ENHANCING THE FUNCTIONAL DESIGN OF A MULTI-TOUCH UAV GROUND CONTROL STATION

By

Jeffrey Haber, B.Eng
Aerospace Engineering
Ryerson University, 2013

A thesis presented to Ryerson University

in partial fulfillment of the
requirements for the degree of

Master of Applied Science

in the Program of
Aerospace Engineering

Toronto, Ontario, Canada, 2015

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners. I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research. I understand that my thesis may be made electronically available to the public.

ENHANCING THE FUNCTIONAL DESIGN OF A MULTI-TOUCH UAV GROUND CONTROL STATION

Jeffrey Haber
Master of Applied Science, Aerospace Engineering, Ryerson University (2015)

Abstract

This thesis presents a reconfigurable Ground Control Station designed for Unmanned Aerial Vehicle use, which utilizes multi-touch gesture inputs as well as the ability for the operator to personalize where the instruments they interact with are located on screen. The Ground Control Station that is presented was designed and developed in Ryerson University's Mixed-Reality Immersive Motion Simulation Laboratory utilizing commercial off the shelf programs supplied by Presagis. Presagis' VAPS XT 4.1 beta was used to design and develop the actual Ground Control Station's User Interface due to its ability to create high quality interfaces for aircraft that harness multi-touch gestures. While FlightSIM 14 was used to simulate a high fidelity aircraft being controlled by the Ground Control Station. The final interface was comprised of six key features and 12 different instrument panels that could be manipulated by the operator to control a simulated aircraft throughout a virtual environment.

Acknowledgements

I would like to thank my advisor, Dr. Joon Chung, for all his support, guidance, and help throughout the last year as I navigated through the thesis process. I would also like to thank Eric Simon and Presagis for providing the resources to conduct my research. Finally, I would like to thank my family and friends for always supporting my efforts through thick and thin.

Table of Contents

Author's Declaration ii

Abstract iii

Acknowledgements iv

Table of Contents v

List of Acronyms vii

List of Tables viii

List of Figures ix

1.0 Introduction 1

1.1 Unmanned Aerial Vehicles 1

1.2 Human Machine Interfacing 5

1.3 Human Factors 12

1.4 Ryerson's Mixed-Reality Immersive Motion Simulation Laboratory 14

1.5 Thesis Overview 17

2.0 Design Considerations 18

2.1 Ground Control Station Considerations 18

2.2 Human Factor Considerations 21

2.3 User Interface Considerations 22

3.0 Flight Integration 25

3.1 FlightSIM 26

3.1.1 FlightSIM Aircraft Modeler 27

3.1.2 FlightSIM Run-Time Setup 30

3.2 VAPS XT 34

3.3 nCom 38

4.0 Design Process 42

4.1 Conceptual Stage 42

4.2 First Design Iteration 45

4.3 Second Design Iteration 47

4.4 Final Design 48

5.0 Results 51
5.1 Ground Control Station Features 52
5.1.1 Alerts and Notifications 53
5.1.2 Instrument Side Tab 54
5.1.3 Multi-Touch Gestures 56
5.1.4 Reconfigurability 60
5.1.5 Top Tab Bar 66
5.1.6 Virtual Keyboard 68
5.2 Ground Control Station Instrument Panels 69
5.2.1 Aircraft General Parameters 70
5.2.2 Angle of Attack Gauge 72
5.2.3 Autopilot/ Waypoint Controller 73
5.2.4 Detailed Engine Monitor 81
5.2.5 Flap Controller 82
5.2.6 Fuel Gauge 84
5.2.7 Gesture Controlled Map 85
5.2.8 Non Gesture Controlled Map 88
5.2.9 Payload Controller 89
5.2.10 Primary Flight Display 94
5.2.11 Simple Engine Monitor 96
5.2.12 Throttle and Gear Controller 97
5.3 Simulator Tutorial 98
6.0 Conclusion 114
6.1 Contributions 115
6.2 Future Work 116
7.0 References 119

List of Acronyms

API	Application Program Interface
HUD	Heads up Display
I/O	Input/ Output
MIMS	Mixed-Reality Immersive Motion Simulation
NASA	National Aeronautics and Space Administration
NATO	North Atlantic Treaty Organization
P2P	Peer-to-Peer
RPM	Rotations per Minute
SDK	Software Development Kit
TCP	Transmission Control Protocol
UAV	Unmanned Aerial Vehicle
UDP	User Datagram Protocol
UI	User Interface

List of Tables

Table 1- List of Presagis Software Programs 26
Table 2- F-16 Fighting Falcon Specifications [84] 29
Table 3 - Word and Bit Numbers for Joystick Input 33
Table 4- Word and Bit Numbers for Rudder and Throttle 34
Table 5- Utilized I/O Buffers 41
Table 6- InstrumentLocation Array Element Allocation 64
Table 7- InstrumentLcation Array Panel Order 64
Table 8- Instruments Found Within Preset Tabs 67
Table 9- WptArray Element Allocation 76

List of Figures

Figure 1 - Sperry Aerial Torpedo [3] 2
Figure 2- Modern UAVs: General Atomic's Predator B [5] (Left) and DJI's Phantom Aerial UAV [6] (Right) 3
Figure 3 - CAE 7000XR Full-Flight Simulator [30] 6
Figure 4 - Oculus Rift Goggles [36] 8
Figure 5- Microsoft Kinect 2.0 Camera Sensor [47] 9
Figure 6- Microsoft Kinect 2.0 Camera Views [48] 10
Figure 7- Frustrated Total Internal Reflection Technology [55] 11
Figure 8 - Workload Effects 14
Figure 9 - MIMS Lab Simulators: a) Fixed Based Simulator, b) Full Motion Simulation, and c) UAV Ground Control Station [70] 16
Figure 10- Presagis Program Integration 25
Figure 11- FlightSIM Modeler User Interface 27
Figure 12- Screenshot of FlightSIM's User Interface 31
Figure 13- VAPS XT User Interface 35
Figure 14- Example of a Data Flow 37
Figure 15- Example of an Internal Transition 37
Figure 16 - State Chart Example 38
Figure 17- FlightSIM nCom Connection 39
Figure 18 - VAPS XT nCom Connection 40
Figure 19- Initial Alert and Notification Concept Drawing 43
Figure 20 - Reconfigurable UI Concept 44
Figure 21- Sample Flight Control Test Panel 45
Figure 22- First Iteration User Interface 46
Figure 23- Radial Menu Concept 47
Figure 24 - ITSEC Demo (Second Iteration) User Interface 48
Figure 25 - Final User Interface Design 49
Figure 26 - Ground Control Station Functionality Flow Chart 52
Figure 27- Sample Alert and Notification Pop-up (Left) and Expanded Panel (Right) 53
Figure 28 - Instrument Side Tab Bar 55
Figure 29- Selecting a Panel to be Inserted into the UI 56
Figure 30- Pinch Gesture 58
Figure 31- Drag Gesture 59
Figure 32- Tap Gesture 59
Figure 33- Swipe Gesture 60
Figure 34 - Sample of a Non-Interactable Panel (Left) Versus an Interactable Panel (Right) 62
Figure 35 - Angle of Attack Gauge Boundary Variable Diagram 63
Figure 36- Closing Panels Pop-Up 66
Figure 37- Top Tab Bar 66
Figure 38-Virtual Keyboard 68
Figure 39 - Aircraft General Parameters Instrument Panel 71

Figure 40 - Angle of Attack Gauge 73
Figure 41- Autopilot/ Waypoint Controller 73
Figure 42- Flight Director Logic [91] 75
Figure 43 - Autopilot State Chart Diagram 77
Figure 44- Trigonometry of an Octagon 80
Figure 45- Detailed Engine Monitor Instrument Panel 82
Figure 46- Flap Controller Instrument Panel 84
Figure 47- Fuel Gauge Instrument Panel 85
Figure 48- Gesture Controlled Map Instrument Panel 86
Figure 49 - Non-Gesture Controlled Map Instrument Panel 88
Figure 50 - Fictional Payload Controller 89
Figure 51- Target Box 1's Movement Pattern With Associated Times 90
Figure 52 - Greyscale Map with High Value Target (Lower Box) and Regular Potential Target (Upper Box) 91
Figure 53- Trigonometry of Determining if a Point is Within a Circle 92
Figure 54 - Primary Flight Display Instrument Panel 96
Figure 55 - Simple Engine Monitoring Panel 96
Figure 56- Throttle and Gear Controller 98
Figure 57- Tutorial Objective Placement 99
Figure 58- Aircraft Has Not Flown Through the Waypoint (Left) and Aircraft Has Flown Through the Waypoint (Right) 100
Figure 59 - Tutorial Payload Controller Setup 101
Figure 60 - Tutorial Custom Tab Setup 102
Figure 61- Tutorial Waypoint Creation – 1) First Waypoint (Keyboard), 2) Third Waypoint (Virtual Keyboard) 103
Figure 62 - Tutorial Take-Off 103
Figure 63 - Tutorial G-Load Warning 104
Figure 64 - Tutorial First Waypoint – 1) First Attempt, 2) Second Attempt 104
Figure 65 - Tutorial Engaging Autopilot 105
Figure 66 - Tutorial Engine Monitoring 106
Figure 67- Tutorial Second Waypoint 107
Figure 68 - Tutorial Aircraft Location Check 1 107
Figure 69- Tutorial Payload Controller IR Mode 108
Figure 70 - Tutorial Payload Controller Missed Shot 109
Figure 71- Tutorial Payload Controller Successful Hit 109
Figure 72- Tutorial Location Check 2 110
Figure 73- Tutorial Autopilot Disengage 111
Figure 74- Tutorial Final Waypoint 111
Figure 75- Tutorial Landing Line-Up 112
Figure 76 - Tutorial Completed Landing 112
Figure 77- Tutorial Final Flight Path 113

1.0 Introduction

1.1 Unmanned Aerial Vehicles

Within the last 15 years, the number of Unmanned Aerial Vehicles (UAV) has increased exponentially. UAVs are being primarily used by the military market however they are slowly making their way into more commercial markets for civilian use. Within the military, UAVs are used to conduct dangerous missions (e.g. close air support for infantry or the elimination of an High Value Target) in an effort to save lives, and long duration missions (e.g. surveillance and intelligence gathering) where fuel levels can be conserved by using low weight, high engine efficiency aircraft. In contrast, civilian markets are exploring the use of UAVs for jobs such as; Search and Rescue, agricultural upkeep, film making, and even parcel delivery. These applications are enhanced by UAVs due to their ability to be rapidly deployed and their fuel efficiency, both of which help to improve company profit.

The first attempt at designing an unmanned aircraft was during the First World War when primitive autopilot systems were developed to help control Sperry's aerial torpedoes (shown in Figure 1), as well as the Kittering Bug, both of which were pilotless aircraft filled with explosives [1]. After the war, Archibald Low was the first person to develop an aircraft system in 1917 that could be controlled through radio signals [1]. This method however proved to be erratic and difficult to control which lead to the technology being relatively unused. Radio control did however prove to be a useful method for controlling small target drones for target practice [1]. During the Cold War in the 1960s, the first UAVs began to appear for military uses due to advances in sensor technology and the ability to reuse the unmanned aircraft. However these early UAVs were only used for military reconnaissance and sometimes electronic warfare [1]. Finally due to the advances in electronic technology in the last 50 years, UAVs have become a staple of military forces around the world and are slowly becoming more common within civilian life. Two such examples of modern UAVs are General Atomics Predator UAV (used for military operations) and DJI's Aerial Phantom (used for aerial photography and film), shown in Figure 2.

UAV deployment in the United States Military started to exponentially increase during the early 2000s. Between 2000 and 2010, the United States Department of Defense increased the number of UAVs they operated from 50, in 2000, to over 7000 in 2010 [2], an increase of over 13 000% in 10 years.

With the vast increase in UAV usage, the number of UAV pilots also had to increase. Since 2009 the number of UAV pilots in training began to outnumber the amount of manned aircraft pilots [2].

Figure 1 - Sperry Aerial Torpedo [3]

This increase in UAV usage does not come without a cost. The number of UAV related accidents outnumbers manned military aircraft crashes by roughly 50:1. Currently the United States Military is experiencing roughly 50 UAV accidents per 100 000 hours of operations, for each type of UAV they handle [4]. This is in contrast to roughly one manned military aircraft accident per 100 000 hours of operation [4]. Of all these unmanned aircraft crashes, 69% were caused from a human factor issue, and of these, 24% (or roughly 17% of all accidents) were caused from a human factor issue directly relating to the Ground Control Station [4]. Based upon this information when UAVs become commonplace within civilian/ private life that there will be roughly 330 unmanned vehicle accidents per 100 000 hours of operation [1]. The most likely reason for this drastic difference in terms of accident rates is because private users of UAVs (i.e. those not utilizing UAVs for their job) do not receive extensive (or any) training in terms of how to pilot their aircraft. This can lead to users not having the proper training when it comes to dealing with problems that may arise during flight, thus leading to more accidents. This contrasts commercial and military users who receive extensive training so that they feel comfortable with controlling the aircraft and know how to deal with potential problems.

Figure 2- Modern UAVs: General Atomic's Predator B [5] (Left) and DJI's Phantom Aerial UAV [6] (Right)

Regardless of the type of UAV being flown (i.e. a large military aircraft or a small aircraft being flown recreationally) all UAVs share three commonalities [1]. The first is that they all require some kind of Ground Control Station to control and monitor the flight status of the vehicle. The Ground Control Station could be anything from a small remote controller typically used for the control of small Radio Controlled aircraft, to large multi-monitor/crew member stations housed within a complex. The second is that they all require some form of communication link to the controller so that the UAV and Ground Control Station can send and receive data to one another. This could be as simple as a short range line of sight radio transmitter typically used for Radio Controlled aircraft, to a direct satellite uplink more frequently used by military and government. The third commonality between all types of UAVs is that they require some form of support equipment.

In terms of actually controlling an UAV, there are three potential levels [1]. The first is manual control. This means that the aircraft's attitude is controlled directly by the operator. The second level is supervisory control. When an aircraft is controlled at this level, it means that the UAV is partially automated. The operator is sending the aircraft certain commands (e.g. waypoint locations) which the UAV will then execute. The final level is when the UAV is fully autonomous. This means that the UAV has full control over its actions. Once a mission is outlined, the UAV will determine how to best complete it. In such a case, the operator will mainly be monitoring the aircraft's status so that if something goes wrong, they can take control. Regardless of the UAV, they will have at least one of these methods of control, if not the potential to be operated in any of the three modes.

UAVs are not perfect systems. Up until the last decade and a half they were rarely used. As a result there are still a number of shortcomings with current UAV systems and their associated control stations. First and foremost is that there is a distinct lack of Situational Awareness [7], [8] due to the fact that the operator is not physically within the aircraft. The operator must make educated guesses on how

the aircraft will react to changes in momentum caused from changes in the attitude of the aircraft [8]. The lack of Situational Awareness is further reduced because onboard cameras tend to provide a low Field-of-View [9] which limits what the operator can physically see, thus reducing what they can determine about the aircraft's current situation. In addition, most UAVs typically lack multi-sensory inputs (i.e. no haptic feedback for a joystick) that can be used by the operator to get a feel for how the aircraft is reacting to its environment [9]. As a result a large number of unmanned aircraft crashes were caused in due part to the lack of Situational Awareness [10], [11].

Lack of Situational Awareness is not the only problem, though it is one of the most prevalent. Another common issue with UAV operation is that there is typically a slight delay between what the operator sees on screen and what the actual aircraft is doing [7]. This is due to the fact that all beyond line of sight data transmission between the two systems (Ground Control Station and the UAV) must first travel to through a series of relay nodes (typically satellites) and then to the desired system [12]. If these delays are larger than one second, manual control of the aircraft is typically deemed impossible. Even 50ms delays are considered incredibly difficult [1]. Finally, there are no standards when it comes to communication protocols, or general Ground Control Station interfaces. Currently the industry is trying to establish standards for communication protocols/ interfaces so that allied countries can easily share information between one another and to help streamline training. In particular, NATO is currently pushing for the acceptance of STANAG 4586 as the standard for UAV communication protocols between UAVs and Ground Control Station to help with the ease of transfer of information [13], [14].

Due to these issues, there has been a lot of research done focusing on these areas. The most common areas of UAV research focus on efforts to improve upon:

1. User Situational Awareness [11], [15], [16]
2. Changing how the user perceives the environment [15], [16], [17]typically through the use of virtual or augmented reality,
3. Human-Robot Team interactions [15], [18]
4. Creating new/ improving upon, autopilot systems that are used by UAVs to increase the functionality of autonomous flight [19], [20], [21]

1.2 Human Machine Interfacing

Human Machine Interfaces are a method for people to interact and transmit data with a machine [22]. Human Machine Interfaces can be a physical interface that the user interacts with, or it can be a method of inputting information into a system. Human Machine Interface technology is important because it allows a wide variety of people to interact with another mechanical system, with the two most common methods being a mouse and keyboard or a joystick [22]. Despite their frequent use in society, Human Machine Interfaces can be difficult to analyze due to the fact that human reliability is not as well understood, as mechanical hardware reliability is when looking at fault analysis [23]. As such, one typically has to look at either the human factors that might affect the user or the machine's own performance separately.

Human Machine Interfaces are possible through the advancement of machines and computers because they can process large quantities of data in a rapid, clear, and effective way [23]. As such one does not have to sift through vast quantities of data to determine how precisely the system is reacting to an input. More importantly, some systems can directly interact with the user simply by representing information in a graphical manner including, the information's location on the screen, size, and / or colour [24]. For instance, drastic contrasts in colour can help pull the eye to a specific location. These methods can be placed into one of two categories, foreground information and background information. Foreground information directly represents the transmission of raw information, typically in the form of text [24]. Whereas background information indirectly represents the data found on screen through an object's choice of colour, size and location [24]. As a result, nearly all information is presented in a combination of background and foreground information when it comes to computer and mobile device interfaces [24].

Simulators are a standard example of a Human Machine Interface. The simulator itself is a mechanical object that helps to represent another object (i.e. a cockpit) allowing a person to gather training experience in a safe environment. Simulators, such as CAE's7000XR shown in Figure 3, are quickly becoming a popular method of training individuals in the use of machines (e.g. a commercial aircraft) [25]. In addition, they are necessary for training military forces during times of peace when users cannot gain firsthand experience [26]. For a simulator to be effective, they have to focus on the actual evaluation of the user and/ or vehicle as opposed to focusing on the minute details of running the actual simulation [27]. When used correctly simulators have been found to enhance overall user training [28] while also helping to reduce training costs and times [29].

Figure 3 - CAE 7000XR Full-Flight Simulator [30]

There are a number of advantages that a Human Machine Interface can provide. One such advantage is that some systems will allow the user to reconfigure their setup. This can be done by physically rearranging the hardware (i.e. moving a joystick from one side of a table to another), or by changing the hardware specifications (i.e. changing out the components within a personal computer). The reconfigurability can also be applied to the software/ User Interface (UI) side of the system where the user might choose to optimize their interface so that multiple people can interact with the vehicle, thus increasing team performance [31]. Most computer based setups allow people to increase (or decrease) the number of available monitors that can be used (typically constrained to hardware limitations). Multiple screens can be used to help reduce overloading the operator with information (i.e. by allowing them to reposition various windows) and help to get around the fact that monitors have set screen size limitations [32].

Human Machine Interfaces are not just methods of representing data to a user. They also encompass how the user physically interacts, or visualizes the data. There are numerous methods for interacting with machines. These include, but are not limited to [32]:

1. 3D audio, used to improve a user's Situational Awareness
2. Eye tracking, where the position of one's eyes are tracked by a machine
3. Gesture recognition, where human gestures are detected through touching a screen or an algorithm that utilizes a camera and are converted to a machine input

4. Haptic devices, used to stimulate people's feelings of touch or by changing the resistance of how something moves
5. Head tracking, where the position of one's head is tracked by a machine
6. Mouse and keyboard devices, where the user can use a mouse to accurately click on objects and a keyboard to type in commands
7. Speech recognitions, where speech commands are converted to machine inputs
8. Speech synthesis, where verbal commands are translated to text
9. Virtual Reality, where the visuals are displayed in a custom headset to immerse the user into the environment

One unique aspect of computer based systems is that they often can support multiple inputs. One does have to be wary about adding too many, however as it has been found that the more types of Human Machine Interface inputs a user is subjected to, the more likely they will become overloaded with information [32]. Of the nine Human Machine Interface inputs listed above, the most commonly found/ researched methods are in the areas of Virtual Reality immersion, mouse and keyboard inputs and gesture recognition.

Currently Virtual Reality has become a hot topic for research as it provides a very unique method of visualizing data. Virtual Reality is only possible through the use of specialized headsets, such as those developed by Oculus Rift [33] (shown in Figure 4). These headsets have a very small, high resolution screen on the inside of the headset. The image is then split into two viewable regions and magnified by two separate eye lenses. One major advantage of Virtual Reality goggles, (and their sister technology Augmented Reality where graphics display on a transparent screen much like a Heads-Up-Display (HUD)), is that various hazards can be visually marked to make them stand out. This is particularly advantageous for pilots when flying in terrain rich environments, where red warning lines could be used to mark the tops of objects such as mountains [9], or to visually show the flight path of an aircraft to the pilot when they look outside of a window [16]. Virtual Reality technology is relatively new and as such is not readily available for vast commercial use. The two biggest hurdles that Virtual Reality still must overcome are the screen's frame rate and the overall Field-of-View they provide. Frame rate is defined as the time it takes for a screen to refresh all of the pixels on the screen to create a new unique image (known as a frame). Currently Oculus encourages developers to create games/ applications that run above 75FPS [34] as anything below that rate can cause eye fatigue. The second hurdle is the loss in

Field-of-View, which is attributed to the fact that the goggles use a flat, non-curved screen. This can lead to training errors [16], [35] because it limits the amount of Situational Awareness available to the user.

Figure 4 - Oculus Rift Goggles [36]

The most common method of interacting with a digital device is the mouse and keyboard. These devices are particularly useful to more accurately interact with small objects on screen, (e.g. objects smaller than 16 pixels in width) [37]. However they have a number of disadvantages. First and foremost they are limited to single inputs (i.e. a mouse click or the press of a keyboard button) compared to more recently developed gesture inputs that allow a user to use multiple inputs at any given time (i.e. multiple fingers on the screen) [38], [39]. These systems also tend to increase the user's reaction times by typically requiring two hands to efficiently type on a keyboard and then having to switch to one hand to use a mouse [40]. And, one needs to interact with both devices in different ways (i.e. typing vs. mouse clicking) which can lead to operator overload during stressful situations [40] . Further, using a mouse and keyboard can lead to physical health problems, typically in the form of a repetitive strain injury [40], which can lead to significantly reduced performance or discontinuation of the activity all together until an injury has healed. Finally mouse and keyboard setups can take up large areas of space due to the size of a keyboard and the amount of room required to move a mouse around [40]. A number of these issues can however be addressed by utilizing more recently developed gesture recognition hardware/ software.

A number of these issues can be addressed by utilizing more recently developed gesture recognition hardware/ software. Currently there are two distinct methods of detecting gestures. The

first is through computer algorithms that utilize data retrieved from a camera, and the second is through a multi-touch monitor that determines the gesture through changes in finger position. Both have their uses however multi-touch screens are the more popular method as they can be easily implemented and do not require the user to use external hardware (i.e. a specialized camera sensor) to determine if there is a gesture.

Regardless of which of the two gesture recognition methods are used, research has shown that gesture commands offer distinct advantages over more traditional mouse and keyboard systems. The greatest advantage that all gesture systems have is that gestures are very natural for the human body to adapt to and use [41], [42]. This comes from the fact that the human body can intuitively keep track of its limbs kinaesthetically, which allows one to utilize the natural dexterity of one's body [37], [43]. People also naturally tend to communicate with one another through gestures [44], [45]. Interacting with the computer based system in a similar fashion gives a stronger feeling of control over one's interactions with the system [46].

Figure 5- Microsoft Kinect 2.0 Camera Sensor [47]

Of the two gesture recognition methods, the least popular method is the one that requires a specialized camera, such as Microsoft's Kinect 2.0 camera sensor shown in Figure 5. These systems can be developed using custom Application Program Interface (API) programs or utilizing Microsoft's Kinect Software Development Kit (SDK). Microsoft's Kinect 2.0 uses a high definition dual infrared/ colour camera (the various camera colour modes are shown in Figure 6) can track up to 25 different joints on a maximum of six people at any given time [48]. These joints can then be analysed within custom

applications (such as those written with Microsoft's SDK) that determine if the motion of the joints match up with a predefined gesture. If the motion is similar to those found within its database a set command can be given to the machine. This form of gesture recognition however has two major issues that make it imperfect for commercial use. The first is that most systems, (excluding the Kinect 2.0), have a very difficult time recognizing 3D gestures as the sensor's cannot accurately measure depth [44]. And secondly the cameras that are used to pick up the gestures can have major issues when it comes to finding/ determining the actual motion of the joints due to background interference, typically caused by inconsistent lighting, static background objects, and joint/ object orientation [41], [44]. These issues however have not stopped people from attempting to use motion gestures as a method for controlling aircraft. *Easy Gesture Recognition for Kinect* shows a system that utilized SVTs and DTs to recognize gesture commands given by United States Air Force personnel for aircraft [49]. While *Control of Flight Operation of a Quad rotor AR.Drone Using Depth Map from Microsoft Kinect Sensor* showcases a UAV that can be controlled by gestures read by Microsoft's Kinect [50].

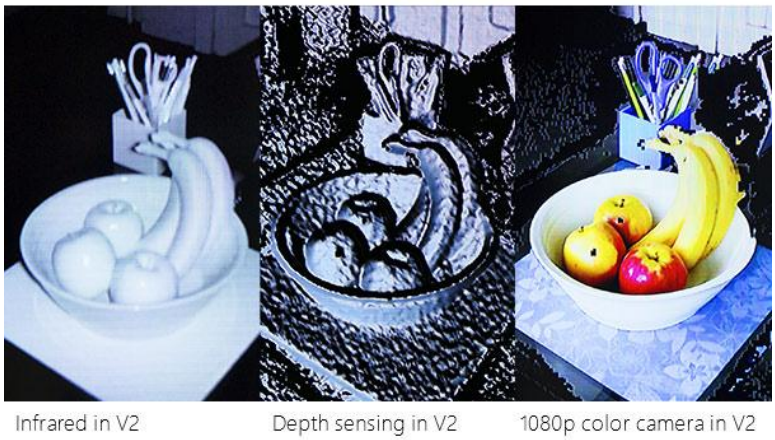

Figure 6- Microsoft Kinect 2.0 Camera Views [48]

Multi-touch monitors on the other hand do not encounter the same identification/ interference issues that motion gestures have. Their screens utilize Frustrated Total Internal Reflection to determine if there is a touch point, and its location [38]. Systems that use Frustrated Total Internal Reflection work by shining an infrared light through a transparent medium. If there is an object touching the screen the infrared light is refracted back towards its source where it is picked up by a diode sensor [38]. Figure 7 shows a visual representation of how Frustrated Total Internal Reflection technology works. Multi-touch devices are seen as very powerful for Human Machine Interfaces because the human hand is the best body part for expression, thus making it a prime candidate for gesture commands [41]. Additionally they

are seen as powerful methods of input because they are commonly used in society for interacting with mobile smart devices. Touch gestures, as a result are very easy for anyone to pick up and learn [43], [51]. Touch gestures can also be used for intuitive and inventive ways to access system features because of the wide variety of gestures that can be harnessed (i.e. taps, drags, pinches, swipes, etc.) [24]. Due to their robust use, aerospace companies, such as Garmin and Thales, have begun to design and integrate multi-touch avionic systems into their concepts for near-future glass cockpits [52], [53]. In addition, other industries have conducted research to see if touch gestures made on everyday mobile devices could be used to control UAVs. *A Human-Machine Interface with Unmanned Aerial Vehicles* shows a UAV that could be controlled based off of code written for Android based phones, whereas *On the Human-Machine Interaction of Unmanned Aerial System Mission Specialists* shows how a Parrot AR.Drone can be controlled using Apple's iOS platform [54], [31].

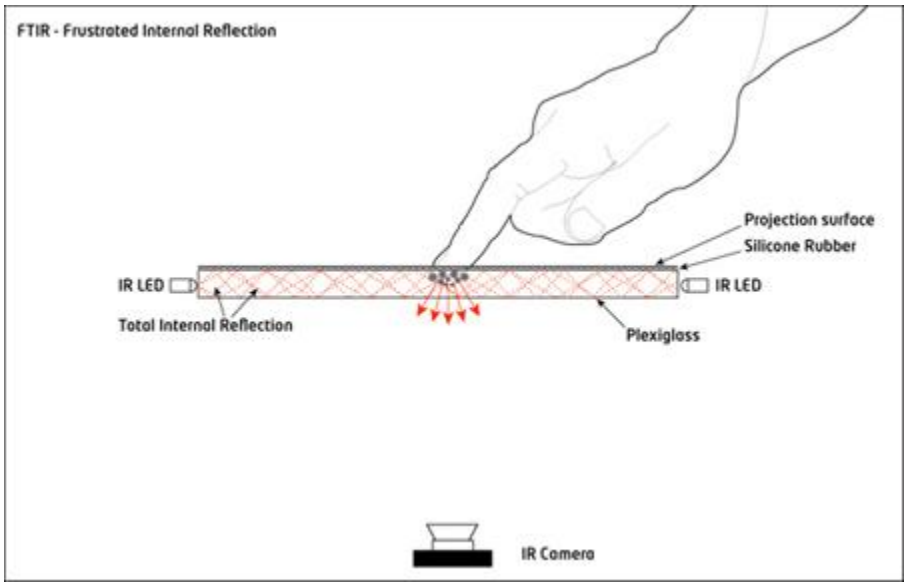

Figure 7- Frustrated Total Internal Reflection Technology [55]

There has been extensive research on how touch systems can help to improve user performance compared to mouse and keyboard devices. Most gestures can be easily committed to muscle memory. This improves operator performance because they can more easily focus on the task at hand [56]. Mouse and keyboard systems can also be easily ported over for multi-touch use with little optimization. When such systems are ported over, it has been found that the user performance was hardly altered [57]. It has also been found that touch gestures reduce the response times of users completing a variety of tasks [58]. This reduction of response time even occurs when users do not think the system is providing them with any advantages. This was found in one experiment where users were tasked with

completing tasks on Microsoft's Surface, a multi-touch capable table [51]. It was found that 75% of participants performed faster using multi-touch gestures then they did with a mouse, even though only 56% of them felt the improvement [51]. Finally it has been found that the most commonly used touch gesture is the tap, where a finger is placed onto a screen and then quickly removed [59]. However, it has also been found that taps cause the most errors amongst users because it can be easy to miss-click the wrong button/ object [60]. This issue of precision is not just limited to tap gestures. It has been found that one of the main disadvantages of touch based systems is their accuracy, which can be partially attributed to the fact that they also lack tactile feedback, since the surface texture does not change based upon the situation [61], [46]. The accuracy of touch gestures can also be attributed to the size of the object the user is attempting to interact with. The smaller the object, the harder it becomes for a touch gesture to interact with it, due to the size of a finger. The use of a mouse is more accurate because it can interact with large and small objects with ease due to their small pixel size [37]. Another disadvantage that touch systems face is that because the user is constantly interacting with object on the screen with their hands/ arms, the operator is much more prone to arm fatigue [46], [51], [61]. Despite these disadvantages, multi-touch systems are becoming increasingly popular methods for interacting with machines.

1.3 Human Factors

Human factors are used to define human capabilities, limitations, and people's behaviours when completing a task [62]. This information is then used to enhance system safety, performance and operator well-being by designing the system around the average limitations of people [62]. Most modern day aviation human factor research and knowledge comes from research conducted during the Second World War [63]. This research was conducted by both the Axis and Allied forces in order to help improve pilot performance while subsequently reducing the number of accidental crashes [63]. Since the end of the Second World War, human factor analysis has determined that training regimes that consider each person's abilities separately tend to improve the effectiveness of individual training results [64]. This is because humans are all different and thus, the measure of any individual's characteristics differs [63].

In terms of accident rates, it has been found that between 70-80% of all manned aviation accidents were caused from some combination of human factors, while, as previously mentioned in

Section 1.1, 69% of all UAV accidents were caused from human factors [65], [4]. One such reason for these high accidental human factor rates, specifically in the case of UAVs, is that humans are very poor long term monitors [66] and the common length of most military UAV missions is 12 hours [4], [12]. Thus it is common for UAV operators to experience fatigue during mission operations, which can lead to more mistakes.

The overall mission complexity can also have a negative effect on the operator. This is because mission complexity can affect the operator's mental workload and this workload can have an adverse effect on Situational Awareness, which finally can affect operator performance [67]. All of these potential sources for stress can ultimately lead to a general degradation of way of life for the exposed operator [68]. Thus improving the conditions and human factors effects are a primary goal of any designed interface. Of all the types of human factors, this thesis will specifically focus on the reduction of operator workload and the factors that have been found to directly affect it. Workload was specifically chosen as the key human factor to be analysed because it, and overall performance (which both influences, and is influenced by, workload), are the two key factors that engineers commonly try to improve upon with new Ground Control Station designs [12].

Workload can be described as the mental and physical cost that a human operator feels subjected to while trying to obtain a particular level of performance [69]. It is an important factor to consider in the design of a UI because even though a task/ mission might appear to be simple to complete the operator may begin to behave like they are overloaded, due to an accumulation of coupled human factors [69]. Since the effects of workload are subjective to each individual it makes it very difficult to look at it as an individual factor. Instead, one must look at a multitude of factors that ultimately can be coupled together to get an idea of the perceived workload response that an operator experiences.

The imposed workload that an operator feels is directly related to the task variables that they are trying to accomplish, which are in turn are impacted by the operator's perception of the task they are completing, their behaviour, and their overall performance (these interactions are all displayed in Figure 8) [69]. The task variables that typically affect the workload of an operator are the objective (the goals and criteria that need to be met for a successful mission), the temporal structure (the time based pressures that can arise from complex procedures and long duration missions), system resources (the equipment, information and support personnel that the operator is working with), and environmental factors (both the social and physical interactions the operator is exposed to over the course of a

mission). The effects of these factors can be increased by any unexpected system failures, operator errors or changes to the operator's state of mind or environment.

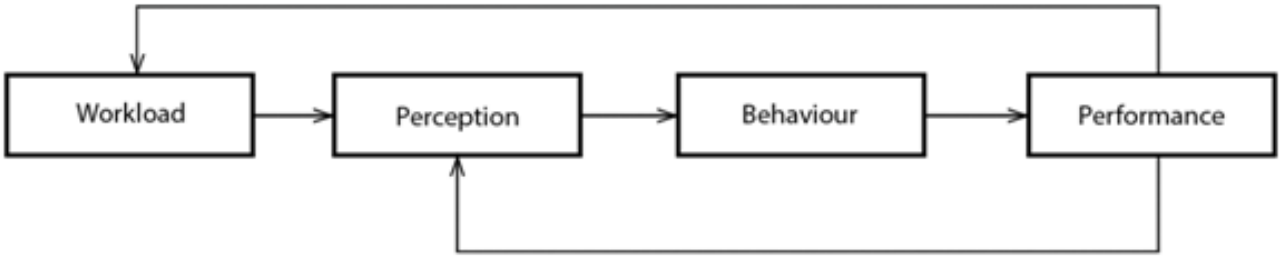

Figure 8 - Workload Effects

As a mission progresses, these task variables begin to affect other factors which in turn begins to affect the operator's workload. The first of these supplemental factors is the operator's perception. As the imposed workload increases, the operator's perception of the mission goals and procedure may change. This can be caused from preconceived bias over the structure of the mission, the environment over which they are flying or their level of experience [69]. This change in perception can then lead to changes in operator behaviour ranging from changes in an operator's mental and physical capabilities as well as how they go about breaking down tasks that need to be completed. This can then impact the operator's overall performance by overloading (or relieving) the operator's sensory and motor functions resulting in changes in response times, accuracy and reliability. Finally the changes in performance can lead to changes in operator perception as well as imposed workload. Thus when any of these factors are influenced by another, the perceived workload that the operator experiences will also be affected.

1.4 Ryerson's Mixed-Reality Immersive Motion Simulation Laboratory

The research presented within this thesis was conducted solely within Ryerson's Mixed-Reality Immersive Motion Simulation (MIMS) laboratory (Contact information for the MIMS laboratory can be found on the website: http://www.ryerson.ca/~j3chung or in the MIMS laboratory video: https://www.youtube.com/watch?v=Yq2XTn0M144) [70], [71]. This laboratory is used for a variety of research purposes ranging from multi-disciplinary design optimization to aircraft flight simulation.

The laboratory has access to three main simulation stations and a variety of unique and innovative technologies that can be harnessed as computer input methods. The first of the three simulators is Ryerson's Fixed Based Simulator, shown in Figure 9, which is used to simulate the cockpit of commercial airliners. The simulator has access to a yoke (on the left pilot's side) and a joystick (on the right pilot's side), a throttle quadrant in the middle of the two pilots and a pair of moveable rudder pedals. The simulator also has a total of nine monitors, three of which are 42inch displays and the remaining six are 22inch multi-touch displays. The non-touch displays are used to show the outside simulation environment whereas the multi-touch displays are used to show the aircraft's various instrument panels (which can also be interacted with through touch gestures).

The second simulator is Ryerson's Full Motion Simulator which is built by MaxFlight [72]. This simulator can rotate 360 degrees on two-axes, pitch and roll, and can be used to simulate the motion of an aircraft. Two users can use this simulator at any time, and are harnessed in the main central pod which rotates. Inside the pod there are two joysticks and throttles (one for each user), a button to switch who is in command, and finally an emergency stop button. This simulator displays all visuals within the cockpit via a rear-projector screen that can, when selected, project images in stereoscopic 3D utilizing 3D glasses from NVidia. Currently the simulator currently can only run Microsoft's Flight Simulator X, however other programs can be loaded onto the onboard and external computers.

The third simulator station within the MIMS lab is the UAV Ground Control Station, which was used for designing the proposed multi-touch capable, reconfigurable Ground Control Station. This station has access to a joystick, throttle and pair of rudder pedals for aircraft control, though the rudder pedals were not used for this Ground Control Station due to sensitivity issues within the simulation program. Instead, the rudder pedal functions were moved over to the rotational Z-axis on the joystick where better control could be given. Finally this simulator station has access to two touch screen monitors; the first is a 46inch single touch monitor, and the second is a smaller 24inch multi-touch monitor capable of recognizing up to ten touch points.

The MIMS lab also has access to a pair of Oculus Rift Virtual Reality goggles, as well as a head tracking sensor. Both of these devices were not used for this specific thesis; however the simulation program that was used, Presagis' VAPS XT and FlightSIM, can be setup to take advantage of Oculus.

