6-2006

Situational Awareness and Synthetic Vision for Unmanned Aerial Vehicle Flight Testing

Joseph M. Dugan

SITUATIONAL AWARENESS AND SYNTHETIC VISION FOR UNMANNED AERIAL VEHICLE FLIGHT TESTING

THESIS

Joseph M. Dugan, Ensign, USN

AFIT/GAE/ENY/06-J03

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GAE/ENY/06-J03

SITUATIONAL AWARENESS AND SYNTHETIC VISION FOR UNMANNED AERIAL VEHICLE FLIGHT TESTING

THESIS

Presented to the Faculty
Department of Aeronautics and Astronautics
Graduate School of Engineering and Management
Air Force Institute of Technology
Air University
Air Education and Training Command
in Partial Fulfillment of the Requirements for the
Degree of Master of Science in Aeronautical Engineering

Joseph M. Dugan, B.S.
Ensign, USN

June, 2006

AFIT/GAE/ENY/06-J03

SITUATIONAL AWARENESS AND SYNTHETIC VISION FOR UNMANNED AERIAL VEHICLE FLIGHT TESTING

Joseph M. Dugan, B.S.

Ensign, USN

Approved:

Maj Paul Blue Thesis Advisor	Date
Lt Col Stew DeVilbiss, Ph.D. Committee Member	Date
John F. Raquet, Ph.D. Committee Member	Date

AFIT/GAE/ENY/06-J03

Abstract

The Advanced Navigation Technology (ANT) Center at the Air Force Institute of Technology (AFIT) is currently exploring ways to develop and advance the employment of autonomous Unmanned Aerial Vehicles (UAV) by the Department of Defense for military purposes. The research in this thesis describes the development of a tool that enhances situational awareness and provides synthetic vision in a program called the Aviator Visual Display Simulator (AVDS) during UAV flight. During flight testing, the Situational Awareness and Synthetic Vision Relay Tool (SASVRT) developed provides the test coordinator and pilot, as well as the safety observers, with the most pertinent information regarding operational safety. In addition to improved safety, the enhanced situational awareness provided by SASVRT provides improved operational capabilities. SASVRT provides users with real-time information regarding hard boundaries, interpolated terrain, flight ceilings, and other aerial vehicles present. The hard boundary is a user defined area within which the UAV is to stay at all times e.g. as designated by a Safety Review Board (SRB). Pertinent data to this boundary is both distance and time until crossing, and is provided to the user during flight by SASVRT. The interpolated terrain part of SASVRT allows the user to input contours of altitude and based on all present information, interpolates a minimum safe altitude for flight. This information is also relayed to the user during flight. If multiple aerial vehicles are recognized by the autopilot operator interface, SASVRT will relay to the user the distances between a vehicle and any other vehicles present. Finally, in order to provide synthetic vision, SASVRT relays telemetry data from the aircraft to AVDS, an aircraft simulation program, to provide a real-time visualization of the aircraft's position and attitude relative to a synthetic terrain constructed based on information entered by the user. The visualization can be thought of as the view from a virtual camera that can be placed anywhere relative to the UAV or ground. Furthermore, the virtual camera can be oriented such that it provides a view from the cockpit, providing synthetic vision for the UAV operator, the fidelity of which is limited only by available maps and GPS accuracy. This report and the accompanying SASVRT program, provide a much enhanced methodology for safe operation during UAV flight testing, as well as improved operational capabilities.

Acknowledgements

I would like to express my deepest gratitude to my faculty advisor, Major Paul Blue, for his support throughout my time here. His guidance provided me the direction and focus I needed in order to accomplish my goals. I would like to thank Dr. John Raquet for allowing me the use of the lab's aircraft and equipment and making my thesis possible. I would like to thank Brent Robinson and Patrick McCarthy for their help throughout my thesis as I encountered difficulties. Their synergistic effects cannot be quantified by the number of hours they saved me. Randy Plate, who provided a baseline program from which I developed my thesis, gave me the start I needed in this research. Second Lieutenant Brett Pagel provided insight into autopilot flight testing allowing me to easily pick up on the methodology of autonomous flight testing. Don Smith, the lab's expert technician provided the mechanical and electrical tools needed in order conduct this research. Steve Rasmussen and his expertise with the AVDS program was always available to answer my questions as I worked with his program. To my shipmates, who provided the moral support I needed in order to complete this thesis.

Joseph M. Dugan

Table of Contents

Page

List of Figures

Figure Page

List of Symbols

List of Abbreviations

SITUATIONAL AWARENESS AND SYNTHETIC VISION FOR UNMANNED AERIAL VEHICLE FLIGHT TESTING

I. *Introduction*

1.1 Motivation

Unmanned Aerial Vehicles (UAVs) have long been researched in order to obtain a operating presence amidst environments too dangerous for pilots. However it has only been in recent years that developing technology has been able to revolutionize the capabilities of UAVs. Their missions can now range from intelligence, surveillance, and reconnaissance (ISR) for troops on the ground or commanders apart from the battlefield, to autonomous search and destroy missions. Because the pilot is removed from the platform, many restrictions are also removed allowing the aircraft to perform longer without risk of fatigue or perform in conditions not suited for the human body such as high accelerations, high altitude, or contaminated areas. It is with these advantages in mind that the Department of Defense (DoD) pursues research of UAVs with such vigor.

In order to produce UAVs capable of fulfilling the high demands of modern warfare, extensive research and flight testing is an absolute requirement. Experimental flight testing is a vital element in aircraft research and development, and although the target end product of research and flight testing is assured and safety or reliability, the testing itself can lack exactly that. Flight testing an autopilot system on an aircraft can go wrong for a countless number of reasons due to the complexity of the system and its reliance on numerous vital elements. However, improving the operator's situational awareness can dramatically reduce the risk of flight testing autonomous UAVs.

Flight testing of unmanned autopilot systems on Area B of Wright-Patterson Air Force Base requires a safety briefing in which an area is designated within which the flight testing is to occur. Because of the nature of the flight test, it is often difficult for observers to monitor where the aircraft is in relation to the boundary of the designated area. However, with a Global Portioning System (GPS) receiver on board, an autopilot can tell where exactly the aircraft is in relation to boundaries. Because of GPS and other sensors on board, the situational awareness of observers can be heightened by processing the on board information and relaying the most relevant data to the observers. Furthermore, the improved situational awareness to improve flight test safety can also increase operational capabilities.

1.2 Related Research

1.2.1 Characterization of UAV Performance Evaluation and Development of a Formation Flight Controller for Multiple Small UAVs. Ensign Patrick McCarthy is working in the Advanced Navigation and Technology (ANT) Center to control multiple aerial vehicles and fly them in formation patterns. As a lead aircraft flies, his work involves using a position based relative to that of the lead aircraft to place a waypoint for a following aircraft to track and fly towards (McCarthy, 2006). The application of this research with regards to Ensign McCarthy's research is discussed in Chapter VI.

1.2.2 An Investigation Into Robust Wind Correction Algorithms for Off the Shelf Unmanned Aerial Vehicle Autopilots. Ensign Brent Robinson is also working in the ANT Center to account for wind effects during UAV flight (Robinson, 2006). The goal of his research is to point an on board sensor at a target amidst the autopilot induced crabbing of the aircraft to compensate for the wind conditions. The application of this research with regards to Ensign Robinson's research is discussed in Chapter VI.

1.2.3 Controlled Flight Into Terrain. The idea of maximum situational awareness is not new nor unique to UAVs. Much research has been done in the area of heightening the situational awareness of commercial pilots in order to reduce the number of incidents of controlled flight into terrain, (CFIT), such as in Barry Breen's paper (Breen). Breen discusses that most airline fatalities are not the fault of the aircraft itself, but rather the fault of the pilot's lack of situational awareness. Yet many attempts of a ground proximity warning system have failed due to excessive alarms leading to pilot agitation and ultimate ignoring of the alarm. Many incidents occurring with UAV flight testing are also due to pilot error, which often results from the pilots lack of situational awareness.

1.2.4 Synthetic Vision. The predominant cause of aircraft incidents involving CFIT and runway incursions has been cited as limited visibility. Because of this, NASA is now conducting research and developing Synthetic Vision Systems (SVS), in order to reduce the risks associated with low visability conditions. In *Synthetic Vision Enhanced Surface Operations and Flight Procedures Rehearsal Tool*, the authors discuss an evaluation of SVS in which two concepts were used, a head-worn display of SVS and a three-dimensional SVS electronic flight bag. After experimenting with several flight modes and situations including CFIT, the research concluded taht numerous future directions should be examined in order to better optimize SVS concepts [Kramer, 2006].

As a result, NASA then conducted a study on the incorporation of Enhanced Vision Systems (EVS) into SVS [Bailey, 2006]. The research is based on complimenting the advantages of SVS, unlimited vision in any conditions, with those of EVS, enhanced vision not subject to navigational error. Because EVS is an imaging sensor presentation and not a database-derived image like SVS, it can be used to correct the display of SVS by incorporating the researched integration and/or fusion techniques. An example of the two displays that are compared is shown in Figure 1 [Bailey, 2006].

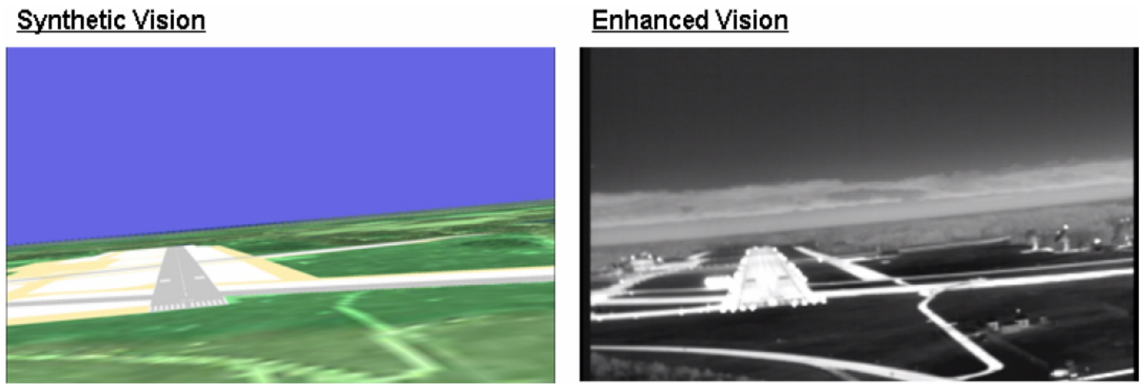


Figure 1   Comparison Between SVS and EVS

1.3 Problem Definition

The primary objective of this research was to develop and test a situational awareness tool that provides a real-time indicator as to the aircraft's current location with respect to the designated borders, including both two-dimensional approved flying areas, and three-dimensional approximated terrain for safe flying. In order to have an immediate impact of UAV flight testing in AFIT's ANT Center, the situational awareness tool must be compatible with the Piccolo II autopilot software currently being utilized. The situational awareness tool will communicate with the Piccolo II autopilot on board the aircraft via the UHF antenna connected to the ground station computer.

