Property (IP). Using game engines enables companies to produce applications inhouse - without reliance on external suppliers for delivery and maintenance - allowing complete control over their IP. This control of IP is particularly important for growing and highly competitive industries where inhouse development is a key priority, for example the Unmanned Aerial Vehicle (UAV) industry.

Like Game Engines, the UAV industry has seen a surge, in both recreation and commercial sectors, with hobbyists and commercial operations all developing and experimenting with potential new uses and functionalities. Recent reports and forecasts on the industry in the

USA, based on assessment by the Federal Aviation Administration (FAA), highlighted key trends in both Model (hobbyists) and Non-Model/Commercial sectors. With the model sector having seen considerable growth but expected to reach a peak in around 5 years. While the non-model sector's growth has exceeded last year's prediction of a 44% growth in registered craft by around 80% more than expected; and growth is predicted to not only continue but accelerate in the near future (Federal Aviation Administration, 2019).

There is already some overlay with UAV use and the gaming industry, namely with the increasing popularity of Drone Racing, with the Drone Racing League (DRL) world championship having racers who have gained their place in the competition through game-based platforms, specifically the DRL simulator (Swatch, 2018). This existing overlap is just one area where games technology is beneficial to UAVs.

This project will look at how a popular Game Engine can be used to drive visualisations of commercial UAV operations via network-based communication with existing systems. Looking specifically at those for which the hardware and data needs can vary based on the operation being carried out. With the aim to create a flexible system, based around a local company's manufacturing and practical uses of UAV systems, defining common requirements and existing limitations which could be addressed through the use of game technologies, such as game engines.

Research Question:

How can Game Engines be used to drive real-time visualisations for commercial UAV systems with varied data stream requirements?

To achieve this the following key objectives have been identified:

- Implement an extension to an appropriate game engine to automatically handle network setup and real-time data parsing
- Identify and implement an appropriate solution for integrating UAV data into engine
- Analyse and evaluate the solutions performance in relation to the handling of varied data stream formats, and its maintenance of a reasonable degree of accuracy for monitoring remote UAVs
- Evaluate the effectiveness of the solution for supporting safe operation of UAVs carrying out flights beyond line of sight communication
- Identify limitations of the solution and suggest areas for improvement

The following literature review will discuss existing trends in the use of games technology outside of the games industry, flexible approaches to networking and data parsing and the current state of the UAV industry. With later chapters discussing the methodology employed for designing and implementing end solution; followed by a discuss of the results achieved when testing the produced artefacts.

1 Literature Review

1.1 Use of Game Technology Outside of the Games Industry

A main influence on this project has been the expansion of Game Technology of various forms beyond the Games Industry; becoming increasingly popular options for solutions in a growing number of industries. With hardware platforms such as the HTC Vive being accessible - even for small companies – for use when developing and presenting visual experiences using VR. This type of use can be seen in architecture firms who have used VR to provide walkthroughs of spaces; and automotive companies allowing customers and engineers to 'see' and walk around vehicles to assess design choices. Alongside hardware uptake, Game Engines such as UE4 and Unity are also gaining popularity as development platforms outside of the Games Industry. Providing powerful real-time rendering capabilities and opportunities to create highly interactive and visual applications for a wide variety of purposes. For example the automotive industry, using the likes of UE4, to drive online car configurators; creating high quality and often photorealistic renders from customer specifications, in real-time; as demonstrated by Epic Games and McLaren Automotive at GDC2016 (Crecente, 2016), and even to create Mixed Reality tools to support the design process (Sloan, 2018).

When looking at how current game technology can be used to support operations with UAVs a few areas had to be researched to determine an appropriate approach to handling the issues discussed with Cloud3D; a local company specialising in bespoke technology solutions including UAVs. Mainly regarding easily viewing information received from UAV systems in flight; with particular difficulties concerning video feeds. Video feeds experience the most degradation over distance and are prone to being lost entirely, leaving operators with little information regarding the UAVs behaviour and orientation. This is the main problem this project aims to address, by providing a visualisation of critical data to allow operators to more easily understand information and reduce the cognitive load involved in doing so, by using graphic displays over constantly changing numerical values. Due to the variety of UAV systems that Cloud3D work with, the solution will need to work with varied data streams and conduct real-time parsing, without needing code changes to accommodate different data streams. For this reason, important areas for research included

adaptive parsing, defining network characteristics for flexible uses, and existing UAV technologies.

1.2 Approach to Specification Based Adaptability

In relation to this project, the main need for an adaptable solution comes from a requirement to be able to handle varied data needs and availability, e.g. UAVs with different purposes will have variations in hardware and the structure of data produced. So, an important area for research was designs and approaches to handling similar problems; namely how to create a flexible solution by having functionalities derived from user specifications.

Looking first at research into creating new network protocols. One method trialled using a declarative language, producing a solution to allow developers to specify requirements and then have the protocol automatically compiled and executed (Loo et al., 2009). In the context of this project, a declarative language is out of scope due to the existing language support offered within the chosen Game Engine. However, the idea that by automatically handling implementation, developers are better able to focus on high level design, rather than the specifics of implementing requirements; helps to provide a context as to why this project can be of practical use. Another approach to protocol creation was using composite protocols (Minden at al., 2002); combining smaller reusable components to provide specific functionalities as required by a given specification in order to create new protocols. Both approaches although different in implementation; looked to simplify the development of protocols – a historically complicated area of development- through the use of specifications and separating the developer from the actual implementation when creating a variety of new protocols.

A further area of interest was approached to parsing data streams; specifically working with heterogenous data streams – where the contents of the stream contains multiple datatypes. The grammar-based method was of particular interest; making use of the high degree of expressiveness offered by a grammar-based system to automatically produce parsers while maintaining generality and technology independence (Campanile et al.,2007). The solution created was applied to and tested using a real-time billing processing system; results stated significant performance improvements, compared to the previous setup. However, when

comparing the streamed implementation using the automatically produced parsers and the original buffer system they only worked with specific, processed log files. Although the results discussed show practical performance improvement, little mention of specific testing for adaptability of the system is made. This approach was of particular interest as the problem is extremely similar to that being considered in this project. However, in the approach and the specific application discussed there is not the same emphasis on the need for the system to be real-time. A billing system can easily cope with delays if necessary, but when handling visualisation of a physical system unexpected delays could be extremely detrimental to the applications basic functionality. This difference in real-time sensitivity will need to be carefully managed if using methods based on this approach.

1.3 Middleware Solutions

Following discussions with a local UAV industry specialist – Cloud3D - regarding the variety of technologies available and current systems in use; an appropriate method of linking existing UAV technology and the game engine had to be selected. Due to the variety of software and hardware available within the UAV industry, a specific solution created within the game engine would greatly limit the flexibility of the resulting application and require significant work to extend. For this reason, a middleware solution to collect data from UAV hardware and send it via a network connection was selected as an appropriate method for the scope of this project as it avoided the plugin or visualisation application becoming limited to a particular UAV platform.