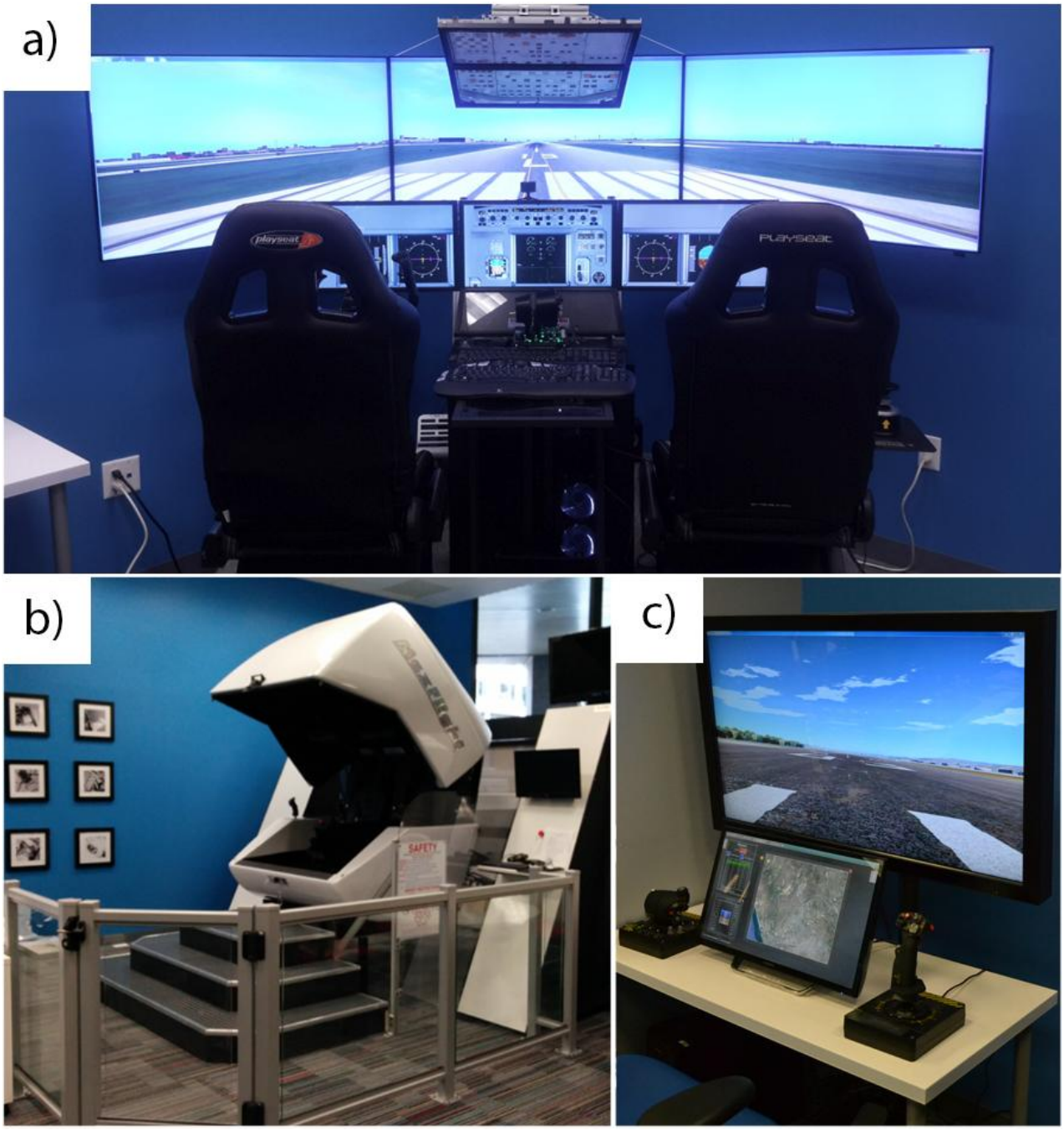

Figure 9 - MIMS Lab Simulators: a) Fixed Based Simulator, b) Full Motion Simulation, and c) UAV Ground Control Station [70]

1.5 Thesis Overview

This thesis will present the design of a reconfigurable, multi-touch capable, ground control station. The Ground Control Station UI used multi-touch gestures because they are a new and contemporary method of interacting with computers that can provide inherent advantages over more traditional mouse and keyboard systems. The Ground Control Station included six key features to help improve user effectiveness and the way they interact with the interface. In addition the system includes 12 instrument panels to showcase how one could utilize multi-touch gestures in order to control a UAV. Finally all of the included Ground Control Station instrument panels can also be repositioned and/or rescaled by the operator to create a customized UI setup that best suits their needs.

The design considerations that were used in order to finalize Ground Control Station UI are outlined and described in Section 2. Section 3 presents the software that was used to design the Ground Control Station UI as well as how they interacted with one another. Section 4 explains the design process and all of the various iterations that the Ground Control Station UI went through until it reached the final design. Section 5 describes the final Ground Control Station UI in terms of all of its features and instrument panels and includes a brief flight tutorial utilizing the features and instruments. Finally Section 6 summarizes completed system, the contributions of the completed research, along with a discussion of potential future work.

2.0 Design Considerations

Before the Ground Control Station UI could be designed, a number of considerations had to be evaluated to ensure the system was designed efficiently and could be easily interacted with. The first are Ground Control Station considerations which looked at the various instruments and functions that should be included in the final design. The second are human factor considerations, which looked at the human impact on the system. The final set of design considerations were with respect to the UI, which took into account the overall layout and design of the interface with which the user interacts.

All of the considerations that are described in the following section were found through the review of extensive literature research and by looking at a variety of open source Ground Control Station designs. The free open source programs that were looked at included: QGroundControl's open source Micro Air Vehicle Ground Control Station and Michael Oborne's open source Mission Planner [73], [74].

2.1 Ground Control Station Considerations

Without looking at the human factor or UI considerations, there are number of Ground Control Station specific elements that have to be looked at to create an effective Ground Control Station. These considerations include a number of common instruments and functions that should be available to operators who are controlling an unmanned vehicle. There were also a number of common issues with current systems that should be improved upon if possible, all of which are described below.

One of the biggest hurdles to overcome when one designs a Ground Control Station system is that the system can be used for prolonged periods of time. As a result, operators can experience high levels of fatigue, burnout and boredom while utilizing the system [1]. These symptoms can lead to a decrease in performance which in turn can lead to accident. Thus it is of the utmost importance that operators stay alert and attentive while utilizing the Ground Control Station system. If they are alert and attentive, they are also less prone to information overload especially when receiving vast quantities of information from the UAV pertaining to its status [23]. With this said, it has been found that receiving too much information at any given time can be equally as bad as receiving too little [63]. This can be caused from the user has to search through large amounts of information leading to a reduction in performance on time sensitive tasks [63]. Thus it is important to present the most important information in the portions of the screen that are most looked at. It has been found that experienced

UAV operators tend to look at the center of the screen the most while only diverting their attention to other locations briefly (such as when they needed to see a HUD element) [75]. Thus it is important to present all critical information in the center of the screen or in the center of a specific instrument panel.

In terms of required instruments, the Ground Control Station operator should have access to the following six functions [1]. Firstly they must have an instrument that shows the current aircraft's location. Knowing the location of the UAV is necessary because the pilot is not onboard the aircraft and thus cannot use more traditional visual cues to navigate the aircraft throughout the course of its mission. Next, the operator must be provided with the aircraft's flight path [1], [76]. This gives the operator an idea of where the pilot has flown the aircraft which is important for surveillance and Search and Rescue missions. A visual flight path also provides the operator with information on where the aircraft will be flying in the future so that they can update the flight plan with any new weather information or changes to the mission. The flight path also helps to indirectly show the operator the heading of the aircraft, its current position and a rough idea of its velocity [76]. The third and fourth requirements are that both the aircraft and physical Ground Control Station must have methods to show the status of the various onboard systems. This information is critical so that the operator can determine whether or not the equipment is functioning properly. As was described in *Ground Control Station Development for Autonomous UAV,* the most vital equipment statuses are for the aircraft's engine, camera, ailerons, elevators, and other control systems [76]. In addition to the status of direct subsystems, the Ground Control Station also needs to show the operator a number of key parameters such as, the aircraft's fuel load and remaining fuel, whether or not the landing gears are extended or retracted, as well as the aircraft's flap settings [12]. Next a display of the time lag between the Ground Control Station output and the UAV receiving the command is necessary [1]. This is required so that the pilot can know if they need to compensate their control methods to pilot the aircraft properly and safely. The final required function for Ground Control Station is a display of the strength of communication to/ from the UAV [1]. This is critical because the operator needs to be aware if the vehicle they are piloting is within line of sight / distance of receiving commands as well as if the aircraft is reaching the edge of its operational area.

Additionally there should be options for the operator to add waypoints at any time during the course of the mission [76]. This is important because tasks could dynamically change over the course of a mission, especially when there are a number of uncertain variables (i.e. the location of a High Value Target). As well the system should be easily adaptable to the overall UI in order to help show the user

what they wish to see [77]. This can help to increase a user's response time and performance because and reduce the time spent searching through information to find what they need. Finally any sensors that can be directly manipulated (i.e. a camera) should display their geographic orientation with respect to either the outside environment or the aircraft itself [77]. This ultimately helps to improve user Situational Awareness because they will have a better idea of how the system is oriented.

There was also a variety of common Ground Control Station issues that were listed in *Human Factors in Aviation* which helped provide insight into areas that could be improved upon [1]. The first of these problems was that contain colour choices within a number of Ground Control Station UI's generated situations where text was difficult to read. This is clearly a situation that should be avoided as it can lead to pilot errors if a display is misread it can also lead to eye fatigue if the colour combinations clash. The second is that more often than not, too much information is displayed as text as opposed to a visual graphic. Graphical representations of data make it very easy to see what is happening to the system and does not force the user to have to search through large areas of text for the information that they are seeking. Thirdly, a majority of Ground Control Station systems place critical controls next to non-critical ones. This can potentially lead to pilots misusing controls, which in turn, can lead to aircraft crashes or other critical problem arising. The fourth common problem is that a number of systems have un-intuitive automation processes, which can lead to unnecessary complexity for the operator, especially since automation is used to help reduce pilot workload and not increase it. The next two common obstacles are the emphasis on MFDs and controls, and hierarchical menu trees. Both of these methods of input and display typically provide unnecessary extra actions that the pilot must complete in order to achieve a goal, but are currently required due to limited hardware space. Ultimately this can add stress in high workload situations and can also lead to confusion and errors when the operator has to navigate through a series of menus to access the information they need. Another potential complication is that systems do not provide adequate feedback to pilots and crew members on the status of system settings and states. This can lead to the user mistaking a system for being activated/ deactivated unintentionally which can cause other more major problems to occur. A final common problem is that crew members tend to be overloaded with raw data. This problem can be a consequence of a number of the previously mentioned issues because when complexity is added and data is not presented in a concise or easily read way, the pilot can become overloaded and may start to experience a decrease in workplace performance.

2.2 Human Factor Considerations

The human factor considerations that are made are used to help gain an understanding of the typical types of errors that an operator might experience. Thus, if one understands the standard causes, they can create a system that ties to reduce the possibilities of a recurrence.

According to *Aviation Psychology and Human Factors* all pilot errors fall under one of six categories [63].

1. Substitution errors, which is when the operator mixes up one control with another.
2. Adjustment errors, where the operator uses the control wrong by either setting it to the wrong position or by moving it too slowly/ quickly.
3. Memory errors, where the control is used at the wrong point in time.
4. Reversal errors, where the control is moved in the opposite direction than is required.
5. Unintentional activation of a system
6. Inability to reach the control, whether that be physically or if they have to divert their attention away from other critical tasks for too long of a period.

It was also found that out of the six potential categories, substitution errors accounted for more than 50% of all reported pilot errors. From this it can be seen that from a design perspective it is necessary for all controls to be labelled to help reduce the potential for substitution errors and to provide methods to help reduce the other potential types too.

As described in *Aviation Psychology and Human Factors* having access to too much information can be just as bad for an operator as having too little [63]. This is caused from having to search through large quantities of information to find what the user is looking for which can lead to poor performance, especially on time sensitive tasks. Similarly, it has been found that people are more likely to remember the first and last pieces of information in a list [63]. This is important to keep in mind when displaying large amounts of information as some of it will be mission critical and some will not. When it comes to memory, it has been found that the average person can remember an average of seven digits (+/- 2) [63]. This is of particular importance for any system inputs as you want the operator too accurately and consistently be able to input the correct information. Thus no system inputs should require the user to input more than seven digits.

Ultimately it has been found that implementing some form of automation into an unmanned system can help to reduce operator workload [78], [79]. Which can be described from the fact that as one increases the amount of multi-tasking that needs to be completed, one runs the risk of degrading the overall task performance [68]. This reduction of performance can then lead an increased chance of total mission failure. A user's overall performance can degrade as they multi-task more because one's mental workload is sensitive to information overload [67]. Thus as one takes on more tasks they will be presented with increasing amounts of information relating to a variety of tasks, which can lead to an overload taking place and thus an increased overall workload.

2.3 User Interface Considerations

The first set of UI design considerations comes from *Microsoft's User Experience Guidelines* for both Windows 7 and Vista [80]. The document itself was created to help Application designers come up with effective interfaces that can be interacted with a mouse and keyboard or through touch gestures. Ultimately the guidelines that are provided by Microsoft give insight into good general design principals as well as methods, control considerations and general text/ error wording and design.

When it comes to generating a new UI for a system, Microsoft believes that a UI should reduce the number of questions that the operator is asked [80]. This helps to increase automation and decreases the amount of interruptions the user experiences while managing the system. Another key UI design consideration is that more often than not, personalization is more important than actual customization. The difference between personalization and customization is that customization tends to deal with changing the way the controls are interacted with, for example changing a knob to a slider. Whereas personalization deals with changing the control input hotkeys and the visual layout. An effective UI should be responsive and simple. This helps to improve the user experience by reducing the time it takes to witness a change on screen which improves efficiency and user response times. Finally keeping the design simple makes it less likely that the user will become confused over the various functions or get caught up in endless menu trees.

The biggest guideline for control schemes is that all controls should be labeled [78], [80]. This helps to identify what can be interacted with. Labelling controls also allows the system to inadvertently tell the user what the control will do without having to give them another window prompt to explain it. This helps to clean up the overall UI and can reduce the number of windows that will take up valuable

screen space. The other major control consideration is that there should always be the ability to cancel an action that that has been inputted by the user. This is used to help avoid accidental control inputs which could be detrimental to a flying system and ultimately help to reduce the number of critical errors the operators will experience.

Text is another key area for overall UI design. According to Microsoft's guidelines all UIs should use ordinary terms whenever possible while also trying to reduce any redundant text [80]. Using ordinary terms helps to increase the usability of the product by using words that both a beginner and more experienced operator can understand. It also helps to increase the simplicity of the system by avoiding potential confusion over words or meanings. The reduction of redundant text is typically used to help decrease the amount of clutter and wasted space on an already limited screen. It also helps to increase simplicity as a whole by displaying what might need to be said in a more concise manner.

The final set of UI considerations that Microsoft recommends has to do with error messages. Whenever possible it is recommended that a solution is proposed to reduce potential stress increases when a problem occurs [80]. Providing a potential solution also gives the operator a starting point to work from when trying to narrow down a working solution, thus reducing their potential reaction time. Error messages should also not be written in a vague manner so as used to avoid confusion over what is the actual problem. By being straight to the point, an error message can provide the user a direct response to what the issue is and how to solve it without being misled. Additionally any displayed error message should not blame the user for the problem [80]. Whether or not the user is to blame for problem or not, not assigning blame can help keep the operator calm and on task. Looking beyond Microsoft's guidelines for errors, it has also been found that most errors should prompt the user in order to help persuade them to actively search for a solution to the issue, while also providing an audible beeping noise to help notify the user in busy situations [76] , [78].

In addition to all of Microsoft's guidelines there are a number of other considerations that have to be made when designing an UI. First and foremost the designer has to consider the number of machine inputs that will be available to the operator. If all the inputs are located in an unorganized manner or all clumped together on the screen the user may become overloaded [78]. Thus it is up to the designer to ensure that all inputs are grouped in such a way that all like objects/ instruments are group together to help limit the chances of overloading the operator [25]. It is equally important to display all

information in real-time so that the operator can react in time and not cause the unmanned vehicle to crash [25].

The final UI should also be as user friendly and easy to use as possible [78]. This helps to limit the chances of the operator getting confused over the functionality of the system. The Ground Control Station UI ultimately acts as the hub for all data transmission and interaction [77]. Thus it should be designed in an ergonomic manner to help promote efficient performance. This can be a difficult task as an operator may be faced with overwhelming amounts of data during a mission [81]. This data could come in the form of crew voice communications, system data, and tactical data.

Finally when it comes to the actual design of specific touch gesture interactions the designer must ensure that the interaction areas are a suitable size. The actual size of touchable objects is one of the most critical factors when designing gesture interactions for mobile or computer devices [82]. It has been found that all touch objects should be larger than 11.5mm in width to promote accurate commands inputs [46]. In addition it was found that as long as objects are larger than 32 pixels in width, the number of selection based errors will be reduced by roughly 66% [37]. Lastly, when it comes to the design of touch areas, one has to remember that the fingers used to complete a gesture can also hide important information from the user [46]. Thus one needs to ensure that the area directly under a touchable object does not show the most important flight parameters.

3.0 Flight Integration

To design the reconfigurable Ground Control Station two software programs were used. Both were designed by Presagis, a company based in Montreal, Canada. Presagis is a subsidiary of CAE and develops a number of Commercial-Off-The-Shelf software products that are all focused around Modeling and Simulation for use training purposes and the design of embedded Human Machine Interface displays for both automotive and aerospace systems. Their products are used by over 1000 companies across the globe, including Boeing, Lockheed Martin, Airbus, BAE Systems, and CAE [83].

Presagis' software programs are divided into one of two categories. The first is their Modeling and Simulation Suite of programs that can create and execute various simulation environments. The second category is for their Embedded Graphics software which includes programs that are used for the design and integration of extensive Human Machine Interfaces. A brief description of all of their programs can be found in Table 1.

The designed Ground Control Station that is described within this Thesis was created using FlightSIM (from Presagis' Modeling and Simulation Suite) and Presagis' VAPS XT (from their Embedded Graphics Programs). FlightSIM was used to simulate an aircraft that would be controlled by a Human Machine Interface that was designed within VAPS XT. It should be noted that all of the programs listed in Table 1 can be integrated together to create a higher fidelity simulation environment, however this comes at the added expense of computation loads and integrated system complexity. The exact program integration can be seen in Figure 10, which shows what each program outputs to the other programs.

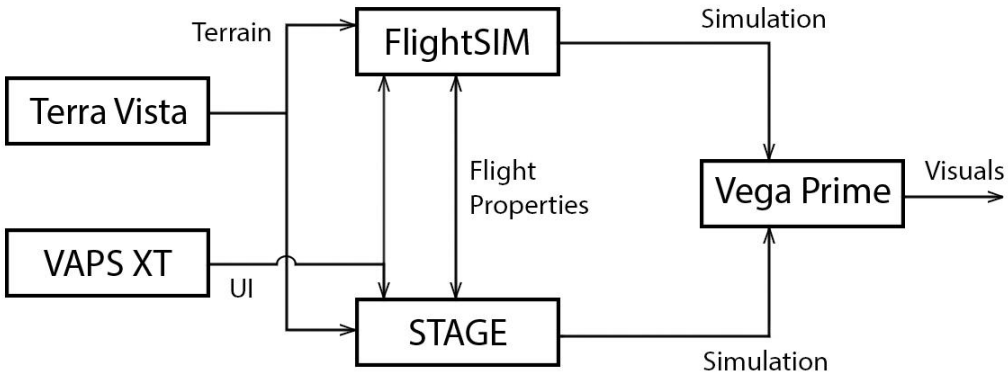

Figure 10- Presagis Program Integration

Program Title	Description of Program
Modelling and Simulation Suite	
Creator	A high quality polygon model generator that allows the user to design building and vehicle models for use in simulations.
FlightSIM/ HeliSIM	A high fidelity flight simulation program that can be used to test aircraft flight dynamics. FlightSIM focuses on fixed wing aircraft whereas HeliSIM focuses on rotary wing aircraft.
TerraVista	Creates randomly generated terrain backgrounds based off of Common Database (.CDB) files. These backgrounds can span from accurately drawn cities to vast forests with random assortments of foliage.
Stage	A high fidelity simulation environment that allows the user to input multiple models that can be user controlled or AI controlled. Can be used to incorporate air, land, and/or sea elements into the same simulation
Vega Prime	A high quality visualization toolkit. It can be used to set up the visual environment for FlightSIM/HeliSIM and Stage. The user can either program the visuals using APIs or through a simple to use UI.
Embedded Graphics	
VAPS XT	Allows the user to design high quality Human Machine Interfaces that can be linked to FlightSIM/HeliSIM or Stage. These interfaces can be directly programmed directly with APIs or through VAPS' UI.

Table 1- List of Presagis Software Programs

3.1 FlightSIM

Presagis developed FlightSIM to act as a standalone simulation environment for fixed wing aircraft. FlightSIM is composed of two components that together create the overall simulation. The first part is an aircraft modeler that allows the user to define as many flight parameters as they have available to them in order to generate a flight model for the aircraft. The fidelity that FlightSIM can

provide is dependent on how well the designer defines the aircraft within the program's modeler (i.e. the more detail they can provide on the system, the more accurate the simulation will be). The second component is the run-time simulation. This component is what is used to define the simulation environment and any dynamic tests the user may wish to conduct.

3.1.1 FlightSIM Aircraft Modeler

FlightSIM's aircraft modeler (the UI of which is shown in Figure 11) allows the user to define in extensive detail the flight dynamics of a fixed wing aircraft. The modeler is separated into three components. The first is the aircraft's flight model which covers all of the aerodynamic surfaces, control laws, and the weight and balance of the aircraft. The second component is the aircraft's engine which is used to define all of the parameters for the aircraft's power plant. The modeler can be used to define a turboprop, turbofan, piston, or performance engine (i.e. a custom engine such as an electric motor). The final component allows the modeler to define any payloads that might be attached or stowed within the aircraft.

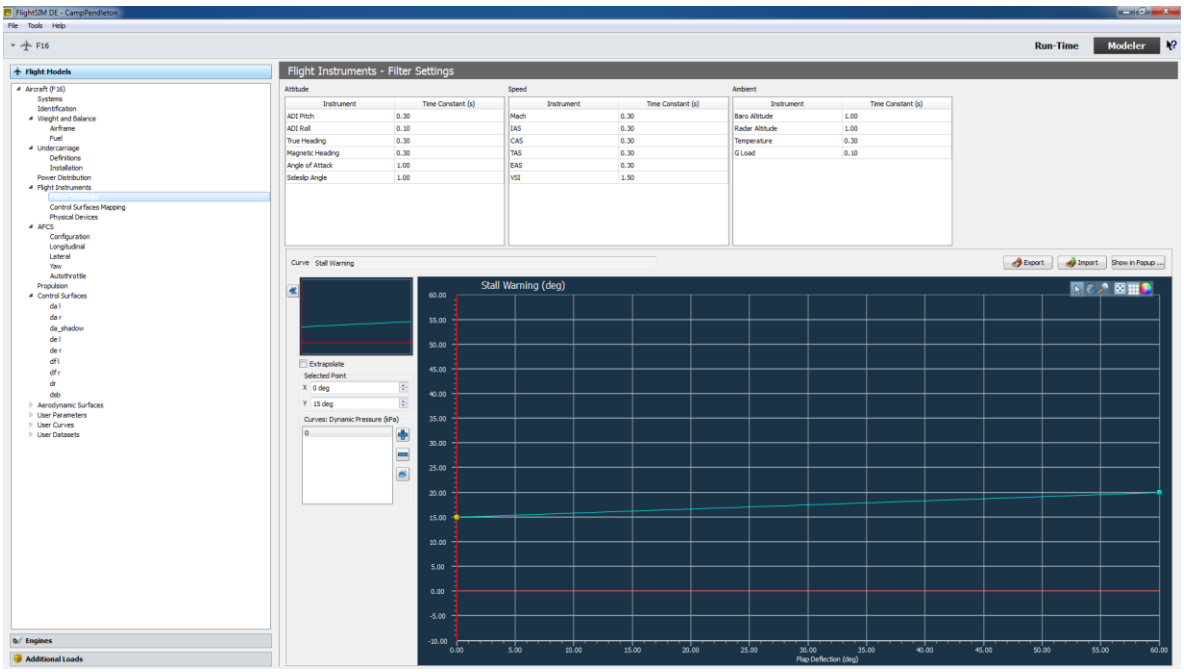

Figure 11- FlightSIM Modeler User Interface

The aircraft's flight model component allows the user to define how the aircraft's flight model is defined. The user can first define the aircraft's weight and balance, including the airframe's center of

mass and moment of inertia as well as the location of the fuel tanks and how a fuel tank's moment of inertia may change depending on the fuel level. The user can also define how all of the various subsystems that are found onboard the aircraft are powered (i.e. is the Attitude Direction Indicator powered by Electrical Bus A/B or is a brake controlled by a hydraulic system, electrical bus, or is it mechanically/ manually controlled). Next the user must define all of the control laws/ parameters for the various flight instruments, control surface and autopilot systems. These are defined by a mix of graphs and control laws that can be imported from MATLAB's Simulink or designed using the modeler's UI. The user must then define the aircraft's propulsive power plants wing locations, while also selecting their profile (which are created within the engine setup). Finally, the user must define any extra parameters that are of use to the flight model, such as lift and drag coefficients. These parameters can be described in terms of defined equations if they are dependent on other parameters or by a graphical plot.

In addition to the flight model, the user can extensively define the characteristics of an engine. The user must first select which type of engine they would like to model (turboprop, turbofan, piston, or performance engine). From this point the user can define the parameters that relate to each individual engine component. For example, if a turbofan is used, the user would start by defining the turbine's starter unit which requires a maximum starting time, disengage Rotations-Per-Minute (RPM) and a starting power graphical plot. They would then move onto the inlet which requires the user to input a graph showing how the airflow is distorted and the compression ratio of the inlet. Next they would define the fuel flow control law through either a MATLAB Simulink code, or through a control law that was defined within FlightSIM's Modeler. The user is required to go component by component through the jet core to define the various component efficiencies, pressure ratios, specific heats, corrected airflows, etc. The user must also define the thermodynamic properties of the cooling oil. This includes the oil's pump flow rates, specific heat values, temperature lags and pressure thresholds. Finally within a turbofan, the user would have to define the control law that controls the shape of the exit nozzle using MATLAB's Simulink or FlightSIM's Modeler UI. After the engine is defined, the user can save the entire profile so that it can be selected in the aircraft's flight model.

The final set of parameters that can be defined within the aircraft modeler are the additional loads. These additional loads represent any payloads that could be mounted to a wing hard point or fuselage undercarriage (i.e. an external fuel tanks, weapon payloads, cameras, etc.) or cargo that is stored within the fuselage. These loads can be defined by mass, surface area, lift and drag coefficients (if

they are attached to the exterior of the aircraft) and the item's moment of inertia. Additionally these loads have to be positioned on the aircraft, and can have a load definition profile attached to them to show how the load will affect the aircraft's flight performance either when released or when it shifts within the body of the aircraft.

FlightSIM does not require the user to define an aircraft before the simulation can be run. Out of the box FlightSIM comes with 17 predefined aircraft. These aircraft range from being incredibly accurate to being minimally defined. This allows users to gain an idea of how much information is needed to create their own flight models. In terms of model accuracy, the highest fidelity model is the Boeing 747-100 which is defined by roughly 1400 variables. Comparatively, the simplest flight model, a fictional microUAV, roughly has 300 defined variables. One can see that while 300 variables is quite substantial, to generate a highly accurate flight model of a larger aircraft one would need exponentially more information.

For this thesis a predefined model of a Lockheed Martin's (formerly General Dynamics) F-16 Fighting Falcon was used. The Falcon was chosen because it contained the most extensive autopilot system of all of the predefined aircraft. A well-defined autopilot was important for the simulation because operators tend to set multiple waypoints for a UAV to follow instead of flying them manually all the time, as such, the most accurate autopilot model was needed. The specifications of the actual F-16 Fighting Falcon can be found in Table 2.

Length	15.03 m
Height	5.09 m
Wingspan	10.0 m
Wing Area	27.87 m^2
Empty Weight	9207.9 kg
Maximum Take Off Weight	21772 kg
Internal Fuel Capacity	2685.2 kg
Speed	1500 mph
Range	1740 n. mi
Power Plant	One General Electric F110-GE-100/129
Engine Thrust	29100 lb

Table 2- F-16 Fighting Falcon Specifications [84]

3.1.2 FlightSIM Run-Time Setup

The run-time component allows the user to setup all of the parameters required to run the FlightSIM simulation. First and foremost the user has to define where the simulation is taking place. To define the location three pieces of information must be given to FlightSIM. The first is the actual database file that defines the simulation's environmental visuals and has to be inputted in the project settings. FlightSIM initially comes with two different databases, with a third one that can be downloaded online. The first is a plain flat environment that is comprised of a basic green ground tile and sky box with a singular airstrip that represents no physical location on earth. The second location is a small portion of California's Camp Pendleton, which features building models for Camp Pendleton itself as well as the surrounding city of Bonsall, and a small portion of the Pacific Ocean coastline. The third map, which can be downloaded, is a high quality map of Yeman that is comprised of a portion the country's coastline and cityscape. For the Reconfigurable Ground Control Station that is described within this Thesis, the database of Camp Pendleton was used as it is the standard map that comes already setup with the system. After the database is defined, the user must set the geographic center of the map in terms of latitude and longitude (for Camp Pendleton, this was set as N33:17:18.585, W117:23:07.440). These parameters can be found under the "Setup" Tab within the "Gaming Area" section within the sub tab called "Map". The final parameter that has to be defined for the gaming space is how the Earth is defined. This can be found under the "Gaming Area" section under the sub section named "Earth". FlightSIM requires the user to input the Ellipsoid Model of the Earth, the Equatorial and Polar radii, and the rotational speed of the Earth. The standard setup for the Camp Pendleton area used a World Geodetic System 84 Ellipsoidal Model to define the Earth. This model is an Earth-Centered-Earth-Fixed reference model where the X-Axis passes through the 0° Latitude and Longitude Point on the earth, the Z-axis being aligned along the northern polar axis, and the Y-axis being perpendicular to both the X and Y axes [85]. In addition, the Equatorial radius is assumed to be 6378.137km, with a polar radius of 6356.752km, and with the Earth rotating at 0.004717807deg/s.

After the simulation environmental area has been defined the user needs to setup the starting conditions of the aircraft (Figure 12 shows the UI page to define the initial conditions, namely the aircraft's position). This is accomplished with the Setup Tab under the sub category of "Initial Conditions". These conditions include the position of the aircraft in terms of latitude and longitude, if the aircraft is starting on the ground or in the air (and if it is in the air, at what altitude it is starting), the heading (in degrees) of the aircraft, and the aircraft's speed (in knots). For the sake of the simulation, it was assumed that the aircraft would start on the ground on Camp Pendleton's main runway which is at

a position of N33:17:43.720, W117:21:42.500 at a heading of 44.1°. In addition to these parameters, the user can also set the aircraft's fuel levels (in terms of a percentage), if any engines are disabled, any NAVAID frequencies, if the Autopilot is setup and any runway conditions (i.e. is the ground dry, wet, snowy, etc.).

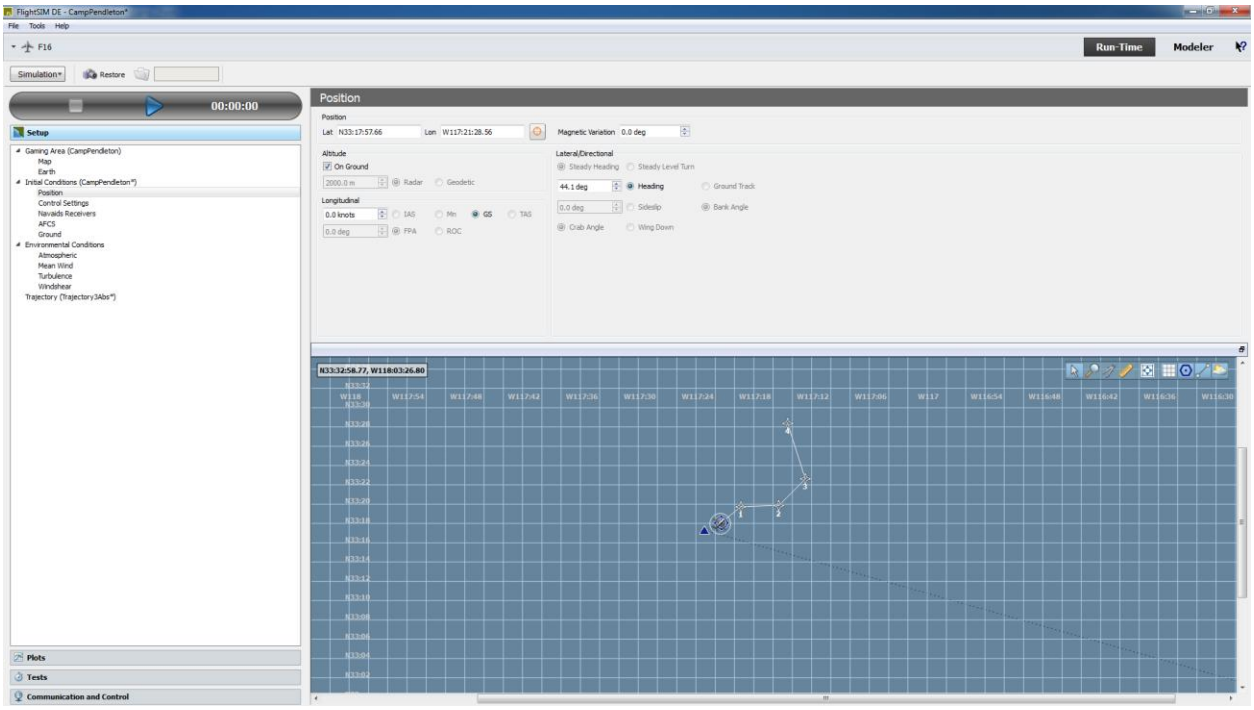

Figure 12- Screenshot of FlightSIM's User Interface

Next the user can define any environmental conditions that might be present within the simulation world. These conditions range can be atmospheric conditions (i.e. how temperature and pressure change based upon altitude), any wind that might be present (i.e. the horizontal and vertical wind speeds and the horizontal wind direction – all of which was assumed zero for the simulation), any turbulence models (i.e. Anomalies, Von Karman, or Dryden models – in this simulation none were taken into account) and finally any wind shear that the aircraft might experience (i.e. the wind shear model, radius of the downdraft, the location of the downdraft and the maximum outflow of the draft – in the simulation no wind shear was present).

The user can setup any predefined waypoints within the "Trajectory" sub section of the "Gaming" tab. When setting up these waypoints the user has a choice between absolute and relative waypoints. Absolute waypoints are waypoints that have pre-defined specific locations (in terms of latitude and longitude) whereas relative waypoints are relative to the aircraft's position when the

waypoint becomes active (in terms of distance from the aircraft and angular bearing). As waypoints are added, they begin to appear on a small map that is shown in the lower portion of the screen to show the user how they will be setup. Finally after the user has inputted all of the waypoints they want to complete, they can save the trajectory so that it can be imported by the aircraft's Flight Management System and used again.

In addition to setting up the aircraft's initial conditions the user can set up a variety of plots and dynamic tests within the run-time component. These plots and tests are used to gather flight test data to see if an aircraft simulation is performing as desired (in the case of a pre-existing aircraft) or to test how an aircraft will fly if it has not entered the manufacturing stage. The tests can also be used to test out the various components of an aircraft in the design stage (i.e. by telling the simulation what the aircraft's thrust production will be if the engine parameters have yet to be finalized) or to re-simulate potential aircraft accidents. Each of these tests can be saved to FlightSIM so that the user can easily retest the aircraft without having to spend time setting it all back up.

To being able to run tests, simulation results can also be saved. This allows the user to revisit and evaluate the flight dynamics of the aircraft to see where something may have gone wrong within the test. The only potential downside to saving the tests is that the user has to specify how long the program will record. As such, the user has to determine how long the aircraft will be flying or if there is a point in time at which they no longer need to record the flight. The user can however speed up the simulation so that a longer flight can be achieved. The final feature that the playback allows is that the user can drop back into the replay at any time and take direct control of the aircraft. This is particularly useful for both training and crash analysis. In training the user could set up a flight path and practise specific portions of the flight using identical environmental conditions (i.e. for practising a landing). For crash analysis the user could have other people take control of an aircraft as the accident is occurring and try and see what methods would have prevented the accidental outcome. Ultimately FlightSIM's recording tool is a very powerful feature that is immensely useful for simulation purposes.

Finally, the user can determine all of the various communication and control configurations with the run-time component. The communication configurations are used to send and receive data between other programs (such as Presagis' VAPS XT). The communication setup is explained in more detail in Section 3.3 as the process involves setting up a connection to VAPS XT. However within this section it was noted the user can also setup any pilot inputs that might come from an external source. All of the various control inputs that FlightSIM can use can be given commands from a variety of different

locations. These can be from FlightSIM itself (typically in the form of keyboard hotkeys), to VAPS XT (in the form of various slider or button commands that are sent through a communication protocol) to external third party hardware devices like a throttle or joystick. If the user wishes to use another hardware device such as a throttle or joystick they have to modify FlightSIM's "PCJOYSTICK_MULTIPLE" command file to include the hardware device's number, its transmitted bit number, and word number which are defined by the PC on which it is installed. These word and bit numbers can be determined by using the PC's command prompt. When a hardware input is added to the computer it is given a device number that initially starts at zero and increases by one for every device. The word number is used to tell FlightSIM which button is being pressed on the input device at any given time, like the device number, it starts at zero and increases for every input option. However, the word number is not unique and thus each new device's inputs start at word number zero and increase from that point. Finally the bit number is used to calibrate the received simulation input variables based upon the range of motion of the particular input. For example, a button would have two unique word numbers, one for when it is clicked and another for when it is un-clicked, whereas a joystick column will have a range of values that correspond to its X and Y position. For the described simulation the aircraft's lateral and longitudinal stick controls, as well as the rudder are controlled by an external PC Joystick, whereas the throttle, flaps and landing gear are controlled by VAPS XT. The actual device, word and bit numbers for the hardware inputs can be found in Table 3 and Table 4. It should be noted that within FlightSIM aircraft can be given up to four engines, however the installed throttle quadrant has only two power levers, and thus multiple engines had to be assigned to the same lever to provide even thrust (i.e. engines placed on the same pylon location on both wings were paired together). The first lever controlled engines one and three, and the second lever controlled engines two and four.

Hardware Input	**Device Number**	**Word Number**	**Max Left/ Forward**	**Middle**	**Max Right/ Backwards**
Pitch	1	1	0	31360	65535
Yaw	1	2	0	32500	65535
Rudder	1	4	0	2010	4095

Table 3 - Word and Bit Numbers for Joystick Input

Hardware Input	Device Number	Word Number	Max	Min
Break Left	0	1	0	255
Break Right	0	2	1	255
Throttle 2 Throttle 4	2	1	1023	0
Throttle 2 Throttle 4	2	2	1023	0

Table 4- Word and Bit Numbers for Rudder and Throttle

Once all of the run-time component parameters are set up the user can define what aircraft they wish to fly and hit the "Play" button in the top left corner of FlightSIM. When this button is pressed, FlightSIM opens up a connection to Vega Prime which initiates all of the graphical objects and displays the simulation using the flight data that FlightSIM provides it.

3.2 VAPS XT

VAPS XT is a software program that solely designs Human Machine Interfaces for the aerospace and automotive industries. The program allows users to design high quality visual user interfaces that can be interacted with in a variety of ways. These UI's can then be linked to other external programs, such as FlightSIM, in order to graphically represent received parameters (i.e. aircraft velocity) or to send commands to the simulation software (i.e. throttle position). VAPS XT uses a simple UI (shown in Figure 13) that displays all of the current Human Machine Interfaces in a side panel on the left hand side of the screen, all of the possible graphical entities the user can employ to the right of the project, a visual representation of the designed UI in the center, and the various object properties on the right side of the screen. Ultimately VAPS XT was used to design the entire Reconfigurable Ground Control Station UI due to its ability to be quickly linked to FlightSIM and because it is an industry program that has been used to design avionic UIs.

There are two ways to design a Human Machine Interface within VAPS XT. The first is to use APIs that are written in C+. These APIs allow the user to design very complex UIs that can be easily linked with non-Presagis programs. APIs are also useful when the various graphical objects are using very similar functions because C+ programming is object oriented and thus can take advantage of parent

functions that are coded elsewhere. The second way to design UIs within VAPS XT is to use the provided UI. VAPS XT provides a clean workspace that directly shows how the UI will look. The user can directly draw and design graphical entities within the workspace or they can drag and drop either premade or user created graphical objects onto it. The user can then code in functions to alter how objects interact with each other or other programs. Once the Human Machine Interface is finished within the workspace, the user can build the code into an executable function that can run without having to boot up VAPS XT. This allows the UI to be placed onto other computers or to be directly installed onto other pieces of hardware for us. The Reconfigurable Ground Control Station that was designed for this thesis was created using the second method, within VAPS XT's UI, because the overall object coding was not very complex.