A secondary objective of this research was to develope a synthetic vision system to provide a three-dimensional visualization of an aircraft's attitude and position relative to its surroundings. This would then give the radio control (R/C) pilot synthetic vision, as if on board the aircraft, in order to enhance situational awareness. Often times, due to the distance between the aircraft and an observer, a potential hazard may look more or less serious from the observer's point of view than it would from the aircraft. A high fidelity synthetic vision system would give observers a much better point of view despite weather or time-of-day conditions, which improves safety and performance.

1.4 Research Objectives

- Establish a method to determine and warn observers when the aircraft is approaching or has crossed over the designated flight area boundaries.
- Establish a method to monitor the aircraft's altitude relative to the ground or a flight ceiling, and relay this information to observers along with warning when the aircraft approaches the limits.
- Incorporate terrain and buildings into a safe operating altitude algorithm in order to maximize the situational awareness of the observers.
- Relay telemetry data to the Aviator Visual Design Simulator (AVDS) in order to give observers synthetic vision from onboard the aircraft.
- Simulate flight tests of aforementioned methods of heightening situational awareness in order to demonstrate a safer way to flight test UAVs, as well as show the applicability to operational use of UAVs.

1.5 Significance

This thesis supports the ANT Center's ongoing efforts to advance the fields of guidance, navigation, and control in support of DoD missions in several ways. First and foremost, the Situational Awareness and Synthetic Vision Relay Tool (SASVRT) provides the ANT Center with improved UAV flight safety by significantly enhancing the UAV test teams situational awareness during flight operations. Of particular interest is the enhanced safety during flights with multiple UAVs. SASVRT continuously displays essential information like distance and time to user defined boundaries, minimum distance between UAVs, etc. At the same time, the synthetic visions enables the operator (or anybody with a computer on the same network) to synthetically view the UAV or multiple UAVs from any fixed position or from a position relative to any of the UAVs, which enables views from perspectives not possible from a fixed position on the ground, especially when the vehicles are at great distances or beyond visual range. SASVRT also enables views not possible with vehicle-mounted cameras, like a bird's-eye view. In addition to improving safety, the increased situational awareness enhances operational mission capabilities, especially when operating in conditions with limited visibility. Finally, the data generated during flight can be logged and analyzed post-flight, including replaying the synthetic movie and even generating new movies from different perspectives than viewed during flight.

1.6 Assumptions

Several assumptions were made in this research. In the coordinate transformations necessary in SASVRT, there are inherent errors due to the distance between the current location and base point due to the curvature of the earth and the assumption of a flat earth. Because the UAV flight testing occurs over a relatively small area, these errors remain small and are neglected, but could be accounted for if flying over large distances. Also, locations provided by the GPS receiver are not exact and have associated errors. As such, the statistics of errors were assumed constant and were included in the calculation of the minimum safe distance from the designated borders and the aircraft. The data used to incorporate the terrain and buildings of the Area B flight space into AVDS were simply to demonstrate the usefulness of the methods, and as such were only estimates; however, if available or generated, extremely high fidelity terrain and building maps can be used. The linear interpolation algorithm used to estimate the altitude above safe operating altitude clearly is not meant to be an exact figure, but rather a conservative estimate to maximize safety during flight.

1.7 Methodology

Methodology for this research with regards to the development of SASVRT involved starting with a compilation of simple problems and concatenating them into a robust algorithm that works in general for all cases. Once the simple problems were shown to work for all cases, they were compiled into a more complex problem and adjusted to produce desired results on the more complex level. Eventually, this led to a set of general functions that could be used by a program to produce desired quantitative results. After SASVRT was developed, it was used in Hardware in the Loop (HITL) simulations to verify that the results produced matched the simulated results.

The development of the synthetic vision portion of SASVRT involved developing an interface program to relay aircraft telemetry data to the AVDS program in real-time. As its name implies, AVDS (Aviator Visual Design Simulator) provides tools specifically developed to visualize aircraft simulations; however, the interface developed in this thesis enables AVDS to be used to synthetically visualize UAVs (or any vehicles) in real-time. Because AVDS uses a different coordinate system, it was necessary to do coordinate transformations when transferring telemetry data from the UAV to AVDS. Therefore, simulations were conducted in cardinal directions to ensure the visualization in AVDS matched the true vehicle direction.

1.8 Thesis Overview

Chapter II examines closely the equipment and software used in this research, and discusses related research. Chapter III explains the mathematical methods used to develop SASVRT and the integration of SASVRT with the Piccolo II autopilot system. Chapter IV discusses AVDS and its use in developing the synthetic vision relay. Chapter V discusses how the developed methods and the program were tested by simulating flight tests. Chapter VI illustrates the results of the simulations and analyzes the results. Chapter VII draws conclusions based on the analyzed results of the simulated flight testing of the situational awareness program.

II. Background

2.1 Overview

Chapter II describes the equipment and software that made this research possible. The Rascal 110 aircraft and its avionics are discussed briefly to put in perspective the immediate application of this research. The Piccolo II autopilot and its accompanying Software Development Kit (SDK) are described to more fully frame the problem of implementation. Finally, a brief explanation of the need to transform between latitude, longitude, and altitude coordinates and north, east, and up coordinates is given.

2.2 Sig Rascal 110 Aircraft

The Sig Rascal 110 R/C aircraft was chosen as the demonstration platform for this research due to the existing data and flight test procedures developed by Captain Nidal Jodeh (Jodeh, 2006). The Rascal 110 was an attractive choice for an autopilot test platform due to its large interior volume and stable flight characteristics (Jodeh, 2006). The Rascal 110 has been used by several groups conducting research on autonomous UAVs including the University of California-Berkeley conducting vision based navigation (Frew, et al., 2004). The aircraft is among the largest R/C planes that come almost ready to fly (ARF) requiring approximately 40 man hours to assemble and tune, compared to the estimated 200+ hours it would take to assemble from a non-ARF kit. A modified fuel tank allows the Rascal to fly for approximately two hours on a single tank of fuel (Jodeh, 2006). In addition to these ideal qualities, a high-fidelity hardware-in-the-loop simulator was developed by Captain Jodeh in his thesis, which facilitates guidance, navigation, and control development pre-flight testing (Jodeh, 2006). A photograph of the Rascal aircraft is shown in Figure 2.

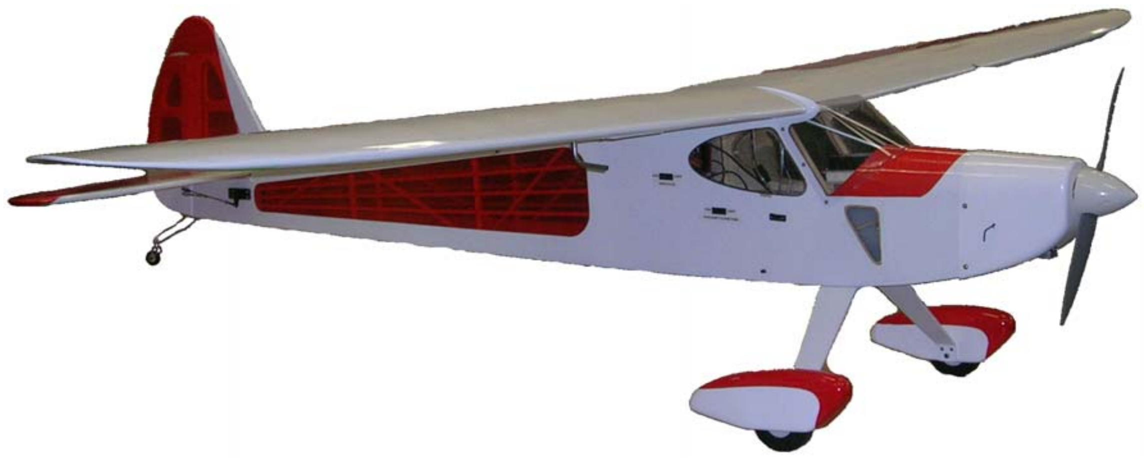


Figure 2 Sig Rascal 110

2.3 Avionics

2.3.1 Radio Control System. The radio control system includes a Futaba 9CAP/9CAF 8 Channel transmitter, which sends commands to a Futaba R149DP PCM 1024 receiver aboard the aircraft. The receiver relays the commands to high torque servos in order to deflect the control surfaces. The Rascal can be controlled by either a standard Radio Control system or the Piccolo II autopilot. Switching between the two is controlled by a Fail Safe Control Relay designed by the Sensors Directorate in the Air Force Research Labs Sensors Directorate (AFRL/SN).