Looking at work done to extend an existing general-purpose networked middleware solution to support real-time systems, highlighted specific approaches which can be taken to improve performance to support the requirements of time sensitive systems. Working with Etherware as a basis and making various extensions to improve performance with real-time control systems (Kyoung-Dae Kim and Kumar, 2013). Specifically introducing an idea of "Quality of Service" to prioritise and schedule messages appropriately and extending existing scheduling methods to allow concurrent execution of multiple components.

While the middleware solution produced during this project will be as simplistic as possible as it is being developed to support development and testing of a prototyped UAV visualisation, so is not the focus of this project. The ideas discussed in the above work

provide important guidance on approaches to maintain necessary performance and are also applicable at least in part to improving performance on the parsing system which will be receiving data for the visualisation.

1.4 UAV Technology

Based on UAV technology decisions made early on; following research and discussion with industry specialists on availability of software, hardware and support for different options. It was decided that a MultiWii flight controller-based system would be an appropriate UAV platform to use for development. This being mainly due to the ease of access to both hardware and software, and the variety of UAV configurations supported by MultiWii. Additionally, with MultiWii being a popular choice for many custom UAV/drone builders it was possible to locate a variety of forums and free to use code modules to work with the MultiWii Serial Protocol (MSP). The main downside to working with MultiWii is the reliance on publicly editable forums and wikis for technical information as there is no official fixed documentation for this platform.

It was also important to consult the Civil Aviation Authority (CAA) website to determine what limitations currently exist across the UAV industry, and critically which if any apply to any part of this project. This has to remain an ongoing process for the duration of the project; due to developing legal limitations and policy regarding use of UAV technology in regard to both private and commercial use. Such as the introduction - in November 2019 – of the requirement to register all drones and pass theory tests before being allowed to conduct flights in the UK (CAA,2019); this applying across the board to all UAV systems including those marketed as toys. This is a critical change in the law during the course of the project and will have an impact on the testing options which are easily available when considering how live testing can be conducted.

Based on the research conducted the following chapter will detail an approach to a flexible solution which can be adapted to the needs of specific data streams through the use of formatting files to provide key data to drive component setup and functionality.

2 Methodology

In order to produce the visualisation application with the ability to support a variety of UAV hardware; implementation has been divided into 3 key areas for development - a plug-in for UE4 providing adaptable parsing, middleware for interfacing with a variety of UAV hardware, and the UAV visualisation application itself. This was done to separate specific functionality; allowing the development process to be better managed, and to improve the overall flexibility of the resulting solution by maintaining the hardware independence of the visualisation application as far a reasonably possible in the scope of this project. This section will discuss the approach taken in the development of these key components, and detail how specific critical functionality has been implemented in each development area.

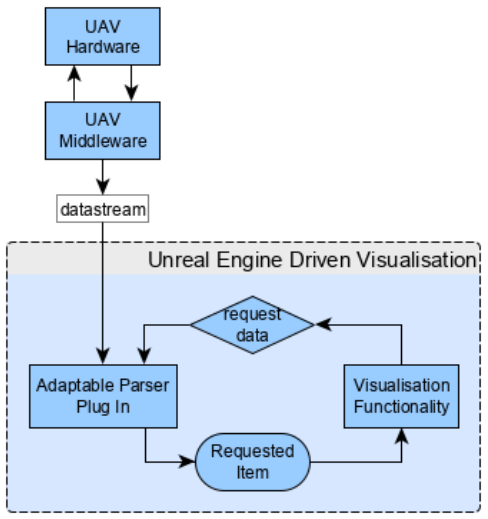

Figure 2-1 General Interaction of Component Systems

2.1 Early Technology Decisions

Due to the nature of the project, there was a large selection of possible technology choices available for various aspects of development. So, it was important to narrow down specific technology to a small selection of appropriate options for this project; while not unnecessarily ruling out future expansion of supported technologies such as different UAV flight controllers and hardware.

Unreal Engine 4.23 was selected as the game engine to be used to drive the visualisation. Using UE4 allowed for the Adaptable Parser to be developed as a plugin which can be easily integrated into multiple projects allowing the parser to be used for other applications in future. Also, with the introduction of Datasmith as part of UE4 and Unreal Studio importing CAD data into projects is a simple process allowing technical models to be used in the visualisation. Additionally, UE4 provides free access to source code and has variety of features designed for use with non-games applications; which may benefit this project.

For UAV related technology all early decisions were made with guidance from Cloud3D, based on how they construct and operate different UAV systems; as these systems are the target platform being used in development and testing due to the variety of custom designs they build providing a suitable test set to assess the flexibility of the solution produced. A key technology decision which had to be made at this stage was which communication protocols to build supporting middleware for to communicate with UAV flight controllers; with MultiWii (including its descendants such as CleanFlight and BetaFlight) and MavLink being selected due to their popularity within the custom UAV industry.

Finally, appropriate technical options for the development of middleware which could be used to support the visualisation had to be selected. The main consideration being the selection of a development language which would be suitable for the purposes of this project; Python has been selected as the development language which was best suited for developing the required middleware solutions. Due to the wide range of free to use additional modules in public circulation offering support for specific hardware, and Python's wide used in a variety of technical fields (including popularity within the UAV industry).

2.2 Adaptable Parsing Plug-In

When looking at an 'adaptable' parser it is first important to define what adaptable means in the context of this solution. In this case, the parser is adaptable when the application is not in use, through the use of a series of files which provide key data used to drive the parsers setup and functionality; the parsing solution as it stands cannot be adjusted at runtime once its setup has been completed.

In order to provide a flexible solution, the message parsing requirement for the UAV visualisation was developed as a Plug-in for UE4, developing in this way allowed for a separation from the visualisation aspect of the project and means that the parsing solution can be easily reused for other applications unrelated to UAVs if needed in the future.

Setting Up Network Components

For developing the parser as a networked solution, the first stage of development was to determine a method to setup sockets and ports through user specification. This is done through the use of a JSON file which allows a user to create a network setup file which holds all required data – this does make some assumption about the user having a basic understanding of network terms such as the IP address, port numbers and how to find this information on their system. As it stands for correct setup of the network component the user needs to provide the following details in the json file:

- The application IP
- The application Port
- The data source IP
- The data source Port

The network component itself has been created using the UE4 built in socket library. While libraries such as Winsock are usable within UE4, using the UE4 specific implementation of sockets allows a level of abstraction from underlying socket systems of different platforms. Allowing the developed solution to have a degree of platform independence without introducing additional work and possible errors into the development process.