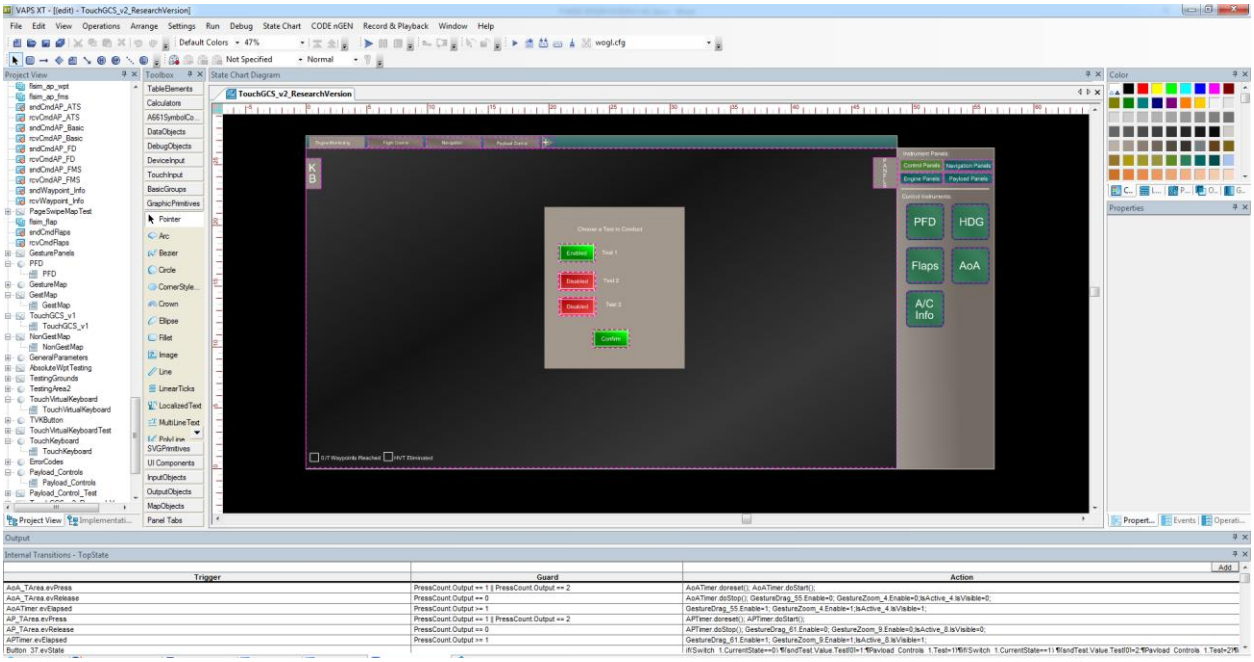

Figure 13- VAPS XT User Interface

VAPS XT allows the user to create two major types of graphical entities. The first are graphical objects which are meant to be called upon multiple times, or are objects that need to either send various property values to another graphical object or require the input of a value from another object. The second type of graphical entity is graphical formats. These are meant to be the final overall Human Machine Interface. The graphical formats cannot easily send or receive direct values from other objects or formats however they can be built into executable files for external use. As such graphical formats

tend to use multiple graphical objects to create a complex system that may experience a lot of interactions.

When it comes to designing graphical formats or objects, the user will have access to all of the same graphical and programming commands. When it comes to drawing the actual graphical entity, the user can select from a number of premade objects that can be visually altered while keeping the same functionality (i.e. the shape of a button can be changed while still keeping the original function of a button) or the user can build an objected from scratch and then program its functions individually. This gives the user unlimited potential when it comes to designing interactions with a system. The predefined objects are all categorized by function. The most common categories are Input Objects, Output Objects, and Graphical Objects (which are the basic graphical building blocks such as lines, rectangles, circles, text, etc.). In terms of input objects, buttons and sliders are the most commonly used objects. Buttons are entities that contain a defined number of states. When the button is interacted with (i.e. via a mouse click or gesture tap) the state changes. Sliders on the other hand have a defined minimum and maximum value, and use a moveable slider to linearly determine what the current output value is, based upon the slider position between the minimum and maximum values. In terms of output objects, dials are the most commonly used object. Dials take in an input value and rotate around a center point by a proportional amount, and as such they tend to be used extensively for gauges.

Each predefined object also has access to a number of properties that can be changed to alter their appearance or function. These appearance properties can range from text font, text colour, line colour, the radius of a circle, to a rectangle filled with a solid colour or a gradient and its associated colour or pattern. In terms of functionality properties the range of minimum and maximum values of dials and sliders, as well as the starting and ending angles of dials can be changed.

There are two methods of linking objects and entities together through the use of coding. The first is through the “Data Flow”, an example of which is shown in Figure 14, which is used when a value is going to be constantly receiving a changing value (i.e. a dial that shows the Angle of Attack). To create this type of interaction the user has to define what parameter/ value is being sent to another object, also known as the “Source”. This value can be further modified if it needs to be. For example, if a slider is outputting a throttle value between 0 and 1, but the text that shows the throttle percentage needs to be between 0 and 100, the throttle value can be multiplied by 100. The user then has to supply the program with the location that the source value will be sent to, known as the “Destination”. For

example, if the previously mentioned throttle value is being shown on a dial it would be sent as the Source to the dials current value (which would be listed as the Destination).

Figure 14- Example of a Data Flow

The second method is through "Internal Transitions" (Figure 15 shows an example of an Internal Transition) which are more complex interactions that are only carried out if certain conditions are met. First the user must define an initialization condition, called the "Trigger" within VAPS XT. This initiation condition could be anything from a timer cycling through a defined period, to a system detecting that the state of a button has changed. Next the user has to state any conditions that must be met for the interaction to occur (known as a guard within VAPS XT). Guards can range from a timer being within a certain time range, to a button reaching a specific state. Finally the user has to define how the system will respond if the guard condition(s) are met (known as the "Action" within VAPS XT). For example if a button is switched to state 1 a circle would change colour from red to green.

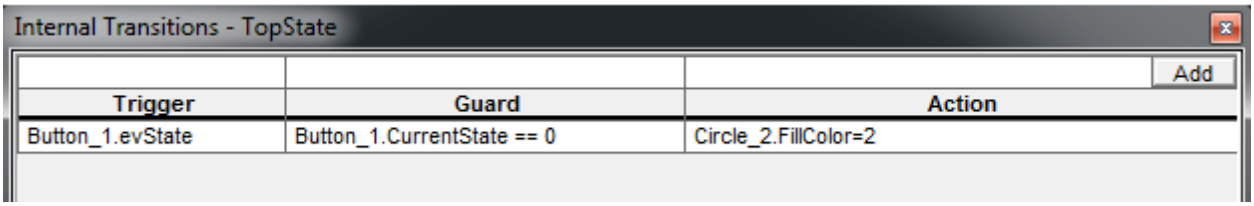

Figure 15- Example of an Internal Transition

The second method can also be expanded into what VAPS XT calls "State Charts", an example of which is shown in Figure 16. These state charts are used to either define various sub-functions that need to run in the background of an object or indicate when the core function of an object will drastically change with an object's state (i.e. the current function in a multi-function display). State charts use defined "states" that each have their own entry and exit actions that are executed when the system goes into the particular state. The system decides when to enter into a state based upon if/else gate parameters or if a state has a guard to determine when it should switch states. In terms of

programming, most state charts follow the same format as Internal Transitions, in the sense that the state chart needs an initiation condition, which is then checked against a guard and then an entry/ exit action must be specified. In terms of the Reconfigurable Ground Control Station, state charts were only used as background sub-functions, namely to determine when and what waypoint information would be sent between the Autopilot/ Waypoint Controller (described in Section 5.2.3) and FlightSIM.

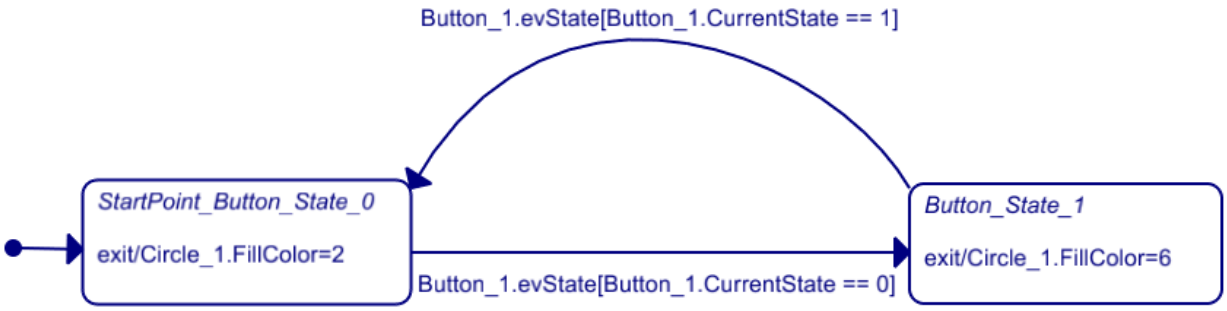

Figure 16 - State Chart Example

Finally, VAPS XT can be linked to other Presagis programs, such as FlightSIM through various communication protocols that must be defined. By linking FlightSIM to VAPS XT, the designed graphical objects can retrieve and send various flight parameters (i.e. the aircraft's Angle of Attack or throttle position). The process to connect VAPS XT and FlightSIM is described in more detail within Section 3.3. However once the connection is setup, the user just has to define what parameter is being linked and what they want to name the linked parameter (known as an Input/ Output (I/O) buffer within VAPS XT). Each I/O buffer that is created has a send and receive entity which is denoted by a prefix attached to the I/O buffer, which is defined as "snd" or "rcv" respectively. These send or receive I/O buffers can then be attached to any graphical object or format so that the system can retrieve or send the desired values to FlightSIM.

3.3 nCom

FlightSIM and VAPS XT can be linked through a number of inter-computer connections. This can be accomplished through direct C+ programming in the form of APIs or can be done through Presagis' in house nCom communication protocol. nCom can use either low level Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) connection protocols to establish a data flow between various

applications. This connection has to be setup within both applications so that the nCom connection can be established through the computer's port or IP address.

The connection that was established between FlightSIM and VAPS XT in the reconfigurable Ground Control Station was setup as an UDP connection. UDP connections are based upon the exchange of data through a defined address port on a computer. In the case of the Reconfigurable Ground Control Station, the port was set to be equal to 7777. The UDP connection can then be set to be either Peer-to-Peer (P2P) or broadcast. P2P is used when the client needs to send the data to a specific IP address port in another computer, whereas the broadcast setting is used when the client needs to be installed onto multiple computers. The final step to setup the UDP connection is to define the received buffer size, which defines how many bytes of information are going to be exchanged through the connection. The default value, which was not changed for the Reconfigurable Ground Control Station, was set to 60 000 bytes.

These connection parameters can be setup within FlightSIM under the Communications and Control tab within the Run-Time mode (Figure 17 shows FlightSIM's nCom editor). The user can then find the nCom settings under the sub-category called nCom. Finally the user needs to select "edit" which is found next to the loaded connection file, and then edit the file to reflect the connection settings that they wish to have.

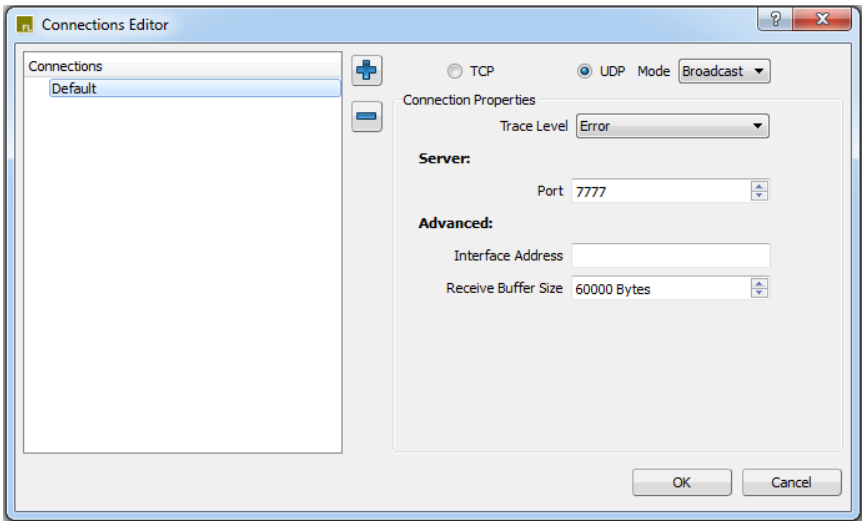

Figure 17- FlightSIM nCom Connection

To setup nCom within VAPS XT, the user first has to setup the direct connection and then link it to a specific I/O process. The first step in this process is to change the connection settings, which is

found within the DataIO sub menu under the name “Connections”. Within the connection settings the user can add any number of connection types. The default connection that needs to be used to create the nCom connection is called “Default”. At this point the user just has to set the “Default” connection properties to the same values that were set within the FlightSIM connection file (i.e. that it is using UDP in broadcast mode; the receiving address port is 7777 and the received buffer size is 60 000 bytes). Once the properties are set the user just has to exit the connection settings tab. All that remains at this point is for the user to select the “Mappings” object within the DataIO sub menu. Within the mappings tab the user has to define how the UDP connection will interact with VAPS XT. The user is given a number of process names which correspond to different potential connection functions. To set up the nCom process, the user just needs to select the “default” process. At this point all that remains is to setup the Process Mapping which describes which I/O buffers will be connected to the connection type, the transmission rate of the connection (in Hz), and if the connection will only be sending data (if the communication protocol needs to both send and receive data it needs to be set as “No”). Figure 18 shows VAPS XT’s two nCom setup pages. After the mapping is completed, VAPS XT and FlightSIM will begin to exchange data over the UDP connection.

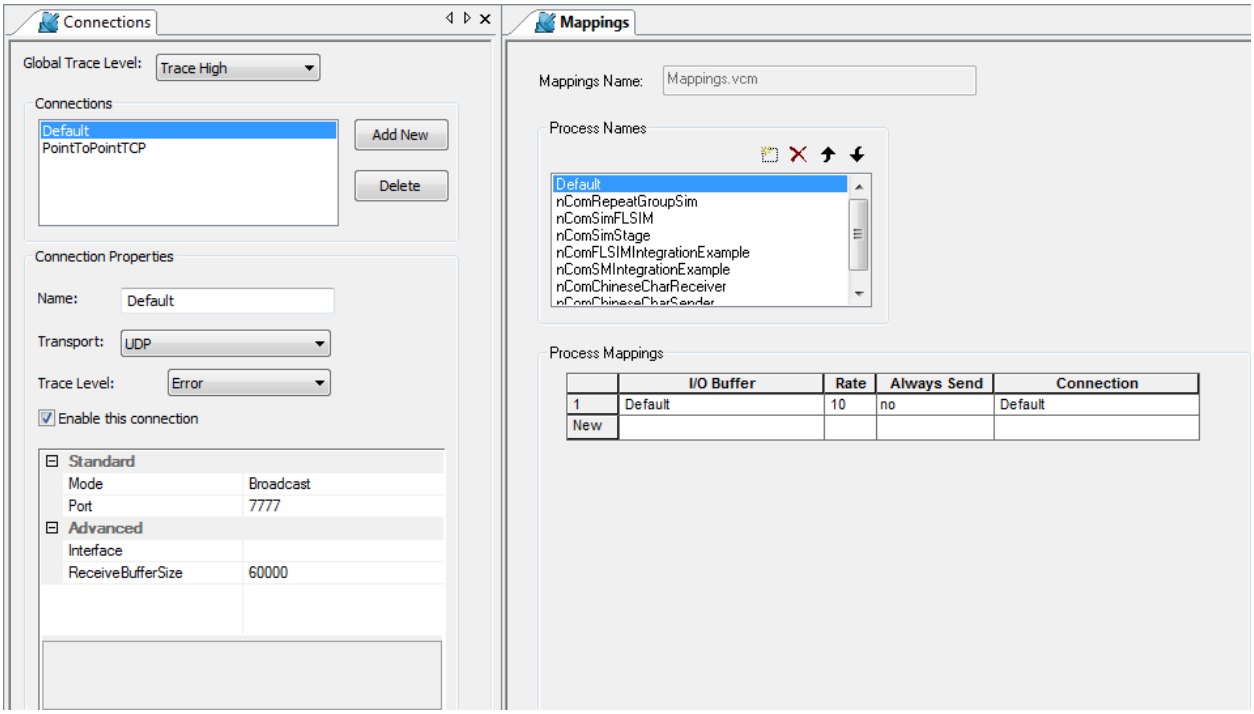

Figure 18 - VAPS XT nCom Connection

The names of the various I/O buffers that were used to exchange data between FlightSIM and VAPS XT are listed in Table 5Table 1 along with a description of what the I/O buffer is transmitting over the UDP connection.

If the user wants to communicate with another program a variety of other connection types can be established. The first is a Distributed Interactive Simulation/ High-Level Architecture connection which is specifically used for a program to communicate directly with Presagis' STAGE. STAGE uses a different connection type because the simulation can be broken down into smaller components that are each handled by a separate computer/ server. As such, a faster and more stable connection is required for the simulation to have little to no lag. In addition it allows the simulation to be run on multiple computer OS' in case the user happens to be running different OS' on different computers. The other connection type is called Common Image Generator Interface and is also designed by Presagis. The Common Image Generator Interface is used to allow FlightSIM, HeliSIM or STAGE to communicate with other visual based simulation programs, such as Laminar Researches X-Plane. The interface ultimately converts variables given off by each visual program so that they can be used by the other program.

Buffer Name	Buffer Description
cmdFlaps	The left and right flap position
ExpBuffer	All of the aircraft parameters
Flsim_ap_ats	The mode and setting of the Autopilot Auto Throttle System
Flsim_ap_basic	The mode and setting of the Autopilot Basic function
Flsim_ap_direct	The mode and setting of the Autopilot Flight Director
Flsim_ap_fms	The mode and setting of the Autopilot Flight Management System
Flsim_ap_wpt	The current waypoint information
Gear	The angle of the gears and the any commands to extend or retract the gears
Throttle	The aircraft throttle position

Table 5- Utilized I/O Buffers

4.0 Design Process

Design of the reconfigureable Ground Control Station had a conceptual stage and three major design iterations. These stages took place over the course of five months (from October 2014 to February 2015). All of these stages are described in detail below in Section 4.1 through 4.4.

4.1 Conceptual Stage

The first step of the design stage was determining the overarching design goals that the Ground Control Station needed to achieve. It was determined that the system must utilize multi-touch gestures due to their inherent advantages with computer based systems. Another design goal that was quickly decided upon was that the system must be reconfigurable to allow the user to determine the placement and size of all the instrument panels found within the Ground Control Station. Reconfigurability was set as a design goal because extensive customization complements touch gesture interaction very well, as can be seen by the ability to reconfigure the layouts of the majority of modern mobile phones and tablet devices.

With these two design goals in mind brainstorming was undertaken in order to determine a series of functions and instruments that could be used to enhance the design goals and to show the effect of using such a system. The initial ideas ranged from (but were not limited to) being able to control payload cameras through drag and pinch-to-zoom gestures and to the ability to drag and drop alerts onto the screen so that more information could be displayed.

After the initial brainstorming was completed, a couple of ideas were selected and expanded upon due to their promise of enhancing the set design goals. The first was an interactive alert and notification system (the concept of which is shown in Figure 19). Most standard alert and notifications systems work off a simple audio cue and a flashing red light to denote that something is wrong with the system [76]. The user must distinguish what the error is solely based off the type of audio cue provided, which can lead to slow response times as the operator has to sift through large amounts of information to find the error. As such, an interactive alert system was conceptually developed to involve the operator more in the error detection process while also giving them information about the error rapidly. The idea behind this feature was that if a malfunction or error was detected within the system, an initial message would appear in the corner of the screen that would briefly explain to the user what the error was. The user could then click or drag the alert pop-up onto the screen where it would expand and

display to the user all of the variables that related to the detected error as well as what range the variables should be between. This type of system would provide two major benefits. The first is that the human eye is rapidly drawn to subtle sudden movements in one's peripheral vision [86]. Thus a display that flashes in the corner of the screen will quickly grab the attention of the user so they can identify the error. The second benefit is that the ability to expand the initial error message allows the user to see more detail and further identify what is wrong with the system and quickly respond to it.

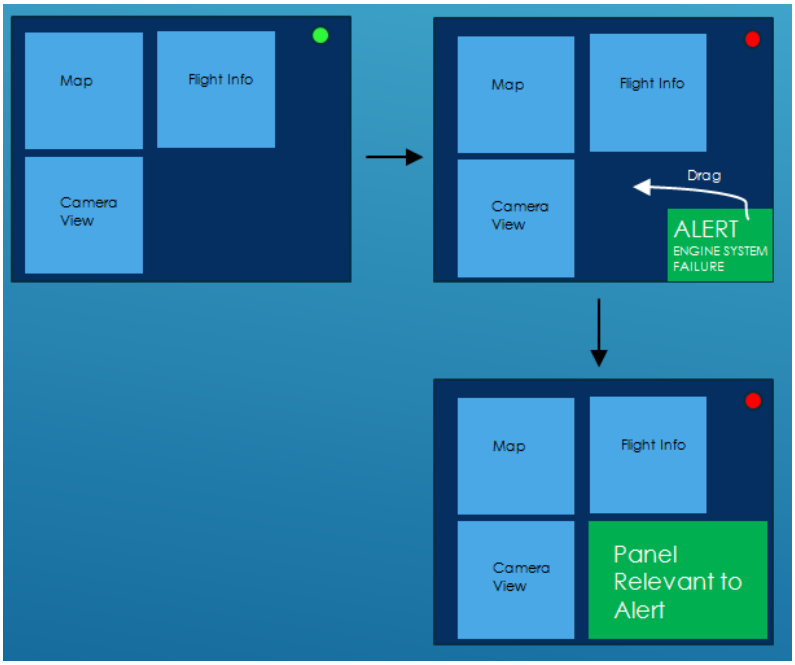

Figure 19- Initial Alert and Notification Concept Drawing

Another key feature that was expanded upon was one that would help enhance the goal of reconfigurability, by developing a centralized location that would house all of the various instrument panels that can be found within the Ground Control Station (a visual design for this concept can be found in Figure 20). This was important for the reconfigurability because it would act as the location that the user would have to go in order to select the instrument panels they wished to have on the screen. This panel would be able to be dragged out of the side of a portion of the screen or pushed back into it in order to save on valuable screen space. All of the panels that would be found within this panel would be categorized to ensure that the operator could quickly find the panel they were looking for. Finally, when the user wanted to have the instrument appear on the screen all they would need to do is tap and drag the associated icon onto the screen. It would then expand and become manipulatable allowing the user to determine where on the screen they want the instrument placed as well as how large or small they want it.

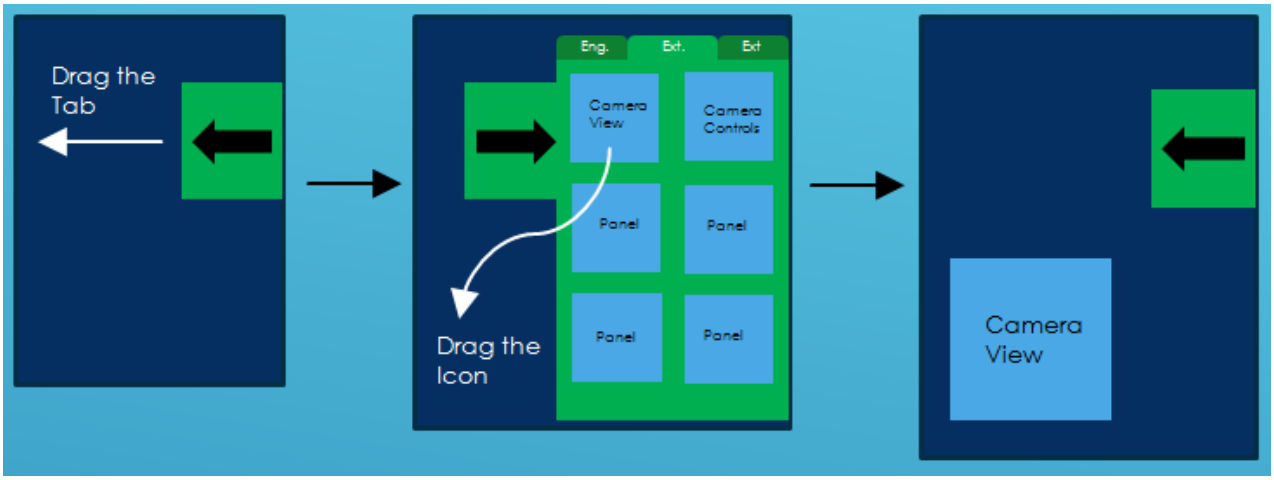

Figure 20 - Reconfigurable UI Concept

In addition to conceptual features, a few concepts for instrument panels were designed. One such concept was for how a user might be able to interact with a camera payload through the use of gesture interactions. It was brainstormed that an intuitive way to move and zoom in and out of an attached camera would be to drag or pinch the screen onto which the camera feed was being outputted. This would also mean the system would not rely on a physical joystick or a set of sliders to move it around. Thus, if the physical joystick could be removed and replaced by simple touches to the monitor, a large amount of physical space could be saved, which is useful for confined stations. Removing the joystick from camera controls would help to eliminate any problems with the operator mistaking what the joystick is controlling (i.e. the aircraft or the camera). In addition the inclusion of gestures would likely improve the accuracy that could be achieved (versus the use of a physical system) because the camera would begin to act as an extension of the arm.

Other emerging forms of input technologies were evaluated in order to determine if they could be used to further enhance the included multi-touch gestures. The two major systems that were looked at were virtual reality and heading tracking. In terms of virtual reality, Oculus Rift goggles were looked at to see if the various camera feeds (i.e. the payload camera and any onboard flight control cameras) could be imposed into a virtual world where the UI could then be over laid, thus removing the need for a simulation display monitor entirely. On the other hand, head tracking could be used to ensure that notifications would directly appear onto the monitor screen that the user is looking at if multiple ones are in use. However since both of these ideas were deemed to be less important features, they were dropped due to limited time constraints and software constraints (the initial version of FlightSIM could not easily support Oculus Rift).

After the initial exploration period, work began on developing some very basic concept panels within VAPS XT 4.0. This was done to expand the designer's knowledge on how VAPS XT functioned, as (access to the program had only been granted at the start of September 2014). One such panel that was generated during this time was a very simple Flight Control Panel which included all the major components of a standard Primary Flight Display. As can be seen in Figure 21, the panel was very simple and was composed mainly of simple shapes and blocks of solid colour. Despite its simplicity it served as a stepping stone for designing the overall Ground Control Station and its associated panels.

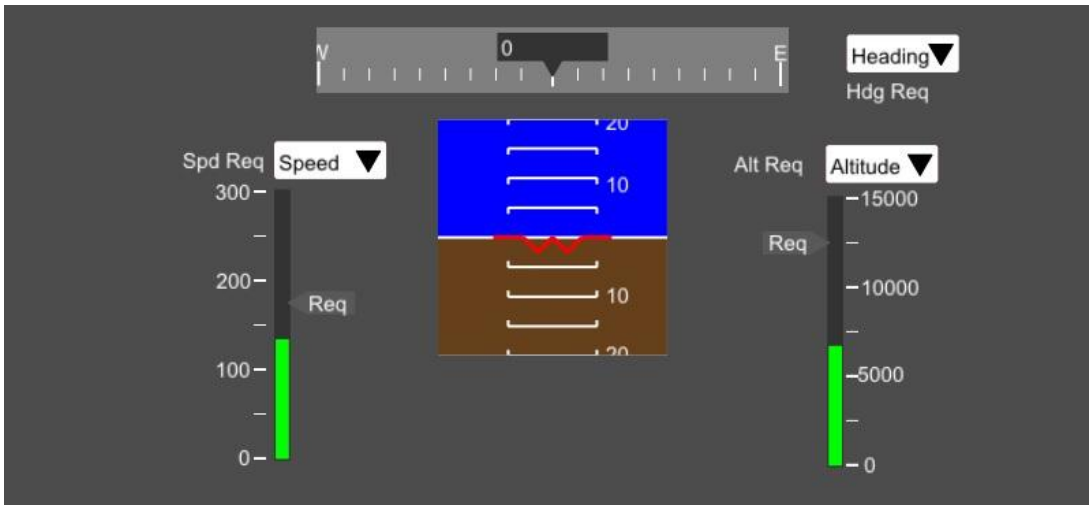

Figure 21- Sample Flight Control Test Panel

4.2 First Design Iteration

After the key concepts and features had been determined, an initial trial version was created, as shown in Figure 22. This version of the Ground Control Station was made within VAPS XT 4.0 and did not contain any real touch features. Its main purpose was to show what was envisioned and to begin creating some of the less complex instrument panels. In this version, simple geometry was used to model all the instruments and the overall UI. Solid blocks of colour were used to help differentiate objects. Despite the lack of any real touch interface integration, this version was able to mimic some simple gestures with the help of a mouse input. A simple left mouse click acted as a tap point, and the actual position of the mouse on the screen acted as the location of the touch point. This first iteration provided a platform to test and setup some of the interactions that would be later expanded upon.

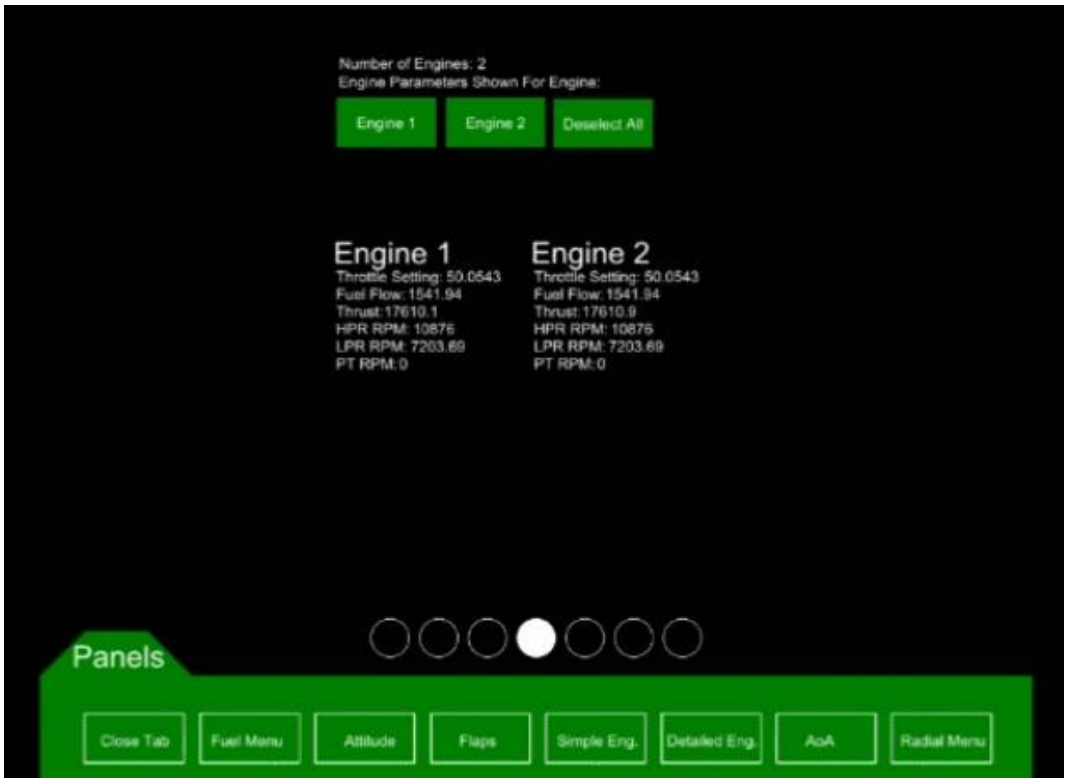

Figure 22- First Iteration User Interface

This test concept had a basic tab system, the ability to swipe between the pages with a mouse and had seven different instrument panels. The included instrument panels were a fuel gauge, a simple engine monitor, a detailed engine monitor, a primary flight display, a gauge showing the aircraft's Angle of Attack, a panel to control the aircraft's flaps, and a concept for a radial menu (which was later dropped from the final version due to time constraints and integration issues). Each instrument panel took up one virtual tab that could be swiped between. In addition, there was a tab on the bottom portion of the screen that when tapped would expand and allow the user to choose which instrument they wished to change to. Finally this concept version included seven small circles in the lower middle portion of the screen to represent the different tabs. At any given time, all but one of these circles would only be outlined whereas the one in use was filled in. This helped to show the operators which of the virtual tabs they were currently using, and helped reduce confusion over the number of available tabs.

The main purpose for the radial menu that was included in this build was to test its function as a means of interacting with a future map panel. The menu would appear when the operator had tapped on a location on the map and could be interacted with afterwards. As can be seen in Figure 23, the radial menu was split into 4 quadrants. Each quadrant had a different option that could be chosen, and the

user would navigate through the various options until all the needed information to complete the desired function had been provided. The functions that could be chosen were either setting both individual waypoints and maneuvers for the autopilot system and/or being able to switch to the previous or next waypoint in cue. If adding a maneuver or waypoint was chosen, the menu would provide a further set of options that were context sensitive and had to do with the needed information to set it up. Ultimately with the time constraints available for this project, this feature was dropped because it could not gather the proper information needed to generate waypoints from a single click (i.e. translating the location of a tap within the map space to a geographical location) which significantly reduced the effectiveness of the radial menu. In addition it was very difficult finding a size for the menu that would not impede the user's actions.

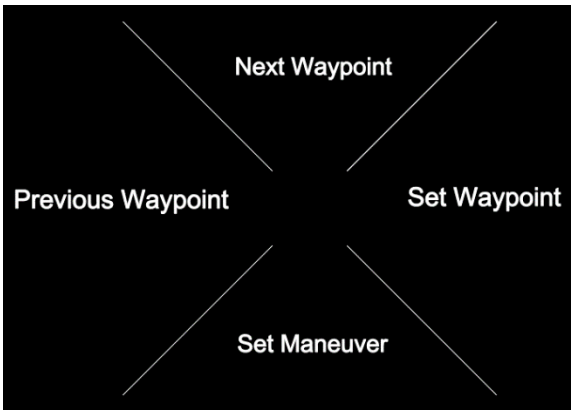

Figure 23- Radial Menu Concept

4.3 Second Design Iteration

The second design iteration (shown in Figure 24) was developed to showcase some of the key design concepts of the Ground Control Station at the Interservice/Industry Training, Simulation and Education Conference (I/ITSEC) that was held in Orlando, Florida from December 1-4, 2014. This build introduced new instrument panels, including a map and a basic autopilot control panel. Furthermore, the number of virtual tabs was changed to five with a number of the previous instrument panels being amalgamated. This allowed for a reduction in the amount of unused space and helped to quicken the operator's ability to look at all of the needed information. The final change over the previous version was a slight improvement to the UI visuals for all buttons. The buttons were initially just solid rectangles, but were changed to look three-dimensional in this build. The buttons would also change colour to better show their respective state. For example, if the autopilot was in the "Disabled" state it would

appear red on screen whereas if it was set to "Enabled" it would be green. This helped to show the Operator the status of the system.

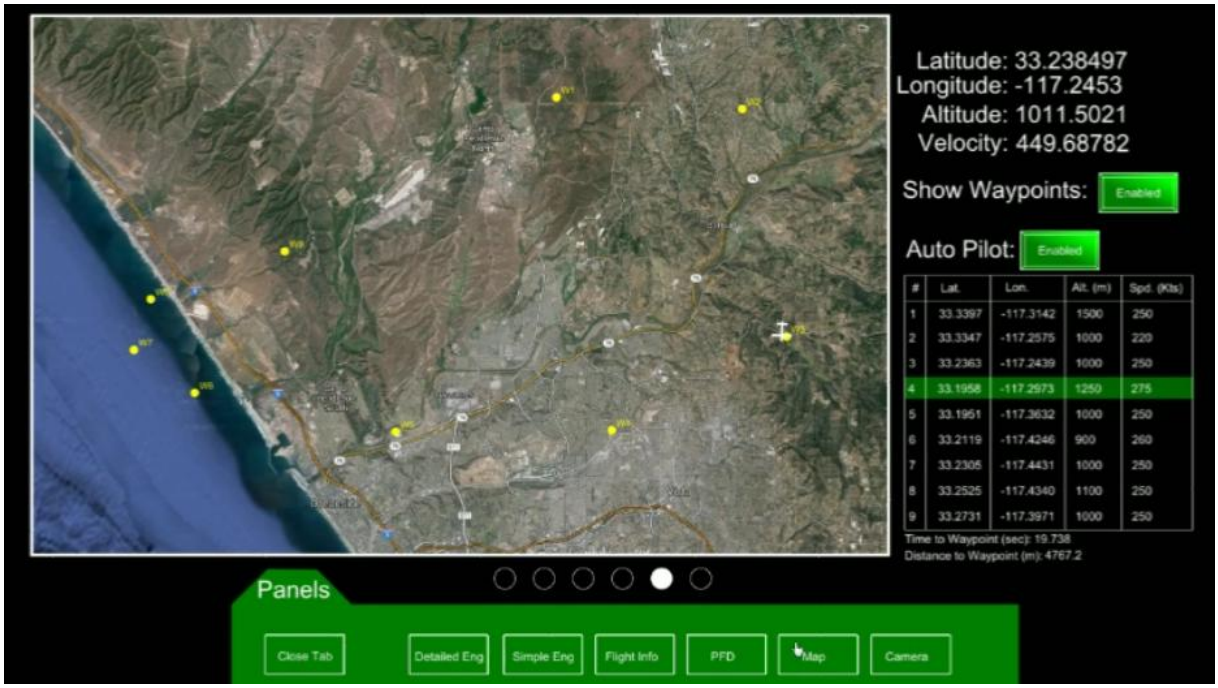

Figure 24 - ITSEC Demo (Second Iteration) User Interface

4.4 Final Design

The most drastic changes to the Ground Control Station came in the third and final iteration of the design process. Initially the first two versions of the Ground Control Station were made using VAPS XT 4.0.1, which was the current commercial version of the program. The final version can be seen in Figure 25 and was created using an updated beta client of VAPS XT (version 4.1) which allowed for the inclusion of true multi-touch gesture support. However because the newer client was an early beta version it did not have access to all of the gestures/ features that would be included in the final commercial release. The beta client was limited to only drag and pinch gestures, but as is described in Section 5.1.3, modifications were made to the gesture drag function in order to utilize tap and swipe gestures.

The final version also brought a number of updates to both the functionality as well as the visual fidelity of the user interface. The new functionality included a host of new instrument panels that took advantage of the touch capabilities, such as a gesture controlled map, as well as more complete panels such as the autopilot/ waypoint controller. In addition to the added instrument panels, the final Ground Control Station iteration implemented the alert and notification system that was described in the

conceptual stage (Section 4.1), a modified instrument tab bar, a virtual keyboard, a top tab bar as well as including the design goals of multi-touch gestures and system reconfigurability. All of these functions and instrument panels are described in more detail in sections 5.1 and 5.2.

Figure 25 - Final User Interface Design

The visual fidelity update allowed the system to go from being rather bland and dark visual appearance to a much more striking visual look. It achieved this first and foremost by replacing all of the solid blocks of colour that were prevalent in the previous two iterations with darker coloured gradients that had splashes of colour to denote the interactable areas. Gradients were used to help provide a more visually interesting background, dark enough to not distract from any tasks that are being completed. The hints of colour were then used to contrast the black background so that the objects that could be interacted with were easily distinguishable and could be rapidly found. The tabs were also changed from a vibrant green to a more subtle beige colour. This was done to help prevent the eye from being pulled away from the instrument panels while still being discernable from the background. Each panel was also given a dark beige backdrop that helped to denote where the panel began and ended. This was beneficial to the user because it meant that they did not have to guess where they would have to press their fingers to enable the repositioning/ rescaling of the panels. Finally the top portion of the top tab bar, virtual keyboard, and exit panel pop-ups were filled with either a teal or green colour to

help differentiate their functionality. Teal was used to denote objects that related to the top tap bar (i.e. custom tab name pop-ups). Teal was also used in these top portions in cases where the user could directly move the pop-up around, and was used to show the active area with which the user could interact. The green, on the other hand, was used to denote non-moveable panels/ pop-ups such as the virtual keyboard and exit panel pop-ups. Green was used for these functions in order to intuitively tell the user that panel/ pop-up had different functions from the others. Finally to help the user recognize which of the tabs was currently selected within the top tab bar or instrument side tab, the selected tab would always be a slightly lighter colour then the other tabs. This was done to help the user quickly identify where they were with respect to the other tabs while not being too visually distracting. Overall the choice of teals, greens, beiges and dark greys were used to provide an aesthetically pleasing colour palette that did not clash. The contrast between light and dark ensured that the user could locate what can be interacted with, with little to no delay. Also, because the choice of colours were similar in tone (i.e. more earthy hues of the colours) the potential for eye fatigue due to high contrasting colours was minimized, thus allowing for longer operations to be completed.