2.3.2 Piccolo II. The Piccolo II autopilot is a commercially available autopilot system from Cloud Cap Technologies that includes an autopilot box on board the aircraft, Operator Interface software, and a Ground Station that relays communications between the serial port of the personal computer (PC) running the Operator Interface, and the UHF Antenna connected to the aircraft. The ground components are shown in Figure 3 and the autopilot box is shown in Figure 4. The Piccolo II is a fully autonomous autopilot system designed for small unmanned vehicles. It obtains information by way of a GPS receiver, a dual ported dynamic pressure sensor, an absolute ported barometric pressure sensor, an on board temperature sensor,

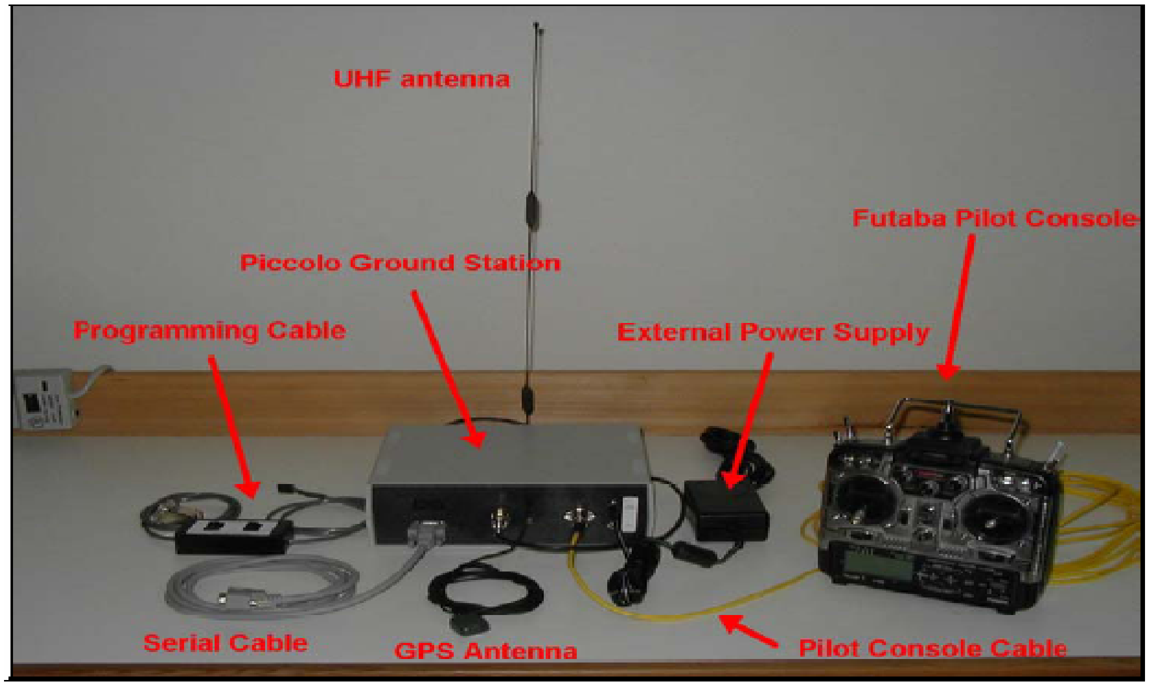


Figure 3 Ground Station Components

onboard rate gyros, and dual two-axis ADXL210e accelerometers. A Kalman filter analyzes the data and estimates gyro bias and attitude in order to minimize the error in telemetry data (Vaglienti, 2005). This telemetry data is relayed through the Ground Station to the Operator Interface, which provides the user with the data and the ability to pass commands back to the autopilot. The various functions and data displays of the Operator Interface are detailed in the Piccolo II System User's Guide Version 1.3.0 available through Cloud Cap Technology (Vaglienti, 2005). The Piccolo II avionics system is depicted in Figure 5.

Various commands can be entered in the Operator Interface to be relayed to the aircraft via the Ground Station and Piccolo II box including altitude, airspeed, etc. However, the most important commands to autopilot navigation are waypoints in the form of waypoint lists. The Operator Interface allows the user to input a loop of waypoints for the aircraft to follow as seen in Figure 6. Through various gain

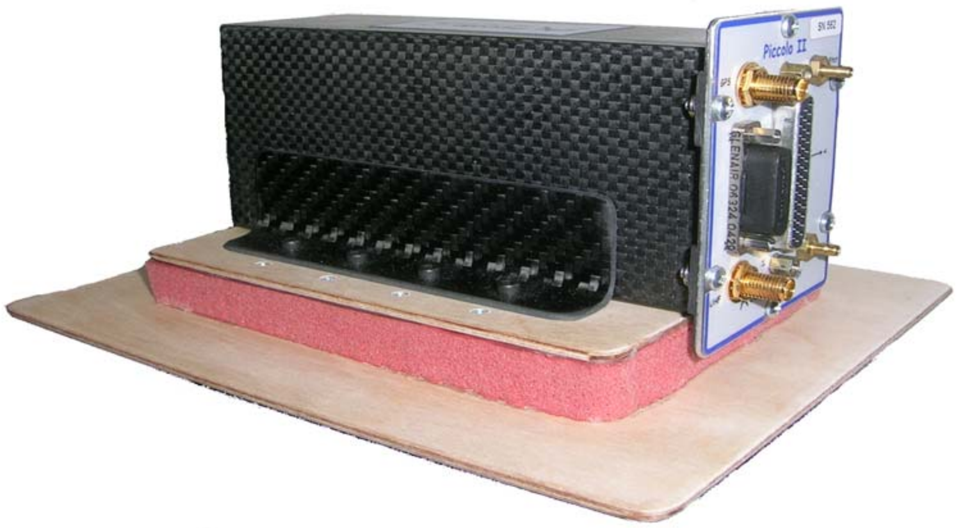


Figure 4 Piccolo II Autopilot Box

settings the aircraft will follow the track more or less tightly and this can be used to tune the autopilot for optimal performance. In order to test the hard boundary and minimum altitude algorithms developed in this thesis, several different waypoint lists were sent and the data analyzed, as described in more detail in Chapter V.

2.4 Hardware in the Loop Simulator

In any kind of flight testing, manned or unmanned, simulation prior to flight testing is vital to fine tuning the items to be tested. Available to the Piccolo II autopilot system is a hardware in the loop (HITL) simulator. The operator interface view, as depicted in Figure 7, is identical for the simulation and actual flight test with the exception of the Piccolo II autopilot box receiving flight data from the simulator (B) rather than the aircraft and its on board sensors(A). When simulating, a PC runs the simulator software and through its Universal Serial Bus (USB) port sends

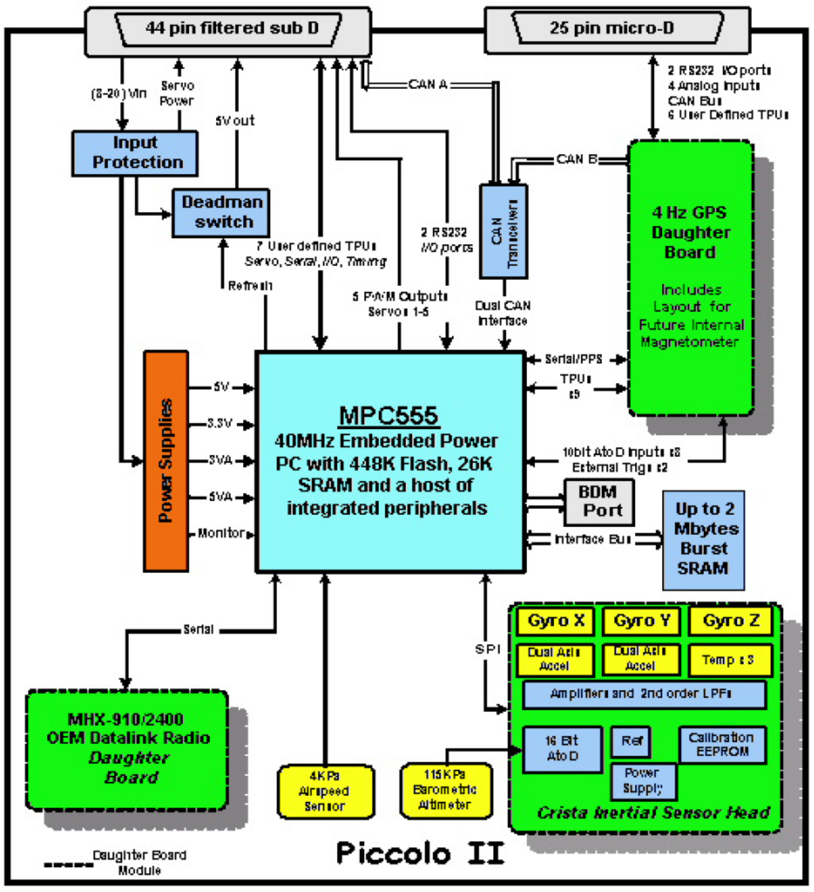
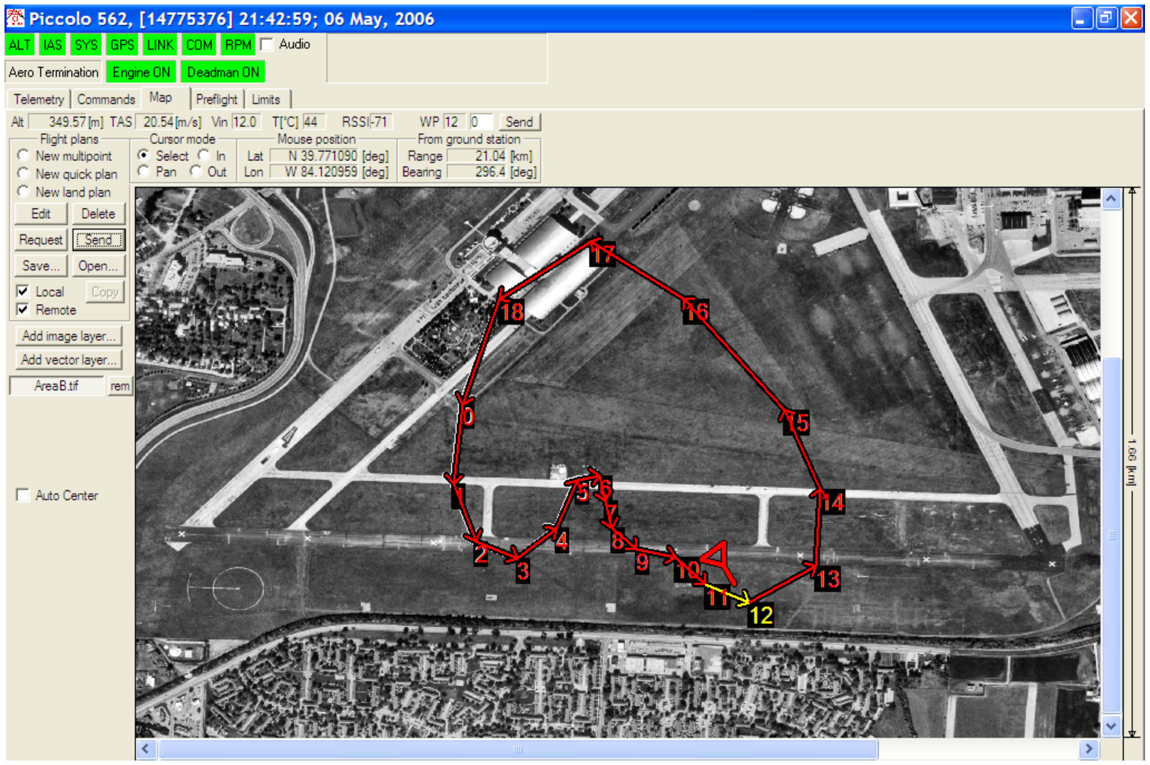