The network component is initialised after the required JSON file has been read and the required data has been identified – should this reading stage fail the socket will be setup using default values and this read failure will be registered in the application log. The application will attempt to bind the port to the IP and port number detailed in the file, again should this fail the reason for failure will be reported in the log.

For protocol use, the system has been setup to use UDP; due to the frequent nature of updates required to support real-time use. While UDP can have issues with lost messages and incorrect delivery order these issues can be addressed relatively easily. Lost messages in a real-time system will quickly become irrelevant so provided good network conditions where message loss is not an overly regular occurrence this would not be a major issue for the type of real-time application being produced.

As for maintaining messages in the correct order- as this is a more critical requirement for a real-time visualisation - support for using either a timestamp or a message id number at the start of the message has been included and can be used by declaring them in the JSON file in the same way as any other data item present in the expected message which should be received. Although only UDP support has been implemented, there is scope to extend functionality to support TCP in future to account for applications where guaranteed delivery of all messages in order would be a key requirement.

In addition to information used for the direct setup of sockets and ports; the application requires the user to provide information on the expected data source (the IP address and Port) in order to introduce a basic check for message integrity. The system implemented will only attempt to interpret a message which has been received from the expected sender. This has been introduced to avoid accidental message received creating errors in the data being used to drive the visualisation, or in extreme circumstances causing fatal errors in the application. This does not however provide any security against deliberate introduction of false data into the data stream as the IP address and Port associated with a message could be relatively easily faked. However, in the context of this project – being only a visualisation and offering no control over hardware - this risk is minimal and so determining other security methods to address it was not a high priority during development.

2.2.1 Message Parsing

For the data parsing aspect of the plug-in there were a number of challenges when considering how to parse a heterogeneous data stream where the number of items, their type and order cannot be strictly defined within the application itself. The first step to being to address the needs of a flexible parsing system was to define a method to describe the expected data stream. Again, making use of a JSON file to store this description of the message format which the user is expecting to receive from the data source; allowing the user to detail the data being sent in the order it is present in the message. Providing a name for each data item - which could be used to identify each item for later for use within application - and the data type of the item; in order for the parser to be able to interpret the data stream and make data received usable at runtime.

Once a description for the expected data could be defined and processed correctly by the plugin, the next issue was how to store the data received. Specifically storing data where key factors such as number of items and the associated type are not known until runtime. Data items would have to be stored in a single structure so that all items could be accessed in the same way to maintain a consistent method for accessing and updating individual items. This was achieved by designing a structure which contained each object's name as define in the descriptor file, an enumerated type to represent the data type needed for this item, and finally - to store the actual data - a pointer of type 'void'. By using this type of pointer, it is possible to reference an item of any datatype; using this to store a pointer to a container of the correct type for that particular data item, creating the container itself at runtime based on the datatype needed for that item. This structure is used to collect all data items together for storage using UE4s built in 'TArray' which can be used like a vector e.g. increasing and decreasing in size without redeclaring, but is also usable within the blueprint system in UE4. This store of data can then be searched for specific items using the stored name associated with the needed data item.

This storage system was used to account for the undefined number of data items which may be used. In the context of this project the 'TArray' was an appropriate structure as the number of data items which may be needed is relatively low, but for large collections of data it could be more efficient to use a map structure where searching based on the value of a key is built in and provides better performance over a large data collection.

2.3 MultiWii Connecting Middleware

In order to support early development, a system for using realistic data in a desktop setup was needed. This requirement was met by building a basic flight controller unit using an Arduino Nano and an IMU sensor (see appendix 1) using the MultiWii Serial Protocol (MSP).

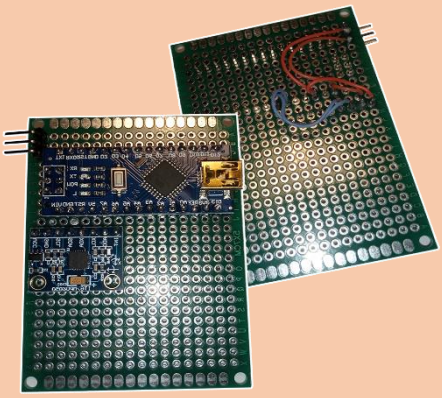

Figure 2-2 Arduino Nano flight controller

In order to maintain platform independence, it was determined that an appropriate method to provide access to the real-time data from this hardware setup was through the development of a small middleware application. This was done using Python to develop a command driven application, due to the existing MultiWii support and modules available publicly online (for additional required libraries installed see appendix 2). Connection to the hardware was done using serial bus connections, which could be easily supported including checks and display on only valid available serial bus options to avoid guesswork when selecting the required COM port for connecting to the flight controller.

The middleware produced works as a networked solution and like the parsing plug in previous discuss it handles socket setup at runtime but using a different method. Due to Python being an interpreted language and the middleware being a console-based solution, it is relatively simple to wait and collect the required IP and port information through requesting user input through the console and using this to set up ports and sending addresses as needed. The middleware solution produced is has support to use additional sensors other than those included on the specific flight controller being used, this is done through a list of possible options for sensor data which can be requested through the MultiWii Serial Protocol. This was to allow this piece of middleware to support various combinations of commonly available sensors on a MultiWii based flight controller but without requesting data unnecessarily e.g. if a sensor is not present only a default value would be returned on every data request.

2.4 Visualisation Application

The visualisation application has been built to act only as a digital reflection of the UAV in use; with the purpose of improving the usability of the existing data already produced by UAV systems. Looking at how to represent data in visual formats to provide ease of understanding and reduce the mental load of interpreting the data to allow operators to better prioritise and understand the data being received.

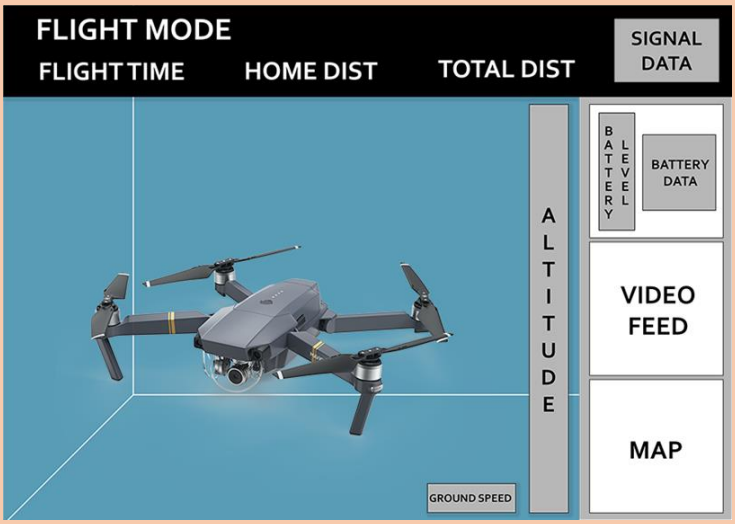

Figure 2-3 layout mock-up of visualization application (further detail see Appendix 3)

2.4.1 Movement Replication

The first step for creating a useful visualisation of UAV in flight is for the digital model to replicate the physical movement of the hardware in use. For orientation of the system, the needed data can be easily sourced from the IMU... which provides gyroscope data as pitch, roll and heading/yaw of the device. The challenge when replicating the movement comes from variations in the heading/yaw value compared to the visible front of a device which will be different between UAVs and even on the same hardware after calibration changes. In order to account for this variation orientation is taken using changes in the heading value compared to the first instance of orientation data which is received, positioning the digital model on the screen with the assumption that the user is beginning with the physical device in front of them facing forward -as this a common and safe practise when operating UAVs in most circumstances.