5.0 Results

The final Ground Control Station included seven key features, such as the ability to reconfigure all of the instrument panels, the ability to interact with the system through multi-touch gestures as well as an intuitive tab bar system allowing for multiple virtual screens. In addition to the variety of features, the system also includes 12 instrument panels. These panels included interactable ones, such as, the gesture controlled map, to non-interactable panels, like the aircraft general parameters panel. The variety of panel types was used to showcase how touch gestures could be easily implemented into a Ground Control Station to control an aircraft. Overall these features and panels helped to create an easy to use and efficient Ground Control Station that could be used to help reduce operator workloads.

The system combined all of the instruments and features to create the actual Human Machine Interface that the user interacts with. These were however only one part of the entire Ground Control Station. The system as a whole required multiple inputs and outputs to simulate a typical mission a UAV. This entire interaction can be seen in Figure 26. From this figure it can be seen that the Ground Control Station UI was comprised of a number of tabs that each contained various instrument panels. The UI that was designed within VAPS XT outputted two parameters. The first was the actual visual video component of the UI itself. This was outputted from the program to the computer monitor so that the user could see what they are interacting with. The second set of outputs were the various flight variables over which the system had control (i.e. the aircraft's throttle or flap settings). The UI also took in two inputs, one from the monitor/ computer system and the other from the flight simulation software, FlightSIM 14. FlightSIM 14 exports all of the monitored aircraft variables to the Ground Control Station UI while taking in inputs from the various other hardware sources (i.e. a joystick for attitude control). FlightSIM also outputs the flight simulation to a separate monitor so that the operator can view the simulation environment through a virtual camera that has been positioned on the nose of the aircraft. Finally the computer system's monitors took the outputted visual information and projected it on the computer monitors. The monitors, if they are touch capable, also output the touch location properties (i.e. the position of the finger touch points) to VAPS XT where the data can be used to create touch gestures for control inputs.

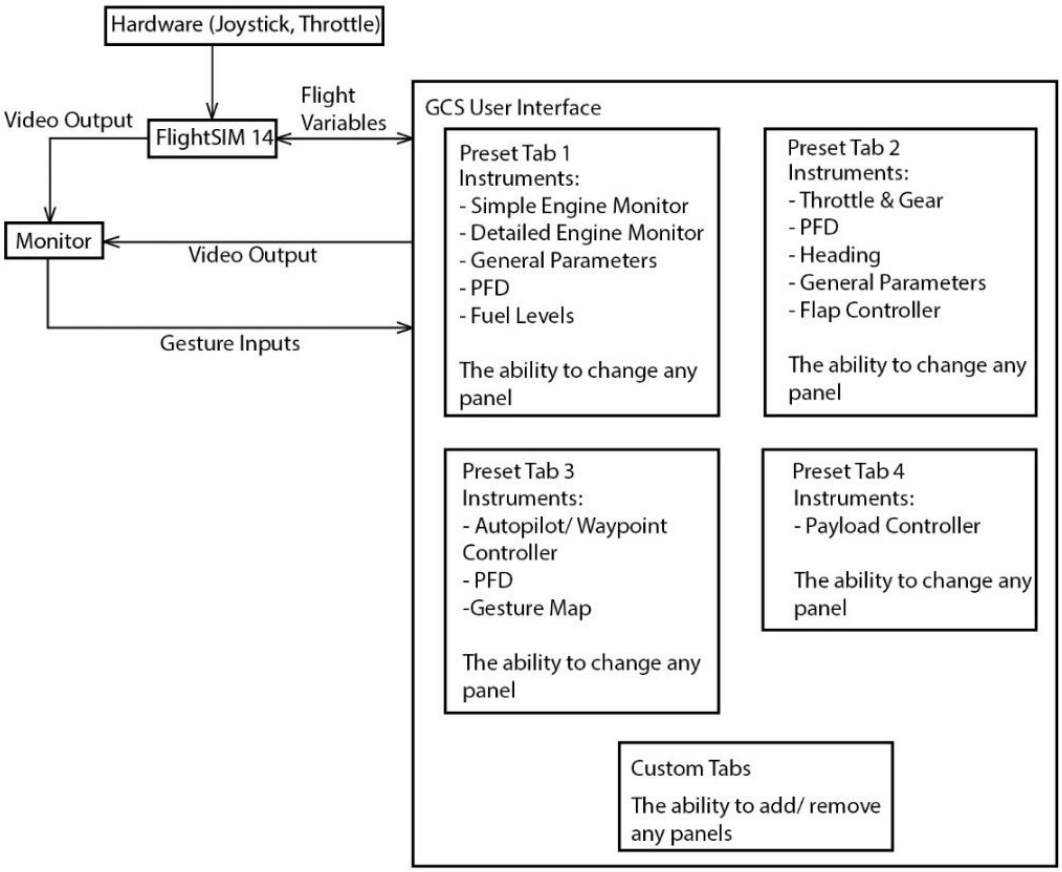

Figure 26 - Ground Control Station Functionality Flow Chart

5.1 Ground Control Station Features

There were a total of six key features that allowed the system to improve the overall workload that the operator experienced. These ranged from a unique Alert and Notification system that allowed for easy identification of errors as well as faster response times when it came to fixing them. The system also included a variety of multi-touch gestures that were used to provide a natural and easy to learn style of inputs that helped to reduce the mental demands of working with the system as well as the ability to reconfigure the setup of the instrument panels which allowed for a unique setup between individual users to suit their needs best. All of these features are described in detail within Sections 5.1.1-5.1.6.

5.1.1 Alerts and Notifications

The Ground Control Station featured a unique two tier alerts and notifications system that was designed around the initial concept described in Section 4.1, with an example of the finalized feature shown in Figure 27. If the aircraft happened to experience an error/ malfunction during the course of its mission an initial error message would appear in the lower right hand corner of the screen. This initial pop-up appeared as a blinking red panel that alternated between a dark and much more vibrant red colour to draw the operator's attention to it. The pop-up would also display a short explanation as to why the pop-up had appeared so that the user could begin to address it.

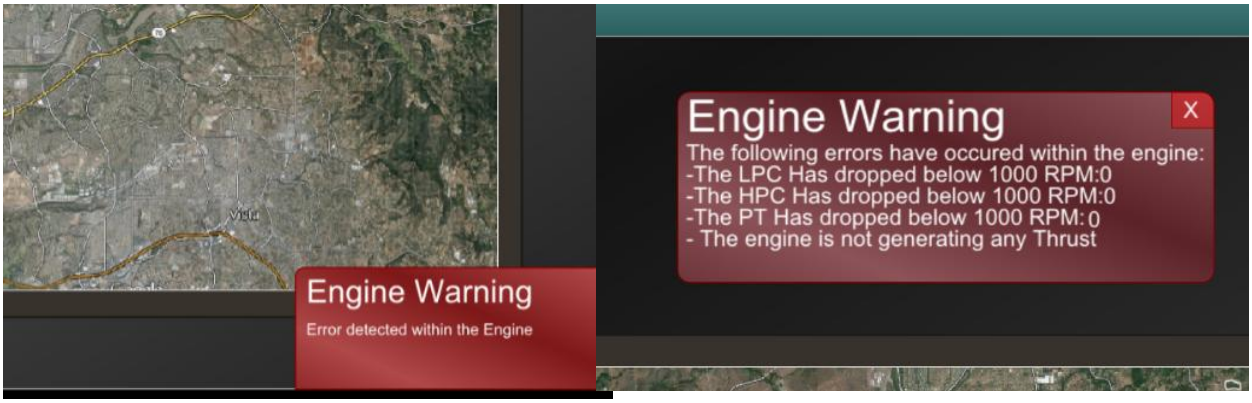

Figure 27- Sample Alert and Notification Pop-up (Left) and Expanded Panel (Right)

The second tier of the alert and notification system was that, when the user tapped the initial pop-up, it would expand into a dark red panel. This new expanded alert panel gave the user a much more detailed explanation of the error. It accomplished this by displaying the exact component that was experiencing the error, what the normal value should be, and what the system was reading the value as. Giving the operator quick access to this information allowed them to quickly assess what was wrong and what needed to be done to fix it, thus helping to reduce operator response times.

In both tier's cases, if the problem was fixed, the pop-up or expanded panel would vanish from the screen. The purpose of eliminating the error message was to help alleviate possible confusion over if the system was indeed fixed or not. In addition, the user could reposition or rescale the expanded panel at any time through the use of drag and pinch touch gestures respectively. By allowing the user to reposition/ scale their error panels, the panel could be placed in a location that best suited the operator. If the operator wished to close the expanded panel at any time, they just had to click the small "X" in the top right corner of the panel. However, if the panel was closed before the error was fixed, it would

reappear in the lower right corner of the screen as a flashing pop-up. It reappeared this way to ensure that the operator would not forget to fix the occurring problem.

Finally within the current build of the Ground Control Station there were six potential error message cases. When an error condition was met, its associated initial pop-up would appear. The six error message cases were:

1. When the aircraft crashed or lost contact with the Ground Control Station.
2. If an issue was detected with any of the aircraft's three landing gears.
3. If the aircraft had exceeded the safe G-Loading region.
4. If the aircraft had exceeded past a safe Mach number.
5. If the aircraft had entered into a stall region.
6. If an error was detected within the engine.

These were chosen as they are all critical values that could be read from the communication with FlightSIM through the "expBuffer" protocol. In addition they were chosen to improve the operator's situational awareness because they all had direct physical effects on the aircraft's flight performance.

5.1.2 Instrument Side Tab

The final version of the instrument side tab (shown in Figure 28) featured a pull out tab that stored all of the available instrument panels. This tab was moved from the bottom of the screen (where it was located within the first two Ground Control Station iterations) to the right side where it could be interacted with more easily. The panel was accessible by the user utilizing a drag gesture on the tab where the user could virtually pull it out or swipe it away from the side of the screen. This drag functionality was not possible in the old VAPS XT 4.0 version, but due to the switch to the 4.1 beta it was possible to take advantage of the gesture commands to create a naturally accessible tab.

Once the user had pulled out the tab, all of the available instrument panels would appear. These instruments were organized into one of four categories to help minimize the number of interactable objects on the screen. With the design considerations in mind and the findings that gesture interactable objects must be greater than 11.5mm for them to be effective, each instrument panel was represented by a 30mm wide labeled box within the side tab bar, to help promote quick and accurate gesture inputs.

However the larger size limited the number of instruments that could be shown on the side tab bar to six. This was not considered to be an issue with the finalized version as no category had more than six instruments. If one did it was felt that a smaller arrow button could be introduced that would allow the operator to view the other instruments. The categorization helped to increase the speed at which the operator could find the instrument they were looking for. The four categories in which the instruments were sorted:

1. Control Panels – Instruments that affected the flight of the aircraft
2. Engine Panels – Instruments that related to the monitoring of the engine
3. Navigation Panels – Instruments used to visualize the flight path and navigate the simulation environment
4. Payload Panels – Instruments that directly controlled any onboard payloads

Figure 28 - Instrument Side Tab Bar

After the user had chosen a category, all of the related instrument panels would appear below. If the operator wished to have one of the panels on the screen they just had to tap and drag the associated labelled box onto the screen. When the box was on the screen, the full instrument panel would appear and could be further positioned and rescaled anywhere on the screen. This process of selecting and adding panels to the main UI area can be seen in Figure 29.

The instrument side tab ultimately helped support the design goal of reconfigurability by acting as the starting point for user customization. Within the tab the user could pick and choose which instruments they needed on screen, where they were located on screen, and how large or small the panels would appear. This feature was further enhanced by the inclusion of multi-touch gestures because it allowed the tab to be both easily accessed and helped to increase the efficiency of screen space used.

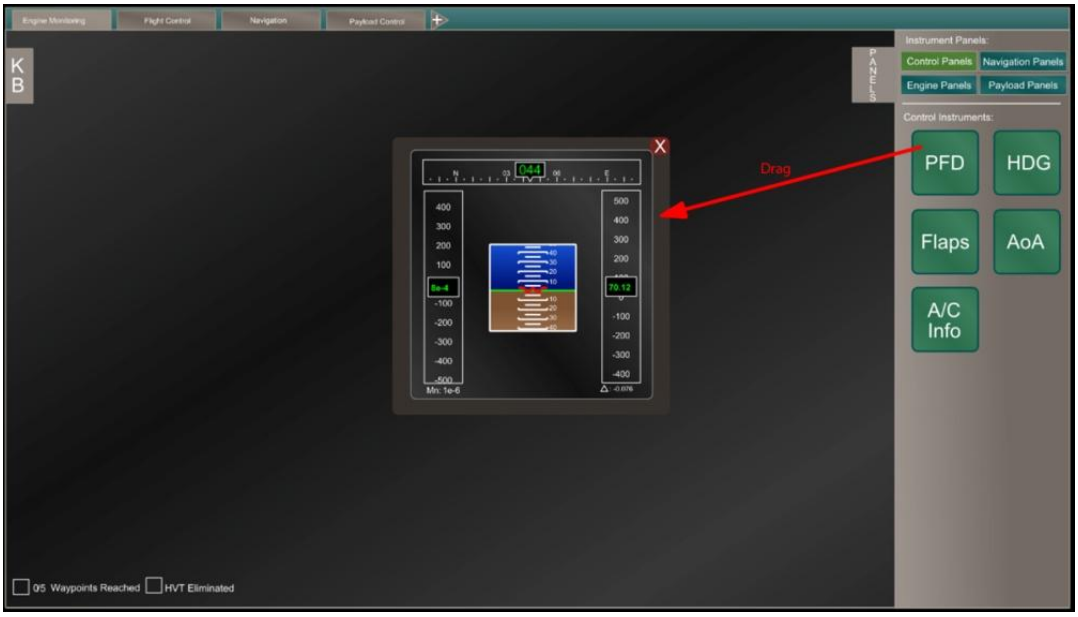

Figure 29- Selecting a Panel to be Inserted into the UI

5.1.3 Multi-Touch Gestures

The VAPS XT 4.1 Beta provided access to gesture inputs for Human Machine Interface interaction. Using this added functionality, a series of gestures were implemented into the Ground Control Station to take advantage of the natural feel they can provide. In total, the system used four gestures; drag, pinch, tap, and swipe, to enhance the overall work capabilities of the Ground Control Station. However, in terms of coding the drag, tap and swipe gestures are all based off the same base gesture drag function with various constraints.

Gestures could be easily added to any UI that was designed within VAPS XT 4.1. This was accomplished by first denoting a rectangular “touch-active-area” space on the screen. This area represented the boundaries within which the specific gesture could be activated. After this area was

defined, the designer could attach any number of gestures to the active area. Before the gesture could be used, the designer had to code the overall interaction between the gesture and the rest of the UI. This coding process first involved programming the initialization of the gesture. This was accomplished by telling the gesture what the initial conditions were (i.e. the start position of the gesture on the screen, or the initial scaling size) when the gesture was recognized by the system. The second step was to set up any guards for the command. These guards could be either the number of touch points that were registered or if another graphical object was visible on the screen. The final stage was to tell the program how the interaction would affect the UI (i.e. if a drag gesture is occurring, how does the distance the finger is dragged effect any other graphical objects). Using these steps the designer could easily integrate any gestures into the system.

The pinch gesture was used solely for rescaling instruments. This gesture, along with a multitude of others, was first popularized with the advent of the original Apple iPhone in 2007. The iPhone was the first mobile phone to incorporate multi-touch gestures while also showing how effective pinch gestures could be for rescaling purposes [87]. The gesture involves multiple fingers being placed on the screen (two or five based on the application) and then spreading the fingers apart (as shown in Figure 30). These pinch gestures rescale objects based upon how far apart the fingers spread from the point that the gesture was initiated. In terms of the Ground Control Station, this gesture was used to rescale all of the panels within the screen space as well as to rescale the payload map and gesture controlled map. The difference between these various pinch to zoom gestures was that to rescale an instrument panel the user had to use only two fingers to complete the gesture, whereas to rescale the various maps within the panels themselves required five fingers. The reason for this differentiation was to avoid accidently resizing the entire instrument panel when all that was desired was resizing a map. An example equation used to change the scale of any onscreen objects (in this case the Angle of Attack Gauge) is shown in Equation 1 below:

$$AoA\_Scl.Scaling.X = e.CurrentScale \quad (1)$$

Where AoA_Scl.Scaling.X was the scale of the Angle of Attack instrument, and e.CurrentScale was the new scale which VAPS XT determined based on the distance between the fingers used for the pinch gesture.

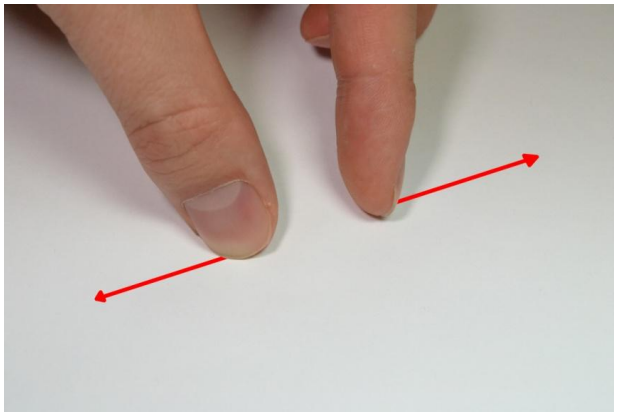

Figure 30- Pinch Gesture

The drag gesture, shown in Figure 31, was used as the base for the remaining gestures used by the Ground Control Station. The base gesture was used to both drag the instrument panels around the screen, interact with sliders, and to pan around the various maps. This gesture was initialized when one or three fingers (depending on the use) are pressed onto the screen. The tethered objects were then affected by the distance the gesture moved from the initialized location. When it came to determining where the gesture was initialized when multiple fingers were used the designer had two choices. The first was to recognize the gesture from the point where the first finger was placed on the screen, and was best used for single finger drags. The second worked better for multi-finger gestures because it determined the location based upon the centroid of all the placed fingers, thus allowing for a more accurate gesture. The one finger drag gesture was used for sliders and panning maps around whereas three fingers were used for repositioning entire panels. The reason that fewer fingers were used for the sliders and pans was to increase the touch accuracy since the sliders are relatively small and more than one finger made it difficult to use. The slider functionality was also very similar to one that Apple has patented in that they both tracked the location of a finger drag which created one set of actions (i.e. whatever action was completed during the drag sequence) followed up by a second action when the finger left the touch event area (i.e. cancelling the gesture) [88].

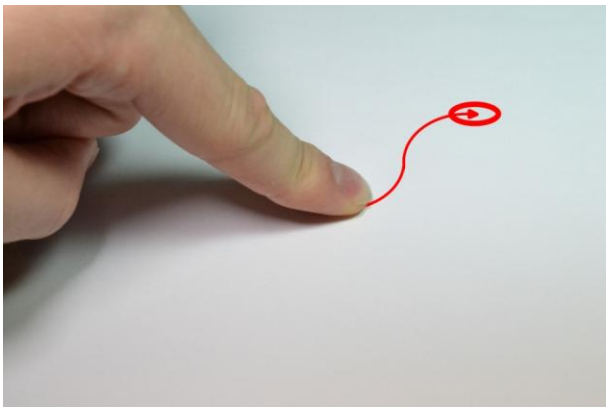

Figure 31- Drag Gesture

The tap and swipe gestures were iterations of the base drag gesture because the version of the VAPS XT beta client that was used did not yet have the gestures programmed. The drag gesture was used as a substitute for these two gestures because of how close the gesture was (i.e. a swipe is fairly similar to a constrained drag). The tap gesture, shown in Figure 32, was a simple change to the drag as it worked solely by registering that the assigned gesture was initiated. This means that no follow up command was given by the system (i.e. the fingers being dragged afterwards would have no effect on any object).

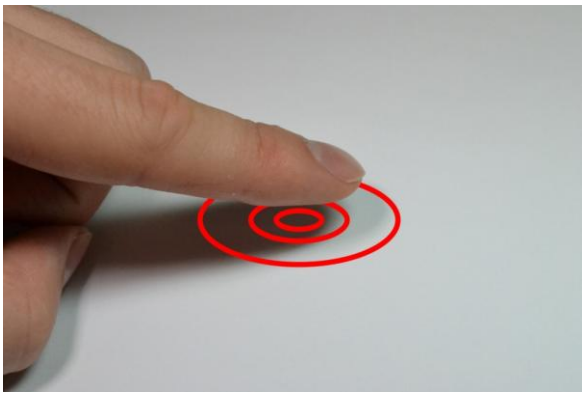

Figure 32- Tap Gesture

The swipe gesture, such as the gesture shown in Figure 33, was initiated as a regular drag gesture as long as four fingers were used, but it only recognized it as a gesture if the distance the drag was tracked exceeded a specified distance of 15 000 units. 15 000 units was selected as the distance to initiate the swipe gesture based off of the trial and error of what felt natural. The equation used to determine if the gesture was indeed a swipe, was based off the distance the hand moved:

$$e.Centroid.X - Swipe.InitialPosition.X \geq 15\,000 \quad (2)$$

Where Swipe.InitialPosition.X was the initial X position of the gesture (which in the case of this system was set to always be the midpoint of the screen so that there would never be an issue with the system misinterpreting the starting and end point of the gesture) and e.centroid.X was the X coordinate of the swipe gesture' centroid (i.e. the center point between the four finger locations). Only the X-coordinate was evaluated because the tabs were located side by side in a horizontal line in the top tab bar. As such it made intuitive sense for an individual to swipe in the same direction to avoid confusion over the way the tabs were aligned. If the user had to swipe in a vertical line, there would have been a mental disconnect between the order that the tabs appeared in the top tab bar and the order that they were swiped.

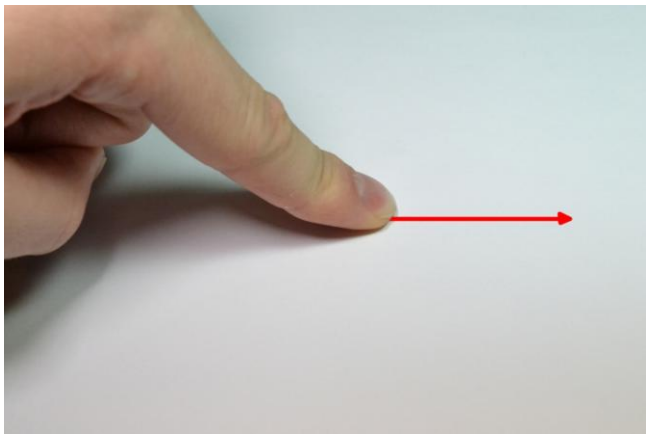

Figure 33- Swipe Gesture

5.1.4 Reconfigurability

The major design goal of the entire Ground Control Station was the ability for the operator to reconfigure the various panels in a way that best suited an individual's needs. This was done by allowing the user to move and resize all of the instrument panels on screen at any given time. This ability to customize the screen space helped to improve response times, reduce substitution errors, and reduce the mental demands experienced because the instruments were in locations that were easily found and sized to allow them to be quickly read.

Since the instrument panels all had to be able to be moved and resized, a method had to be made to lock the instrument panels in place when the user did not want to move them. If they were not locked into place the operator could experience issues when they tried to interact with the various buttons and sliders that were found within each panel. For instance, if a user tried to tap a button, the

panel would shift as the finger hit the button on screen because the program would recognized that as the same initiating command as for a gesture drag. To get around this problem, two functions were introduced. The first was that the number of fingers required to initiate the gestures to move and resize the panels were set to a different number than the gestures required to operate the features within each panel. This was done to minimize in flight errors. The second function was to require the user to tap and hold at least three fingers down on the panel that they wanted to reposition/ rescale. This was done to ensure that the user was not accidently moving the instrument panel by simply tapping it. Three fingers were chosen for the actual movement of the panels because it provided a common gesture and follow up reaction to other multi-touch devices, such as Apple devices, where three fingers are used to move windows around the screen [89]. This helped to avoid potential confusion over what a gesture might be used for as an operator might have owned another multi-touch device that used a similar gesture but for a different response. Thus utilizing common gestures for similar actions helped to lower potential substitution errors. Once the user had held the panel with three fingers for 1.5 seconds the panel would light up (as shown in Figure 34) to display to the user that the panel could be manipulated at will. As soon as the user released their hand from the panel, it would return to its original colour to show that it had been locked into place. Again this ability to go from gesture (a tap and hold) into another gesture (either a pinch to rescale or a drag to reposition) came from one of Apple's common multi-touch functions, where a user could utilize two different gestures subsequently to create a different interaction from either of the individual gestures in order to manipulate a UI object [90]. This helped to reduce the chances of the operator accidently moving an instrument panel around the screen. In addition, the ability to do sequenced gestures allowed the user to utilize more unique operations without creating brand new gestures, which helped to reduce the number of gesture inputs an operator had to memorize.

Figure 34 - Sample of a Non-Interactable Panel (Left) Versus an Interactable Panel (Right)

The next consideration for the reconfigurability of the system was that boundaries had to be placed within the screen space to ensure that the user did not accidently move the panel they were interacting with off the screen. To ensure that this did not happen, a number of timer variables were used. These timers were set at repeating 5ms cycle periods, and every time a timer repeated itself it would check to ensure that its assigned panels were not off the screen. It was found that no timer could handle checking more than two guards at any instant. If it was assigned more than two it would resort to checking the last guard that was assigned to it. A set of equations were then created to see if the one of any four corners of the rectangular panels had gone past the coordinates at the screen's edge. The coordinates used to describe the lower left hand corner and upper right hand corner of the screen were found to be (-5,-5) and (48770,26400) respectively. Using these screen points the system checked to see if either the lower left corner or top right corner of the instrument panel had gone past these points. Using the Angle of Attack Gauge panel as an example, the equations that were used to ensure that the panel stayed within the screen space were:

(AoA_move.position.X + (AoA_TArea.Point1.X*AoA_Scl.Scaling.X))<=-5 (3)

(AoA_move.position.X + (AoA_TArea.Point2.X*AoA_Scl.Scaling.X))>=48770 (4)

(AoA_move.position.Y + (AoA_TArea.Point1.Y*AoA_Scl.Scaling.Y))<=-5 (5)

(AoA_move.position.Y + (AoA_TArea.Point2.Y*AoA_Scl.Scaling.Y))>=26400 (6)

Where AoA_move.position was the graphical location of the instrument panel with respect to either X or Y (denoted by .X and .Y respectively), whereas AoA_Scl.Scaling was the panels scale with respect to either the X or Y axis (denoted by .X and .Y respectively) , AoA_TArea was the active touch area that the touch gestures could be activated within, the .Point1 and .Point2 specified the location of the bottom left corner and top right corner of the touch area with respect to either X or Y.

The touch area was used to determine the bounds of the panel itself, and was multiplied by the scale size to correct for any rescaling that the panel might have encountered. Finally these multiplied values were added to the middle point of the panel to determine the actual corner locations because the various graphical objects were with respect to different coordinate frames (i.e. the AoA_move was with respect to the overall screen dimensions, whereas the AoA_TArea was with respect to the graphical object it was attached to). If it was found that the equations met the conditions of equations 3-6, then the position of the panel would be modified to that of the maximum screen dimensions minus half of the panel's overall width or height (depending on which edge of the screen it was on). For all of the other panels equations 3-6 were modified so that they had the correct panel variables, but the overall equation format remained the same.

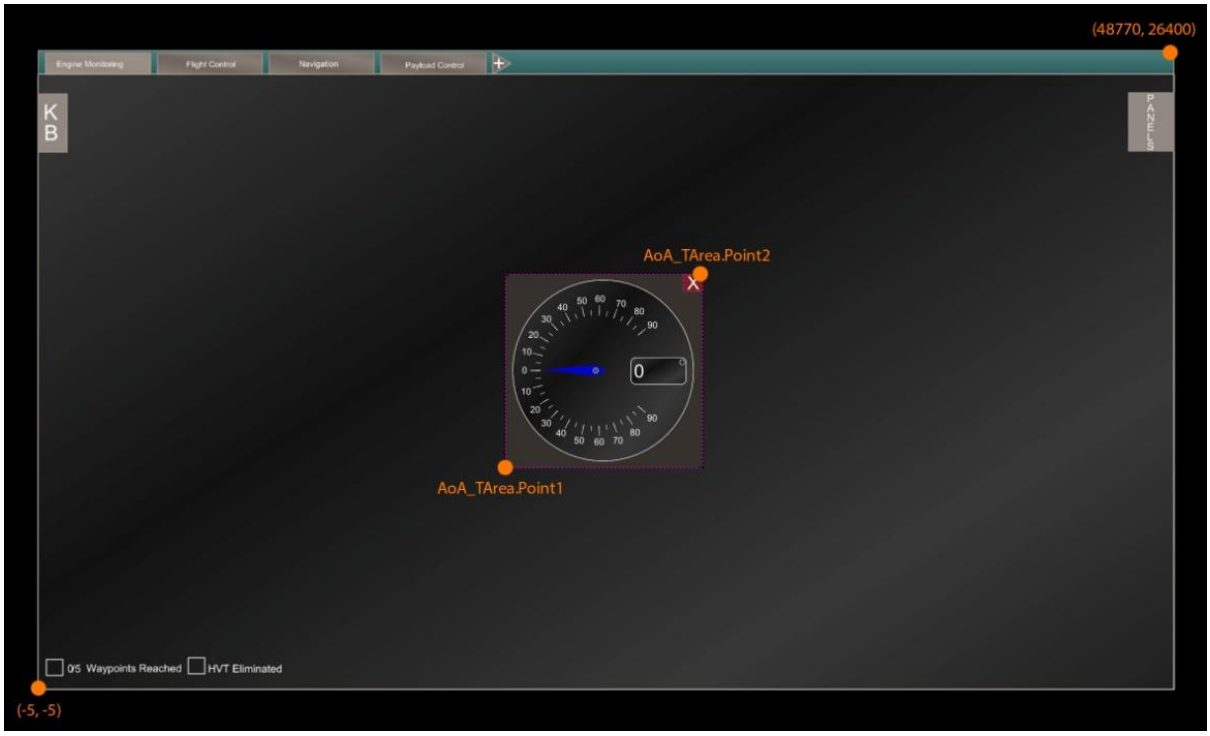

Figure 35 - Angle of Attack Gauge Boundary Variable Diagram

The final reconfigurability consideration that needed to be addressed was how the panel location and scale would be stored. It was decided that all of this information would be best stored within a large float array that was named “InstrumentLocation”. This array stored every panel’s X and Y screen coordinate, its scale and if the panel was currently visible in a tab. Because there were 10 potentially open tabs and four variables that needed to be recorded per tab, each panel took up a total of 40 array elements. The organization of the panels can be seen in Table 6.

Panel Variable	InstrumentLocation Array Element numbers
Is the panel visible within each tab?	0-9
Panel’s X coordinate within each tab	10-19
Panel’s Y coordinate within each tab	20-29
Panel’s scale within each tab	30-39

Table 6- InstrumentLocation Array Element Allocation

Using the array allocation method shown above, each panel could be included into the same array following the same format, with the exception that the element number would be 40 more than the previous panel. Table 7 shows the order and element allocation for each of the 12 instrument panels within the InstrumentLocation array.

Panel Name	Associated Elements
Primary Flight Display	0-39
Throttle And Gear Controller	40-79
Flap Controller	80-119
Angle of Attack Gauge	120-159
Aircraft General Parameters	160-199
Gesture Controlled Map	200-239
Non-Gesture Controlled Map	240-279
Autopilot/ Waypoint Controller	280-319
Simple Engine Monitor	320-359
Detailed Engine Monitor	360-399
Fuel Gauge	400-439
Payload Controller	440-479

Table 7- InstrumentLcation Array Panel Order

Using the order purposed in Table 7 and the panel element allocation shown in Table 6, all of the instrument panels could store its information efficiently as it was updated. The program knew which elements to update based upon a "CurrentTab" counter variable. This counter was assigned a value from 0 to 9 which corresponded to which tab the user had selected (0 being the first tab and 9 being the tenth tab). The variable was used in conjunction with the selected panel to determine which element to directly update. The first set of 10 variables showed if the panel was active within a specific tab; if the panel was open, the corresponding tab element was set as 1, whereas if it was not open it would get set to 0. The other tabs worked by recording the known graphical object's center position in terms of X and Y, as well as the scale whenever the position or scale changed.

An example of how this system stores a single panel variable is shown in equation 7:

AoA_Move.Position.X = InstrumentLocation.Element[CurrentTab.Value+130] (7)

Where AoA_Move.Position.X was the Angle of Attack Panel's X Center Coordinate, InstrumentLocation.Element[Z] was the specific array element that is being modified (with [Z] being the element number), CurrentTab.Value was the current tab that was selected (between 0 and 9) and the addition of 130 came from the starting element of the Angle of Attack's X Coordinate elements (found using both Table 6 and Table 7).

Since the "CurrentTab" value could only range from 0-9, the Angle of Attack X position could only be stored in elements 130-139. Using this method but working in the opposite manner, the system could also read and update what was shown on the computer screen whenever a tab was changed.

Because the instrument panels needed to be switched out at will, the user had to have some way of removing any unneeded panels from the UI. To accomplish this, every panel that could appear on the UI had a small red box with an X in it placed in the top right hand corner of the panel (an example of which is shown in Figure 36). When tapped, a small pop-up would appear that would ask the user if they did indeed want to close the panel. This pop-up was included to reduce the potential for accidental closures due to a miss-click on the screen and presented the user with two options; they could either cancel the closure, in which case the pop-up would simply disappear, or they could select "confirm" which would remove the panel from the specific tab while also updating the instrument array accordingly.

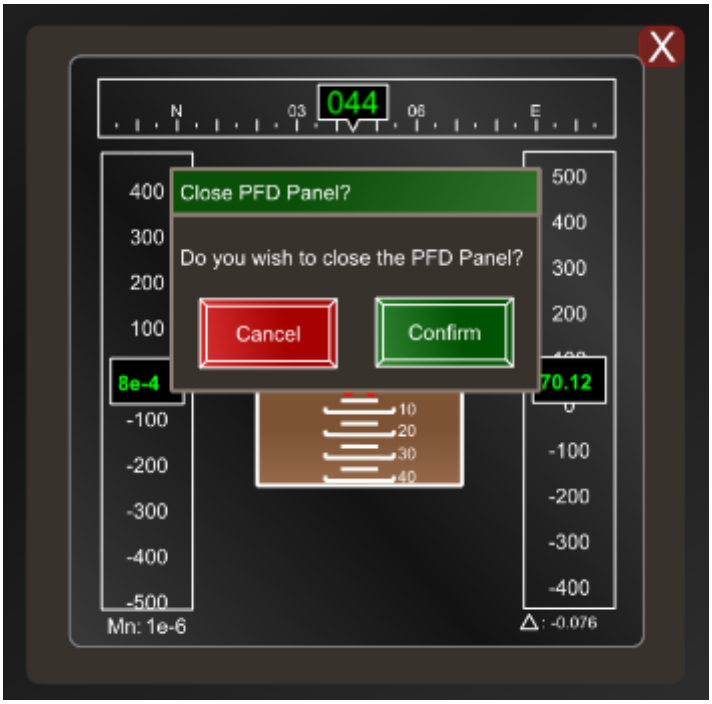

Figure 36- Closing Panels Pop-Up

5.1.5 Top Tab Bar

The idea for the top tab bar (shown in Figure 37) was inspired by modern internet web browsers which use a series of file-like tabs to allow users to quickly navigate between various virtual screens. Each tab provided the user with a unique virtual screen that they could alter (in terms of shown instruments) to suit their individual needs best. Each individual tab also stored information on the location and size of every instrument panel that was open on it so that when the user navigated back to it from another tab they did not have to spend time setting it all back up. These tabs appeared as light beige rectangles at the top of the screen, with the currently selected tab being lighter in colour then the rest. The tabs were placed at the top of the screen to take up as little screen space as possible and so that the functionality could be easily recognized as being similar to the tabs found on internet browsers.

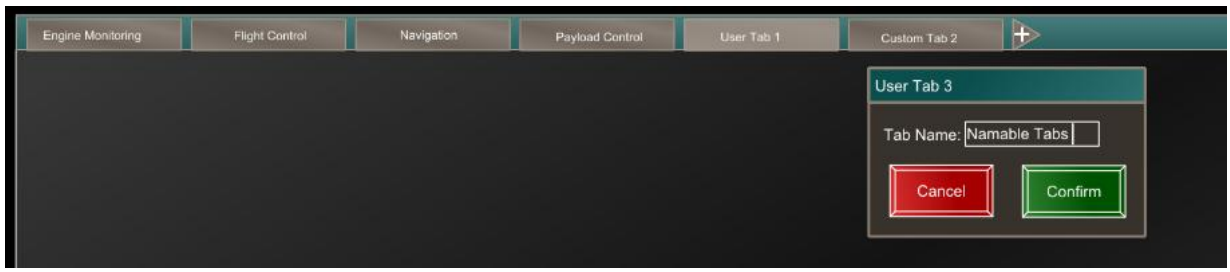

Figure 37- Top Tab Bar

When the Ground Control Station was initialized the user was presented with four pre-set tabs. These were included to allow the system to be used immediately without any setup. The four pre-set tabs each corresponded to the instrument categories found within the instrument side bar described in Section 5.1.2. A list of which instrument panels were found within each of these preset tabs is found in Table 8; a detailed description of the functionality of each instrument panel can also be found in Section 5.2.

The Ground Control Station was not limited to just the four preset tabs. The user could easily add up to six more "custom" tabs (for a grand total of ten tabs) by pressing the "+" button found on the tab bar. If the add tab button was pressed, a separate pop-up contextually appeared so that the user could give their new tab a custom name. This naming process helped to minimize what the operator had to remember by allowing them to give tabs names that corresponded to the instruments they selected. Once a name was inputted, the new tab appeared as a blank slate with nothing in it. At this point it was up to the user to determine what they wish to have displayed.

Preset Tab Name	Included Instrument Panels
Engine Monitoring	- Aircraft General Parameters - Detailed Engine Monitor - Fuel Gauge - Primary Flight Display - Simple Engine Monitor
Flight Control	- Aircraft General Parameters - Angle of Attack Gauge - Flap Controller - Primary Flight Display - Throttle and Gear Controller
Navigation	- Autopilot/ Waypoint Controller - Gesture Controlled Map - Primary Flight Display
Payload Control	- Fictional Payload Controller

Table 8- Instruments Found Within Preset Tabs

5.1.6 Virtual Keyboard

A virtual keyboard was also included within the final Ground Control Station, and can be seen in Figure 38. The virtual keyboard allowed the system to still be used even if there was no physical one attached. This is useful because it allows the system to be used in very space limited environments. The included keyboard layout followed the QWERTY layout as it is the most commonly found type of keyboard. The keys themselves were designed to be interacted with by gesture taps and as such the keys were relatively large on the screen to improve the click accuracy. However if the user did not wish to use gestures to interact with the keyboard every key could also be clicked on with a mouse. The dual input modes allowed the user to not feel constrained to simply gestures or mouse inputs. To further facilitate user customization and creating the best experience possible for the operator, the system was not limited to having to use the virtual keyboard for text inputs. If a keyboard was attached to the computer the user could swap seamlessly between the virtual and physical keyboards. Overall the virtual keyboard provided added functionality to the system so that it could be used in more space limited environments.

Figure 38-Virtual Keyboard

The virtual keyboard was programmed using a series of buttons that had unique properties. Each button had to have an American Standard Code for Informational Interchange (ASCII) value (ranging from 0 to 127) assigned to it, that corresponded to the specific key the button represented. The second unique property determined if the character key was in capital or lower case mode (defined as a 2 or 1 respectively). The capital mode could be changed when the user clicked or tapped the “Shift” or

“Caps Lock” buttons. The only difference between the two buttons was that the “Caps Lock” button would keep all of the keys in mode 2 until it was deselected, whereas “Select” would only keep the keys in mode 2 until another key had been selected.