Figure 5 Avionics System Flow Diagram


Piccolo 562, [14775376] 21:42:59; 06 May, 2006

Alt 349.57 [m] TAS 20.54 [m/s] Vin 12.0 T[°C] 44 RSSI -71 WP 12 0

Cursor mode: [✓ Select] In Pan Out

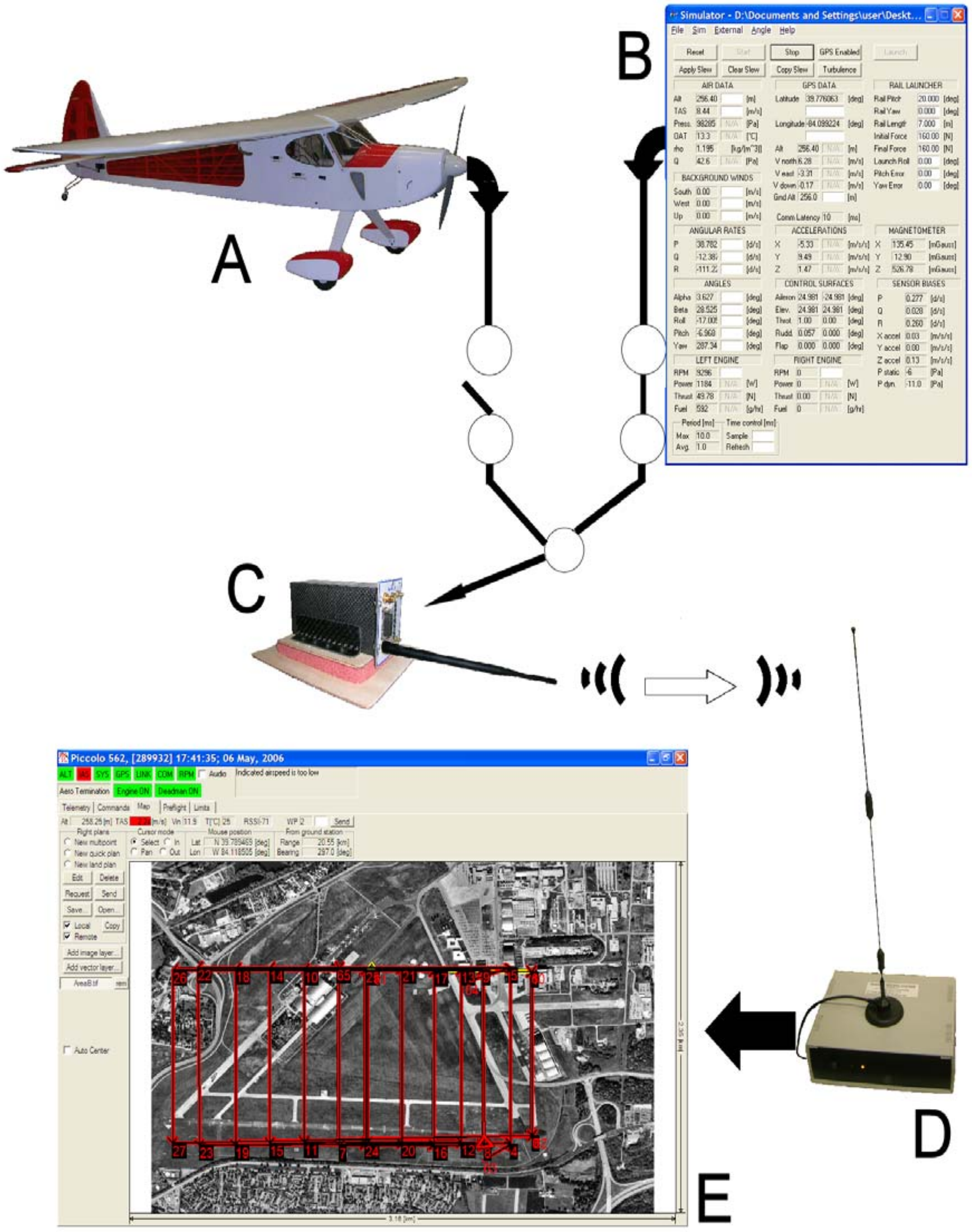
Mouse position: Lat N 39.771090 [deg] Lon W 84.120959 [deg]

From ground station: Range 21.04 [km] Bearing 296.4 [deg]

[✓ Local] [✓ Remote]

AreaB.tif

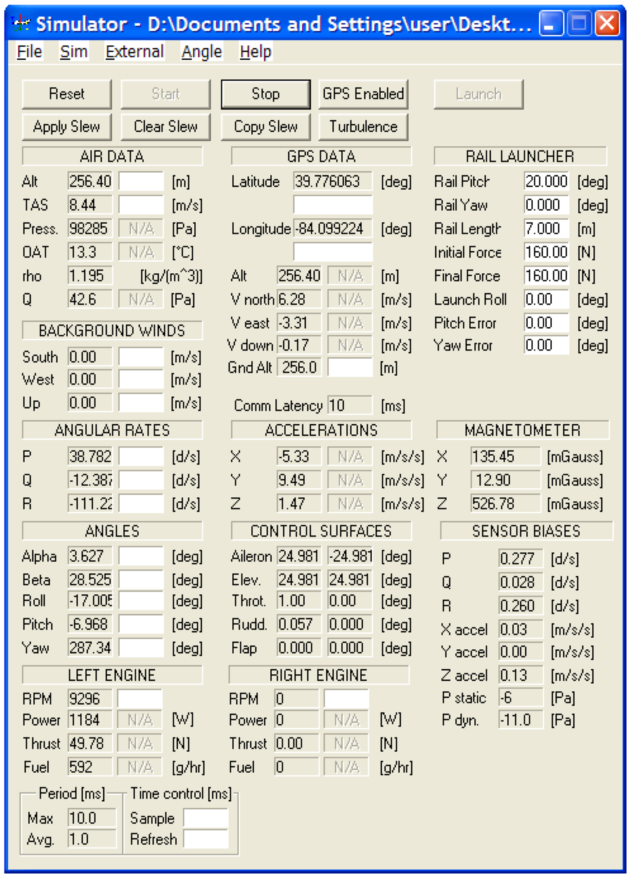
Figure 6 Waypoint Tracking


A
B
C
D
E
Simulator - D:\Documents and Settings\user\Deskt...
File Sim External Angle Help
Reset Start Stop GPS Enabled Launch
Apply Slew Clear Slew Copy Slew Turbulence
Piccolo 562, [289932] 17:41:35; 06 May, 2006
ALT IAS SYS GPS LINK COM RPM Audio
Aero Termination Engine ON Deadman ON
Telemetry Commands Map Preflight Limits
Edit Delete Request Send Save... Open... Copy
Add image layer... Add vector layer...
Auto Center

```
AIR DATA                      GPS DATA                          RAIL LAUNCHER
Alt    256.40        [m]      Latitude  39.776063   [deg]       Rail Pitch    20.000  [deg]
TAS    8.44          [m/s]                                      Rail Yaw      0.000   [deg]
Press. 98285  N/A    [Pa]     Longitude -84.099224  [deg]       Rail Length   7.000   [m]
OAT    13.3   N/A    [°C]                                       Initial Force 160.00  [N]
rho    1.195       [kg/m^3]]  Alt     256.40  N/A  [m]          Final Force   160.00  [N]
Q      42.6   N/A    [Pa]     V north 6.28    N/A  [m/s]        Launch Roll   0.00    [deg]
                              V east  -3.31   N/A  [m/s]        Pitch Error   0.00    [deg]
BACKGROUND WINDS              V down  -0.17   N/A  [m/s]        Yaw Error     0.00    [deg]
South  0.00          [m/s]    Gnd Alt 256.0        [m]
West   0.00          [m/s]
Up     0.00          [m/s]    Comm Latency 10      [ms]

ANGULAR RATES                 ACCELERATIONS                     MAGNETOMETER
P      38.782        [d/s]    X   -5.33   N/A  [m/s/s]          X  135.45  [mGauss]
Q      -12.38        [d/s]    Y   9.49    N/A  [m/s/s]          Y  12.90   [mGauss]
R      -111.2        [d/s]    Z   1.47    N/A  [m/s/s]          Z  526.78  [mGauss]

ANGLES                        CONTROL SURFACES                  SENSOR BIASES
Alpha  3.627         [deg]    Aileron 24.981  -24.981 [deg]     P        0.277  [d/s]
Beta   28.525        [deg]    Elev.   24.981  24.981  [deg]     Q        0.028  [d/s]
Roll   -17.00        [deg]    Throt   1.00    0.00    [deg]     R        0.260  [d/s]
Pitch  -6.968        [deg]    Rudd.   0.057   0.000   [deg]     X accel  0.03   [m/s/s]
Yaw    287.34        [deg]    Flap    0.000   0.000   [deg]     Y accel  0.00   [m/s/s]
                                                                Z accel  0.13   [m/s/s]
LEFT ENGINE                   RIGHT ENGINE                      P static -6     [Pa]
RPM    9296                   RPM    0                          P dyn.   -11.0  [Pa]
Power  1184   N/A  [W]        Power  0     N/A  [W]
Thrust 49.78  N/A  [N]        Thrust 0.00  N/A  [N]
Fuel   592    N/A  [g/hr]     Fuel   0     N/A  [g/hr]

Period [ms]    Time control [ms]
Max  10.0      Sample
Avg. 1.0       Refresh
```

```
Indicated airspeed is too low
Alt 258.25 [m]  TAS [m/s]  Vin 11.9  T[C] 25  RSSI-71  WP 2  Send
Flight plans        Cursor mode          Mouse position            From ground station
( ) New multipoint  (•) Select ( ) In    Lat  N 39.789469 [deg]    Range   20.55 [km]
( ) New quick plan  ( ) Pan    ( ) Out   Lon  W 84.118505 [deg]    Bearing 297.0 [deg]
( ) New land plan
[✓] Local
[✓] Remote
```