In addition to mimicking orientation in real-time, replication of the UAVs translational movement is needed for communicating position and movement information to operators. For this application, translational movement exists where there is GPS data which provides the needed values to calculate device positioning within the digital space. This conversion of GPS data into a local Cartesian coordinate system is a key part of maintaining a meaningful representation of the UAVs movement, and requires a series of calculations. GPS data must first be converted into a Geocentric format (also called Earth Centred, Earth Fixed -ECEF), this allows Latitude, Longitude and Altitude to be represented in a Cartesian format where the centre of the earth is the origin. This is to large scale for this application to make use of; so this much then be converted into a smaller region of local coordinates, using the first ECEF coordinate received as the origin of the local system where all following locations will be calculated relative to this first point, for the purposes for this application the North East Down (NED) coordinate system has been used (see figure 3.5).

Converting GPS to ECEF

$A = radius\ at\ the\ Equator = 6378137$

$f = flattening = 1/298.257224$

$e = eccentricity = \sqrt{(2f * f^2)}$

$C = 1/\sqrt{(\cos^2(Lat)+(1-f)^2 * \sin^2(Lat))}$

$S = (1-f)^2 * C$

ECEF:

$$X = (aC+h)*\cos(Lat)*\cos(Long)$$

$$Y = (aC+h)*\cos(Lat)*\sin(Long)$$

$$Z = (aC+S)*\sin(Lat)$$

ECEF to NED

$NED = R^T(P_{ECEF} - P_{ref})$

$$R = \begin{bmatrix} -\sin(\phi)\cos(\lambda) & -\sin(\lambda) & -\cos(\phi)\cos(\lambda) \\ -\sin(\phi)\sin(\lambda) & \cos(\lambda) & -\cos(\phi)\sin(\lambda) \\ \cos(\phi) & 0 & -\sin(\phi) \end{bmatrix}$$

$\lambda = Longitude$

$\phi = latitude$

Figure 2-4 GPS to NED conversion

2.4.2 GPS map

An important aspect for a useful flight visualisation, is the ability to monitor the world location of the UAV which can be important in case of accidents which may cause equipment to become inoperable. For this purpose, it was important to include map functionality to reflect the location received from the UAV as GPS coordinates, as part of the User Interface.

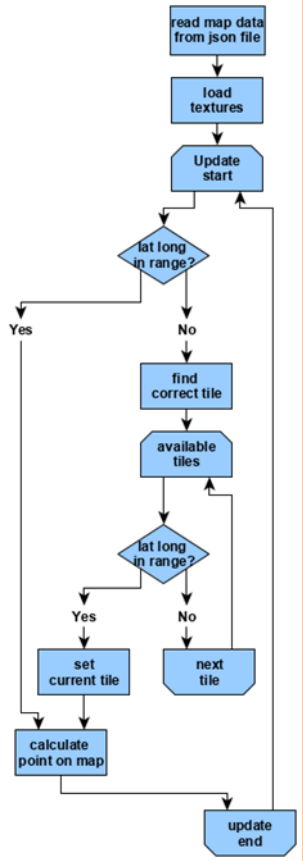

Figure 2-5 2D map functionality

The mapping component of the user interface use a selection of prebuilt tiles which are image files loaded when needed during runtime; using an image loading node from the 'Victory Plug-in'(Rama) as Texture2D types within the application. Tiles loading information is provided by a JSON file which is stored alongside the image files; this JSON file details key information about each tile including:

- Image file name
- Minimum Latitude and Longitude covered
- Maximum Latitude and Longitude covered

Tiles were created using a series of static images covering the desired area, extracted from the Open Street Map project. In this instance this has been done manually, but an automated solution could be developed to handle tile creation as a supporting tool for the visualisation applications long term usability and practicality. Storing tiles locally was determined to be a more practical solution than remote storage or an internet-based system due to the context of this project; UAV systems are often operated on sites where data connectivity and suitable speeds can be common issues.

For tracking the location of the UAV against the map, the appropriate tile is selected based on a comparison between the current Latitude and Longitude, and the maximum and minimum bounds covered by each tile (see figure 3.6) until one with the require location in its ranges identified or no tile is able to cover the needed coordinates (this is registered in

the log file should this occur). The known location is then used to create a positioning value for the marker on the map.

$$MarkerLoc = \frac{MaxLatLong - currentLatLong}{MaxLatLong - MinLatLong} * MaxDisplayPos;$$

Figure 2-6 2D map marker position

2.4.3 Speed Display

Using a digital format to display the speed data; while extremely simple for creating the asset and updating it with new data as it is received. In order to make this element as useful as possible; additional consideration is needed for how display data is updated. The raw data being received provided floating point accuracy; however, this level of accuracy can be excessive for a real-time display and can be detrimental to the overall usability due to constantly changing values unnecessarily drawing the users attention. For this reason, using only integers in the display is a more suitable design choice; but if using standard rounding methods this fluctuation in display values is still present. To reduce this visible fluctuation a rounding method using hysteresis has been employed (see figure 3.7), widening the range of values which will trigger rounding direction changes taking account of the general trend of the previous data received.

```
1.0 Determine Rounding method
        1.0 Get decimal component of Speed
        1.1 If decimal component ≥ 0.7
                1.1.1   ROUND UP
        1.2 Else if decimal component ≤ 0.3
                1.2.1   ROUND DOWN
        1.3 Else
                1.3.1   Leave rounding method unchanged
2.0 Determine display value
        2.0 If ROUND UP
                2.0.1   Display int = int Component of speed +1;
        2.1 Else
                2.1.1   Display int = int Component of speed
```

Figure 2-7 rounding with hysteresis

2.4.4 Measuring Distances in Flight

When operating a UAV there are 2 main distances which are important and typically monitored by the user:

- Distance to Home
- Total Ground distance covered

The distance to home measures the straight-line distance between the current location and the starting location, so that operators can ensure they have enough battery power to safely return. While the total ground distance covered, measures the total distance flown by the UAV including areas where the operator may have backtracked over multiple times.