The user could select which text inputs the virtual keyboard would interact with based upon a focus command that every text input box had. When the user clicked or tapped on the text box they wished to enter text into, the program would request focus for that specific input. When this request was sent, the program knew to insert any text into the specific box, whether it came from the virtual keyboard or from a physical keyboard. The virtual keyboard then used a function called “DiscreteEvtSender” to actually send the specific key presses to the text boxes. This function works like any nCom connection, except that it cannot send the data between programs and thus does not need to have its connection set up. Anytime a key was pressed and then released (denoted by a .doDiscreteEvtPress and .doDiscreteEvtRelease respectively, after the DiscreteEvtSender handle) it would send the specific character to the text input. Utilizing this function the virtual keyboard could be easily added to the overall UI and integrated with any other designed format/ object.

The virtual keyboard could be accessed at any time during operation. All the operator would have to do was tap/ click on the “KB” button, found in the top left hand corner of the screen. When this button was pressed the keyboard appeared in the lower center of the screen (it was always in the same place). It would also disappear from the screen when the user pressed the red “X” in the top right corner of the panel.

5.2 Ground Control Station Instrument Panels

To complement all of the various key features that the Ground Control Station had, the system also had twelve different instrument panels that could be accessed by the operator. These instrument panels were a mix of interactable and non-interactable panels. The interactable panels were used to directly control the aircraft or the amount of information that was being exposed to the operator. The non-interactable panels were used to display critical flight parameters. Each panel featured a similar aesthetic to help promote uniformity throughout the entire system. For example, each panel was bordered in dark beige to indicate the boundaries of the active area for each panel. Within this defined area was a rectangle that was filled with a dark grey gradient. This dark area was chosen to provide contrast to the white display text and more vibrantly coloured instrument graphics. The vibrant spots of

colour were also used to intuitively show the user what could actually be interacted with and where the focus points on each panel were located.

5.2.1 Aircraft General Parameters

The aircraft general parameters instrument panel (shown in Figure 39) was used to display a number of the common aircraft variables that needed to be monitored during flight. All of the variables that were monitored and displayed were taken from the "ExpBuffer" I/O buffer in real-time. Of the twelve panels that were included within the Ground Control Station, the general parameters panel was only one of two that did not include any graphical representation of data (the other being the Simple Engine Monitor described in section 5.2.11). It was designed to be used in conjunction with the other panels to help give the operator a comprehensive idea of how the UAV is behaving. An inherent issue with displaying large quantities of text was that it becomes difficult for the reader to sort through it all and find what they specifically needed with ease. To help negate this problem, two methods were considered. The first was that each variable would be grouped with other similar variables. In total there were seven groupings that the instrument panel utilized;

1. Position-based variables. The position variables included the aircraft's latitude, longitude and altitude.
2. Atmospheric Conditions. The atmospheric conditions included the air's density, temperature pressure, wind direction and wind speed.
3. Control Surface Settings. The control surface settings included pitch, roll and yaw trims as well as flap, slat and speed break deployment percentages.
4. Attitude variables. These included the aircraft's pitch, row and yaw (heading) angles as well as their angular rates of change. It also included the aircraft's ground speed and indicated airspeed, true airspeed, Mach number, vertical speed, g-loading and side slip.
5. Engine Parameters. The engine parameters included, the fuel flow rate, fuel temperature, High Pressure Compressor RPM, Low Pressure Compressor RPM, turbine temperature, jet thrust, manifold pressure, discharge pressure, oil pressure, oil temperature, throttle percentage and the fuel ratio.
6. Landing Gear Status. The landing gear parameters included the gear's position, wheel angle, the gear status (0 was stowed, 1 was extended, 2 was that the gear was transitioning, and 3

was that the gear has experienced an issue), and if the gear was on the ground (0 was off the ground and 1 was on the ground) for each of the three aircraft landing gears.

7. Autopilot settings. The autopilot settings included the AP Mode, Auto Throttle System Mode, Flight Director Mode, and the Flight Management System Mode.

The second way the general aircraft parameters instrument panel helped to quickly show the user what they needed to be looking at was that whenever a variable dropped below the recommended level, the parameter text and its value changed from being white to red. The change of colour allowed the user to quickly recognize that there is an error that needed to be addressed. It also allowed the user to quickly find the error amongst the other parameters because the change from white text to red was such a drastic and noticeable change.

Figure 39 - Aircraft General Parameters Instrument Panel

5.2.2 Angle of Attack Gauge

The angle of attack gauge, which can be seen in Figure 40, shows the aircraft's Angle of Attack. Within the Ground Control Station, the gauge was represented by a real-time updating dial, which showed the user the positive or negative Angle of Attack of the aircraft. In addition, the entire gauge changed colour to red if the aircraft entered into a described stall region (which is aircraft dependent). This change of colour was intended to help to inform the operator quickly that the aircraft was stalling so they could change how they were flying the aircraft before an accident arose thus improving operator response time and lowering the accident rate. The gauge also included a numerical depiction of the Angle of Attack. To help limit the operator's chances of misreading the Angle of Attack during the flight, while also providing them with a more accurate measure.

The actual design of the gauge was done using a combination of pre-defined graphical entities within VAPS XT. The dial markings were added using VAPS XT's "RotTick" entity which allowed the designer to add tick markings in a circular pattern. This specific entity gave one the option to change the radius of the circle that the markings would be positioned around, the number of tick markings, the length of each tick marking, as well as the starting angle and the arc that the markings will be separated across. For this specific instrument, four sets of markings were used. The first two sets represented the positive Angle of Attack, while a negative Angle of Attack was represented by a pair of mirrored tick markings. All the markings started at an angle of 40 degrees and then spanned an arc of 140 in order to get it to not span across a full semi-circle. In addition, one of each pairings of ticks corresponded to a major tick mark, representing a change of 10 degrees in Angle of Attack, while the other set of smaller tick marks related to a change in five degrees. The actual dial was taken from a pre-defined dial entity with just the colour transparency changed from being completely solid to being slightly transparent to help it match the aesthetics of the UI. The dial itself would rotate based upon the Angle of Attack that was provided from the "ExpBuffer" I/O buffer. In addition, the numerical value that was displayed also used the same I/O buffer variable to determine the value.

Figure 40 - Angle of Attack Gauge

5.2.3 Autopilot/ Waypoint Controller

The autopilot/ waypoint controller (shown in Figure 41) allowed the operator to both activate/ deactivate the aircraft's autopilot system with the tap of a button while also giving the ability to add waypoints at any point during flight. In total there were three buttons the user could interact with when they used this panel. The first enabled/ disabled the autopilot, the second allowed the user to add singular waypoints, and the third allowed the user to input a maneuver (a series of consecutive waypoints).

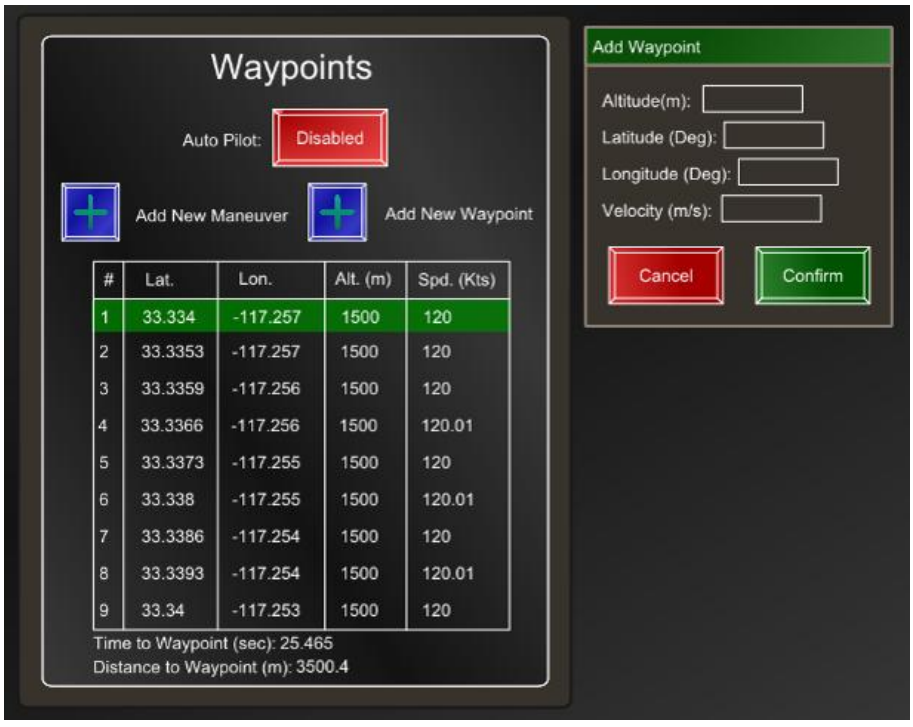

Figure 41- Autopilot/ Waypoint Controller

When the autopilot was engaged the button changed from a red colour (signifying that it was disabled) to the colour green (signifying that it was enabled). The colour change occurred to help augment the activation process so the operator did not have to spend time trying to determine if the system had indeed been activated. When this button was pressed the autopilot had to enable four parameters so that the autopilot engaged properly. The first parameter was the Auto Throttle System. The Auto Throttle System was typically set to "off" during manual flight, denoted as state 0 in FlightSIM and VAPS XT, and it controlled the aircraft's throttle position automatically. The Auto Throttle System had six potential modes that could be set, which were:

1. State 0 – The Auto Throttle System was considered deactivated
2. State 1 – The aircraft would reach and hold at a specific speed
3. State 2 – The aircraft would reach and hold at a specified Mach number
4. State 3 – The aircraft's throttle was slowly set to idle
5. State 4 – The Flight Management System Control was activated allowing the system to match the waypoint velocity
6. State 5 – The aircraft went to the set low pressure compressor spool speed (N1)

When the autopilot was engaged however the Auto Throttle System' state switched from 0 to 4. This was required to ensure that FlightSIM and VAPS XT were informed that the aircraft would be controlled by the Flight Management System as opposed to manually. The second parameter that needed to be altered was the actual Flight Management System. The Flight Management System had only two modes, state 0 which denoted that it was off, and state 1 which denoted that the Flight Management System was on. The Flight Management System ultimately determined what the pitch, roll, and yaw rates should be in order to reach the current waypoint along with any change in velocity to achieve the desired waypoint speed. Despite the Auto Throttle System mode being set to Flight Management System control, the Flight Management System could not truly be activated unless it was directly changed. The third parameter that needed to be changed was the Flight Director. Initially the Flight Director started in a turned off state (state 0) and needed to be switched to on (state 1). The Flight Director controlled the flow of waypoint commands by utilizing the logic found in Figure 42. The final parameter that needed to be activated was the actual autopilot. This parameter was denoted as the "Autopilot Basic" function. Like the previous two parameters, this parameter had to be switched from state 0 (off) to state 1 (on). If this was not done, the other previously activated systems would not

function. In addition, the AP function controlled the aircraft's pitch, roll, and yaw (based upon the Flight Management System commands) as long as the autopilot was engaged.

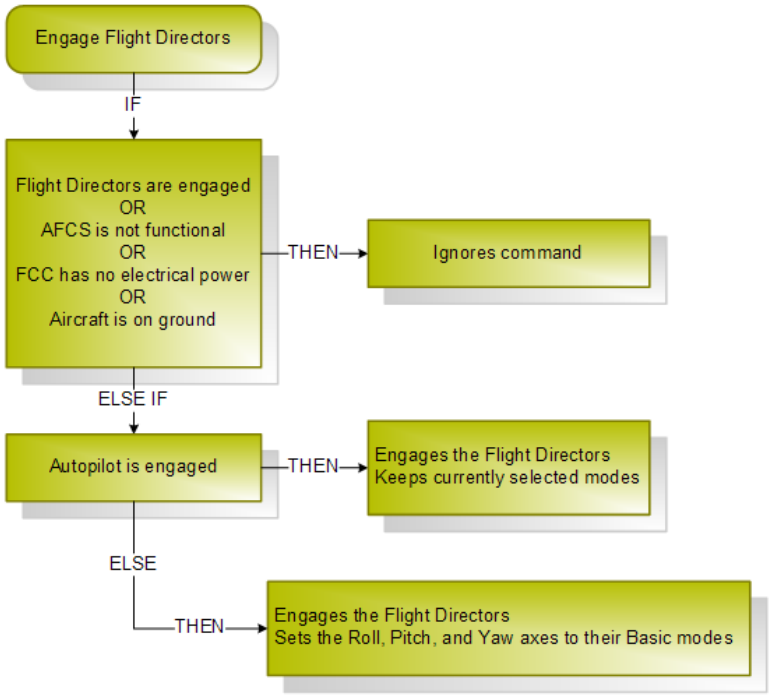

Figure 42- Flight Director Logic [91]

Once the autopilot was enabled, FlightSIM and instrument panel would begin to exchange waypoint data. This was done through the "FLSIM_ap_wpt" I/O buffer. This buffer required the Ground Control Station to send FlightSIM the current waypoint's latitude, longitude, altitude, the required aircraft velocity and whether the waypoint was absolute or relative to the aircraft. Since FlightSIM could only send the current waypoint, all of the waypoint information was stored within an array named "WptArray". This array stored every inputted waypoint's latitude, longitude, altitude and velocity. A counter was then used to keep track of which waypoint the aircraft was currently travelling to. Each waypoint took up six elements within the array with the element allocation being shown in Table 9.

Variable	Array Element
Waypoint Number	0
Waypoint Type (0 = Absolute, and 1 = Relative)	1
Altitude (meters)	2
Latitude (degrees) [Replaced with Range (in Kilometers) if Relative Waypoint]	3
Longitude (degrees) [Replaced with Bearing if Relative Waypoint]	4
Velocity (meters per second)	5

Table 9- WptArray Element Allocation

As such, whenever a new waypoint was added to the system six more elements were used. The exact element that the information was placed was based upon the total number of waypoints and what the new waypoint number was. This was accomplished using the following equations:

Waypoint Number = WptArray.Element[0+((WaypointNumber-1)*6)] (8)

Type of Waypoint = WptArray.Element[1+((WaypointNumber-1)*6)] (9)

Altitude =WptArray.Element[2+((WaypointNumber-1)*6)] (10)

Latitude (or Range) = WptArray.Element[3+((WaypointNumber-1)*6)] (11)

Longitude (or Bearing)= WptArray.Element[4+((WaypointNumber-1)*6)] (12)

Velocity = WptArray.Element[6+((WaypointNumber-1)*6)] (13)

Where the -1 after the WaypointNumber variable was used to counteract the fact that the first waypoint would be Waypoint 1 and not Waypoint 0, and the *6 multiplier was used to account for the fact that each new waypoint would add 6 new elements to the array.

Using equations 8-13 all of the waypoint data could be added into the system where it was used to control the attitude and navigation of the aircraft as long as the autopilot was engaged.

When the distance to the current waypoint reached zero, a waypoint counter increased by one and the next set of waypoint information was sent to FlightSIM through nCom. This distance to the next waypoint parameter was taken from FlightSIM's "expbuffer" I/O buffer where it was analysed within a

state chart, shown in Figure 43, to determine what the system should do next (i.e. does it send the next waypoint's information or do nothing)

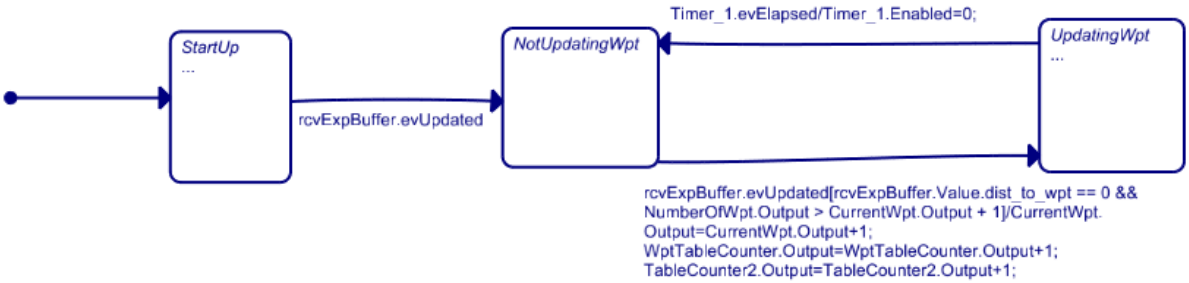

Figure 43 - Autopilot State Chart Diagram

The waypoints themselves were divided up into two different types. The first were absolute waypoints which were waypoints that had an already defined location within the simulation environment. The second type was relative waypoints which did not have a specific location but were instead relative to the aircraft's current position.

When the operator chose "Add New Waypoint" within the panel, the system generated a single absolute waypoint since the user should know the exact location of the waypoint. The user was then prompted to insert the relevant information for this waypoint (i.e. the latitude (in degrees), longitude (in degrees), altitude (in meters), and velocity (in meters per second)). This data entry was accomplished using VAPS XT's pre-made "Text Input" entities. These test input boxes allowed one to type in information using either a keyboard or the created virtual keyboard, and would store the information as a data string that could be accessed by other entities. After, the new waypoint was added to the waypoint array that was previously mentioned so that it could be used to direct the aircraft when needed.

If the operator chose "Add New Maneuver", the user was able to generate a series of ten waypoints that defined a circular maneuver. This maneuver first directed the aircraft to the center of the desired circular maneuver, at which point it would fly out to the desired radius of the maneuver and then it would fly through eight different waypoints that defined an octagon that was transcribed with the desired circle. An octagon was chosen because it was not possible to conduct a true circular maneuver within FlightSIM because the aircraft would fly to each waypoint in a straight line. The limitation of the aircraft taking the shortest route to its waypoint meant that it was not possible for the

aircraft to take advantage of constant radius turning, since the aircraft would take as straight line as possible to the waypoint as opposed to following a defined circular radius. In addition, great circle navigation was not looked at because of how it is typically used for long distance flights that are several hundred kilometers in length, and the current simulation area was less than 100 kilometers in both North/South and East/West directions making it less viable [92]. As such an octagon allowed a rough circular pattern to be obtained. Of the ten waypoints, only one was an absolute waypoint that corresponded to the center of the waypoint maneuver. All the other manoeuvres were defined as relative waypoints because it was far more accurate for the system to tell the aircraft to turn a specified number of degrees (45˚ in the case of an octagon) and travel a specified distance to the next waypoint as opposed to trying to first define each waypoint in terms of absolute coordinates and then sending them to FlightSIM. When the user wished to create a new maneuver, they were prompted to give the center point of the circular maneuver in terms of latitude and longitude, the radius of the desired circular maneuver, the altitude it should be executed at, and the velocity. Once this was done, FlightSIM stored the waypoint information into the same waypoint array as the absolute waypoints.

One other piece of information needed for all of the relative waypoints was the bearing. The bearing was defined as the angle between the nose of the aircraft and where the current waypoint was located. Due to the fact that the relative waypoints were all points on the octagon that defined the circular maneuver, it was known that the bearing between each waypoint would consistently be 45 degrees. However, an issue arose at this point because FlightSIM would not properly turn the aircraft 45 degrees even when it was sent the command to do so regardless of whether it was sent in radians or degrees. It was found that the values that FlightSIM needed to read changed based upon the radius of the maneuver and partially by the velocity at which the aircraft was travelling. The reason for it changing based off of these variables as opposed to it being constant could not be found. However it was believed to be caused from the fact that FlightSIM's supplied aircraft do not have perfectly accurate autopilot systems due to company-customer product confidentiality. As a result the pre-supplied aircraft that come with FlighSIM (such as the F-16 model that was used for the simulations) sometimes have systems that would not react in a realistic manner. As such, an equation was needed to be defined in order to accurately define the required bearings for each relative waypoint.

To define this bearing equation, a series of trial and error points needed to be completed. FlightSIM was sent various values for the bearing until the aircraft turned 45 degrees at multiple radial distances. The bearing was then recorded along with its associated radius. This trial and error process

occurred for radial distances of 7km (the minimum waypoint distance that FlightSIM could execute), 10km, 15km, 20km, and 30km. The values were then graphed in excel with the X axis representing the maneuver radius and the Y axis being the associated required bearing. It was found from this graph that the points were roughly exponential in nature. Since it was exponential in behaviour, it was known that the equation to define the bearing would be in the form $Y=aX^b$. The same points were then inputted into a mathematical regression program called *CADRE Analytic* [93]. This program quickly and accurately generated the "a" and "b" parameters that defined the standard exponential expression and it was found that the bearing would be roughly equal to:

$$Bearing = 0.553957707313 * Range^{0.0720572884593} \quad (14)$$

However, it was found that this equation was not perfectly accurate as the experimental error ranged from 0-7% as the velocity changed. As such, a small correction term was added to equation 14. This correction term was found to be equal to +0.017453292, through trial and error. The final equation used to define the bearing was determined to be:

$$Relative\ Bearing = 0.553957707313 * Range^{0.0720572884593} + 0.017453292 \quad (15)$$

The radius of the maneuver was also used to determine the "range" between each relative waypoint. This range was calculated using trigonometry and could be defined with two equations:

$$Range = 2 * R \quad (16)$$

$$R = R_C sin(22.5) \quad (17)$$

Where R_c was defined as the radius of the circular maneuver. Figure 44 shows how equations 16 and 17 were determined.

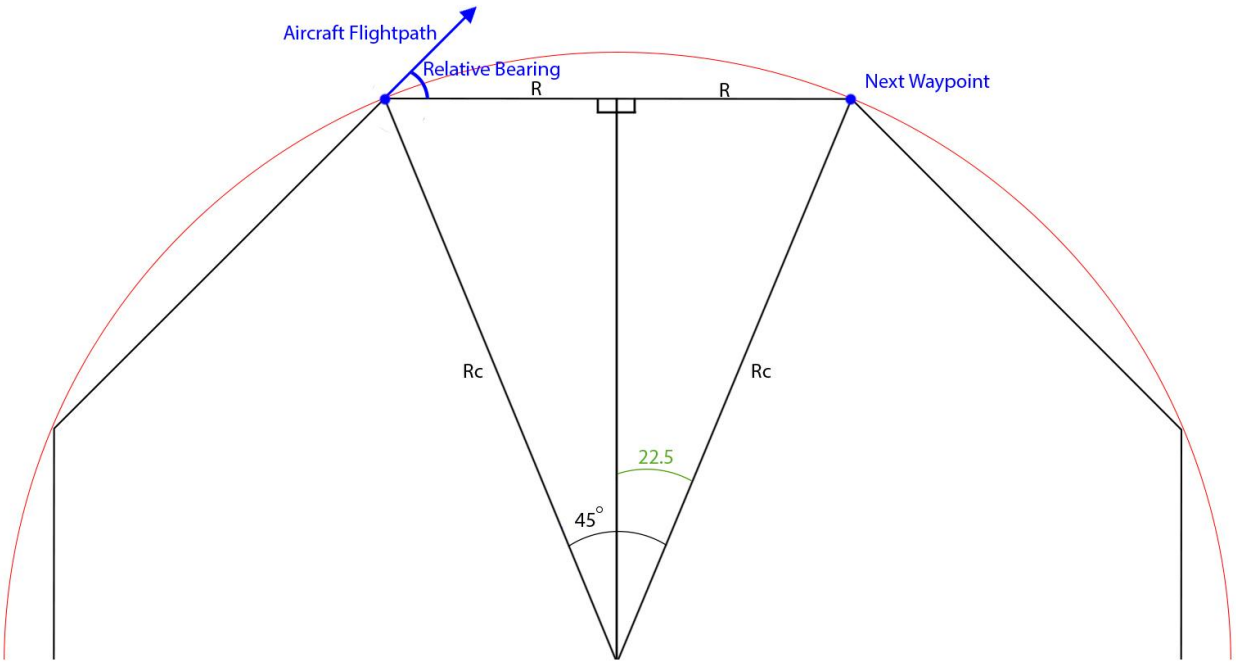

Figure 44- Trigonometry of an Octagon

Every waypoint that was created within the panel was also added to the waypoint table that was found within the center of the panel. Since there was a limited amount of space, only nine waypoints were shown at a time, with the currently set waypoint being highlighted in green for quick reference. The Once the system went through each of the nine waypoints that were visible, the table was updated with the next set of nine waypoints. The table itself showed the waypoint numbers, the latitude and longitude of each waypoint (in degrees), the altitude (in meters) that the waypoint was at and the desired velocity of the aircraft (in meters per second).

Since the waypoint table needed to display all of the waypoints in terms of latitude and longitude, the program had to estimate the location of each relative waypoint in terms of the specific coordinate frame. This was accomplished using the previous waypoint's latitude and longitude as well as the range to the next waypoint and the relative turning angle (bearing) to it. Using these values two equations were used to solve for the next waypoint's latitude and longitude respectively [94]:

$$Lat\ 2 = \ \text{asin}\left[\sin(Lat\ 1) * \cos\left(\frac{d}{R}\right) + \cos(Lat\ 1) * \sin\left(\frac{d}{R}\right) * cos\theta\right] \quad (18)$$

$$Long\ 2 = Lon\ 1 + atan2[sin\theta * \sin\left(\frac{d}{R}\right) * \cos(Lat\ 1)\ , \cos\left(\frac{d}{R}\right) - \sin(Lat\ 1) * \sin(Lat\ 2)] \quad (19)$$

Where Lat and Long 2 were the next waypoint's Latitude and Longitude respectively (in radians), Lat and Long 1 were the previous waypoint's Latitude and Longitude respectively (in radians), d was the distance between the waypoints, R was the radius of the earth in kilometers which was equal to 6371km and θ was the relative turning angle between the two waypoints (in radians).

The final feature in this panel was that it showed the distance and time to the next waypoint just below the waypoint table. These values were taken directly from the "expBuffer" protocol that was sent from FlightSIM to the Ground Control Station. This feature allowed the operator to get an idea of how close they were to the next waypoint, which provided them with more situational awareness.

5.2.4 Detailed Engine Monitor

The detailed engine monitor, which was shown in Figure 45, instrument panel is one of the more interactive panels. The panel itself showed a graphical depiction of the mounted engine in terms of each major component (i.e. for a turbojet: Intake, Compressor, Combustor, Turbine, Afterburner, and Nozzle). When any of these components were tapped on a computer monitor, a pop-up appeared that showed the operator all of the relevant information about the specific component. The information that was shown on screen was taken from FlightSIM's "ExpBuffer" receive protocol (explained in more depth within Section 4.3). Giving the operator the ability to show as much or as little information about the engine on screen helped to limit the chances of overloading the pilot with text and information. This was accomplished by designing each of the engine components within VAPS XT using simple polygons to define the shape of each engine component. The components then had a "touch active area" added around the graphic so that a touch tap gesture interaction could be recognized. After which, whenever the operator tapped onto a component a separate graphic that showed the specific parameters would appear either above or below the graphical entity, depending on its location and how much space there was. This change of visibility was possible by allowing the tap gesture to modify whether or not the graphic's visible parameter was set to on (defined as 1) or off (defined as 0). Thus whenever a touch gesture was recognized the respective graphic would have that parameter changed to the other (i.e. 0 would be changed to 1 and vice versa).

To further help the pilot process information, whenever an engine component had a parameter that entered into a non-optimal region (i.e. the Low Pressure Compressor RPM dropped below a certain value), both the specific parameter's text as well as the graphical component representation changed

colour from white to red. This too helped the operator quickly recognize where the error was occurring within the engine should they not have all of the pop-ups expanded. In addition, because each component pop-up showed multiple parameters, the parameter which was experiencing the fault was also shown in red to help the user identify what they needed to focus on fixing. Overall both these colour changes helped to lower the reaction time of the operator because less time was required to determine what was wrong and how to go about fixing it. In addition, when a parameter started to operate nominally again, it returned to its original colour so that the operator knew the problem was resolved. This panel was in contrast to the simple engine monitor tab explained in Section 5.2.11 because instead of showing only the major engine parameters the user could see how the entire engine was functioning.

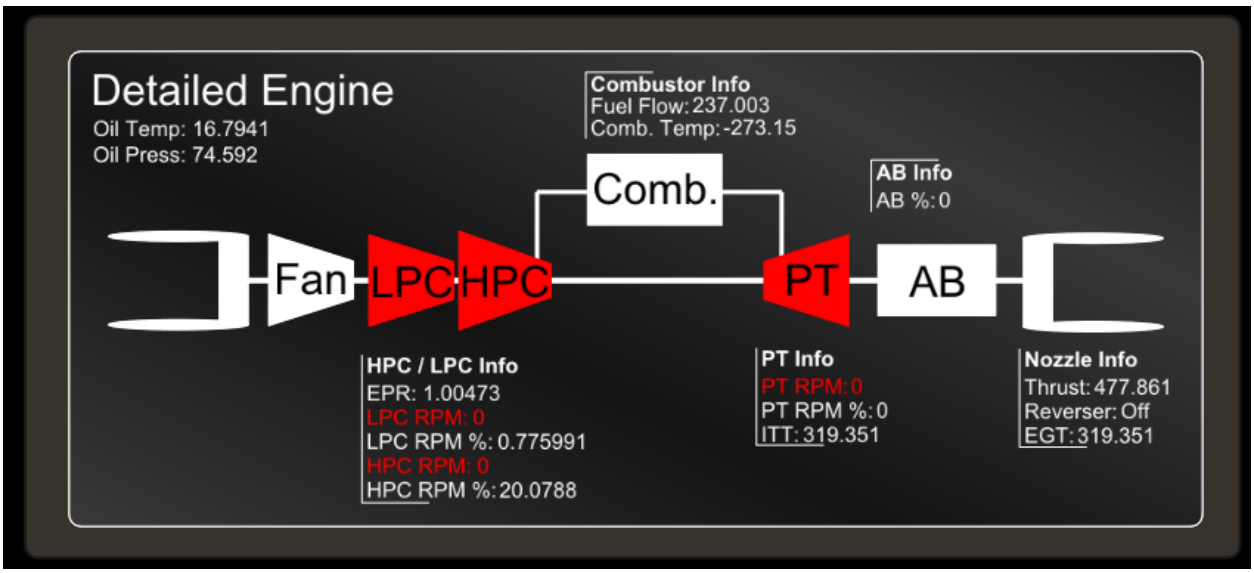

Figure 45- Detailed Engine Monitor Instrument Panel

5.2.5 Flap Controller

The flap controller, shown in Figure 46, was an instrument panel that was used to change the aircraft's flap angle in real-time during the simulation. The panel had a graphic showing the wing along with a blue polygon section that represented the flap. This graphic helped to increase the operator's situational awareness because the wing of the aircraft was not visible to them and thus, some method was needed to get that information. The angle could be changed by dragging a slider that was found above the wing graphic. The value that was sent to FlightSIM within the "cmdFlaps" I/O buffer ranged from 0 (un-deployed) to 1 (fully deployed)

The slider that was used for this instrument panel, along with the other panels that utilized sliders, was designed within VAPS XT. VAPS XT does contain a pre-made slider entity, however it was not possible to interact with it via touch gestures within the VAPS XT 4.1 Beta client that was used for the design process. As a result a slider graphic was created to represent a slider knob and was further programmed to move only along a specific X or Y axis, like a slider would. Setting the motion to only a single axis was possible by having the touch and drag gesture change only one of the graphical entity's position variables (i.e. if the motion of the slider was along the X-Axis, the drag gesture would only change the X-position of the graphical entity). Since each slider had to have a limit as to how far it could move in either direction a set of boundary conditions had to be created. These were simply if the graphical entity moved beyond a specified value (along its X or Y axis) it would stop moving and would be constantly reset to the boundary position the graphic was at. The reason the graphic would not stutter when it passed beyond these boundaries was because there was a timer that would check the slider's location at a rate of 200Hz (a 5 millisecond period) which means that the graphic would be resetting up to 20 times a second if it was told to move beyond the boundary point. Finally the boundary limits on all of the custom made sliders were shown to the operator as a filleted box that the slider graphic was positioned within.

An equation also had to be determined that would describe what the sent flap angle value was supposed to be equal to when it was not at the maximum boundaries of the slider. This equation was developed through the linearization of the slider position so that when the slider was positioned at the left side of the slide it would not be deployed (i.e. commanded to be at 0) and when it was at the right, it would be fully deployed (i.e. commanded to be at 1). Since the slider positions were constantly negative (going from -2880 to -4060) due to the slider's location within the panel, the difference between the starting position (-2880) and the current position had to be multiplied by -1 to ensure that a positive command was sent to FlightSIM.

$$Flap\ Value = \frac{-1*(-2880-Slider.Position.X)}{6940} \tag{20}$$

The moving portion of the flap graphic was designed utilizing a modified dial entity within VAPS XT. The dial's polygon was changed from being a diamond (like the one found in the Angle of Attack Gauge in section 5.2.2) to being more of cone. This was accomplished by changing the polygon from having four sides to three and then by adding in a circular entity to the dial, thus creating a cone. This

dial would then rotate based upon the value of the flaps that were read from the “ExpBuffer” I/O buffer. The displayed numerical value was also found using the same buffer parameter.

Complementing the flap graphic was a small indicator that output the actual true flap angle. This feature was added to the instrument panel because it was difficult to quickly determine what the angle was simply by looking at the flap graphic or the slider tick marks. As such, showing the numerical angle of the flap deployment allowed the user to quickly see what it was set to, and allowed the user to accurately set it to a desired level.

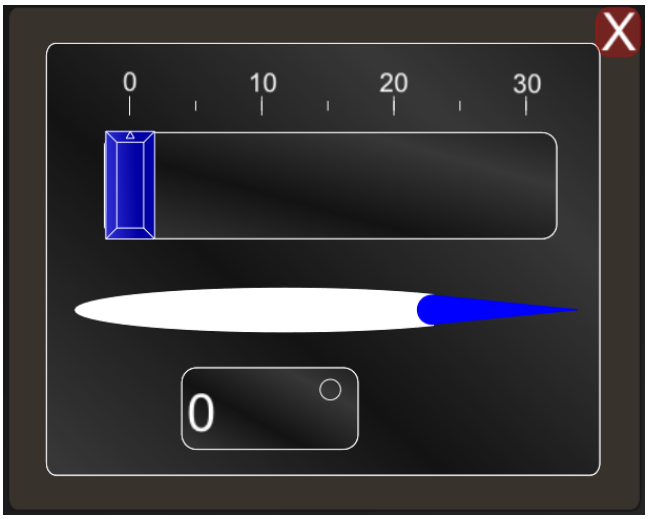

Figure 46- Flap Controller Instrument Panel

5.2.6 Fuel Gauge

The aircraft’s fuel gauge was a graphical representation of the amount of fuel remaining in the aircraft’s fuel tanks (or battery). The remaining fuel percentage was shown by both a depleting graphic as well as a numerical percentage value to eliminate vagueness. The displayed graphic showed a coloured bar inside of an outline of a battery. This bar would shrink as the fuel source was depleted and was worked based upon taking the “Fuel_Ratio” parameter from the “ExpBuffer” I/O buffer which represented the percentage of remaining fuel onboard the aircraft, ranging from 0 to 100, and then that value was used as the “Current Value” for a bar graph entity within VAPS XT. The bar not only shrank, but also changed colour depending on how much fuel remained. Initially the graphic started out green in colour and remained that way as long as the fuel percentage stayed between 100% and 30%. When the percentage dropped below 30% it changed to orange until it dropped below 10%. Orange was chosen to help notify the user that they needed to start considering ending the current mission or should find

somewhere close by to land or refuel. Once they reached 10% of the fuel reserve, the graphic changed to red to indicate the urgency of landing the aircraft. These various fuel colour states can be seen in Figure 47. Finally, like with the flap controller, the percentage of remaining fuel was also shown numerically so the operator did not mistake how much fuel was remaining by taking a quick look at the panel. The percentage was found by using the "expbuffer" I/O buffer that was received from FlightSIM. Within the "expbuffer" was a variable called fuel_ratio which represented the remaining percentage of fuel onboard the aircraft.

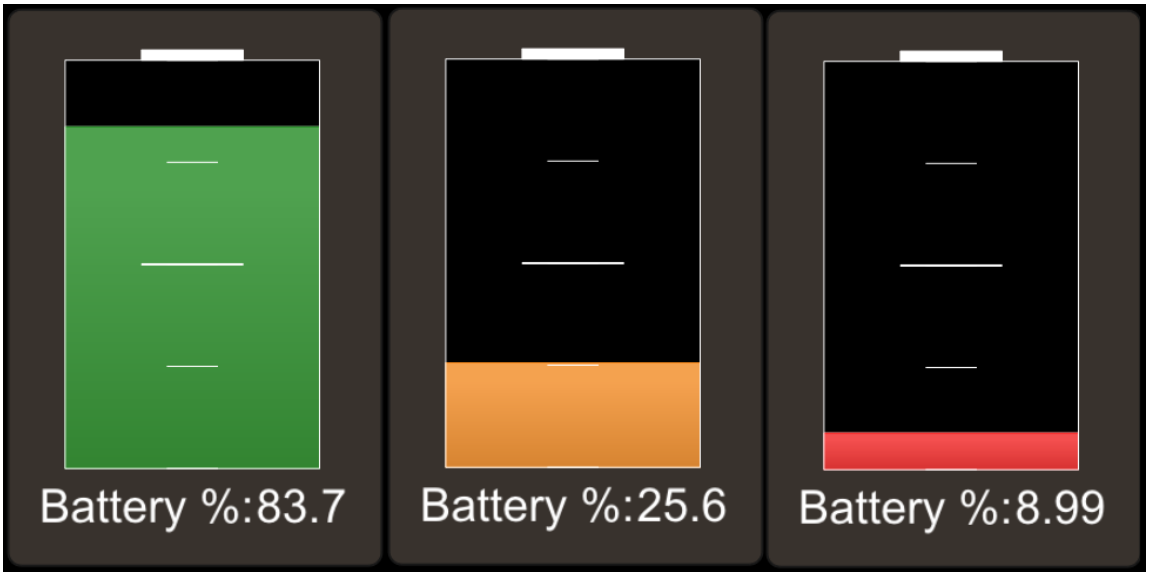

Figure 47- Fuel Gauge Instrument Panel

5.2.7 Gesture Controlled Map

The gesture controlled map instrument panel (shown in Figure 48) solely showed a map of the "playable" simulation area. Within the current build of FlightSIM 14, the basic map area that was included was a small area within South California, situated around Camp Pendleton and was shown in the center of the instrument panel.

The first major feature of this instrument panel was the inclusion of gesture inputs. The user could pan the map image around with a simple drag gesture or zoom in/out of it at will with a pinch gesture. These gestures required the user to use three and five fingers respectively. These numbers were chosen to differentiate the start of the gesture from the gestures that allowed the user to move and rescale the individual panels. These gestures enhanced the system by giving the operator control over what they were specifically looking at in terms of geography. This was considered useful for

operations where the UAV might have flown to a waypoint that was far away from its current position because it allowed the operator to gain an understanding of what kind of terrain was going to found within the area. Giving the operator more access to geographic information helped to increase their Situational Awareness. This was also particularly important for UAV operators because they were not located within the aircraft's cockpit and thus could not physically see the surrounding area.

Besides the gesture inputs, the map panel included a variety of smaller features which enhanced the awareness of the operator. The first was that any Ground Control Station located within the shown map area appeared as orange satellite dish icons. When this icon was tapped or clicked, an orange circle appeared around it. This circle indicated the communication range of that particular Ground Control Station. Within the current build, the Ground Control Station communication range had to be estimated since the simulation did not have any specific equipment setups. As such, a low range of seven kilometers was chosen. This range was chosen because it was the approximate distance from Camp Pendleton's airfield to the edge of the simulation's map. This feature allowed the operator to see how close the aircraft was to potentially losing direct line of sight connection with the Ground Control Station (without taking into consideration any satellites it may be communicating with).