Figure 7 HITL Simulator


Simulator - D:\Documents and Settings\user\Deskt...
File Sim External Angle Help
Reset
Start
Stop
GPS Enabled
Launch
Apply Slew
Clear Slew
Copy Slew
Turbulence

AIR DATA

Alt	256.40		[m]
TAS	8.44		[m/s]
Press.	98285	N/A	[Pa]
OAT	13.3	N/A	[°C]
rho	1.195		[kg/(m^3)]
Q	42.6	N/A	[Pa]

BACKGROUND WINDS

South	0.00		[m/s]
West	0.00		[m/s]
Up	0.00		[m/s]

ANGULAR RATES

P	38.782		[d/s]
Q	-12.387		[d/s]
R	-111.22		[d/s]

ANGLES

Alpha	3.627		[deg]
Beta	28.525		[deg]
Roll	-17.005		[deg]
Pitch	-6.968		[deg]
Yaw	287.34		[deg]

LEFT ENGINE

RPM	9296		
Power	1184	N/A	[W]
Thrust	49.78	N/A	[N]
Fuel	592	N/A	[g/hr]

Period [ms]

Max	10.0
Avg.	1.0

Time control [ms]

Sample	
Refresh	

GPS DATA

Latitude	39.776063		[deg]
Longitude	-84.099224		[deg]
Alt	256.40	N/A	[m]
V north	6.28	N/A	[m/s]
V east	-3.31	N/A	[m/s]
V down	-0.17	N/A	[m/s]
Gnd Alt	256.0		[m]
Comm Latency	10		[ms]

ACCELERATIONS

X	-5.33	N/A	[m/s/s]
Y	9.49	N/A	[m/s/s]
Z	1.47	N/A	[m/s/s]

CONTROL SURFACES

Aileron	24.981	-24.981	[deg]
Elev.	24.981	24.981	[deg]
Throt.	1.00	0.00	[deg]
Rudd.	0.057	0.000	[deg]
Flap	0.000	0.000	[deg]

RIGHT ENGINE

RPM	0		
Power	0	N/A	[W]
Thrust	0.00	N/A	[N]
Fuel	0	N/A	[g/hr]

RAIL LAUNCHER

Rail Pitch	20.000	[deg]
Rail Yaw	0.000	[deg]
Rail Length	7.000	[m]
Initial Force	160.00	[N]
Final Force	160.00	[N]
Launch Roll	0.00	[deg]
Pitch Error	0.00	[deg]
Yaw Error	0.00	[deg]

MAGNETOMETER

X	135.45	[mGauss]
Y	12.90	[mGauss]
Z	526.78	[mGauss]

SENSOR BIASES

P	0.277	[d/s]
Q	0.028	[d/s]
R	0.260	[d/s]
X accel	0.03	[m/s/s]
Y accel	0.00	[m/s/s]
Z accel	0.13	[m/s/s]
P static	-6	[Pa]
P dyn.	-11.0	[Pa]

Figure 8 Simulator Screen Shot

the data to the Piccolo II that it would be receiving were it flying in the prescribed simulated condition. This data is also available on the simulator's operating screen as shown in Figure 8. Then, just as in an actual flight test, the Piccolo II box (C) communicates the telemetry data to the Ground Station (D) which relays it to the Operator Interface (E) running on a separate PC. Captain Jodeh in his research determined and entered the inputs to the simulator in order to correctly model the characteristics of the Rascal 110 aircraft (Jodeh, 2006). This simulator was vital in testing and debugging software developed in this thesis to increase situational awareness.

2.5 Software Development Kit

Along with the Operator Interface, the Piccolo II system comes with a Software Development Kit (SDK). This is a C++ based set of codes that provide the ability to intercept and decode data packets sent between the Operator Interface and the on board Piccolo II box. This gives the user access to the telemetry data measured by the Piccolo II autopilot as well as the ability to send data packets back to the aircraft. This communications managing technique was the basis for developing and implementing SASVRT.

2.5.1 Existing C++ Code. Randall Plate, an intern in the ANT Center in the Summer of 2006 began developing a C++ code with the SDK that allowed a user to view the telemetry data in an external program. Key points in the code that would be used later in this research were implementation of the SDK to decode data packets and a conversion program from latitude, longitude, and altitude, to north, east, and up coordinates.

2.5.2 Latitude, Longitude, Altitude, and East, North, Up. Latitude, longitude, and altitude (LLA) are excellent parameters for navigation and describing any location on the earth. With its origin at the mass center of the earth, the LLA coordinate system defines location as the angles from reference vertical and horizontal planes, for which the Prime Meridian and Equator respectively are commonly used, and also the the distance along a line normal to a reference ellipsoid estimating the surface of the earth (Baleri). However, in order to determine distance or time to a boundary or terrain, it is more convenient to assume a flat earth and use a Cartesian coordinate system rather than a spherical system. Cartesian coordinates for example, allow velocities to be described by three distance rates of the same units rather then two angular rates and a distance rate making interpretations more intuitive and calculations much simpler. A common Cartesian coordinate convention is known as the East, North, Up (ENU) coordinates system. Conversion between these systems is discussed in detail in Section 3.4.3.1.

2.6 Aviator Visual Display Simulator

The Aviator Visual Display Simulator by Rasmussen Simulation Technologies, LTD., is a program incorporating multiple operating modes designed to be universally applicable to a multitude of aircraft simulation visualization needs (Rasmussen, 2005). It can operate independently of other programs and simlulate a variety of aircraft with inputs from a joystick, mouse, or keyboard. It can also obtain telemetry data from a MATLAB script or C++ program using shared memory. These methods can also be used in network mode in which the data is broadcast over a network and can be read by any program running on the network. Several models of common aircraft are available for visualization, or users can define a new aircraft using the corners of a triangle or planar quadrilateral in three-dimensional Cartesian coordinates. These models can also be in fixed locations in order to model terrain, and can have texture image files applied to them. Furthermore, text files of telemetry can be used in the program's playback mode to visualize past flights in which the flight data was recorded. These are just some of the features available in the Aviator Visual

Display Simulator (AVDS), but this encompasses the scope of which this research makes use of the program.

2.7 Summary

Given the presented aircraft, equipment, and software information, the reader has a general sense of the circumstances under which the presented research was started. The Rascal 110 modelled by Captain Jodeh, the C++ code developed by Randall Plate, and the AVDS program provide the foundation to extend the situational awareness and safety of flight testing presented in the following chapters.

III. Development of a Proximity Warning System

3.1 Overview

As previously mentioned, the end product of this research is a Situational Awareness and Synthetic Vsision Relay Tool (SASVRT) to aid UAV testers and operators as they perform their mission. This chapter presents the mathematical foundation in Cartesian coordinates that provide the basis for the SASVRT. It starts off with teh equations required to have situational awareness in two dimensions, and then extends thos results to the three-dimensional case.

3.2 Two Dimensional Proximity Warning

3.2.1 Equations of Lines. Given two points, (x_0,y_0) and (x_1,y_1) as seen in Figure 9, an equation can be determined for the line connecting these points by first determining the slope m of the line given by the equation

$$m = \frac{y_1 - y_0}{x_1 - x_0}, \tag{1}$$

which can then be used to determine the y intercept b by the equation

$$b = y_1 - mx_1, \tag{2}$$

ultimately leading to the general form of the line relating all values of x and y in the equation

$$y = mx + b. \tag{3}$$

Given an arbitrary point (x_a,y_a) not on the line, the closest distance between this point and the line described by Equation 3 can be determined by first describing the

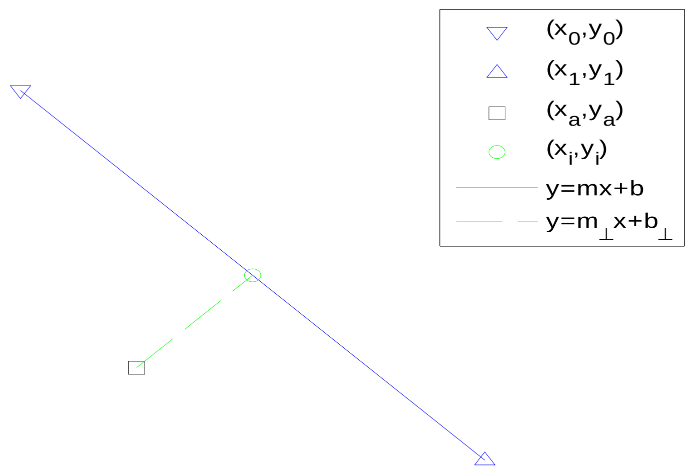


Figure 9 Distance from Point to Line

line through the point (x_a, y_a) perpendicular to the line in Equation 3, for it is along this line that the shortest distance is measured. This perpendicular line is described by the equation

$$y = \frac{-1}{m}x + b_a, \tag{4}$$

where b_a can be solved for by the equation

$$b_a = y_a + \frac{1}{m}x_a, \tag{5}$$

with m being equal to the slope of the line in Equation 3. The intersection of the two lines in Equations 3 and 4 at point (x_i,y_i) satisfies the equation

$$mx_i + b = \frac{-1}{m}x_i + b_a, \tag{6}$$

found by equating the equations for y_i. Solving Equation 6 for x_i shows

$$x_i = \frac{b_a - b}{\frac{1}{m} + m} \tag{7}$$

which can then be used to solve for y_i

$$y_i = mx_i + b. \tag{8}$$

Then the shortest distance, d_a between point (x_a, y_a) and the line described by Equation 3 is the distance between the points x_a, y_a and x_i, y_i, which is found by the equation

$$d_a = \sqrt{(x_a - x_i)^2 + (y_a - y_i)^2}. \tag{9}$$

It is important to note here that in the special cases where $m = 0$ or $m = \infty$ in the given coordinate system, this problem breaks down into a simpler solution, where $d_a = x_a - x_1$ for $m = \infty$, or $d_a = y_a - y_1$ for $m = 0$. When solving this problem in the for all lines this is an important check to have in a program so it does not attempt to solve for infinity. Furthermore, if the equation

$$\sqrt{(x_0 - x_i)^2 + (y_0 - y_i)^2} + \sqrt{(x_1 - x_i)^2 + (y_1 - y_i)^2} = \sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2}, \tag{10}$$

is satisfied, then the point (x_i, y_i) falls between points (x_0, y_0) and (x_1, y_1) on the line described by 3. If it is not satisfied, then the shortest distance from point (x_a, y_a) to the line segment between (x_0, y_0) and (x_1, y_1) is the lesser of

$\sqrt{(x_0 - x_a)^2 + (y_0 - y_a)^2}$ and $\sqrt{(x_1 - x_a)^2 + (y_1 - y_a)^2}$, the distances to the end points of the segment.