Both of these distances are calculated using a spherical model of the earth (this is not entirely accurate due to the actual shape of the earth) and the haversine formula (see figure 3.8) which provides a reasonable degree of accuracy for the purposes of this application.

$$a = \sin^2(\Delta latitude/2) + \cos(lat1) * \cos(lat2) * \sin^2(\Delta longitude/2)$$

$$c = 2 * atan2(\sqrt{a}, \sqrt{(1-a)})$$

$$d = R * c$$

$$R = Radius\ of\ the\ Earth$$

Figure 2-8 haversine formula (GIS MAP INFO, 2015),

Planned Testing

2.4.5 Adaptable Parser Testing

In order to assess the functionality of the adaptable parser a series of tests have been planned to address key requirements for the plug-in. An important area for testing is the quality of the results produced by the parser, in terms of the accuracy and the predictability of production of accurate data. This will be assessed by logging data of core data types on both the sender and the results produced by the parser for comparison. This testing will be conducted over a variety of network conditions to assess the parsers performance under common network conditions such as the local machine, local network and between setups using different hardware e.g. processors from different manufacturers.

2.4.6 Visualisation Testing

The main testing planned for the visualisation was to conduct field testing using a custom-built drone with a variety of safety features including stabilisation and an autopiloted return to home functionality. The presence of these additional safety features on the planned hardware was useful for planning field testing as using these features would allow more attention to be given to the function of the visualisation and checking live data values while maintaining safe operational conditions.

However due to current events which have prevented field testing of the visualisation under operational conditions from being able to take place; testing has to be conducted by using a collection of dummy data combined with desktop safe UAV hardware, to replicate what would be expected from a UAV in flight. With the assumption of functionality working as expected based on results of how elements of the visualisation respond to extreme and exceptional data being provided in this controlled desk-based environment. In addition to testing the predictability, and functionality of individual elements behaviour, the visualisation will be assessed on its overall design, including the ease of locating specific data and comparison to existing UAV monitoring systems.

The following chapter will discuss these planned testing methods carried out in more detail, and the results achieved.

3 Results and Discussion

This section provides further detail on planned testing previously discussed; including the testing methods used, any adjustments to the planned methods which had to be made to account for various restrictions which were in place and the results which were seen. Further to discussion of the results seen; additional testing, implementation changes or future work which would still be of benefit to this project will also be covered.

Due to the nature of the solution produced, testing looked at two of the main components produced separately, the Adaptable Parsing Plugin, and the visualisation, before making an assessment on the function of the complete solution produced.

3.1 Measuring the Accuracy of Conversion Within the Parser

When testing the parser, the main requirement which had to be assessed was that the parser should maintain accurate conversions of data from the bytes received into usable items of the required types. To establish if this has been achieved the parser was tested under a variety of conditions, including different network setups and using different hardware of various ages from different manufacturers where possible. This was done to better assess the accuracy and stability of the parser as a networked solution across systems with commonly seen differences. Some of the specifics relating to the hardware used has been detailed in table 1.

Test Setup		Component	Type
Local Machine	Sender & Receiver	CPU	Ryzen 7 3800X 8-Core (3.9Ghz)
		RAM	32Gb
		GPU	GeForce RTX 2070 Super
Local Network.1	Sender	CPU	Ryzen 7 3700X 8-Core (3.6Ghz)
		RAM	32gb
		GPU	GeForce GTX 980 Ti
Local Network.2	Sender	CPU	Intel Core i7-5500U 2-Core (2.4Ghz)
		RAM	16Gb
		GPU	GeForce 920M

Table 1 Hardware used for parser testing

Each setup was used to run the same collection of tests, with a Python script generating random data for each datatype being used, which was then packed into single collection of bytes for sending to the parser as a UDP. It was important to ensure testing produced sufficient data samples in order to provide enough data to be able to properly assess the accuracy achieved by the parser, the number of samples produced when testing a single hardware setup has been listed here:

- Boolean :: 12000 samples
- Char :: 8000 samples
- Short :: 12000 samples
- Integer :: 12000 samples
- Float :: 12000 samples
- Double :: 12000 samples

To assess the accuracy obtained when converting from the byte array received by the parser, back into data items of the required datatypes; both the sending application and the parser maintained a log of their data to be used for the comparison. This comparison found that across all tests and setups, no accuracy was lost on any of received data for Boolean, Char, Short or Integer values. However, across all tests there was some loss of accuracy seen in regard to the Float and Double values. This accuracy loss remained stable across all test sets, and both datatypes saw extremely similar degrees of accuracy being lost. While this loss of accuracy is not ideal it is predictable and stable so can be accounted for during application design and implementation by considering the predicted error margins this accuracy loss creates. The specific results seen when looking at the data for Floats and Doubles can be seen in Table 2 and figure 4.1. With the accuracy loss only being seen in these two data types, and considering the consistency of the loss observed, it is possible that the cause of this is coming from the logging stage rather than the parsers byte conversion. As the logging stage of the parser is converting the float and double values into a text representation, which can introduce rounding errors; which would account for the consistency seen in the results.

		Mean	Standard Dev.	Min	Max	Q1	Median	Q3
Local Machine	**Float**	2.49143E-07	1.45846E-07	0	5E-07	1.25E-07	2.5E-07	3.75E-07
	Double	2.50836E-07	1.44753E-07	0	5E-07	1.27575E-07	2.50465E-07	3.76662E-07
Local Network.1	**Float**	2.49931E-07	1.44131E-07	0	5E-07	1.25E-07	2.5E-07	3.75E-07
	Double	2.4826E-07	1.44548E-07	1.89175E-10	4.9995E-07	1.23384E-07	2.46736E-07	3.74213E-07
Local Network.2 (laptop based)	**Float**	2.51046E-07	1.45026E-07	0	5E-07	1.25E-07	2.5E-07	3.75E-07
	Double	2.52579E-07	1.4536E-07	4.00178E-11	4.99949E-07	1.2596E-07	2.54085E-07	3.79227E-07