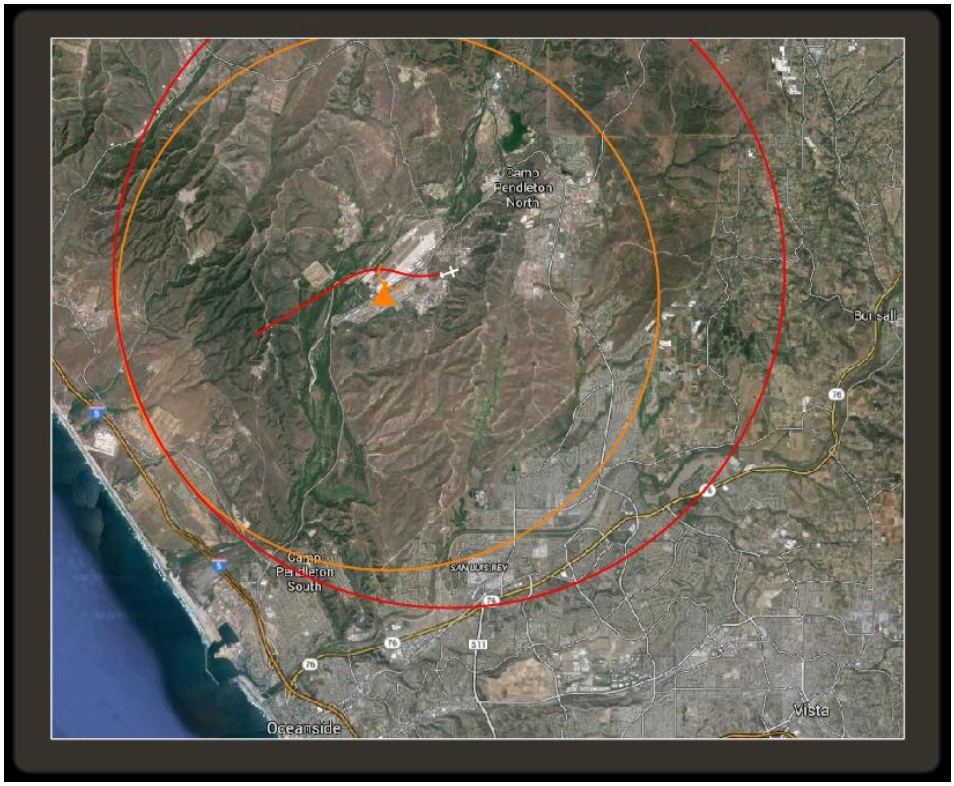

Figure 48- Gesture Controlled Map Instrument Panel

The next feature was the aircraft indicator. This indicator was a small aircraft shaped icon that showed the aircraft's current position on the map as well as its heading, both of which were updated in real-time for an accurate representation of the flight. The flown flight path was shown as a red line that trailed the aircraft and updated in real-time giving an accurate representation of where the aircraft had already flown. This feature would be particularly useful for Search and Rescue type operations where the aircraft might have to search an area, as the operator can directly see the area the aircraft has already covered. This pathline was created using a graphical entity called SVGPathline. It utilized a defined number of points, determined by the designer, where each point corresponded to a unique point on the line based off of an X and Y coordinate. To create an expanding and dynamic path, a timer had to be used. Whenever the timer repeated its cycle time (which was set to half a second) it redefined the total number of points by adding a single new point to the SVGPathline entity, which was then given coordinates equal to the aircraft's current position.

The final feature included in the panel was the addition of a graphic that displays the remaining range of the aircraft based upon the fuel level. This was shown as a red circle around the aircraft icon and could be displayed or hidden from view by tapping or clicking the aircraft icon. The circle itself represented how far the aircraft could travel based upon current fuel flow and how much fuel remained. The equations that were used to describe this distance were:

$$\text{Time Till Empty} = \text{Fuel\_Mass} / \text{Fuel\_Flow\_1} \tag{21}$$

$$\text{Remaining Range} = \text{GroundSpeed} * (\text{Time Till Empty} * 3600) \tag{22}$$

Where Time Till Empty was the amount of time (in hours) that the aircraft could fly until it ran out of fuel, Fuel_Mass was the total mass of fuel onboard the aircraft (in kilograms), Fuel_Flow_1 was the first engine's fuel flow rate (in kilograms per hour) (Note: The fuel flow could be defined per engine within FlightSIM since any aircraft could be designed to have a total of four onboard engines), Remaining Range was the distance the aircraft could roughly fly before it would run out of fuel (in meters), GroundSpeed was the aircraft's Ground Tracked Speed (in meters per second) and thus 3600 was used to convert the Time Till Empty variable from hours to seconds.

In terms of graphically describing this, it was found that 1000 units within VAPS XT was roughly equivalent to 1000 meters. As such no conversion was needed to convert the range which was in terms of meters to the drawn units within VAPS XT.

5.2.8 Non Gesture Controlled Map

The non-gesture controlled map (shown in Figure 49) had all of the same features as the gesture controlled map except that it did not have the ability to pan or zoom in/out of the map via gesture inputs. Instead, the entirety of the flyable simulation area was shown within a significantly larger instrument.

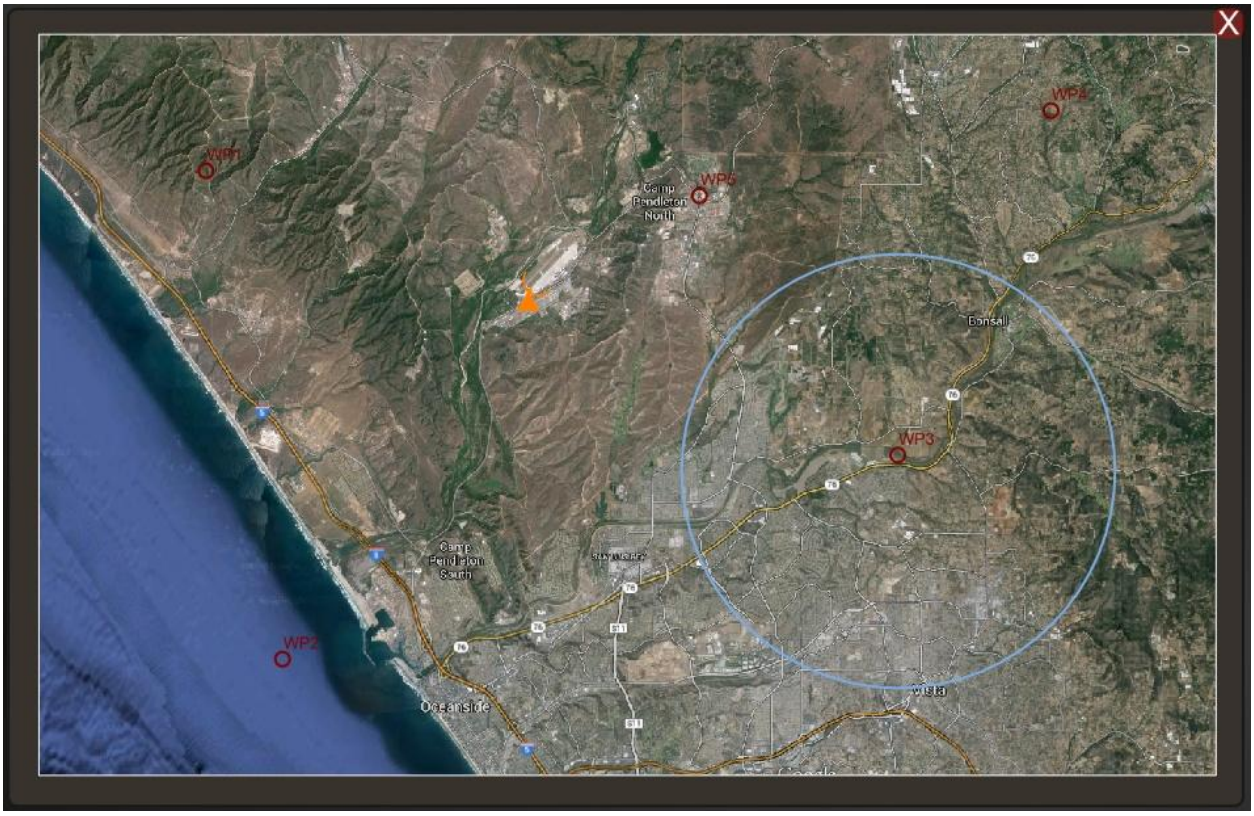

Figure 49 - Non-Gesture Controlled Map Instrument Panel

Giving the operator access to two different styles of maps helped to promote the goal of reconfigurability because the operator could choose how they wished to view the surrounding area. The gesture controlled map took up much less space on the screen allowing the user to have more instruments on the same screen. However, the trade-off for this was that the map also required the user to interact with it more often because they needed to pan it around. The non-gesture map took up significantly more screen space, but did not require the user to move the map around to view the surrounding area allowing them to spend more time focusing on monitoring other aspects of the aircraft's flight.

5.2.9 Payload Controller

The fictional payload controller (which can be seen in Figure 50) was included in the final Ground Control Station to provide an instrument panel that could showcase how touch gestures could be applied to more mission specific instruments. The payload controller that was designed was meant to roughly simulate the neutralization of a random High Value Target. Ultimately, the user had to identify the proper target, arm the payload and then neutralize it without hitting any local friendly targets.

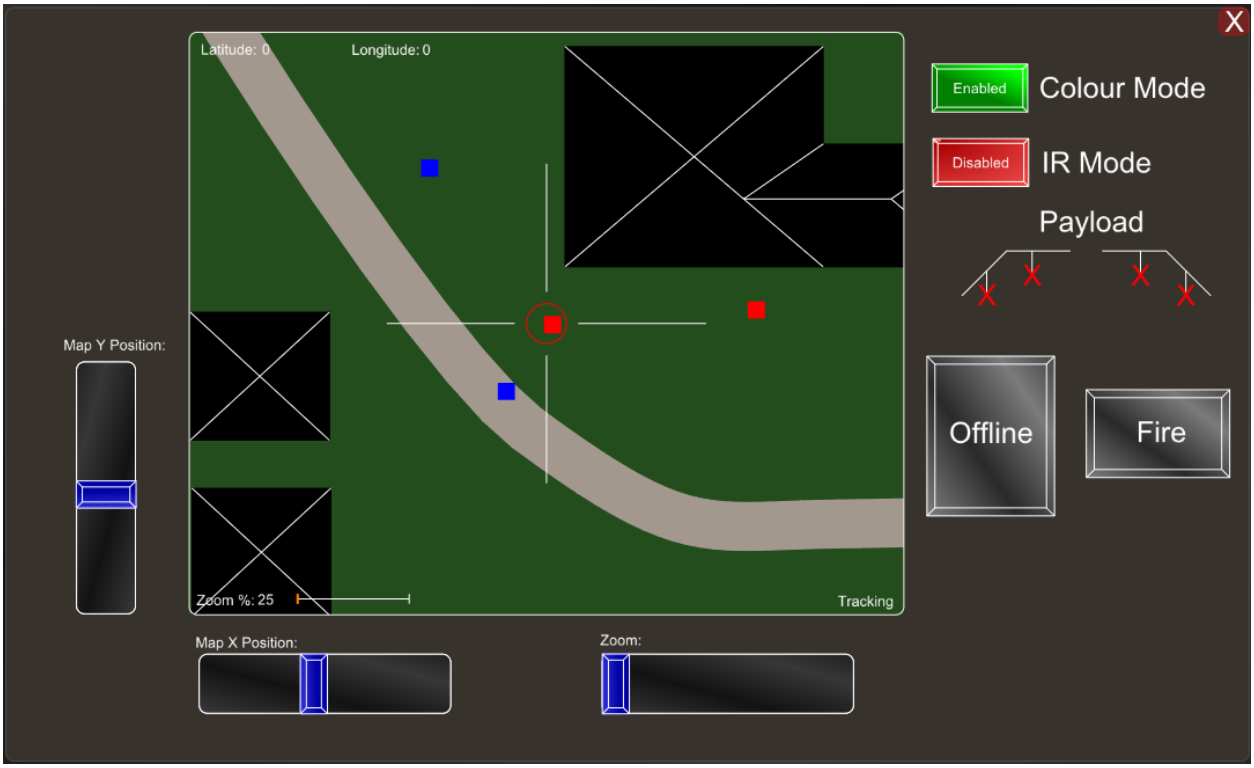

Figure 50 - Fictional Payload Controller

In the center of the panel was a very simple graphical map with seven coloured boxes. This map was meant to act as the fictional camera that could be used by an operator to track a target. A map had to be drawn up because VAPS XT does not have the ability to directly import live video feeds outputted by FlightSIM. The boxes represented the potential targets, of which three were blue and considered friendly targets, and four were red and labelled as enemies. These boxes roamed the simple map through a series of timed commands. Each box moved in a linear fashion between the various predetermined points (i.e. in a straight line from point A to B) over a set amount of time. Due to the fact that VAPS XT does not possess any distinct AI functionality, each of these movements had to be defined by a set of individual linear equations that were each a function of time. Equations 23 and 24 present a

portion of these linear movement equations. These two equations use the first target box as an example, between its timed movement cycle of 1600ms to 2400ms (the full movement cycle is shown in Figure 51):

Target_1.Position.X = -39295.6+(10.427*(Movement_1.Time-1600)) (23)

Target_1.Position.Y=1353.8+(-18.468*(Movement_1.Time-1600)) (24)

Where the Target_1.Position was the first box's position with respect to X and Y, and Movement_1.Time was a timer variable that was used to create a constant looping motion for the boxes.

Both the X and Y position equations were essentially just linear equations in the form of Y=aT+b, where in this case Y is the position in terms of either X or Y, T was the time determined by the timer (with the segment's starting time being subtracted from it), the "a" component was the slope of the linearized path, and the "b" component was the starting point of the tracked movement.

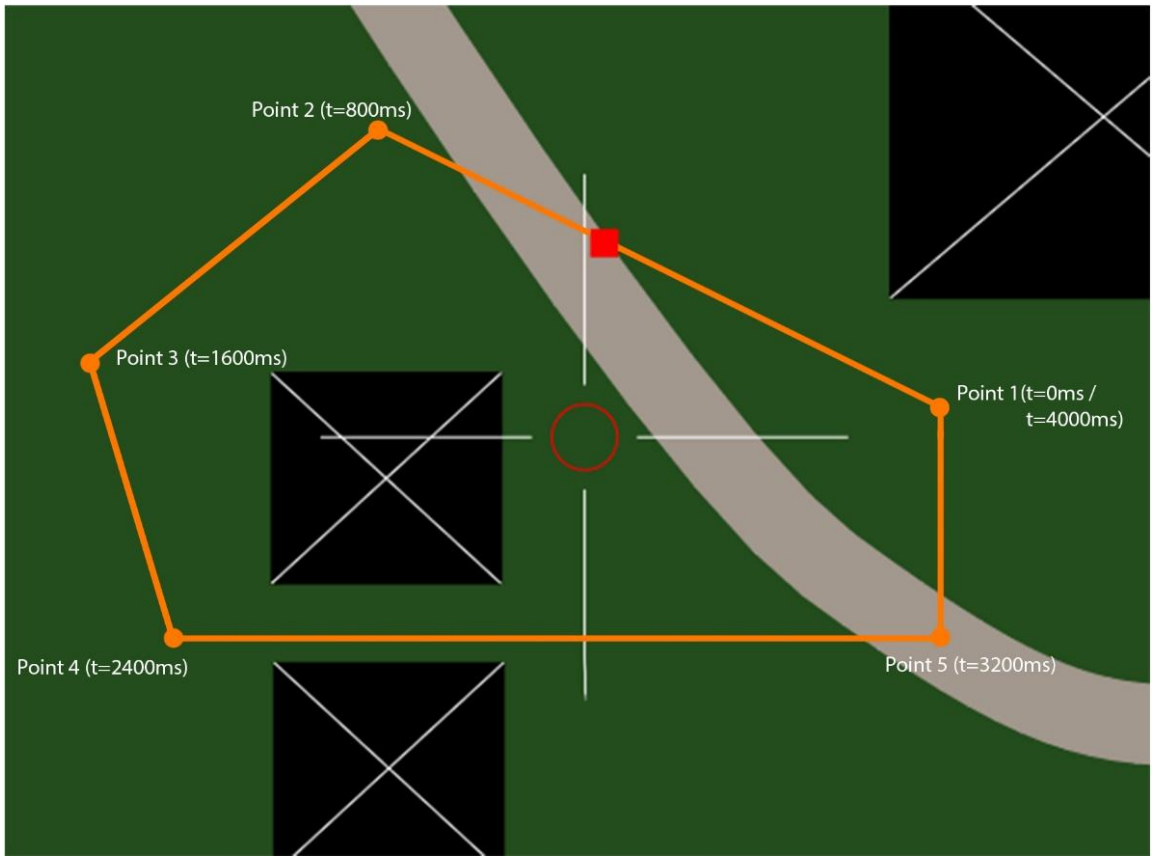

Figure 51- Target Box 1's Movement Pattern With Associated Times

Next to the map, there were four buttons. The top two changed the "colour mode" of the fictional camera. The first mode was a standard colour mode making all of the objects and the map colourful. The other mode was a fake "Infa-Red" mode (shown in Figure 52) where everything was set to a greyscale colour scheme. The two colour schemes were implemented to create an added level of difficultly for determining the High Value Target. Of the four red enemies that would roam the map, only one was considered the High Value Target. This specific model was determined by switching over to the greyscale map where a black flashing light would appear on the High Value Target. However within this mode the user could not see which moving targets were friendly. As such the user was forced to use both colour modes which promoted UI interaction.

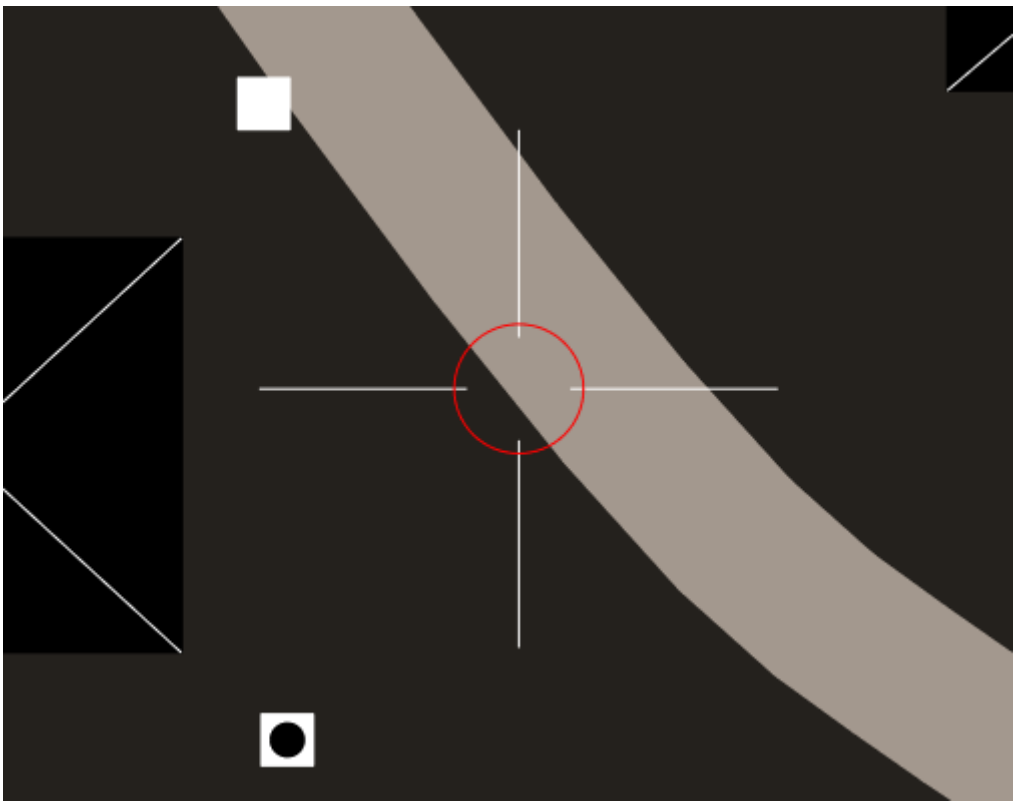

Figure 52 - Greyscale Map with High Value Target (Lower Box) and Regular Potential Target (Upper Box)

The second pair of buttons dealt with the arming and firing of the payload. When the user was within a pre-defined area of the flyable map area that represented the area where the High Value Target would be located (denoted on both sets of maps as a light blue circle) the operator could click/ tap the "Arm" button. At this point the second button would change colour from grey to red. This button was labelled as "Fire" and allowed the user to launch the payload. The payload would hit the map in the center of a set of crosshairs that was always in located within the center of the displayed map. In

addition, the “blast radius” of the launched payload was shown as a red circle within the crosshairs and resized if the map was zoomed in or out of. Whenever the fire button was pressed the system would also expend one of four payloads, and was updated on a graphic (denoted by the removal of a single red X under the “Payload” heading). If all four payloads were launched without hitting the High Value Target the mission would be considered a failure.

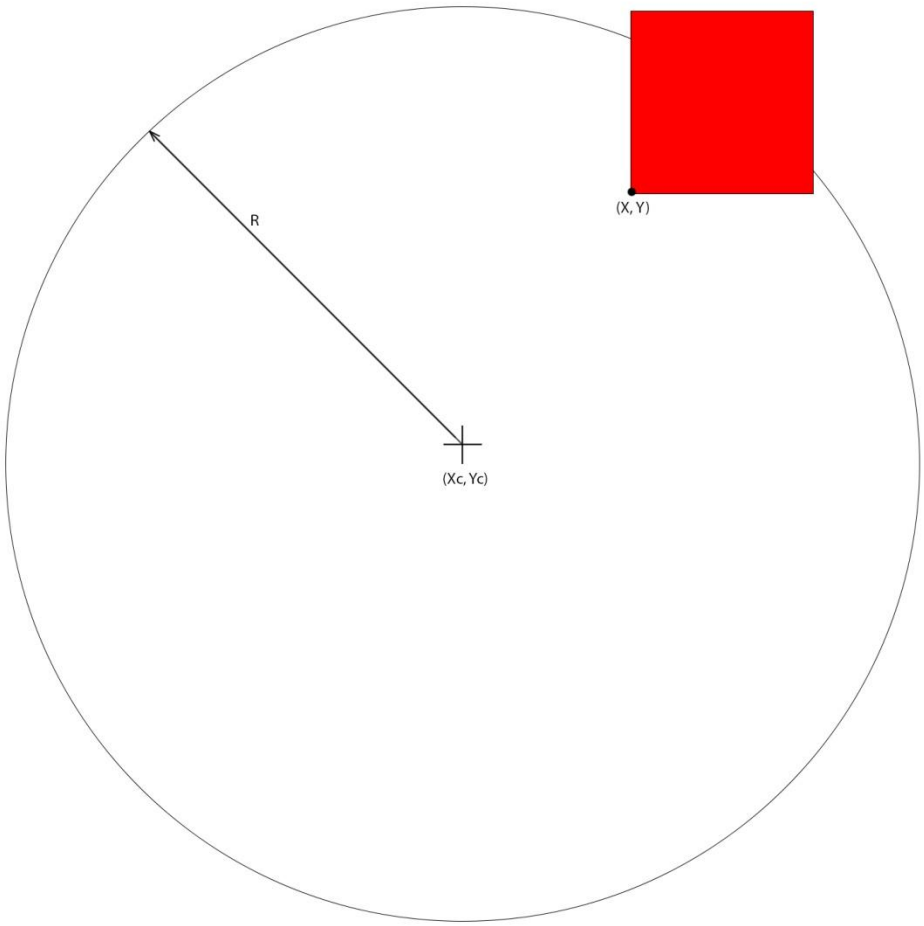

Figure 53- Trigonometry of Determining if a Point is Within a Circle

To decide if a target was hit by the payload blast (a defined circle), and to see if the aircraft was actually within the specified flight region(also a defined circle) to arm the payload trigonometry had to be used as shown below in Figure 53. The following equation was used to see if a point was found to be within the blast circle:

$$(X-X\_c)^2+(Y-Y\_c)^2<R^2 \quad (25)$$

Where X and Y were the test point's X and Y coordinates respectively, X_C and Y_c were the coordinates for the center of the circle that the point was being tested against, and R was the radius of the circle.

If this equation proved to be true, then the point was indeed within the circle. Whenever the user hit the "Fire" button, all of the moving target boxes would be immediately tested to see if any had a corner that was within the red blast circle's radius.

The panel also had three sliders that could be dragged with either a touch gesture or by clicking and dragging it with a mouse. These sliders were the same as the custom made ones described in section 5.2.5 (Flap Controller). These sliders directly controlled where on the map the camera was centered. Two of them controlled the X and Y coordinates and the third changed the level of zoom. These sliders were included so as not to limit the use of the panels to gesture inputs. However, the ability to pan around the map and zoom in/out could also be done with gesture inputs. These gestures helped to show how a camera could be naturally controlled through such inputs. The ability to drag or zoom the camera to the exact location the operator wished to view was simple, efficient, quicker, and more accurate than using a slider. Both the sliders and gestures were coupled in this panel. When the map was dragged around with a gesture, the associated X and Y sliders would move accordingly. This helped to create an environment where either input could be used or switched out instantaneously. An example of the coupled equations that were used for each of the sliders is shown in equations 26-27. These equations were based off of the zoom slider which rescaled the map between 25% (0.25) and 100% (1.0). In addition, the format of equations 26-27 was very similar to the other slider that was previous discussed in section 5.2.5, in the sense that they were also linear in nature with respect to the starting and ending slider position (or gesture position).

The clickable/ drag-able slider equations were:

SliderNew_2.Position.X = -3060+((SliderHoriz_1.CurrentValue-0.25)*9373.33 (26)

SliderHoriz_1.CurrentValue = ((SliderNew_2.position.X+3060)/9373.33)+0.25 (27)

Where SliderNew_2.Position was the slider graphic's position along the track (it started at -3060 and went to 3970), and SliderHoriz_1.CurrentValue was the scale that was being outputted by the slider.

As can be seen, these two equations were directly coupled thus and needed to be solved in unison (which was not a problem for a computer that was computing the values)

The Pinch to Scale equation was:

SliderHoriz_1.CurrentValue=(e.currentPosition.X+3060)/4017.143)+0.25 (28)

Where e.currentPosition.X was the centroid of the pinch gesture.

As can be seen, this equation also changed the value of the outputted scale value, thus as the gesture was completed, equations 26 and 27, would also be updated so that the scale slider did not get de-synced.

The actual map scale was proportional to the outputted scale value (solved by equations 27 or 28) and thus can be found using:

MapArea.Scaling.X =SliderHoriz_1.CurrentValue (29)

MapArea.Scaling.Y =SliderHoriz_1.CurrentValue (30)

Where MapArea.Scaling is the scale of the camera's map graphic.

Both the X and Y scaling factors used the same value to ensure that the object was scaled proportionally to keep the graphical fidelity from being reduced/ skewed.

5.2.10 Primary Flight Display

The Primary Flight Display was a non-interactable panel that showed the aircraft's altitude, heading, Mach number, pitch rate and velocity, and can be seen in Figure 54. The layout of the Primary Flight Display was based around common industry Primary Flight Display layouts. The heading was shown on a wrapping tape display found at the top center of the panel. This tape was a pre-made VAPS XT entity that was altered to include the values for the Primary Flight Display. The heading was also shown in degrees with major markings every ten degrees and semi-major markings every five degrees. Every 90 degrees, the associated compass bearing was shown (0 was North, 90 was East, 180 was South, and 270 was West) to help the operator quickly determine their aircraft heading. To make the displayed heading more accurate, a numerical heading value was added to the pre-made tape. This was added by adding in a simple rectangular graphic above the tapes tick marking in order to have the markings not appear where the numerical value is being shown. In addition, the graphic included a small arrow in the lower middle portion of the rectangle to show to the user where along the tape the value being shown

was from to avoid any potential confusion from the operator. Finally the text within the numerical representation was shown in a bright green text within the middle of the displayed tape so that it could be easily read.

The middle of the panel displayed the Attitude Direction Indicator showing the aircraft's pitch in increments of five degrees. The Attitude Direction Indicator was another pre-made VAPS XT entity that was altered. The original entity had a white artificial horizon that separated two solid coloured rectangles, one blue and one green. The only other thing included within the original entity was that it included some major tick marking for the Angle of Attack, however there were no values associated with the markings. The original Attitude Direction Indicator was as a result modified to be much more user friendly. These changes included an artificial horizon that depicted the ground as a brown gradient and the sky as a blue gradient to make it more visually appealing and were chosen to avoid confusion over what they represented. The orientation of the aircraft could also be seen from the "W" icon in the center of the Attitude Direction Indicator. A number of additional pitch rate tick markings were also added to increase the accuracy while also adding in the numerical value next to each major tick marking (each major tick mark was an increment of 10, and each minor tick mark was each to five degrees) so that the operator could know what their pitch rate was set to. These pitch markings all re-orientated themselves in real-time around this icon to show the operator how the aircraft was positioned with respect to the environment. All of these small improvements helped to improve the operator's performance and to help improve their situational awareness through the addition of the aircraft's orientation and added flight data.

To the left of the Attitude Direction Indicator was the velocity indicator which showed the aircraft's velocity in meters per second on a finite tape in increments of 100m/s. Like the wrapping tape, the numerical value of the velocity was added to the finite tape in green text to again help to make it stand out and be easily read. Just below the tape is the Mach number indicator which displayed the aircraft's Mach number as received from FlightSIM's "expBuffer" protocol.

Finally, to the right of the Attitude Direction Indicator was the altitude indicator. This indicator was the same as the velocity indicator in that it was a finite tape showing the exact numerical altitude in the center. The indicator also increased in steps of 100. However, unlike the velocity indicator, the change in altitude was shown directly under the tape represented by the Greek letter Delta, Δ, and was used to show the operator how quickly they were either ascending or descending.

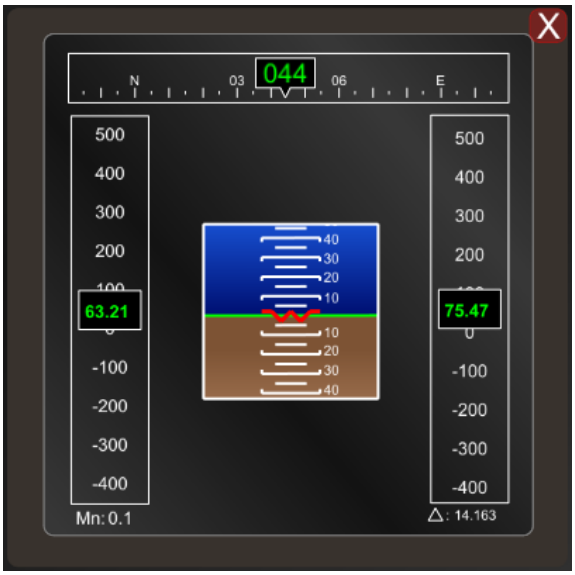

Figure 54 - Primary Flight Display Instrument Panel

5.2.11 Simple Engine Monitor

The simple engine monitor instrument panel (shown in Figure 55) was used to display the key engine parameters within the confines of a small panel. It showed the operator the aircraft's fuel flow rate, throttle position, Low Pressure Compressor RPM, High Pressure Compressor RPM, Turbine RPM and the overall thrust being produced by the engine. All of these values were updated in real-time based upon the values that were received from FlightSIM's "expbuffer" I/O buffer. This panel was ultimately used to give the operator a general overview of the operation of the engine. It was also meant to act as an alternative to the significantly more descriptive "detailed engine monitor instrument panel" described in Section 5.2.4, in the sense that it could be easily placed onto any of the virtual tabs without taking up a lot of room like the detailed engine monitor instrument panel.

Figure 55 - Simple Engine Monitoring Panel

5.2.12 Throttle and Gear Controller

The final instrument panel included within the Ground Control Station was the throttle and gear controller, which can be seen in Figure 56. This panel served as a digital throttle so that a physical one did not have to be used. The digital throttle was ideal for operating environments where space was limited. The throttle could be dragged like all of the other sliders and would send a value ranging from 0 (engine off) to 1 (engine at full) through the "sndThrottle" I/O buffer to FlightSIM so that the aircraft knew that it was acting as the throttle. The equation used to determine the throttle setting was described as:

Throttle Value = SliderNew.Position.Y/7000 (31)

The throttle slider ultimately used the same linearization process and slider model that the other sliders had used to ensure that the send value was within the range that FlightSIM needed (which for the throttle was between 0 and 1). Unlike most of the other sliders that the Ground Control Station used, this slider was positioned vertically. As such, the starting position was at a Y-coordinate of 0 and ended at 7000 units. Thus, the position just had to be divided by the maximum Y-coordinate to achieve the proper values. In addition to the throttle slider, the exact throttle percentage was shown just below the slider to ensure that the operator was setting it to the proper location. This helped to alleviate the possibility of the operator experiencing an adjustment error, because it would show the exact value the throttle was set to instead of an estimated position. In addition, showing the throttle position percentage helped to mitigate some of the issues with using a virtual throttle over a physical throttle. A physical throttle gave the added advantage of having haptic feedback in terms of how easily it was to able to be moved to a new position as well as giving the operator an idea of the throttle setting from its physical position. The virtual throttle on the other hand does not have any haptic feedback due to the touch based nature of the instrument. This lead to a slider which could be very quickly and easily moved, which as a result made it much more difficult to set it to a very specific position. Thus showing the throttle position as a direct value underneath the slider served to improve the operator's effectiveness when utilizing this instrument.

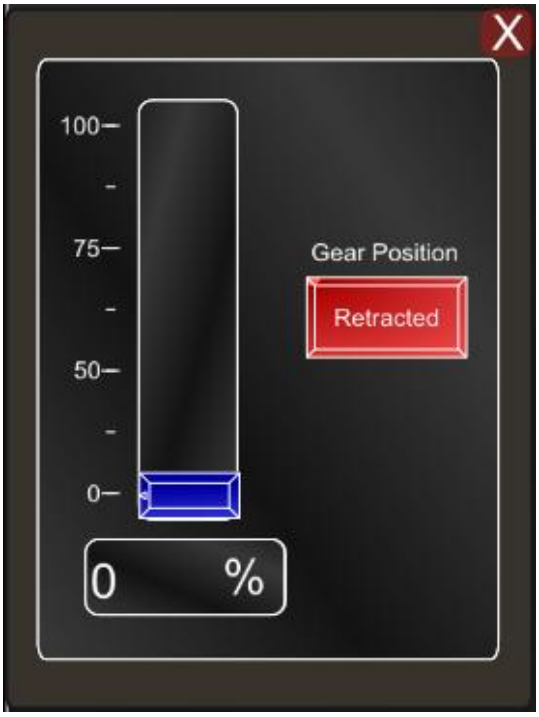

Figure 56- Throttle and Gear Controller

The gear controller allowed the user to extend or contract all of the aircraft's landing gears at the press of a single button. This command was sent through the "sndGear" I/O Buffer to FlightSIM and was equivalent to 1 when the gears needed to extend or 0 when they need to be retracted. The ability to extend or retract the gears was added to the throttle instrument panel because it was a feature required for both take-off and landing. Thus the operator would always be using the throttle when they needed to set the position of the landing gears. With this feature the user did not have to search the UI for the button to initiate the gear position process which ultimately helped to reduce their reaction time.

5.3 Simulator Tutorial

To showcase how all of the included features and instrument panels worked in conjunction with one another to create a unique and effective user experience, a tutorial example was created. The tutorial utilized an F-16 Fighting Falcon (described in Section 3.1) as the aircraft flight model, due to it having the most accurate autopilot system. The objective of the tutorial was to fly the aircraft through a series of five waypoints while also flying to a specified flight region, which was 8 kilometers in radius, in

order to utilize the fictional Payload Controller (described in Section 5.2.9) and eliminate a random High Value Target. The objective placement for the tutorial can be found in Figure 57.

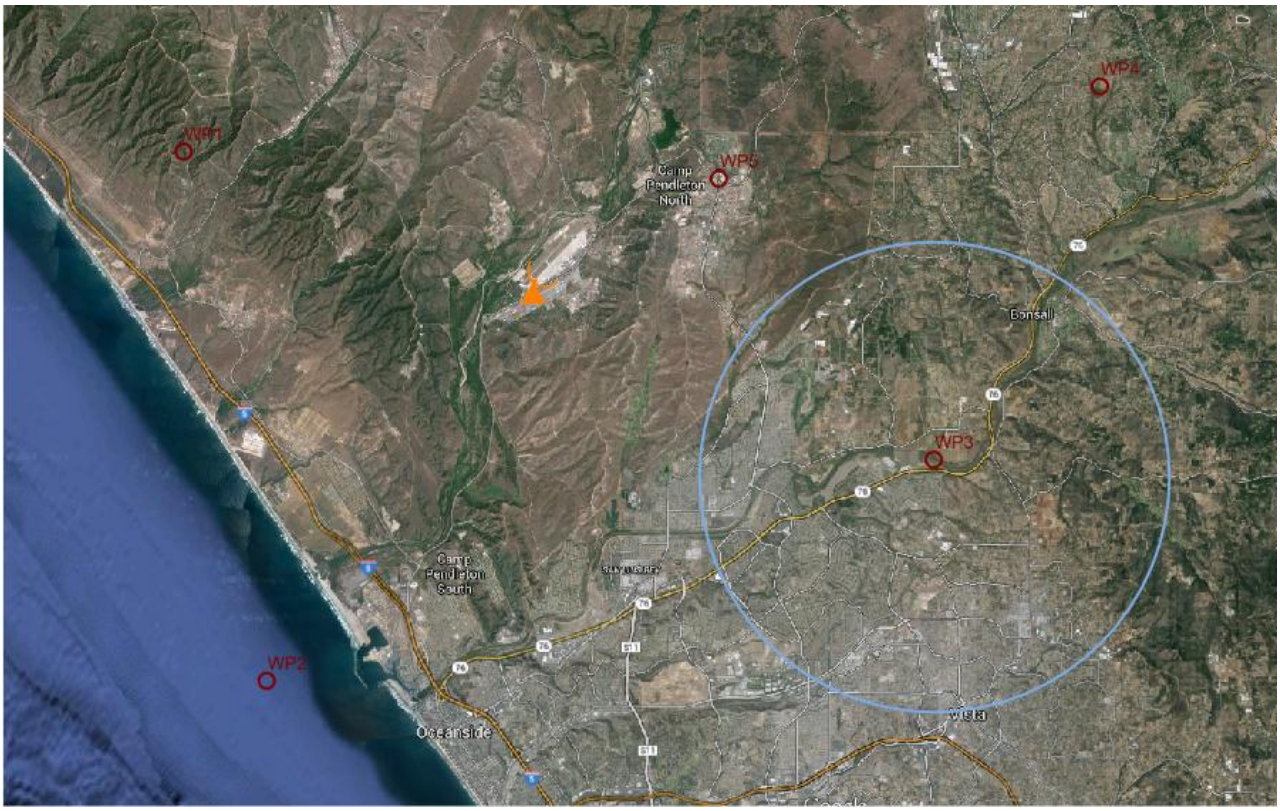

Figure 57- Tutorial Objective Placement

To determine if these objectives were met, three new I/O buffers had to be added to the system. The first was called “IsInRange” and was used to send a single float value command to the fictional payload controller (described in Section 5.2.9) when the UAV entered into the predescribed flight region. The second was “CompletionConditions” that was an array of seven elements, with each element corresponding to a different goal that had to be achieved. Element 0 corresponded to whether or not the High Value Target had been neutralized, and elements 1-5 corresponded to whether or not each of the potential five waypoints had been flown through or not. Finally the third I/O buffer was called “Test” and was a single float value that was used to change up which target within the Payload Controller would be considered the High Value Target. The value was set when the program was initialized in a small pop-up menu.

All three of these I/O buffers were added to the non-gesture map, which would be used to determine if the aircraft was within specific regions of the map. The reason the buffers were added to a

single map was because the functions of the map would still be running in the background of the program even if it is not visible on the screen. By limiting the coding to a single map it helped to reduce computational strains on the testing computer. The test buffer was used to determine the location and size of the flight region, as well as the number of waypoints that would appear. The "IsInRange" buffer was then set based upon the position of the aircraft icon on the non-gesture map, whereas the "CompletionConditions" buffer was solely used to determine when waypoints had been reached.

For both the flight region and waypoint conditions, the map could determine if the aircraft icon had flown into either the described flight region (graphically shown as a light blue circle on the maps), or if it had come within 200 meters of a waypoint (shown as either a maroon or vibrant green circle depending on if the aircraft had gone through it yet – both of which are shown in Figure 58). The map would determine if the aircraft had entered one of these regions by using equation 25 and the trigonometry found in Figure 53. The map would set the appropriate buffer value to either 0 (when outside of the payload zone or it had not yet flown through the waypoint) or 1 (when within the payload zone or when having flown through the waypoint).