An extension of this mathematics lends itself to the problem of a a polygon comprised of line segments connecting a series of points. It is clear that one can determine the minimum distance between an arbitrary point (x_a, y_a) and a polygon.

3.2.2 Closed Polygons. At the very root of a boundary warning system is the necessity of a systematic method to determine at a given time, if the current location, (x_a, y_a) is within or outside of a boundary defined by line segments connnecting the points defining a closed polygon that can be easily coded into an algorithm. If a line was extended in a single direction from the (x_a, y_a) out to infinity and the number of intersections with the line segments defining the edges of the polygon were counted, it could be determined whether or not (x_a, y_a) is within the bounds of the polygon. If the line crosses an even number of boundaries, then (x_a, y_a), is located outside of the bounds of the polygon. If the line crosses an odd number of boundaries, then the current location is within the bounds of the polygon. Figures 10 and 11 show an arbitrary polygon defined by seven points. For the sake of simplicity, in the algorithms used in this research, the arbitrary line from the test point was given zero slope and taken to be all values greater than the test point's x value. Clearly Figure 10 shows that line extending from (x_a, y_a) horizontally to the right crosses four sides of the polygon and the three other intersection points are not contained on the line segments that make up the polygon. Based on the aforementioned criteria, this point is determined to be outside of the bounds. Using the same polygon, but changing the location of (x_a, y_a), Figure 11 shows that the horizontal line extending from (x_a, y_a) to the right only crosses three boundaries to the right, with two intersection points not being contained on the line segment, and the other two intersection points being to the left of (x_a, y_a). Therefore this (x_a, y_a) is determined to be within the bounds of the polygon.

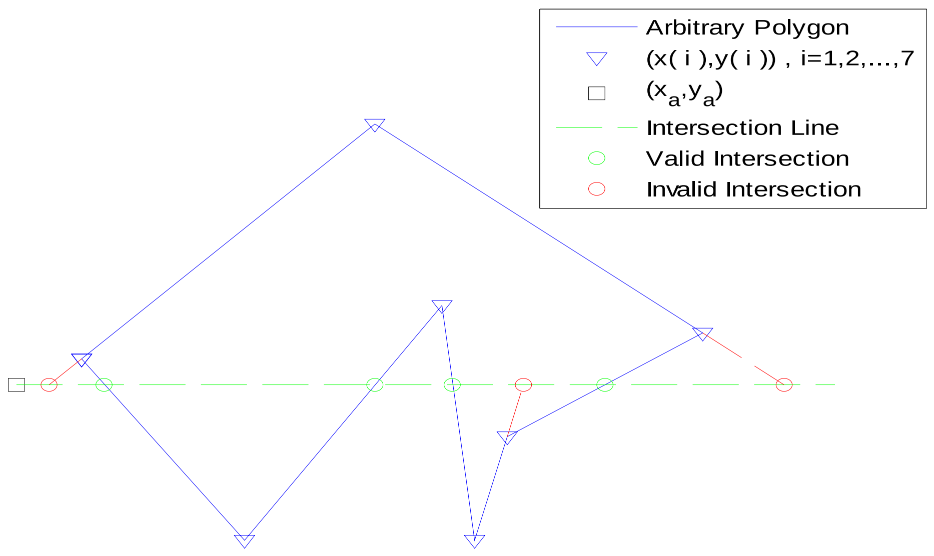


Figure 10 Arbitrary Polygon with Point Outside Bounds

The mathematical techniques presented in the previous sections of this chapter can be coded as to systematically determine if a position is within defined boundaries. With regards to geographic position, given a current latitude and longitude, a program can reference a prescribed set of latitudes and longitudes for a designated area and determine if the location is within the bounds of the flying area, or as it will be called hereafter the *hard boundary*.

3.3 Three Dimensional Proximity Warning

3.3.1 Linear Interpolation. In this section, the two dimensional mathematics will be extended to three dimensions to develop an altitude proximity warning system. If one were to define several closed polygons of different altitudes, they could be thought of contours of altitude on a topographic map describing the terrain of a given area. Then it would be feasible to interpolate between the lines to estimate

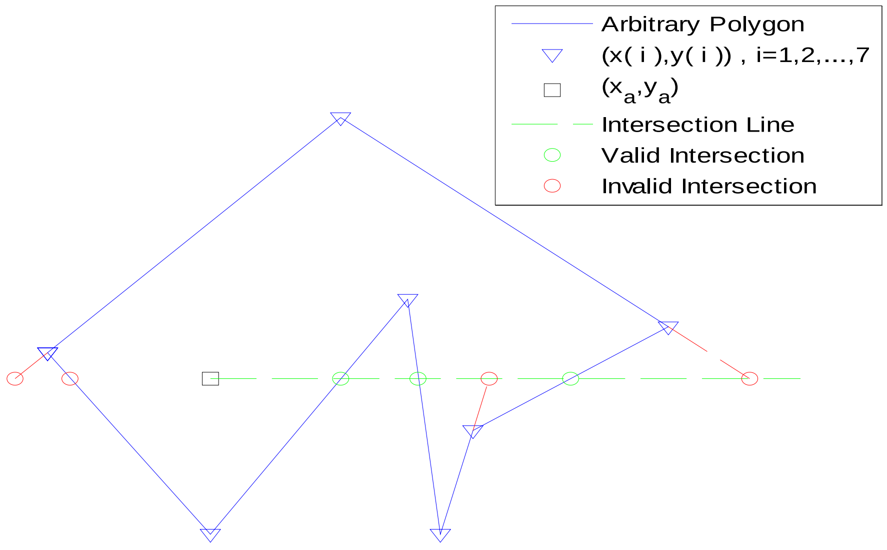


Figure 11 Arbitrary Polygon with Point Inside Bounds

the altitude of the ground at a test point, and call this the minimum altitude for that point. The linear interpolation algorithm created to do this does so by taking a current two-dimensional location (x_c,y_c) and calculating the shortest distances to every line describing a side of a defined polygon representing a contour of altitude, as well as the distances to the points describing the corners of the polygons. If

the shortest distance to a point on a line describing the side of a polygon is not contained on the line segment actually making up the polygon, this value is thrown out. The result is a set of points of known altitude, but most of the points are too far away the current location to be used in a minimum altitude calculation for the current location. As such, the algorithm then determines the shortest remaining distance, either to a side or corner of a polygon, and stores the coordinates of the point corresponding to this shortest distance as (x_1,y_1). The algorithm then determines the second shortest distance and subjects the corresponding point to several

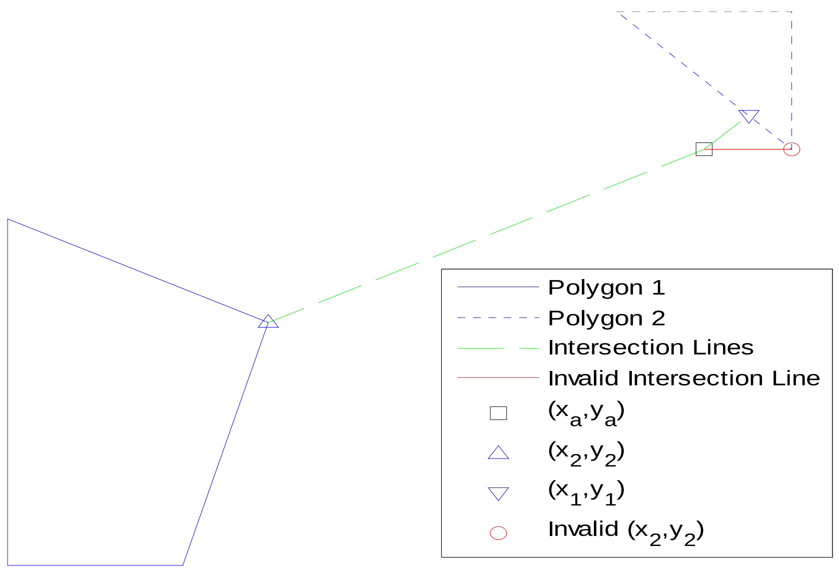


Figure 12 3-D Interpolation Restriction 1

restrictions. The first restriction, shown in Figure 12 dictates that the second closest point, (x_2,y_2) , must come from a different polygon than (x_1, y_1), to ensure that an interpolation occurs between two different contours. The next restriction, shown in Figure 13, states that if (x_c, y_c) is located outside the bounds of the polygon containing (x_1, y_1), then (x_2, y_2) must also be located outside the bounds. This ensures that the interpolated value is not accidentally extrapolated and ends up with a estimated ground height larger than the larger of or smaller than the smaller of the altitudes of the polygons containing (x_1, y_1) and (x_2, y_2).