Table 2 conversion accuracy results – floats and doubles

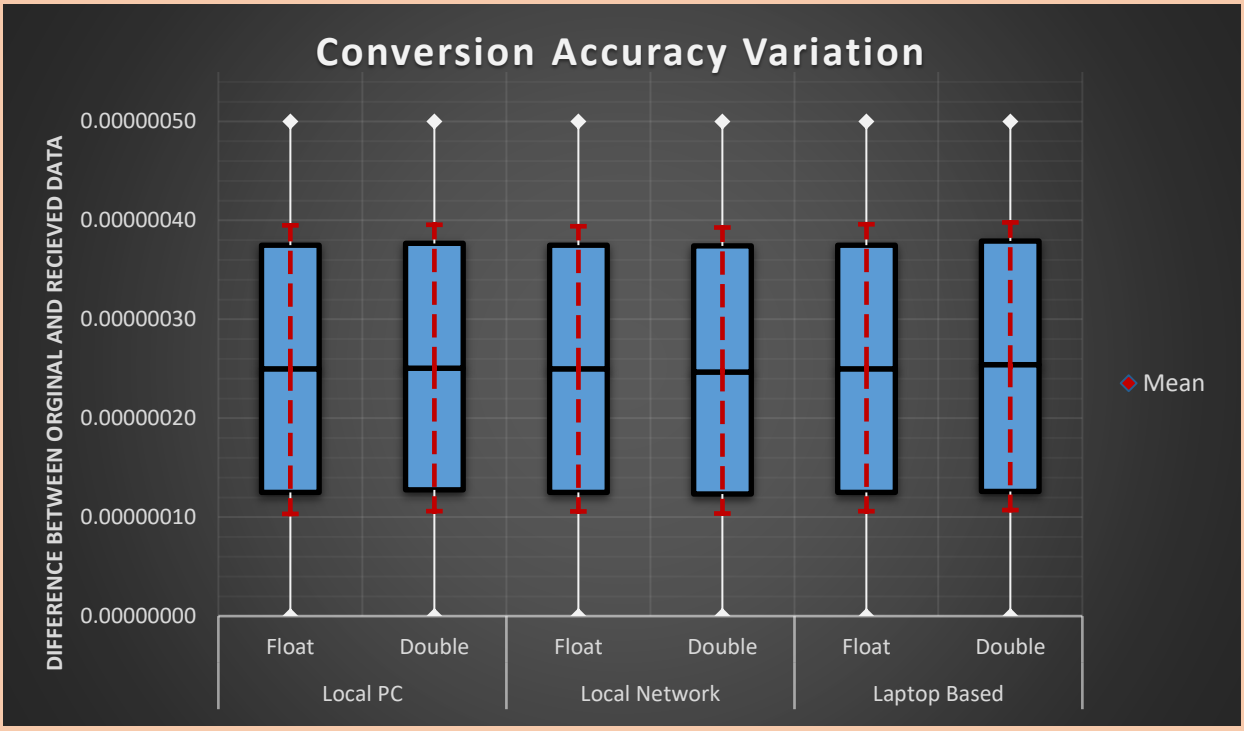

Figure 3-1 Conversion Accuracy Charted Results

3.2 Visualisation Testing

For testing the visualisation application – the final layout can be seen in figure 4.2 - all testing had to take place in a desktop environment due to external restrictions. This has created some limitations on what aspects of the visualisation solution could be thoroughly tested; as operational conditions could not be completely replicated within the testing environment in use. The main areas of the visualisation application which were considered during testing were the overall UI design. The functionality of specific UI elements and the complete application using live data from a variety of hardware options.

Due to the limitations imposed by conducting all testing in a desktop environment there were particular data items which could not be produced in real-time from the target hardware; specifically, GPS and connection data. While UAV setups with the required hardware and protocols were available, in the test environment being used this data would not be stable due to being indoors preventing a proper satellite lock, nor would it have a wide enough variation to support the degree of testing needed for specific functionality. In order to account for these limitations, a collection of test data was compiled to provide the necessary variation for functionality dependant on the data affected to be properly tested.

Looking first at specific UI elements of the visualisation, elements were tested to ensure that the functionality, calculations and behaviour remained correct and predicable even when dealing with exceptional data and circumstances.

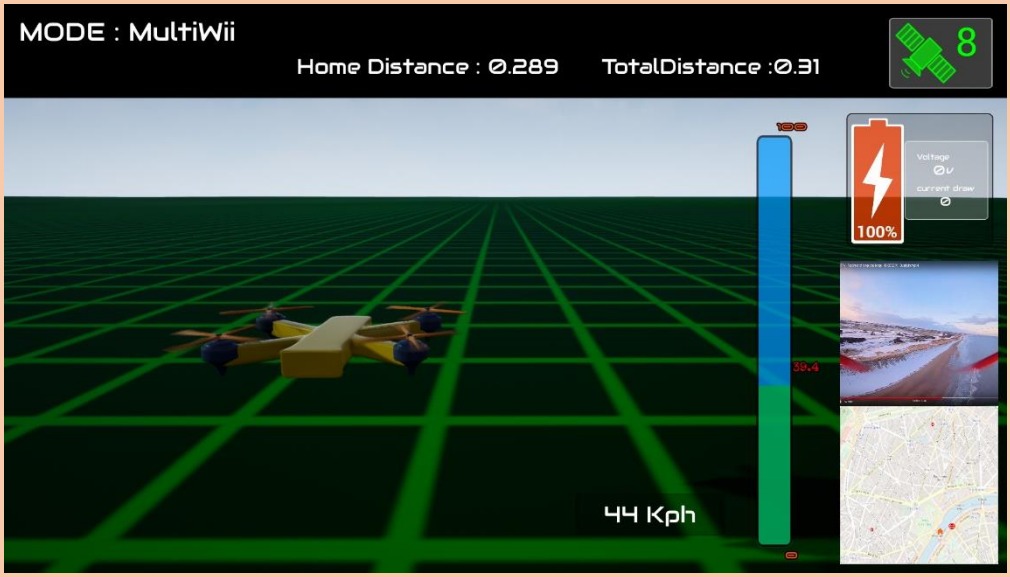

Figure 3-2 visualisation in use

3.2.1 GPS Driven Functionality

There were 3 key UI elements which made use of GPS data to drive functionality, the 2D map, flight distance and speed monitoring functionalities. These elements had to make use of collected test data rather than real-time UAV data. The test data provided a number of routes using real GPS locations to allow for the distance, speed and mapping calculations produced by the application to be compared with known values collect from Strava - a popular mobile application used to log activities such as running using GPS tracking. Data was collected from various activity logs created using the mobile Strava app, by using data exported from this platform it was relatively easy to perform the needed comparisons to confirm functionalities were operating as expected. It also allowed test data from a wider variety of locations than would have been possible using only data from UAV hardware, e.g. data covering London, Paris and Loch Ness was used during testing.

3.2.2 2D Map Location

The mapping functionality, when tested using the collected GPS data, does show a good level of accuracy while placing markers relative to the map tile being displayed. With the marker maintaining the expected path on the streets and roads on the map, in all test cases. When testing for potential sources of error using rounded and extreme values, the marker placement was predictably affected by the quality of the data received (referring to the floating point accuracy of the GPS data), this would be expected in any system using Latitude and Longitude values due to the level of accuracy needed to pin point world locations, even small variations in values will result in miscalculated locations.