Figure 58- Aircraft Has Not Flown Through the Waypoint (Left) and Aircraft Has Flown Through the Waypoint (Right)

Finally, in addition to the non-gesture map, the fictional payload controller had all three of the buffers added to it too. The test buffer was used to determine which red enemy target would be set as the High Value Target. The "IsInRange" buffer was used to see if the aircraft had entered into the payload zone so that the payload could be armed and fired. And finally the "CompletionConditions" buffer was used to tell the program if the user had successfully eliminated the High Value Target without hitting any blue friendly targets. If the High Value Target was hit and avoided all the friendly targets, the first element of the buffer was set to 1; if they had failed, it was set to 0.

The tutorial began with the aircraft on the runway at Camp Pendleton. When it was initialized the “Payload Control” preset tab was reconfigured from just the Payload Controller to that shown in Figure 59, which included a Primary Flight Display, to monitor the aircraft’s attitude dynamics, and the Virtual Throttle, to control the speed. This new setup allowed one to monitor the aircraft while they utilized the controller to ensure that it would not crash. In addition, a custom tab, shown in Figure 60, was created and personalized to showcase how one might utilize other panels. The custom tab was titled “Nav n’ Control” because within it was placed the Gesture Controlled Map to monitor the aircraft’s position, an Angle of Attack Gauge to monitor and ensure that the aircraft was not stalling, a Primary Flight Display to monitor its attitude dynamics, and a Virtual Throttle to control its speed. This personalized grouping of instruments, like the name suggested, allowed one to control and navigate the aircraft through all of the tutorial objectives.

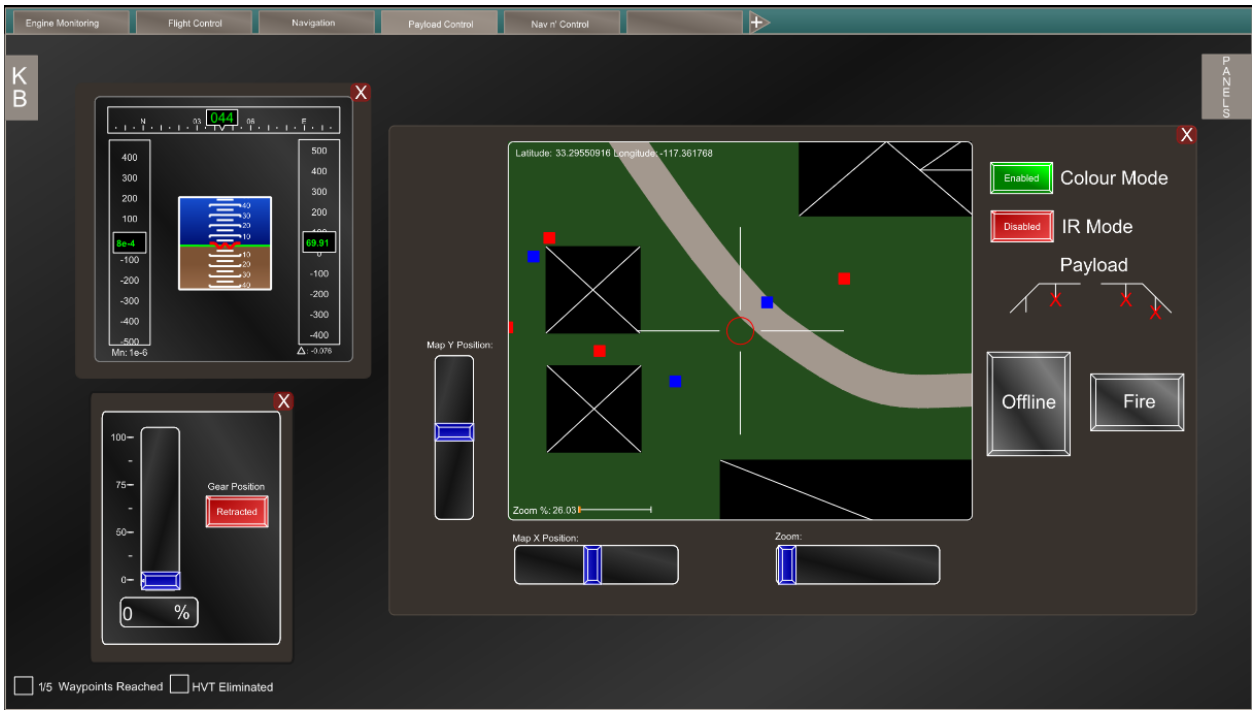

Figure 59 - Tutorial Payload Controller Setup

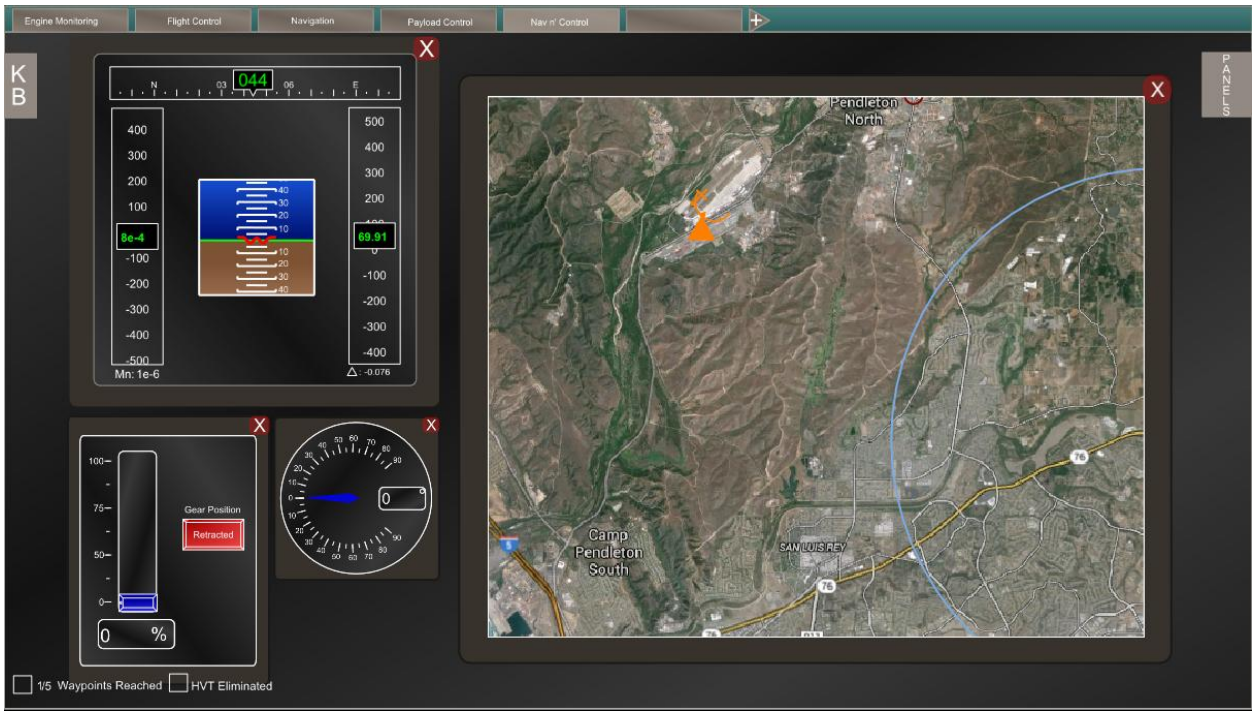

Figure 60 - Tutorial Custom Tab Setup

After the tabs had been configured, three waypoints were set within the Autopilot/ Waypoint Controller. The aircraft was not completely guided by waypoints to promote other features within the system. Ultimately the Autopilot was utilized to guide the aircraft through the second, third, and fourth waypoints. Thus leaving the take-off and landing to the pilot in addition to the waypoints that would come directly after, or before those events (i.e. waypoints one and five). The waypoints were added using both a standard keyboard for the first two waypoints and the virtual keyboard for the third and final waypoint. The addition of these waypoints can be seen in Figure 61. The three waypoints were all created as singular waypoints, and not maneuvers, due to the simple nature of the objectives, and the fact that the aircraft did not have to circle a location.

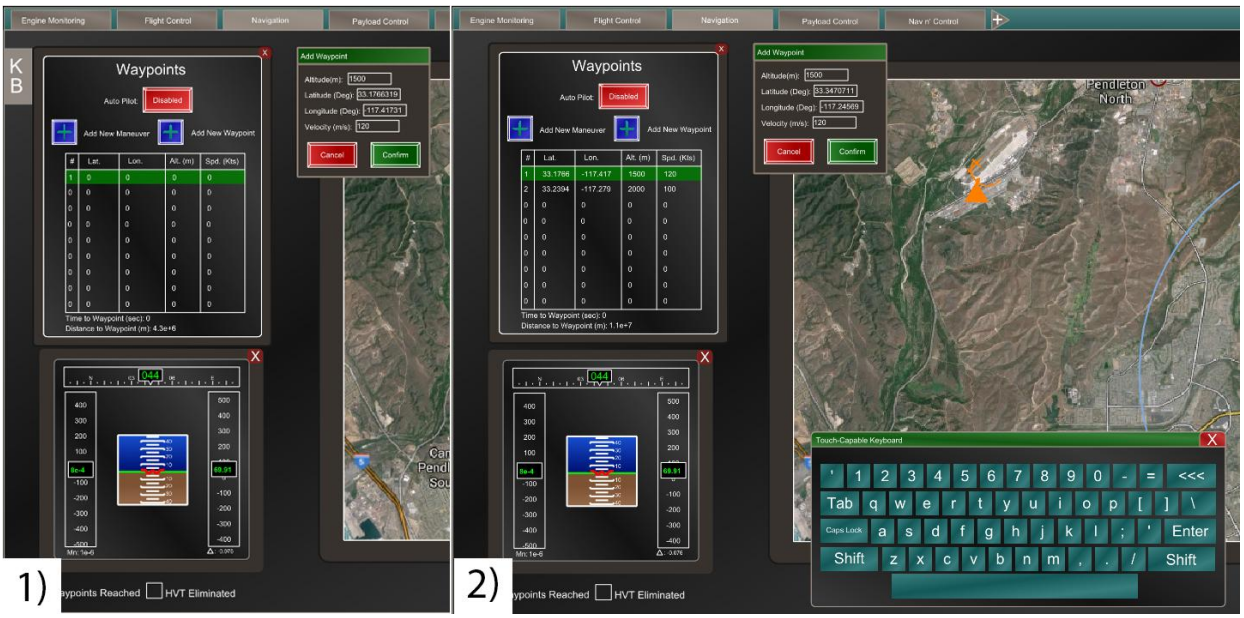

Figure 61- Tutorial Waypoint Creation – 1) First Waypoint (Keyboard), 2) Third Waypoint (Virtual Keyboard)

After the waypoints had been created, the aircraft was ready to fly. The custom tab was then switched to in order to control the aircraft upon take-off. The Virtual Throttle was then set to the 100% position, and once the aircraft was in the air the gears were retracted, as shown in Figure 62.

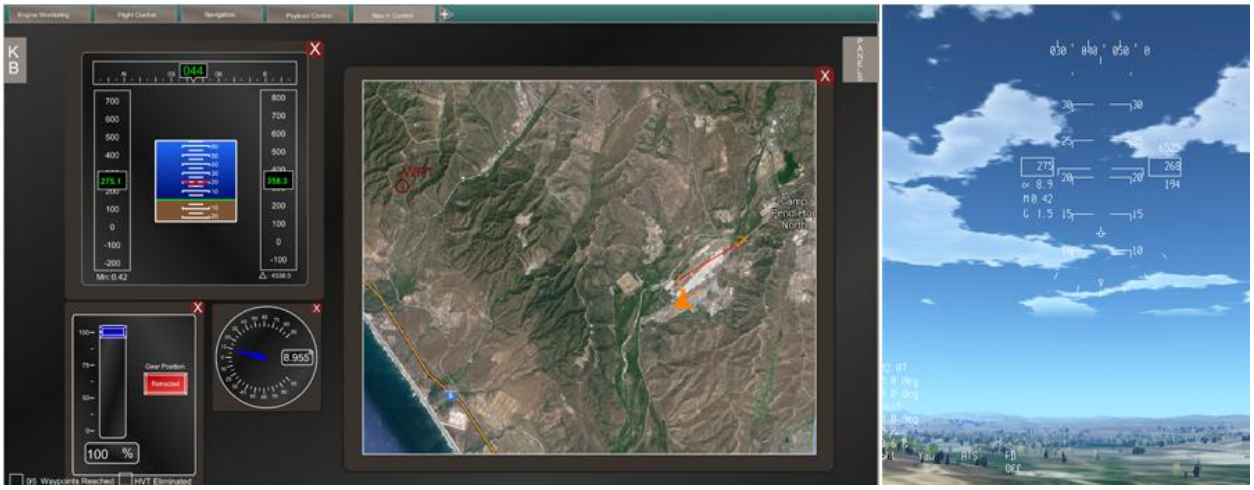

Figure 62 - Tutorial Take-Off

The aircraft then had to turn westward to reach the first waypoint. As the aircraft was manually turned to face this desired heading, it experienced a G-Load warning due to the sharpness of the turn, shown in Figure 63. This warning was engaged because the aircraft had exceeded a G-Loading over 2G's.

This warning prompted a less steep banked turn to get out of the warning zone. Ultimately this helped to correct the maneuver, which might have damaged an actual UAV should it have been sustained.

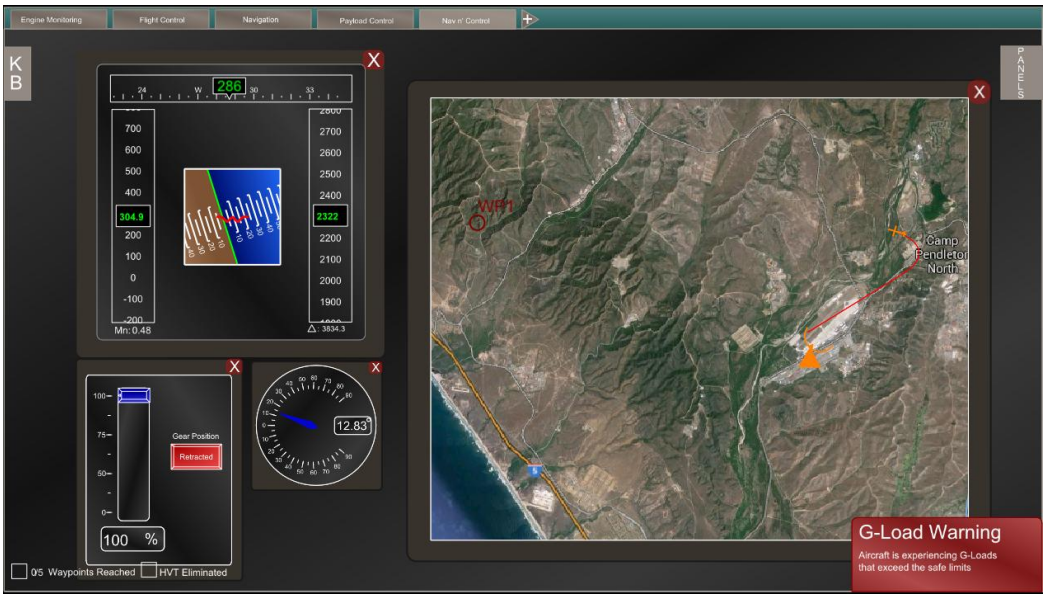

Figure 63 - Tutorial G-Load Warning

After the warning had been dealt with, the aircraft approached the first waypoint. Initially the Gesture Controlled Map was relatively zoomed out, showing the aircraft and a wide area around the waypoint. This caused the aircraft to not be properly lined up with the waypoint and thus caused a miss. As a result, the aircraft had to fly back around in order to properly reach the waypoint. During the second attempt, the Gesture Controlled Map was zoomed in further to help navigate the aircraft through the small waypoint zone. This process can be seen in Figure 64.

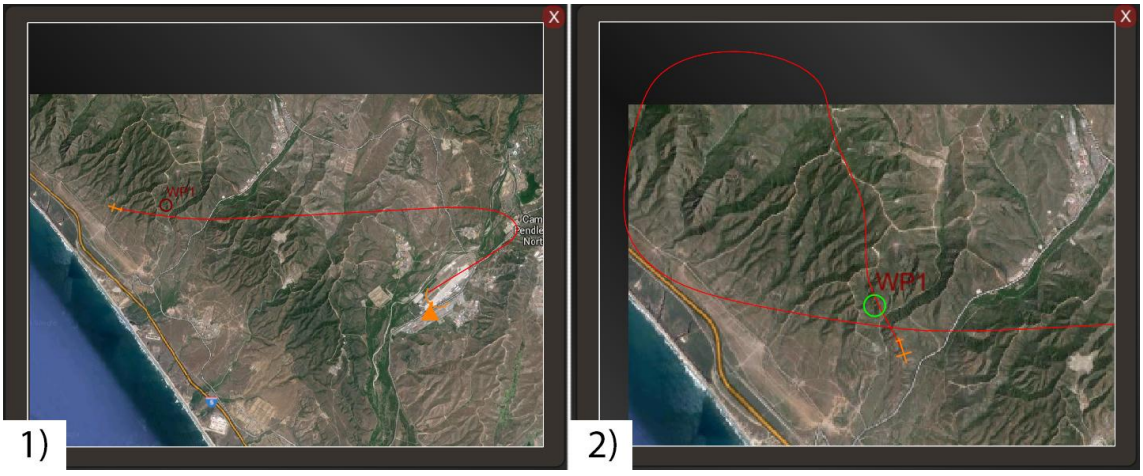

Figure 64 - Tutorial First Waypoint – 1) First Attempt, 2) Second Attempt

Once the first waypoint had been reached the Autopilot was turned on to guide the aircraft through the next three waypoints. This was accomplished by going into the “Navigation” preset tab and then tapping on the Autopilot button to engage it, as shown in Figure 65. Once the Autopilot was engaged, the first waypoint would be highlighted by the green bar to show that it was the current waypoint.

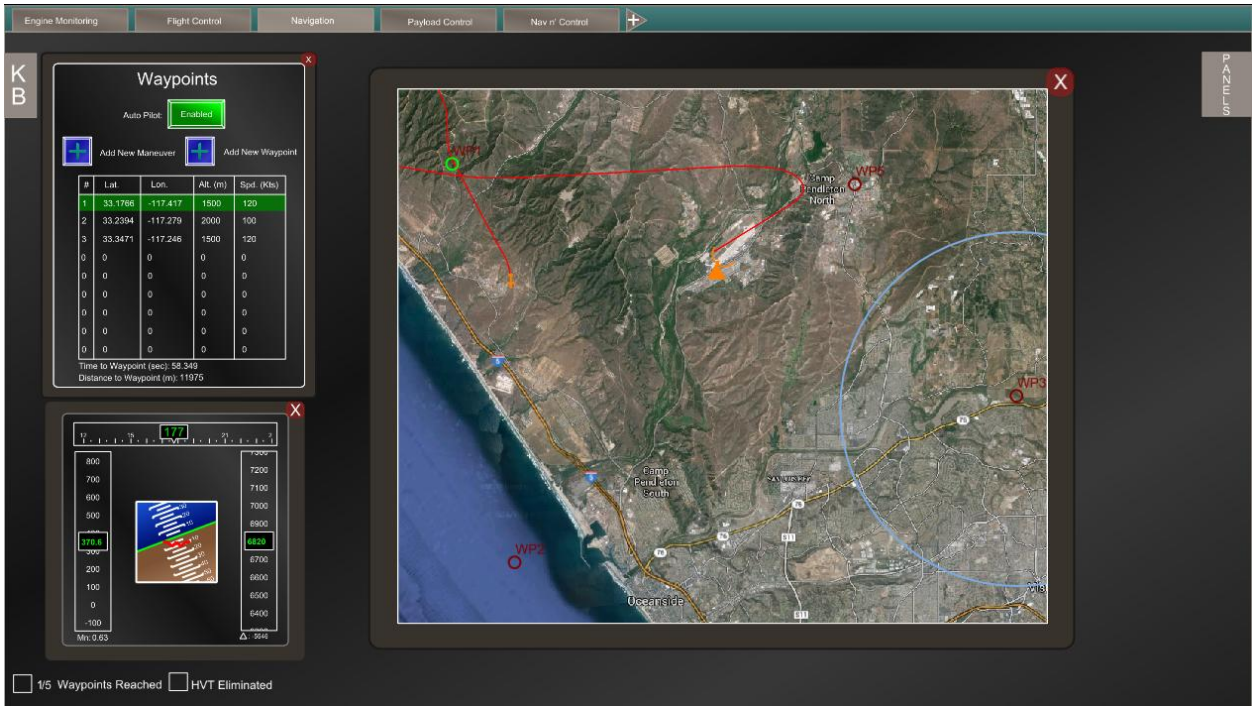

Figure 65 - Tutorial Engaging Autopilot

At this point the aircraft did not need to be directly controlled, thus providing an opportunity to check on the status of the aircraft. This was completed by going into the “Engine Monitoring” tab, shown in Figure 66. Utilizing this sole tab allowed for a more efficient monitoring of flight variables because it stored all of the engine related instruments in addition to having the complete General Aircraft Parameters panel. It should be noted that the chosen aircraft did not simulate its engine’s Low/ High Pressure Compressor’s RPMs or the Power Turbine’s RPM, thus they showed up as zero and in need of fixing, when in reality there was no issue actually present with these components

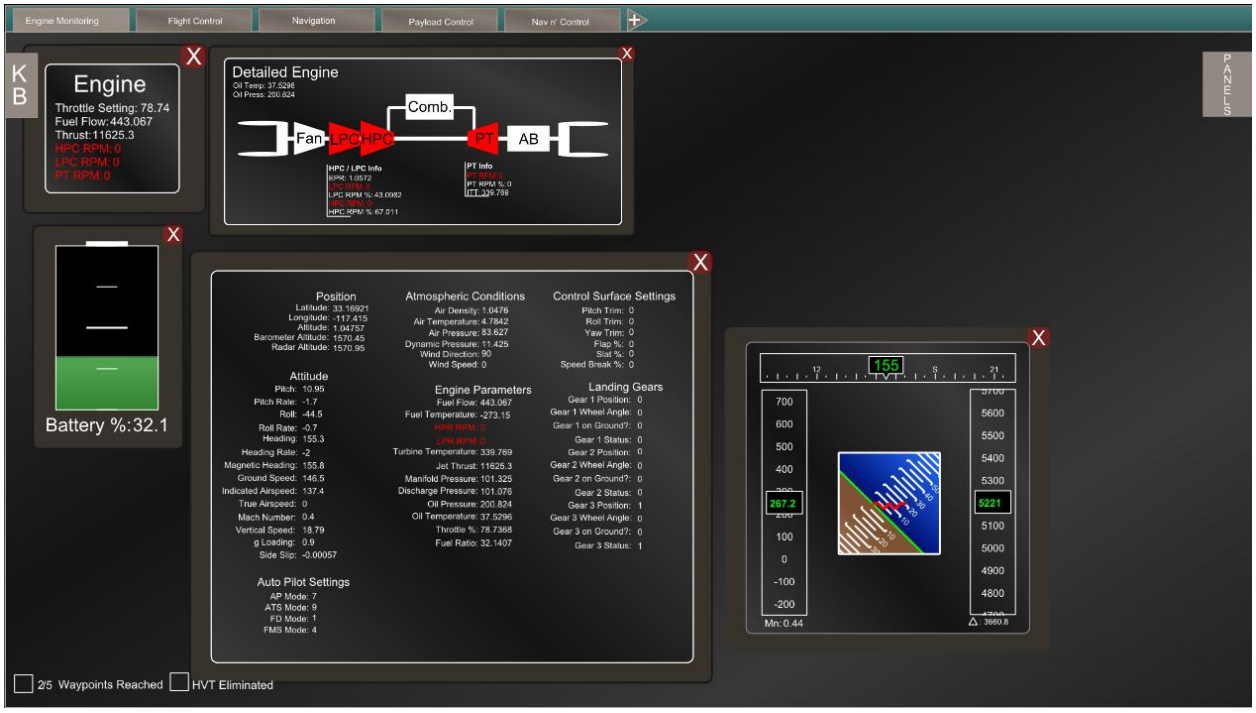

Figure 66 - Tutorial Engine Monitoring

The aircraft then flew through waypoint two without any trouble and began to turn towards the third waypoint, as shown in Figure 67 , which happened to be located at the center of the area where the Payload Controller could be used. As a result, the engine and flight variables were checked once more, utilizing the same procedure as before, to ensure that the aircraft was not going to experience an error while utilizing the payload controller. Finally before switching to the Payload Controller, the "Navigational" tab was swapped to, to ensure that the aircraft was actually within the desired area, shown in Figure 68.

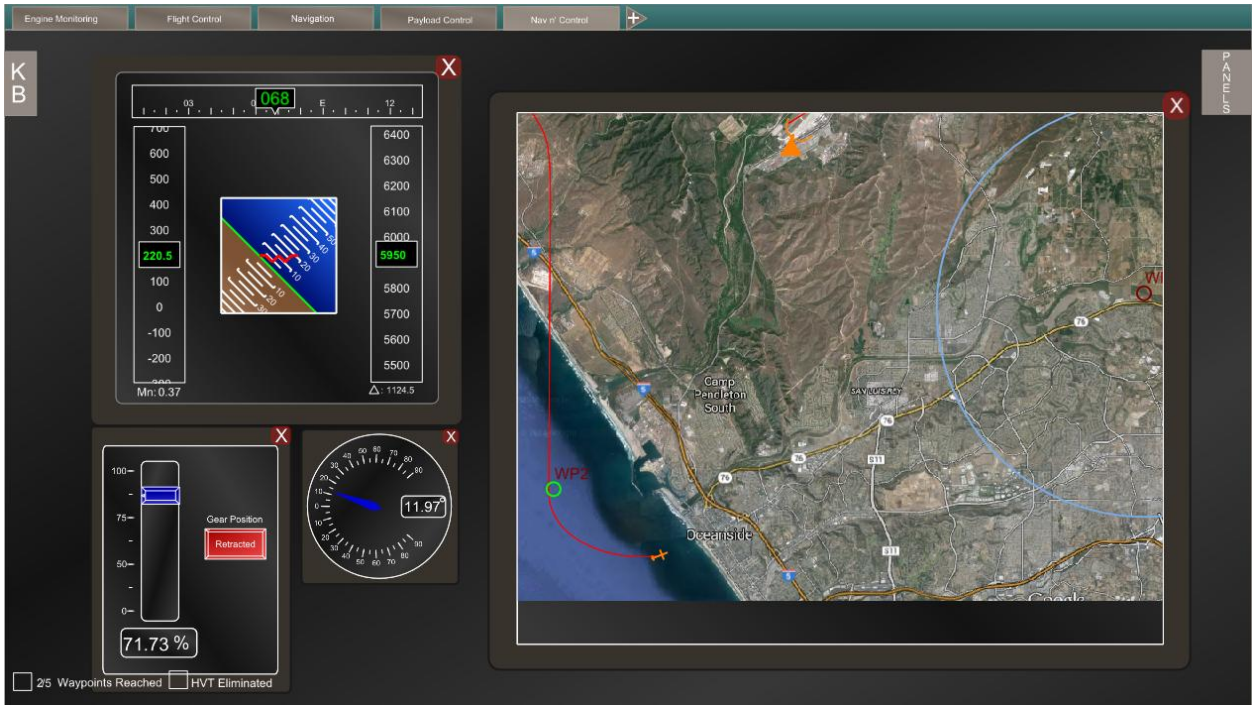

Figure 67- Tutorial Second Waypoint

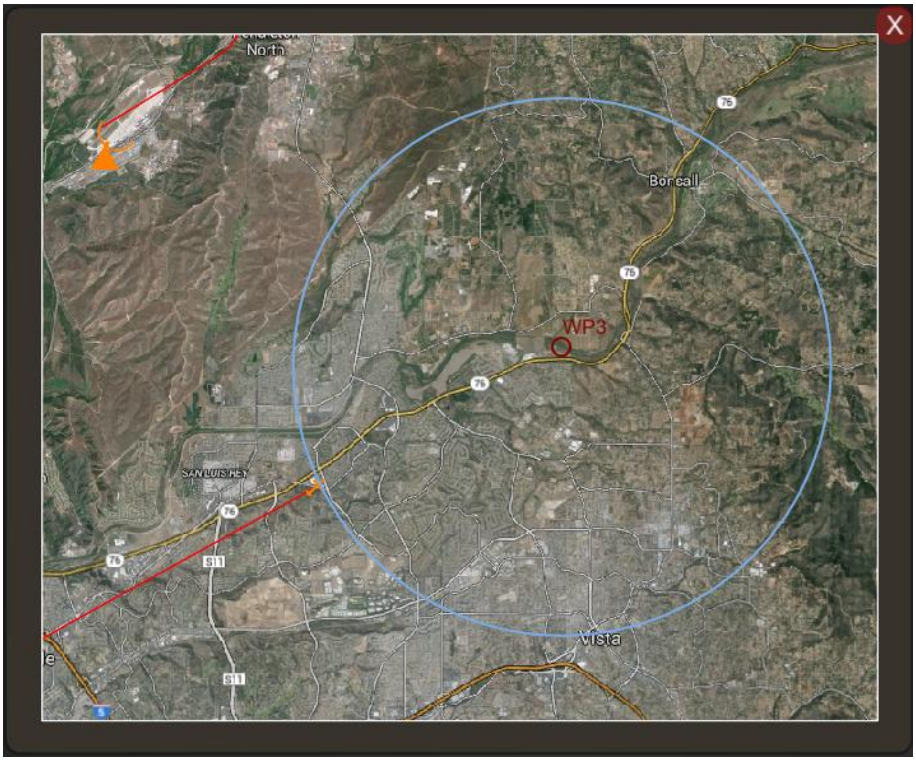

Figure 68 - Tutorial Aircraft Location Check 1

Once the location of the aircraft was confirmed to be within the Payload area (defined by the light blue circle) the UI was swapped to the reconfigured “Payload Controller”. At which point the colour mode was immediately set to IR Mode to determine which target had to be eliminated (shown in Figure 69). It was found that the High Value Target was in the upper right hand corner, denoted by the flashing black dot.

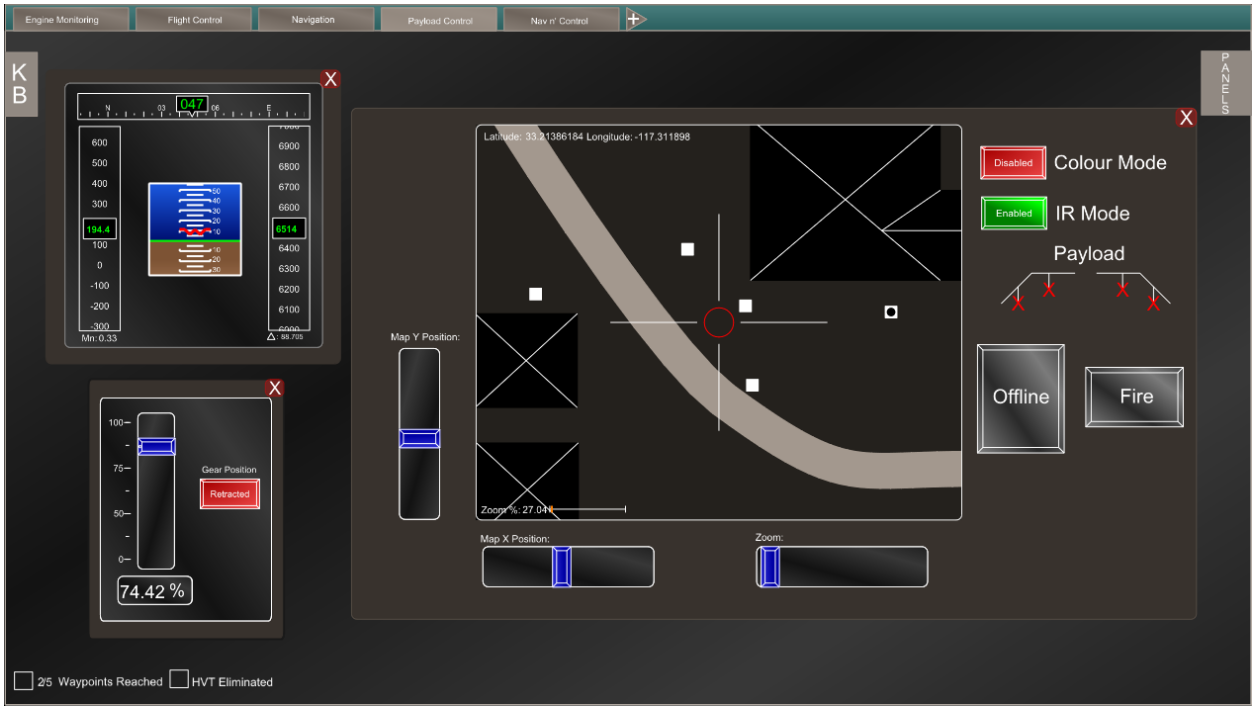

Figure 69- Tutorial Payload Controller IR Mode

The payload target was then zoomed into utilizing a pinch gesture to get a more accurate firing solution. The payload was then armed by tapping the arm button. Finally the fire button was tapped as the target approached, however the timing was slighting off resulting in a miss, shown by the missing “X” in the Payload Controller Figure 70. As a result a second shot had to be taken, which resulted in a successful hit, and elimination of the High Value Target (shown in Figure 71).

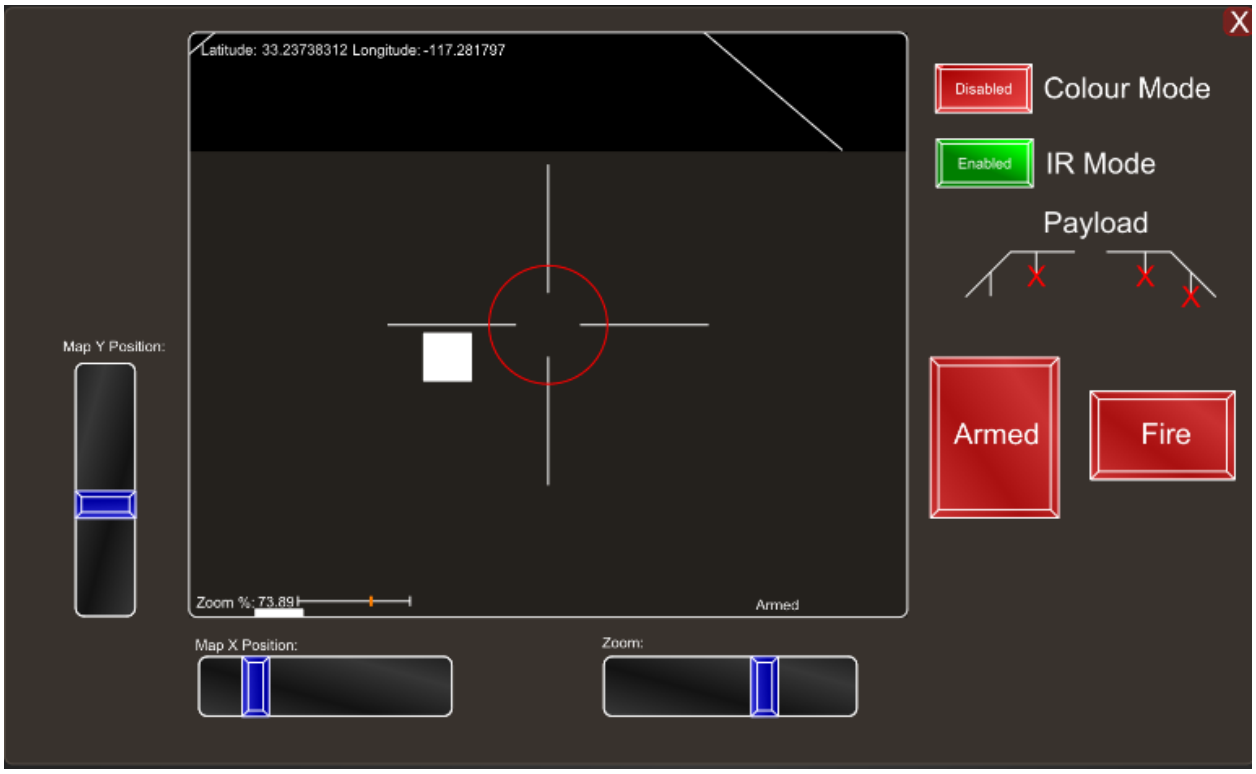

Figure 70 - Tutorial Payload Controller Missed Shot

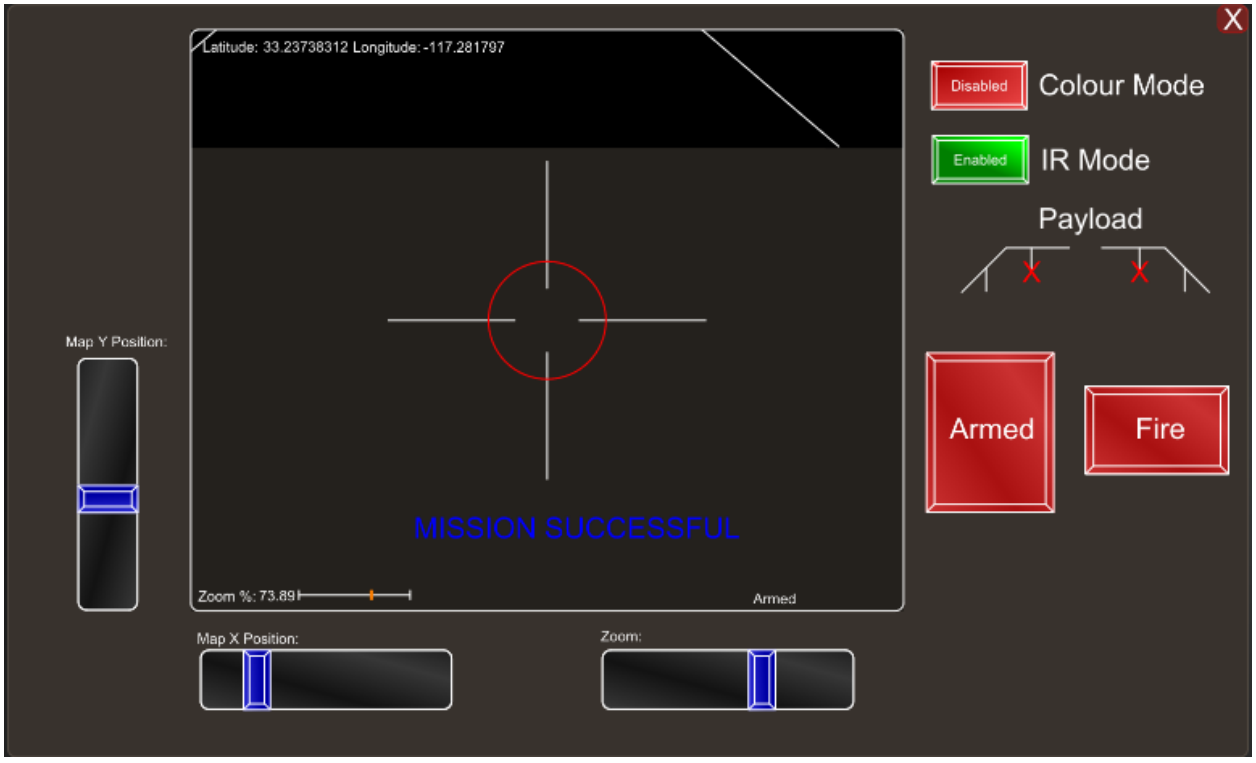

Figure 71- Tutorial Payload Controller Successful Hit

Once the target was eliminated, the “Payload Controller” tab and “Navigational Tab” were swapped. This was to check on the position of the aircraft. It was found that after completing this objective that the aircraft had passed through the third objective and was en route to the fourth, as can be seen in Figure 72. Due to the use of the Autopilot for navigation through this portion of the map, more than one objective was efficiently met. This helped to reduce the overall mission time, which helped to reduce operational stresses. In addition, control of the aircraft was given to the Autopilot, which helped to improve the overall performance by reducing the chance of operator overload while multiple objectives were met.