Similarly, Figure 14 shows that if (x_c, y_c) is located within the bounds of the polygon containing (x_1, y_1), then (x_2, y_2) must also come from within the polygon for the same reasons as before. Figure 15 demonstrates the next rule which can be stated, the distance between (x_c, y_c) and (x_2, y_2) cannot be greater than the distance between (x_1, y_1) and (x_2, y_2), again to avoid accidentally extrapolating a

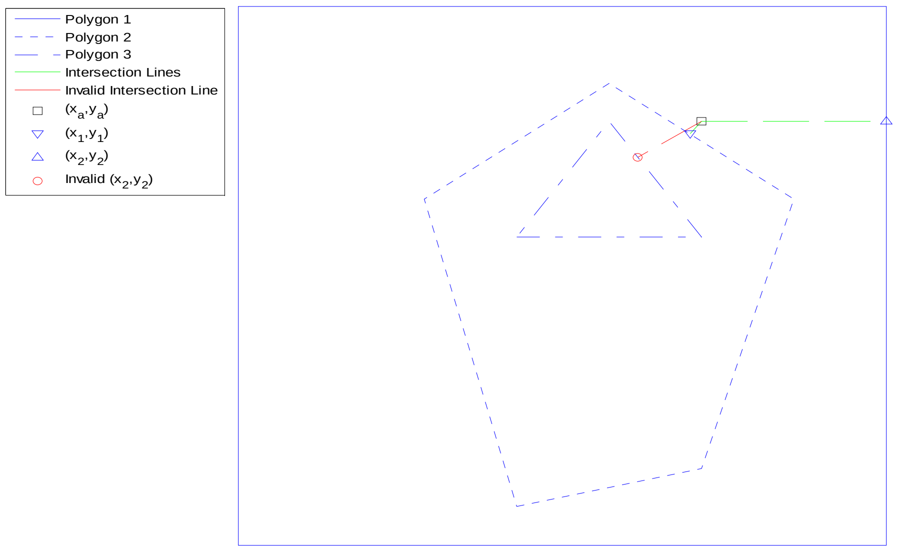


Figure 13 3-D Interpolation Restriction 2

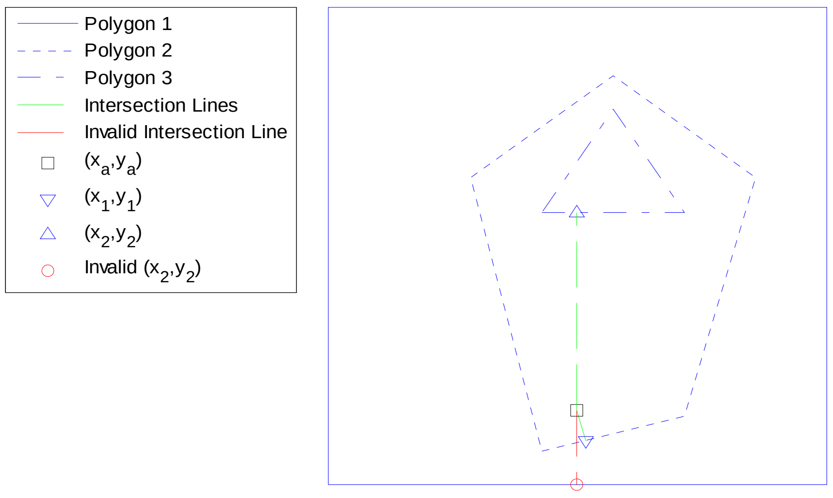


Figure 14 3-D Interpolation Restriction 3

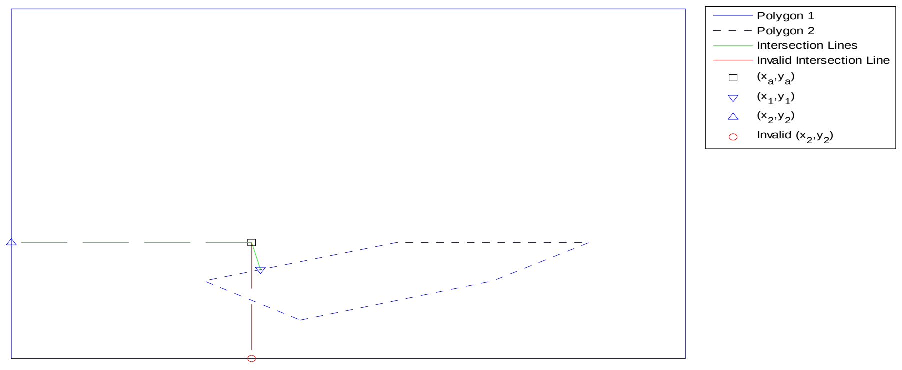


Figure 15 3-D Interpolation Restriction 4

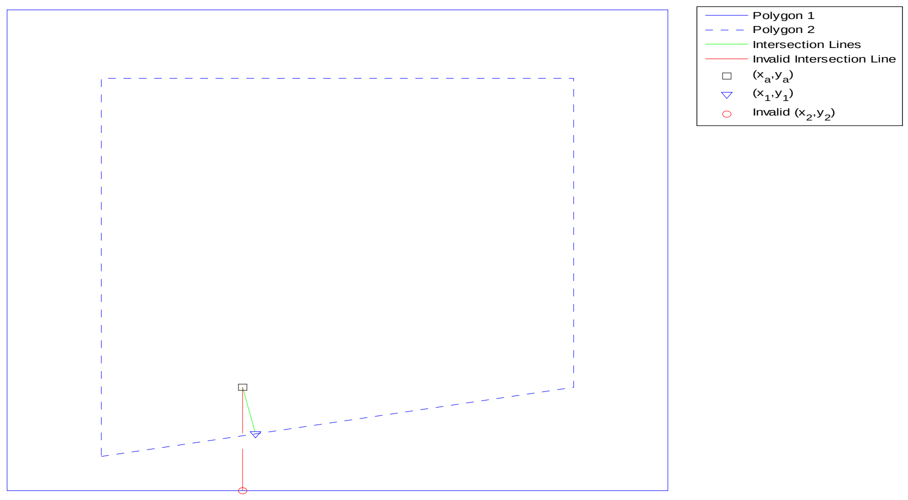


Figure 16 3-D Interpolation Restriction 5

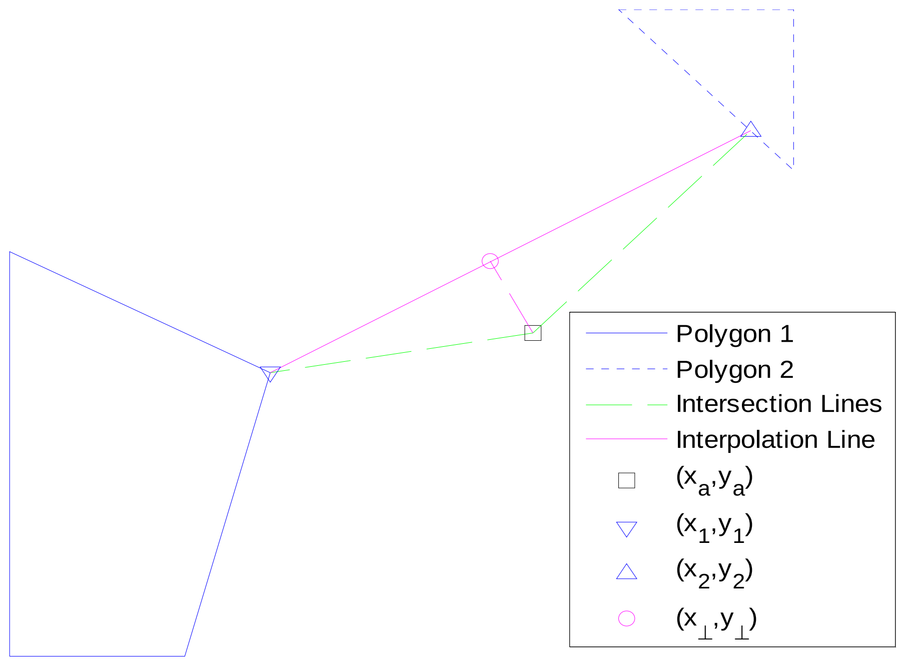


Figure 17 Three Dimensional Interpolation

minimum altitude. If a point violates any of these restrictions, it is thrown out and the next closest point is examined by the algorithm. Finally, if no points satisfy the restrictions for (x_2, y_2) as shown in Figure 16, the minimum altitude is assumed to be that of the polygon containing (x_1, y_1). Otherwise, the algorithm has stored the two points, (x_1, y_1) and (x_2, y_2) as the two valid points closest to (x_c, y_c).

In order to estimate the minimum altitude, the equation for the line between the points (x_1, y_1) and (x_2, y_2) is then determined. Next, the point on this line closest to (x_c, y_c) is determined and called $(x_\perp, y_\perp)$, as shown in Figure 17. The distances between $(x_\perp, y_\perp)$ and (x_1, y_1), $d_{1\perp}$, $(x_\perp, y_\perp)$ and (x_2, y_2), $d_{2\perp}$, and (x_1, y_1) and (x_2, y_2), d_{12} are then used in the equation

$$h_{est} = \frac{d_{1\perp}}{d_{12}} h_2 + \frac{d_{2\perp}}{d_{12}} h_1, \tag{11}$$

to determine the estimated height of the ground at (x_c, y_c), h_{est}, if h_2 is the altitude of the polygon containing (x_2, y_2), and h_1 is the altitude of the polygon containing (x_1, y_1).

3.3.2 Urban Canyon Check. An advantage of using the linear interpolation algorithm is that it also includes a maximum slope check. Hypothetically, an aircraft could be flying along calculating the ground height and be comfortably above the minimum flight altitude, and then crash into a tall object, (e.g. a building). Therefore, a check was built in to the altitude interpolation algorithm to determine the slope of the line from the current location to all points of known altitude. If based on the maximum climbr rate performance, the aircraft cannot climb at the at the steepest slope, the minimum altitude is defined as the distance above the ground at which the aircraft would be able to climb to that countour of altitude. Thus, as the example in Figure 18 shows, a steep increase in terrain, like the simulated buildings represented by the blue surface, can be smoothed in the synthetic terrain like the red surface, ensuring that at any instant the aircraft is able to climb out of

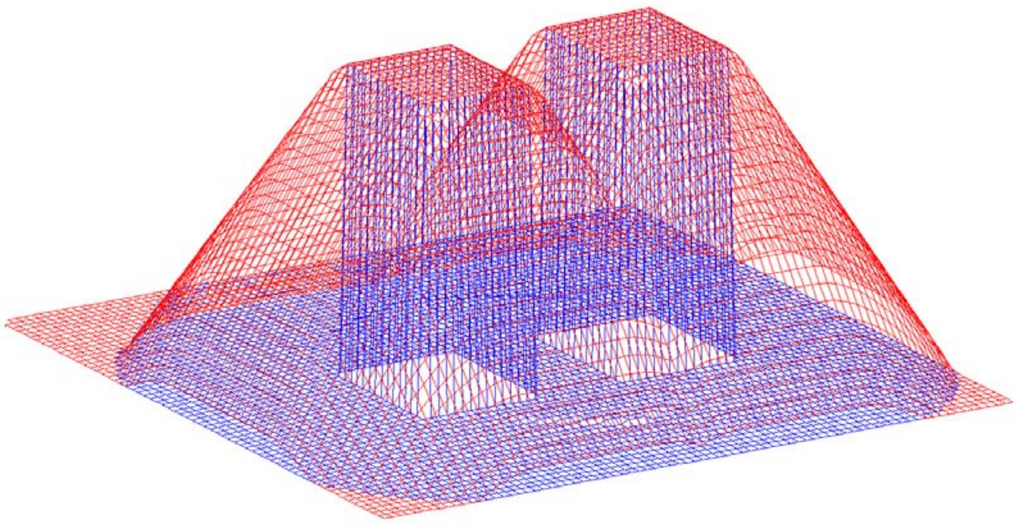


Figure 18 Urban Canyon Check

an imminent collision. Clearly this is on the conservative side as it is independent of relative heading to the buildings, and in fact assumes the aircraft is flying directly toward the building.