3.2.3 Distance Monitoring

For distance calculations, the overall distance calculated did occasionally lose a small degree than what would be expected if using live data, due to the significantly lower update rate of the test creating a more generalised line when tracking the UAV. However, it has to be considered that due to the speeds which can be achieved by a UAV in flight the distances covered between updates could be much larger than those seen in data which was collected from human movement. This combined with the occasional lost UDP message means that this small loss of accuracy may still be seen in operational conditions, however it was not a major issue and again is commonly seen in GPS driven calculations.

It is also worth considering that with the target update rate of 15-20hz from a UAV, it may not be an efficient practise to be updating the distance calculations with every update as the degree of accuracy produced may not be sufficient to justify the overheads of the calculations required. Unfortunately, it was not possible to fully test for this potential efficiency issue and if recalculating distances every updated offered particular benefit under operational conditions.

3.2.4 Ground Speed Calculations

Following on from monitoring distances travelled, it was important to communicate the current ground speed of the UAV to the operator. While some flight controllers may have this value as a requestable data item, not all provide this data. So to provide this functionality the speed is being calculated using the time between updates and positional changes to provide the speed in KPH, these values were again compared against those obtained from the original Strava data and it was seen this method maintained a good level of approximation, as the value displayed was being run through the rounding with hysteresis there was the occasion discrepancy but no severe variations. This rounding method was being used to reduce the flickering which can otherwise be seen when using standard rounding methods. Its use did greatly improve the usability of the element by preventing the user's attention from being unnecessarily drawn to the speed display by near constant movement. The hysteresis rounding method itself was tested early in development by feeding a separate collection of false speed values with small variations which would normally trigger rounded values to flicker, this effect was greatly reduced through the use of hysteresis to control the rounding direction in use.

3.2.5 General Observations during Element Testing

While of great use during this testing stage, the use of Strava data did not completely replace the expected data from an operational UAV particularly when considering the update rates being used. With the collected Strava data typically having one update every second, while the target update rate from an operational UAV being considered is 15-20 updates per second. This difference in data update rate did result in much more sporadic movement and calculations, but this was taken into consideration when assessing the results of testing.

However, the presence of this sporadic movement allowed for smoothing and predication systems to be developed and tested to help reduce visual jitters due to missed data updates from the network. Despite the value of using Strava data during testing, further testing with live data is still desirable to properly review the functionality of UI elements in true operational conditions. Although it can be reasonably expected from the results seen that elements will continue to function correctly when provided with live data, but may encounter issues stemming from the live update process and this may only be noticeable in specific operation conditions which could not be replicated in the test environment.

3.3 Whole Solution

Multiple hardware systems have been used while testing the produced components as a functional system, details on the UAV hardware used is provided in table 3. This was done in order to test and demonstrate the flexibility of the system produced, through working with a variety of hardware and protocols in both development and testing. This was the last stage of testing conducted, integrating hardware, middleware and the visualisation together as a complete solution. Working in a manner as similar to its intended use under operation conditions, to demonstrate and test elements functionality as a single unified solution. Paying particular attention to maintaining a useable framerate and smooth motion of 3D elements, and accounting for predictable issues such as missing individual data items or message loss, without the system becoming unusable.

Setup Name	Protocol Available	Data Available
Arduino Flight Controller	MSPv2	IMU
Racing Drone	MSPv2	IMU Motors Battery
Cloud3D Small UAV	MSPv2 & Mavlink	IMU, Battery, GPS

Table 3 testing hardware

3.3.1 Functional Testing

Overall functional testing of the complete system was successful, regularly maintaining a framerate above 90fps and easily exceeding the target frame rate of 60fps. The main issue observed during the functional testing when using different hardware, is that the UI elements as they currently stand are predictably dependant on receiving the correct data

through the network messages received. When working with systems where particular data items such as the current battery level or GPS data was not available these elements were non-functional, but are still present on the screen and would be periodically requesting data from the parsers and receiving default values to indicate that data items were not present. This can be both confusing to the user and does not support maintaining an efficient application. Whether data is available is dependent on both the hardware in use on the UAV system, but also on the protocol being used. During testing both MSPv2 and MavLink protocols were used, with MavLink being the preferred target protocol for the end solution due to the wider selection of data items available through its use.

While this did not cause any noticeable functionality issues for the visualisation application; some elements could be misleading to an operator as they are still present. When working with systems where it is known that particular data items will not be available – e.g. racing drones typically don't support GPS data -it would be a desirable feature to be able to disable the associated displays for absent data. The disabling of unused UI elements would help to support maintaining an efficient system by reducing or removing the overhead generated by these unused elements; and could improve the end users experience by decluttering the viewport leaving only the data relevant to the UAV system in use.

The adaption of displayed items is an interesting area for further development, where identifying an appropriate approach is an important aspect for creating a practical system for operators. Two general approaches could be viable options but need further consideration before attempting to implement within the visualisation. Either looking to integrate a setup stage for the selection of UI elements or automatically culling unused elements could be workable solutions, but both have drawbacks. The first method would introduce more setup work in addition to that already existing when setting up the parser and the connection to hardware, this additional setup work may not be desirable attempting to carry out operation flights. While the second approach risks introducing additional debugging work for operators, if an element is just not present when it was expected, the operator would have to spend time attempting to identify if its absence is due to a fault in the software execution, setup or a hardware fault.

4 Conclusion

This project aimed to build on the current growth of game technology-based solutions in non-games industries; specifically looking at the applications of commercial UAVs a sector where the hardware, software and purpose of systems can vary hugely. Part of this variation includes the data requirements and data streams produced. This project aimed to demonstrate a potential application of games technology within the UAV industry; by developing a game engine driven real-time visualisation of a UAV system, while maintaining a high level of flexibility in the data stream being used to drive functionality.

The solution produced provided a parser which could be adapted to allow different message structures to be used when sending data via UDP messages. This enabled the visualisation to be used with data streams with varied data formats to support different UAV systems use. The adaptable parser has been developed as an engine plug-in and remains separate from the purpose of the application, and so could be used for developing other applications within the engine with no connection to the complete solution produced for this project.

Overall as a visualisation, the solution did provide and maintain the key data needed when monitoring a remote UAV, and has demonstrated that game engines are capable of providing the necessary resources and functionality for developing these types of applications. While desk-based tests have proved promising, a key objective of this project was to evaluate the effectiveness of the solution for supporting safe operation of UAVs. However due to restrictions put in place during development and testing, it was not possible to carry out the flight tests needed to be able to evaluate effectiveness of the solution under operational conditions so no comments can be reasonably made on this.

4.1 Limitations and Future Work

There are limitations and scope for future improvements across the components produced as part of this solution, some of these are discussed below.

The parser is limited to handing a select list of data types commonly used in C++ applications, and data types out with this currently are not supported, expanding the supported data types is an obvious area for expansion. As a networked solution the parser is also limited by the minimal security offered by the current setup, further development to

support the use of common network message security measures such as checksums would be valuable for creating a more robust solution with a wider variety of possible applications.