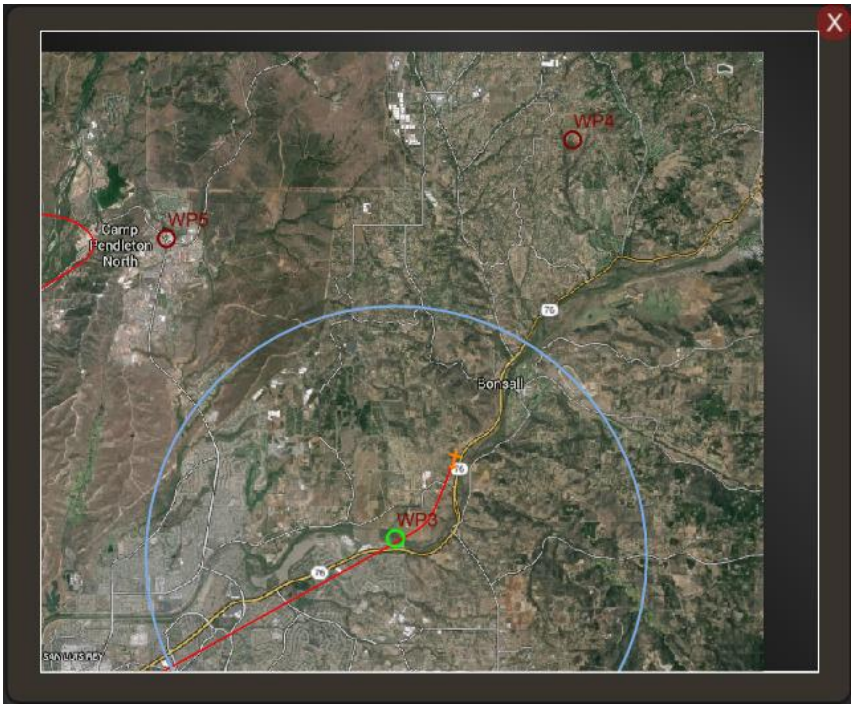

Figure 72- Tutorial Location Check 2

After the aircraft passed through the fourth waypoint (the final Autopilot waypoint) the Autopilot system was disengaged (as is shown in Figure 73), and the throttle was once again set to 100%. Thus the final waypoint had to be reached with manual control. Like the first waypoint, the final waypoint was missed and required the aircraft to fly back around to reach it (shown in Figure 74). During this time, the aircraft’s speed was also reduced to just over 80% in preparation for the landing procedure.

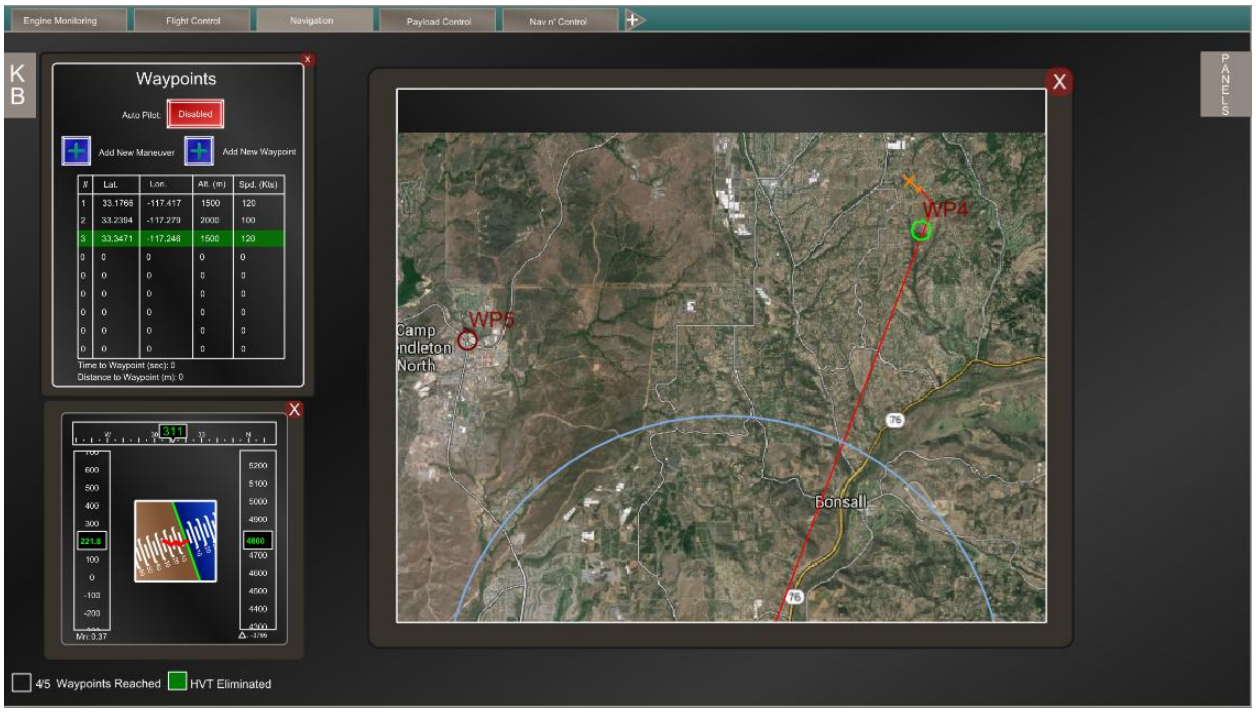

Figure 73- Tutorial Autopilot Disengage

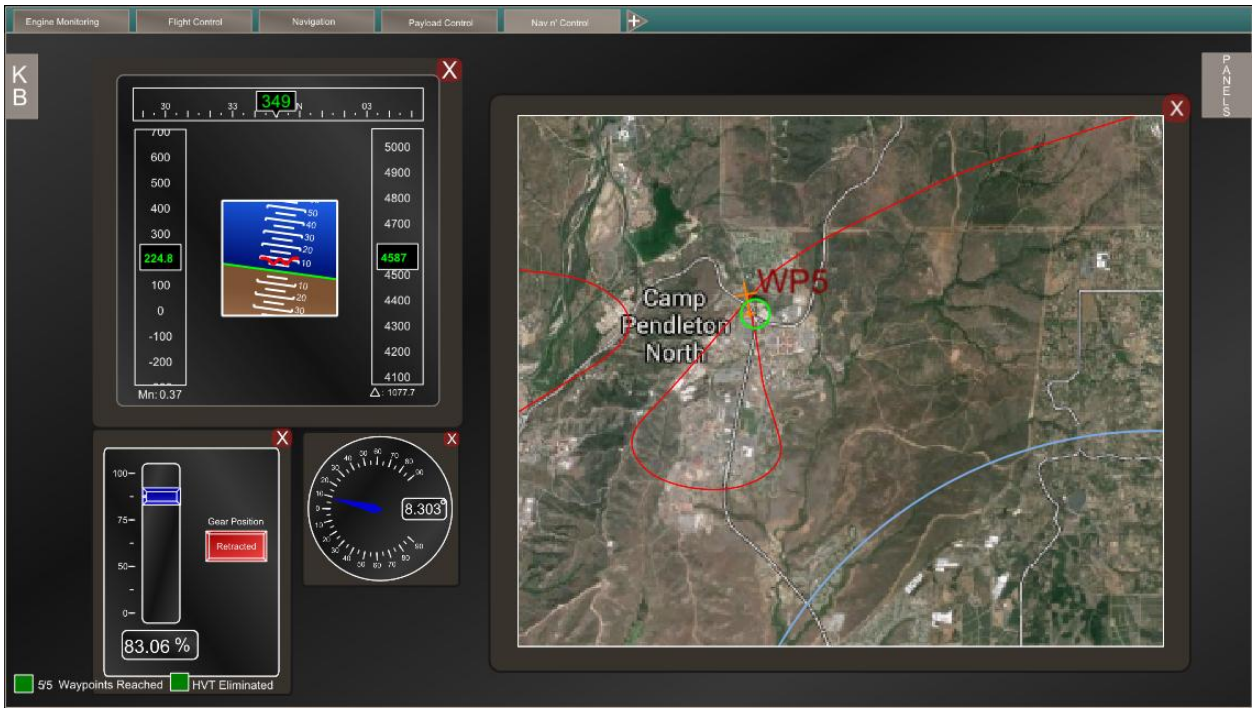

Figure 74- Tutorial Final Waypoint

Finally after all the objectives had been reached, the aircraft was lined up with Camp Pendleton's runway. The landing gears were extended, and the speed was further reduced. To actually

reach the runway from the final waypoint the aircraft had to loop around as there was not enough room to safely slow down or lineup for the landing (shown in Figure 75).

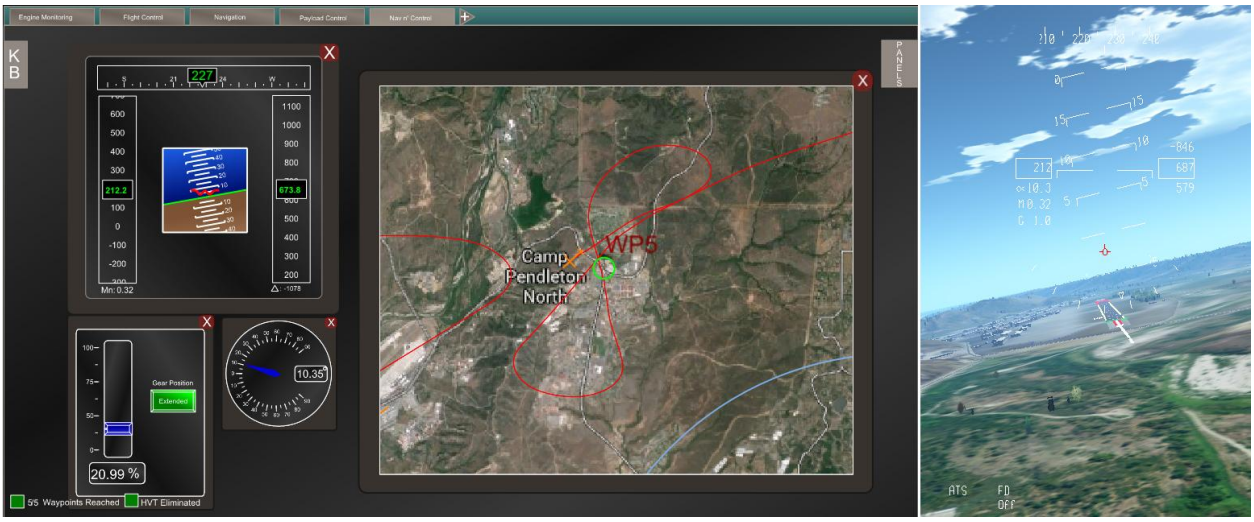

Figure 75- Tutorial Landing Line-Up

The aircraft was then guided down while the Virtual Throttle was used to slow the aircraft down. Once the aircraft touched down the breaks were applied and the aircraft was stopped (shown in Figure 76) resulting in a successful mission.

Figure 76 - Tutorial Completed Landing

The final aircraft flight path can be seen in Figure 77, which accurately showed where the aircraft flew. From this flight path it can be seen just how accurate the autopilot system can be for the navigation of the aircraft. It achieved 3/3 of its waypoints without needing to reattempt any of them. This is in contrast to the manual control of the aircraft, which required the aircraft to fly back around in both of its attempts.

Figure 77- Tutorial Final Flight Path

Ultimately the tutorial showed how some of the various instrument panels and features could be utilized in unison to complete a set of objectives while also starting and ending the aircraft on a runway.

6.0 Conclusion

Due to advances in touch screen technology in the past decade, multi-touch monitors and devices have become more common place in society. They provide a number of inherent advantages that include ease of use, a quick learning curve, the ability to increase the accuracy of interactions, while also reducing a user's response times. As a result, a number of aerospace companies, including Thales and Garmin, have begun to develop futuristic glass cockpits that allow users to interact the various instruments through touch gestures. Despite this progress, there has been a limited amount of research conducted on the effects of touch gestures on their operators when it comes to the control of aircraft. As such, a multi-touch capable, reconfigurable Ground Control Station was developed and presented in this thesis.

Utilizing Commercial-Off-The-Shelf Modeling and Simulation software provided by Presagis, a Ground Control Station UI was developed using VAPS XT 4.1 beta. This program allowed the author to design and implement all of the included instrument panels as well as UI elements into a single comprehensive executable program. The UI and instrument panels were then linked to FlightSIM 14 utilizing nCom's inter-computer communication protocols in order to simulate how the UI could be used to control an unmanned vehicle. The overall UI allowed users to customize the location and size of each designed instrument panel according to their preferences while giving them the ability to use touch gestures to physically interact with the UI elements.

The ability to reconfigure the setup was not only used to increase the effectiveness of the operator, but was also implemented to allow the system to be used for a wide variety of mission profiles as new instruments could be rapidly swapped out. Reconfigurability also allowed operators to interact with a system that was setup around their individual needs as opposed to one that was designed to be used by a wide variety of operators. This would ultimately help to improve operator performance as the panels would be the proper size for the specific user, allowing them to read the displayed information with more ease. In addition, the panels would all be located in a way that gives the individual operator the best results, thus improving their efficiency which would affect their performance in turn.

Finally the UI allows users to interact with all of the instruments and UI elements through common touch gesture commands. These multi-touch gestures allowed an operator to take advantage of the naturalness that gestures are known to provide by combining them with specific instrument inputs to create intuitive interactions. Some examples of these interactions include the ability to pan

navigational maps by dragging them around with a single finger, tap on buttons to engage systems or display important information, and utilizing pinch gestures to resize entire instrument panels. Implementing these multi-touch gestures could help to improve an operator's overall mission performance by giving them a more efficient method of interacting with their computer systems that could also improve their response time. These touch interactions ultimately help to transition older Ground Control Stations inputs from less effective mouse and keyboard setups that can be more time consuming and stressful to use, to more modern and efficient multi-touch inputs.

6.1 Contributions

The multi-touch capable, reconfigurable Ground Control Station was designed to enhance the capabilities of the operator. As such the primary contribution of this thesis was the multi-touch functionality of the Ground Control Station. The touch controls that this Ground Control Station present allowed a user to single-handedly operate a UAV and all its accompanying sensory instruments. This was possible because of the inherent advantages that touch gestures provide the system. First and foremost, they allowed the user to interact naturally with the UI elements so they could rapidly learn how to use the system. Gestures are typically how humans tend to interact with each other, and so it is a natural progression to be able to interact with an electronic device in a similar manner. In addition, to date touch gestures have not been used extensively for the control of aircraft. As such, creating a Ground Control Station that utilized these functions provided an opportunity for research into how touch gesture interactions could be used to improve the human interact element of a control station.

Secondary to the touch capabilities, was giving the user complete control over how they wanted to have their UI setup. This allowed a user to choose where every instrument was located on screen along with how large (or small) it would appear. Giving users this type of flexibility allows them to interact with their environment more efficiently which would help to improve their overall performance. This efficiency comes from the user being able to pick and choose where instruments and variables were located, meaning that the overall interface would be custom tailored to their specific needs as opposed to utilizing an interface that was designed to be used by anyone. In addition, reconfigurable systems can be easily designed around multi-touch gestures, so that the user could quickly set their environment up. Without the gesture support, it would still be possible to create such a system, however it would increase the setup time and would make it much more difficult for users to change their interface at any

point in time (e.g. after the aircraft is in flight). Finally a custom interface could provide beneficial operator human factor improvements because currently the majority of aircraft utilize standard instrument layouts due to the physical equipment that is required to view the data. However by changing the instruments readouts to a virtual screen, the user has the unique opportunity to change where that information appears.

Thirdly, the overall UI and instrument panels that were presented within this Thesis helped to showcase how various features (such as Multi-Touch gestures) could be implemented into the graphical objects that an operator would interact with. These instruments took advantage of the actual touch gestures to provide unique and innovative ways of accessing and interacting with various aircraft parameters and control functions. These included, but were not limited to, a map that could be easily panned around or zoomed into using drag and pinch gestures, a payload controller that allowed a user to pan a factious payload camera around while also releasing a payload to hit a target using drag, pinch and taps, as well as an instrument that could display as much or as little information about the engine and how it was functioning. All of these helped to improve user performance by allowing the user to take advantage of the naturalness of touch gestures while also helping to give them control over how much information they would be exposed to, could help to reduce operator overload. Finally the UI and instruments presented in this Thesis provide a starting point for future studies into how instrument panels could be adapted to other types of human-machine interface inputs (e.g. non-touch screen based gesture controls, eye tracking, etc.) to see if there are more efficient ways of controlling the aircraft and viewing flight data.

6.2 Future Work

No system is perfect when it is created and thus there are areas in which the reconfigureable Ground Control Station could be improved.

First and foremost, the Ground Control Station should be actually analyzed in terms of how it could actually affect operator conditions. This would allow one to see if the addition of multi-touch gestures and full UI reconfigurability are actually of benefit and would better give the industry an idea of whether or not they should be putting more effort into adopting these types of inputs. One potential method for determining the effects of a UI on a user is NASA's Task Load Index. The Task Load Index was initially designed by NASA in the late 1980s to determine the workload that pilots experience while

completing various tasking inside of a cockpit. While it is used by a wide range of fields in modern times, the Task Load Index is still used to determine a user's workload while they interact with a system. As such it would provide a unique opportunity to explore how the Ground Control Station can affect a user's workload, which is a very important factor for operators conducting long duration flights.

Secondly, a number of small improvements could be done to the current instrument panels/ simulation screen based upon user comments. The first issue was that the gesture map sometimes jittered when a user tried to interact with it. This can made it difficult to initially interact with, at times. Improving the stability of the interactions would ultimately help to improve the user experience. In addition, a North arrow could be added to the both maps in order to help the operator stay oriented during flight. Another improvement could be to implement a new graphical overlay in FlightSIM for the visual simulation screen. One such way of improving FlightSIM's UI would be to display the waypoints within the simulation environment. This could have a positive impact on user performance as they would not have to constantly shift focus from the simulation screen to the Ground Control Station UI to determine where the waypoint they were heading to was.

Another set of improvements could be the implementation of new instrument panels to complement what has already been developed. An example of a new panel that could be added is an onboard navigation camera payload controller. This would allow the Ground Control Station to not have to rely on having a second monitor upon which to display the simulation. This panel could also have the added functionality to swap between a first person view of what the aircraft is actually seeing and a virtual third person (or chase) view, where the user could see the aircraft and how it is behaving. Having a chase view camera option would help to improve an operator's Situational Awareness because they can get an idea of what is directly around the aircraft, something that is particularly useful when an aircraft is in a cluttered environment like a city.

New functionality could also be implemented into either the UI or into any of the instrument panels to further enhance them. An example of one such potential new function would be to add the ability to add waypoints directly on either of the two developed maps. Utilizing tap gestures the user could just tap on the exact location upon which they wished to place a waypoint, while filling in the remaining needed information in on a small contextual radial menu. Having this functionality might help to decrease pilot errors because they would not have to rely on remembering the exact latitudinal and longitudinal coordinates. There may also be times when the user does not know the exact waypoint

location, or have time to search for it, so having the ability to quickly create a waypoint on the map in such a situation would help to reduce the response time of the operator.

Another potential improvement would be to switch from using Presagis' FlightSIM for the aircraft simulation to Presagis' STAGE. This change would give the added flexibility of creating more complex, and higher fidelity simulations due to the ability to add other objects into the environment such as ground forces, other aircraft, and civilians. The higher fidelity training simulations would allow more a more accurate training environment that forces the user to consider how they are connected to the other elements in the flight region as opposed to being a single entity.

Ultimately there has been little direct research conducted on how multi-touch gestures and reconfigurability can help to improve the various human factors that can affect a pilot. As such there are a lot more possibilities for what can be done to enhance the operator's level of control and their work environment.

7.0 References

[1] D. Maurino and E. Salas, Human Factors in Aviation, 2 ed., Burlington, MA: Elsevier, 2010.

[2] L. Weiss, "Autonomous Robots in the Fog of War," *IEEE Spectrum,* vol. 48, no. 8, pp. 30-57, 2011.

[3] Society, Long Island Republic Airport Historical, "Sperry Gallery," [Online]. Available: https://sites.google.com/site/lirepublicairporths/fairchildgallery22. [Accessed 08 May 2015].

[4] Q. Waraich, T. Mazzuchi, S. Sarkani and D. Rico, "Minimizing Human Factors Mishaps in Unmanned Aircraft Systems," *Ergonomics in Design: The Quarterly of Human Factors Applications,* vol. 21, no. 1, pp. 25-32, 2013.

[5] Defense Update, "MQ-9 Reaper Hunter/ Killer UAV," [Online]. Available: http://defense-update.com/products/p/predatorB.htm. [Accessed 08 May 2015].

[6] HisPotion, "DJI Phantom Aerial UAV Drone Quadcopter for GoPro," [Online]. Available: http://www.hispotion.com/dji-phantom-aerial-uav-drone-quadcopter-for-gopro-7394. [Accessed 05 May 2015].

[7] R. Sward, C. Sparkman and D. Pack, "Challenges in Developing Unmanned Aerial Vehicle Software Systems," in *Infotech@Aerospace*, Arlingto, VA, 2005.

[8] R. Valimont and S. Chappell, "Look Where I'm Going and Go Where I'm Looking: Camera-Up Map for Unmanned Aerial Vehicles," in *6th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, Lausanne, Switzerland, 2011.

[9] T. Lam, H. Boschloo, M. Mulder, M. van Paassen and F. van der Helm, "Effect of Haptic Feedback in a Trajectory Following Task with an Unmanned Aerial Vehicle," in *IEEE International Conference on Systems, Man and Cybernetics*, 2004.

[10] J. Hing and P. Oh, "Mixed Reality for Unmanned Aerial Vehicle Operations in Near Earth Environments," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Taipei, Taiwan, 2010.

[11] T. Carretta, D. Perry Jr. and M. Ree, "Prediction of Situational Awareness in F-15 Pilots," *The International Journal of Aviation Psychology,* vol. 6, no. 1, pp. 21-41, 1996.

[12] J. Gundlach, Designing Unmanned Aircraft Systems: A Comprehensive Approach, 2nd ed., J. A. Schetz, Ed., Reston, VA: American Institute of Aeronautics and Astronautics , 2014.

[13] B. Kayayurt and I. Yayla, "Application of STANAG4586 Standard for Turkish Aerospace Industries UAV Systems," in *IEEE/AIAA 32nd Digital Avionics Conference (DASC)*, East Syracuse, NY, 2013.

[14] B. Rivers, "NATO establishing UAV standards," *Journal of Electronic Defense,* vol. 25, no. 11, p. 24, 2002.

[15] M. Perhinschi, M. Napolitano and S. Tamayo, "Integrated Simulation Environment for Unmanned Autonomous Systems - Towards a Conceptual Framework," *Modelling and Simulation in Engineering,* vol. 2010, pp. 1-12, 2010.

[16] G.-A. Hector, "Neurovision; The Way To Merge Visual Reality with Navigational and Military Systems," in *IEEE/AIAA 32nd Digital Avionics Systems Conference (DASC)*, East Syracuse, NY, 2013.

[17] B. Walter, J. Knutzon, A. Sannier and J. Oliver, "VR Aided Control of Unmanned Vehicles," Iowa State University , Ames, IA, 2004.

[18] R. Murphy, K. Pratt and B. J.L, "Crew Roles and Operational Protocols for Rotary-Wing Micro-UAVs in Close Urban Environments," in *3rd ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, Amsterdamn, Netherlands, 2008.

[19] D. Erdos and S. Watkins, "UAV Autopilot Integration and Testing," in *IEEE Region 5 Conference*, Kansas City, MO, 2008.

[20] R. Sharma, "Fuzzy Q Learning Based UAV Autopilot," in *Innovative Applications of Computational Intelligence on Power, Energy and Controls with their impact on Humanity (CIPECH)*, Ghaziabad, India, 2014.

[21] A. K. Yadav and P. Gaur, "AI-based adaptive control and design of autopilot system for nonlinear UAV," *Sadhana,* vol. 39, no. 4, pp. 765-783, 2014.

[22] M. Hamedi, S.-H. Salleh, T. Tan, K. Ismail, J. Ali, C. Dee-Uam, C. Pavaganun and P. Yupapin, "Human facial neural activities and gesture recognition for machine-interfacing applications," *International Journal of Nanomedicine,* vol. 6, pp. 3461-3472, 2011.

[23] C. Sandom, "Operator Situational Awareness and System Safety," in *IEE One Day Seminar on Systems Dependency on Humans*, London, UK, 2000.

[24] H. Shin, J.-M. Lim, J.-u. Lee and K.-U. Kyung, "Background Display for Visually Impaired People in Mobile Touch Devices," in *IEEE Conference on Consumer Electronics*, Las Vegas, NV, 2013.

[25] M. Jovanovic, D. Starcevic and Z. Jovanovic, "Improving Design of Ground Control Station for Unmanned Aerial Vehicle: Borrowing from Design Patterns," in *36th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*, Lille, France, 2010.

[26] K. S. Yoon, I. G. Kim and K. S. Ryu, "AN EFFICIENT EMBEDDED SYSTEM ARCHITECTURE FOR PILOT TRAINING," in *IEEE/AIAA 30th Digital Avionics Systems Conference*, Seattle, WA, 2011.

[27] P. Zipfel, "A C++ Architecture for Unmanned Aerial Vehicle Systems," in *AIAA Infotech@Aerospace Conference*, Rohnert Park, CA, 2007.

[28] G. Fein, "Growing Use of UAVs Among Marines Requires Common Ground Control Station," *Helicopter News,* vol. 31, no. 22, p. 1, 2005.

[29] B. Kayayurt, I. Yayla, A. Yapici and C. Kucukoguz, "Ground Control Station Avionics Software Development in ANKA UAV," in *IEEE/AIAA 30th Digital Avionics Systems Conference (DASC)*, Seattle, WA, 2011.

[30] CAE, "Civil Aviation Photo Gallery," [Online]. Available: http://www.cae.com/civil-aviation/media-centre/photo-gallery/?LangType=1033. [Accessed 08 May 2015].

[31] J. Peschel and R. Murphy, "On the Human-Machine Interaction of Unmanned Aerial System Mission Specialists," *IEEE Transactions on Human-Machine Systems,* vol. 43, no. 1, pp. 53-62, 2013.

[32] I. Maza, F. Caballero, R. Molina, N. Pena and A. Ollero, "Multimodal Interface Technologies for UAV Ground Control Stations: A Comparative Analysis," *Journal of Intelligent and Robotic Systems,* vol. 57, no. 1-4, pp. 371-391, 2010.

[33] Oculus, "Oculus VR," [Online]. Available: https://www.oculus.com/. [Accessed 30 April 2015].

[34] Oculus, "Oculus Best Practises Guide," 09 January 2015. [Online]. Available: http://static.oculus.com/sdk-downloads/documents/Oculus_Best_Practices_Guide.pdf. [Accessed 30 April 2015].

[35] J. Hing, K. Sevcik and P. Oh, "Development and Evaluation of a Chase View for UAV Operations in Cluttered Environments," in *Selected papers from the 2nd International Symposium on UAVs*, Reno, NV, 2009.

[36] LabReporter, "Mark Zuckerberg Acquires Oculus VR," [Online]. Available: http://www.labreporter.com/mark-zuckerberg-acquires-oculus-vr/. [Accessed 08 May 2015].

[37] C. Forlines, D. Wigdor, C. Shen and R. Balakrishnan, "Direct-Touch vs. Mouse Input for Tabletop Displays," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, San Jose, CA, 2007.

[38] M. Nimbarte, "Multi-Touch Screen Interfaces and Gesture Analysis: A Study," *Advanced Computing: An International Journal,* vol. 2, no. 6, pp. 113-121, November 2011.

[39] T. Moscovich and J. Hughes, "Indirect Mappings of Multi-touch Input Using One and Two Hands," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, Florence, Italy, 2008.

[40] W. Westerman and J. Elias, "Multi-Touch: A New Tactile 2-D Gesture Interface for Human Computer Interaction," *Proceedings of the Human Factors and Ergonomics Society 45th Annual Meeting,* vol. 45, no. 6, pp. 632-636, 2001.

[41] B.-J. Chen, C.-M. Huang, T.-E. Tseng and L.-C. Fu, "Robust Head and Hands Tracking with Occulsion Handling for Human Machine Interaction," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Vilamoura, Portugal, 2012.

[42] F. Jiang, S. Wu, G. Yang, D. Zhao and S. Kung, "Viewpoint-Independent Hand Gesture Recognition with Kinect," *Signal, Image and Video Processing,* vol. 8, no. 1, pp. S163-S172, 2014.

[43] K. Ponto, K. Doerr, T. Wypych, J. Kooker and F. Kuester, "CGLXTouch: A multi-user multi-touch approach for ultra-high-resolution collaborative workspaces," *Future Generation Computer Systems,* vol. 27, no. 6, pp. 649-656, 2011.

[44] Y. Gu, H. Do, Y. Ou and W. Sheng, "Human Gesture Recognition Through A Kinect Sensor," in *Proceedings of the 2012 IEEE International Conference on Robotics and Biomimetics*, Guangzhou, China, 2012.

[45] D. Fiorella, A. Sanna and F. Lamberti, "Multi-Touch User Interface Evaluation for 3D Object Manipulation on Mobile Devices," *Journal of Multimodal User Interfaces,* vol. 4, no. 1, pp. 3-10, 2010.

[46] S. Bachl, M. Tomitsch, C. Wimmer and T. Grechenig, "Challenges for Designing the User Experience of Multi-Touch Interfaces," in *Proceedings of Engineering Patterns for Multi-Touch Interfaces Workshop of the ACM SIGCHI Symposium on Engineering Interactive Computer Systems*, Berlin, 2010.

[47] Middle of Nowhere Gaming, "Benefits of the Kinect 2.0," [Online]. Available: http://middleofnowheregaming.com/2014/02/07/benefits-of-the-kinect-2-0/. [Accessed 08 May 2015].

[48] Microsoft, "Kinect for Windows features," [Online]. Available: http://www.microsoft.com/en-us/kinectforwindows/meetkinect/features.aspx. [Accessed 30 April 2015].

[49] R. Ibanez, A. Soria, A. Teyseyre and M. Campo, "Easy gesture recognition for Kinect," *Advances in Engineering Software ,* vol. 76, pp. 171-180, 2014.

[50] K. Boudjit, C. Larbes and M. Alouache, "Control of Flight Operation of a Quad rotor AR.Drone Using Depth Map from Microsoft Kinect Sensor," *International Journal of Engineering and Innovative Technology,* vol. 3, no. 3, pp. 15-19, 2013.

[51] M. Nord and H. Vestgote, "Multi-Touch in Control Systems: Two Case Studies," Uppsala University Publications, Uppsala, Sweden, 2010.

[52] Thales, "THALES UNVEILS AVIONICS 2020, THE COCKPIT OF THE FUTURE," [Online]. Available: https://www.thalesgroup.com/en/worldwide/aerospace/press-release/thales-unveils-avionics-2020-cockpit-future. [Accessed 30 April 2015].

[53] Garmin, "Garmin Powered Glass Cockpit System Announced by Volvo Penta," [Online]. Available: http://ph.garmin.com/news/pressroom/news20130801/. [Accessed 30 April 2015].

[54] D. Soto-Guerrero and J. G. R. Torres, "A Human-Machine Interface with Unmanned Aerial Vehicles," in *10th International Conference on Electrical Engineering, Computer Science and Automatic Control*, Mexico City, 2013.

[55] NUI Group Community Wiki, "Frustrated Total Internal Reflection (FTIR)," [Online]. Available: http://wiki.nuigroup.com/Frustrated_Total_Internal_Reflection_(FTIR). [Accessed 08 May 2015].

[56] A. Bragdon, E. Nelson, Y. Li and K. Hinckley, "Experimental Analysis of Touch-Screen Gesture Designs in Mobile Environments," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, Vancouver, BC, 2011.

[57] B. Keyes, M. Micire, J. Drury and H. Yanco, "Improving Human-Robot Interaction Through Interface Evolution," in *Human-Robot Interaction*, G. Chugo, Ed., Rijeka, InTech, 2010, pp. 183-202.

[58] D. Li, J. Xie, D. Weng and Y. Li, "Touch Experience in Augmented Reality," in *IEEE Virtual Reality (VR)*, Lake Buena Vista, FL, 2013.

[59] T. Guerreirom, H. Nicolau, J. Jorge and D. Goncalves, "Towards Accessible Touch Interfaces," in *Proceedings of the 12th International ACM SIGACCESS Conference on Computers and Accessibility*, Orlando, FL, 2010.

[60] H. Lee and J. Park, "Touch Play Pool: Touch Gesture Interaction for Mobile Multifunction Devices," in *IEEE International Conference on Consumer Electronics*, Lafayette, LA, 2012.

[61] J.-S. Kim, D. Gracanin, K. Matkovic and F. Quek, "iPhone/iPod Touch as Input Devices for Navigation in Immersive Virtual Environments," in *IEEE Virtual Reality Conference*, Lafayette, LA, 2009.

[62] J. Koonce and A. Debons, "A Historical Overview of Human Factors in Aviation," in *Aviation Human Factors*, J. Wise, V. D. Hopkin and D. J. Garland, Eds., Boca Raton, FL: Taylor & Francis Group, LLC, 2010, pp. 1-1 to 1-11.

[63] D. R. Hunter and M. Martinussen, Aviation Psychology and Human Factors, Boca Raton, FL: Taylor and Francis Group, LLC, 2010.

[64] J. Shmelev, "Simulator Training Optimization of UAV External Pilots," in *IEEE 3rd International Conference on Methods and Systems of Navigation and Motion Control (MSNMC)*, Kiev, Ukraine, 2014.

[65] K. Williams, "A Summary of Unmanned Aircraft Accident/ Incident Data: Human Factors Implications," U.S. Department of Transportation, Washington, DC, 2004.

[66] J. Wilson, "UAVs and the Human Factor," *Aerospace America,* vol. 40, no. 7, p. 54, 2002.

[67] E. Svensson and G. Wilson, "Psychological and Psychophysiological Models of Pilot Performance for Systems Development and Mission Evaluation," *The International Journal of Aviation Psychology,* vol. 12, no. 1, pp. 95-110, 2002.

[68] C. Murray and W. Park, "Incorporating Human Factor Considerations in Unmanned Aerial Vehicle Routing," *IEEE Transactions on Systems, Man, and Cybernetic Systems,* vol. 40, no. 4, pp. 860-874, 2013.

[69] S. Hart and L. Staveland, "Development of NASA-TLX (Task Load Index): Results of Emperical and Theoretical Research," *Advances in Psychology,* vol. 52, pp. 139-183, 1988.

[70] Ryerson University, "Mixed-Reality Immersive Motion Simulation Laboratory," [Online]. Available: http://www.ryerson.ca/~j3chung/. [Accessed 08 May 2015].

[71] Ryerson's Mixed-Reality Immersive Motion Simulation Laboratory, "Youtube (Video Title: MIMS PROMO VIDEO Final Edit HR)," [Online]. Available: https://www.youtube.com/watch?v=Yq2XTn0M144. [Accessed 10 June 2015].

[72] MaxFlight, "MaxFlight Corporation," [Online]. Available: http://www.maxflight.com/. [Accessed 10 June 2015].

[73] QGroundControl, [Online]. Available: http://www.qgroundcontrol.org/. [Accessed 30 April 2015].

[74] Mission Planner, [Online]. Available: http://planner.ardupilot.com/. [Accessed 30 April 2015].

[75] A. Amresh, J. Femiani, J. Fairfield and A. Fairfield, "UAV Sensor Operator Training Enhancement Through Heat Map Analysis," in *17th International Conference on Information Visualisation (IV)*, London, 2013.

[76] Y. Hong, J. Fang and Y. Tao, "Ground Control Station Development for Autonomous UAV," *Intelligent Robotics and Applications,* vol. 5315, pp. 36-44, 2008.

[77] A. Burkle, F. Segor and M. Kollmann, "Towards Autonomous Micro UAV Swarms," *Journal of Intelligent and Robotic Systems,* vol. 61, no. 1-4, pp. 339-353, 2011.

[78] D. Perez, I. Maza, F. Caballero, D. Scarlatti, E. Casado and A. Ollero, "A Ground Control Station for a Multi-UAV Surveillence System: Design and Validation in Field Experiments," *Journal of Intelligent and Robotic Systems,* vol. 69, no. 1-4, pp. 119-130, 2013.

[79] B. Ludington, E. Johnson and G. Vachtsevanos, "Augmenting UAV Autonomy," *IEEE Robotics and Automation Magazine,* vol. 13, no. 3, pp. 63-71, 2006.

[80] "Windows User Experience Interaction Guidelines for Windows 7 and Windows Vista," [Online]. Available: http://www.oamk.fi/~teraisan/K1033BI/Exercises/styleguide/WIN7_UXGuide.pdf. [Accessed 30 April 2015].

[81] T. Lovell, "Affordable Multisensory Situational Awareness for Aircraft Applications," in *IEEE/ AIAA 30th Digital Avionics Conference*, Seattle, WA, 2011.

[82] Y. Park and S. Han, "Touch Key Design for One-Handed Thumb Interaction With a Mobile Phone - Effects of Touch Key Size and Touch Key Location," *International Journal of Industrial Ergonomics,* vol. 40, no. 1, pp. 68-76, 2010.

[83] Presagis, "Presagis," [Online]. Available: http://www.presagis.com/. [Accessed 30 April 2015].

[84] Lockheed Martin, "F-16 Specifications," [Online]. Available: http://www.lockheedmartin.ca/us/products/f16/F-16Specifications.html. [Accessed 30 April 2015].

[85] "World Geodetic System 1984 (WGS84)," QPS Software Products, [Online]. Available: https://confluence.qps.nl/pages/viewpage.action?pageId=29855173#WorldGeodeticSystem1984(WGS84)-WGS84definitions. [Accessed 30 April 2015].

[86] R.-F. Day, G. C. Shyi and J.-C. Want, "The Effect of Flash Banners on Multiattribute Decision Making: Distractor or Source of Arousal?," *Psychology and Marketing,* vol. 23, no. 5, pp. 369-382, May 2006.

[87] P. Cohen, "Macworld Expo Keynote Live Update," Macworld, 9 January 2007. [Online]. Available: http://www.macworld.com/article/1054764/liveupdate.html. [Accessed 5 August 2015].

[88] S. Herz, S. Forstall and M. Matas, "Portable multifunction device, method, and graphical user interface for interpreting a finger gesture". United States of America Patent 8,665,225, 4 March 2014.

[89] Apple Inc. , "Turn on "three finger drag" for your Force Touch trackpad," [Online]. Available: https://support.apple.com/en-ca/HT204609. [Accessed 5 August 2015].

[90] B. M. Victor, "Device, method, and graphical user interface for manipulating user interface objects". United States of America Patent 8,456,431, 4 June 2013.

[91] Presagis, *Presagis FlightSIM 13 Software Help Library,* 2014.

[92] Encyclopædia Britannica, "Great Circle Route," 2015. [Online]. Available: http://www.britannica.com/technology/great-circle-route. [Accessed 28 July 2015].

[93] "CADRE Analyric," [Online]. Available: http://www.cadreanalytic.com/. [Accessed 30 April 2015].

[94] MTL Movable Type Scripts, "Calculate distance, bearing and more between Latitude/ Longitude points," [Online]. Available: http://www.movable-type.co.uk/scripts/latlong.html. [Accessed 30 April 2015].

[95] S. Hart, "NASA-Task Load Index (TLX); 20 Years Later," *Proceedings of the Human Factors and Ergonomic Society Annual Meeting,* vol. 50, no. 9, pp. 904-908, 2006.

[96] Anonymous, "NASA Task Load Index (TLX) v1.0 Paper and Pencil Package," Human Performance Research Group, Moffett Field, CA, 1988.

[98] S.-H. Kim, L. J. Prinzel, D. B. Kaber, A. L. Alexander, E. M. Stelzer, K. Kaufmann and T. Veil, "Multidimensional Measure of Display Clutter and Pilot Performance for Advanced Head-Up Display," *Aviation, Space, and Environmental Medicine,* vol. 82, no. 11, pp. 1013-1022, November 2011.

[99] J. Y. Chen, "UAV-Guided Navigation For Ground Robot Tele-Operation In A Military Reconnaissance Environment," *Ergonomics,* vol. 53, no. 8, pp. 940-950, 2010.

[100] D. V. Gunn, J. S. N. W. T. Warm, R. S. Bolia, D. A. Schumsky and K. J. Corcoran, "Target Acquisition With UAVs: Vigilance Displays and Advanced Cueing Interfaces," *Human Factors: The Journal of Human Factors and Ergonomics Society,* vol. 47, no. 3, pp. 488-497, 2005.