The visualisation, while suitable as a demonstration of a potential application of games technology for real-time monitoring of varied UAV systems; it is far from a complete solution as it currently stands. Expansions to alter UI elements being displayed, the choice of digital model at runtime would all be beneficial to the function of the application and the User experience. There is also potential to improve the mapping solution implemented to be an internet-based request of maps rather than a predefined collection of tiles, to reduce the preparation work needed by an operator prior to flying in a new area. When considering representation UAVs there is also a huge variety of potential data which is not being used which could be of benefit – such as replicating lights and peripheral devices fitted to systems.

While the aim of this project was to produce as flexible system as was reasonable, there is the possibility to develop a more UAV focused system for incorporating data streams from specific protocols such as MultiWii or MavLink, and thus negate the need for middleware or the parser implemented.

5 List of References

CAA (2015) Unmanned Aircraft and Drones. Available at: https://www.caa.co.uk/Consumers/Unmanned-aircraft-and-drones/ (Accessed: Nov 3, 2019).

Campanile, F., Cilardo, A., Coppolino, L. and Romano, L. (2007) Adaptable Parsing of Real-Time Data Streams.

Crecente, B. (2016) McLaren now uses Epic's Unreal Engine to help design its high-end cars. Available at: https://www.polygon.com/2016/3/16/11246586/mclaren-unreal-engine-design (Accessed: Feb 4, 2020).

Federal Aviation Administration (2019) Unmanned Aircraft System. USA: Federal Aviation Administration. Available at: https://www.faa.gov/data_research/aviation/aerospace_forecasts/media/unmanned_aircraft_systems.pdf (Accessed: Dec 4, 2019).

Haversine Formula – Calculate geographic distance on earth. (2015) Available at: https://www.igismap.com/haversine-formula-calculate-geographic-distance-earth/ (Accessed: Apr 6, 2020).

Kyoung-Dae Kim and Kumar, P.R. (2013) 'Real-Time Middleware for Networked Control Systems and Application to an Unstable System', IEEE Transactions on Control Systems Technology, 21(5), pp. 1898-1906. doi: 10.1109/TCST.2012.2207386.

Loo, B., Condie, T., Garofalakis, M., Gay, D., Hellerstein, J., Maniatis, P., Ramakrishnan, R., Roscoe, T. and Stoica, I. (2009) 'Declarative Networking', Communications of the ACM; Commun.ACM, 52(11), pp. 87-95. doi: 10.1145/1592761.1592785.

Minden, G., Komp, E., Ganje, S., Kannan, A., Subramaniam, S., Tan, S., Vallabhaneni, S. and Evans, J. (2002) Composite protocols for innovative active services.

Sloan, K. (2018) BMW Brings Mixed Reality to Automotive Design with Unreal Engine. Available at: https://www.unrealengine.com/en-US/spotlights/bmw-brings-mixed-reality-to-automotive-design-with-unreal-engine (Accessed: Dec 29, 2019).

Swatch (2018) The Swatch DRL Try-outs - Swatch® United Kingdom. Available at: https://www.swatch.com/en_gb/explore/swatch-sports/articles/the-swatch-drl-tryouts/ (Accessed: Feb 13, 2020).

6 Bibliography

(2017) OpenStreetMap. Available at: https://wiki.openstreetmap.org/wiki/About_OpenStreetMap (Accessed: 03/11/2019).

Microsoft Built-in types (C++). Available at: https://docs.microsoft.com/en-us/cpp/cpp/fundamental-types-cpp (Accessed: Jan 9, 2020).

MultiWii Wiki. Available at: http://www.multiwii.com/wiki/index.php?title=Main_Page (Accessed: 03/11/2019).

Rama (2014) (39) Rama's Extra Blueprint Nodes for You as a Plugin, No C++ Required! Available at: https://forums.unrealengine.com/development-discussion/blueprint-visual-scripting/4014-39-rama-s-extra-blueprint-nodes-for-you-as-a-plugin-no-c-required (Accessed: Mar 15, 2020).

Rama. (2019) VictoryPlugin23 [1]. Available at: http://www.mediafire.com/file/2jedu15w6p3emz8/VictoryPlugin23.zip/file (Downloaded: Mar 15, 2020).

7 Appendices

7.1 Appendix 1 - Desktop flight controller

A MultiWii based flight controller was built to support the initial development, allowing limited live data to be collected and sent over the network. Using a specific set up of a popular UAV flight controller software, connected to the testing pc through a serial port.

Components Used	Purpose
Arduino Nano V3.0	Main component for running flight controller software
GY-521 MPU-6050	Gyroscopic and angular acceleration data

Table 4 flight controller components

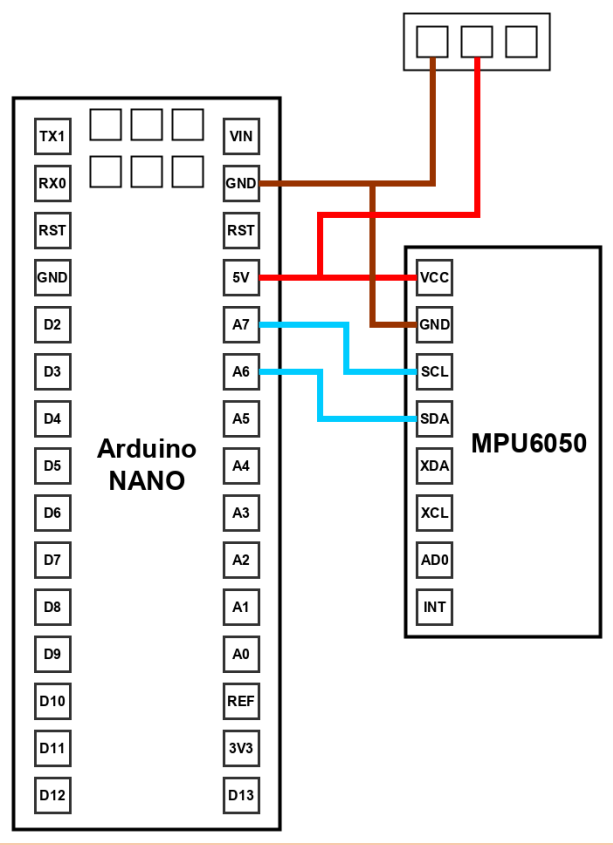

Figure 7-1 flight controller wiring

7.2 Appendix 2 – Required Python Libraries

When building middleware for connecting to specific hardware – such as the Desktop flight controller – particular Python libraries has to be installed to support or provide the development of the required functionality.

- Keyboard
- Schedule
- Socket
- Serial
- WiiProxy
- csv
- MavLink

7.3 Appendix 3 – Visualisation User Interface Design