3.4 Integrating Proximity Warning with the Piccolo II

3.4.1 Overview. In order to make use of the mathematical techniques described above in any real time situation, a program needs access to an aircrafts location relative to boundaries and contours of altitude. This section describes how SASVRT uses the Software Development Kit (SDK) provided with Piccolo, to intercept telemetry data, transform it into data easily used in calculations, and apply the minimum altitude interpolation and proximity warning algorithms to quantify information pertinent to situational awareness.

3.4.2 Intercepting Piccolo II Packets. As mentioned in Section 2.5, the SDK allows for the interception of data packets travelling between the Piccolo II and the Operator Interface. As such, retrieving the inputs for the above algorithms becomes quite straightforward. First, the packets coming from the Piccolo II are intercepted. Then, depending on what type of packet was intercepted, whether it be telemetry, control, etc. the packet is decoded and the information stored in C++ data structures. From there it is simply a matter of retrieving the data from the structure and using it in the algorithms to calculate pertinent information such as distance to the hard boundary, time until crossing the hard boundary, distance above minimum altitude, time until crossing below the minimum altitude, or whatever else the observer may want to know.

3.4.3 Determining Current Location. Obtaining the data containing an aircraft's current location from the Piccolo II autopilot is straightforward for latitude, longitude, and altitude. However, as mentioned a Cartesian coordinate system is favorable over a spherical coordinate system for proximity calculations. Thus, there is one other step to determining the current location that is to be used in the algorithms, and that is transforming latitude, longitude, and altitude (LLA) to East, North, Up (ENU) coordinates. This is done by a separate algorithm and then used as the input for the linear interpolation algorithm.

3.4.3.1 Convert Latitude, Longitude, Altitude to East, North, Up. In order to easily calculate local rates and distances between the aircraft and boundaries or terrain, the easiest coordinate system to use is the ENU system. So to convert LLA to the ENU system, one must first determine a base location in an intermediate coordinate system known as the Earth Centered, Earth Fixed coordinate system (ECEF). The ECEF system is a three-dimensional Cartesian representation of a location relative to the mass center of the earth, and is fixed with the earth as it rotates. The Z-axis of this system is along the axis of rotation of the earth from the center through the North Pole. The X-axis travels from the center through the intersection of the Prime Meridian and the Equator. The Y-axis is then defined following the right-hand-rule convention of $k \times i = j$, if i,j, and k are unit vectors along the X,Y, and Z axes respectively. So the Y-axis is in the direction 90° east of the X-axis and 90° south of the Z-axis. If a cross-section of the earth is estimated by the ellipse shown in Figure 19, then the parameters describing the ellipse can be defined as,

$$a = 6378137, \tag{12}$$

$$b = 6356752.31424518, \tag{13}$$

$$e = \sqrt{\frac{a^2 - b^2}{a^2}}, \tag{14}$$

where a is the semi-major axis in meters, b is the semi-minor axis in meters, and e is the eccentricity of the ellipse. These parameters can then be used to define the radius of curvature N,

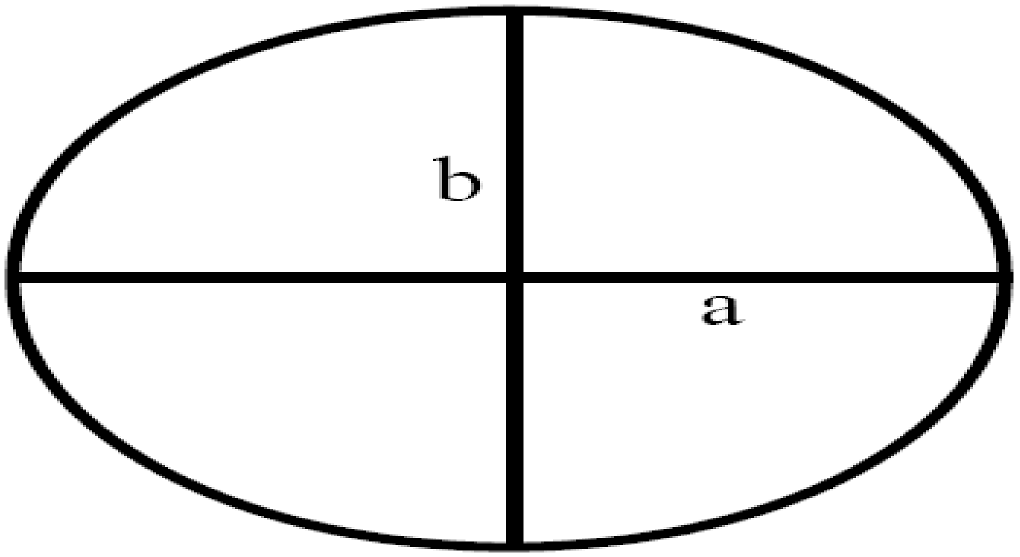


Figure 19 Ellipse Representing Earth

$$N = \frac{a}{\sqrt{1 - e^2 \sin^2 \Phi}} \tag{15}$$

where Φ is the latitude of the base location. This can then be used with the longitude, λ, and the altitude, h in the equations,

$$X = (N + h)\cos(\Phi)\cos(\lambda), \tag{16}$$

$$Y = (N + h)\cos(\Phi)\sin(\lambda), \tag{17}$$

$$Z = (\frac{b^2}{a^2}N + h)\sin(\Phi), \tag{18}$$

where X, Y, and Z are the distances in meters from the mass center of the earth along their respective axes[9]. The ECEF coordinate system can then be used to easily calculate distances and rates between objects by using vector algebra, yet it is still not an intuitive coordinate system when dealing with flight testing UAVs. However, with a single base point in ECEF coordinates, one can define a new coordinate system, placing its origin at the base point. The simplest and most intuitive basis for a new coordinate system for flight testing would then be a flat earth approximation, and would use coordinates for distance north of the base point, east of the base point, and above the base point. With this ENU coordinate system, it is easy to comprehend where the aircraft is relative to the base point, as well as determining relative velocity.

3.4.4 Altitude Comparison. Using the above methods then, at any given point during flight, SASVRT can provide the aircraft's current distance above a base point, as well as the estimated terrain distance above a base point using the linear interpolation and coordinate transformations. By subtracting these two, SASVRT can alert an observer to the aircraft's altitude relative to the ground, and if this distance surpasses a threshold, will give a indication for the pilot to "pull up!" Similarly, working in ENU makes it straight forward to implement distance and time to boudnary warnings.

3.5 Summary

As stated above, the main issue with heightening situational awareness with regards to minimum altitude and relative position to hard boundaries was coming up with a theoretical way to resolve these problems that could be converted to C++ code for integration with the Piccolo II autopilot. From there it was simply a matter of debugging and fool proofing the code in order to have it work in all situations.In addition, an investigation was performed to study which parameters would most improve situational awareness.

IV. Synthetic Vision in the Aviator Visual Design Simulator

4.1 Overview

The Aviator Display Simulator (AVDS) was used in this research to visualize an aircraft in flight. This chapter explains the process of the communication relay between the intercepted packets from the Piccolo II autopilot and the telemetry information passed to AVDS. Also, the development of three dimensional models of the Sig Rascal and surrounding terrain is discussed.

4.2 Communication Relay

By intercepting the data packets sent from the Piccolo II, the attitude and position of the aircraft relative to a fixed point can be determined. For this six degree of freedom model of the aircraft, the only necessary variables are displacement from the base point in X, d_x, displacement from the base point in Y, d_y, displacement from the base point in Z, d_z, bank angle, ϕ, pitch angle, θ, and yaw angle, ψ. Note also that these X, Y, and Z are according to the convention of AVDS.

4.2.1 Piccolo II to C++. As was the case for the telemetry data used in the proximity warning algorithm, the variables listed above can be determined by simple calculations from the data packets from the Piccolo II. The variables d_x, d_y, d_z can be measured by a simple subtraction of the current ENU coordinates from the ENU coordinates of a reference point. The reference point is the corresponding ENU coordinates to a latitude, longitude, altitude chosen at a specific location to be discussed below. The Euler angles ϕ, θ, and ψ can be determined by retrieving the information from the decoded telemetry data packet. Thus, all the variables needed to display the aircraft position (relative to a basepoint) and attitude, are readily available during flight.

4.2.2 C++ to AVDS. Before initiating this research, there already existed C++ example programs that demonstrated the functions used to pass the flight data directly into the visualization program. Based on these example programs, the functions were incorporated into the Piccolo II packet decoding program and thereby relayed real-time telemetry information from the autopilot to the visualization program. Another variable to be passed from Piccolo II to AVDS was the time that passed between data packets. AVDS has a built in interpolation algorithm to smooth the visualized motion of the aircraft between time steps. The smaller the time step of the information passed to AVDS, the smoother this motion appeared. The Operator Interface for Piccolo II allows a toggle between data broadcast rates of 1 Hz and 20 Hz with the 20 Hz data broadcast rate corresponding to a smoother visualization in AVDS. Also, a model of the Sig Rascal 110 was written and included in the AVDS program folder according to the AVDS manual, in order to visualize the aircraft being tested. Comparison images between this model can be seen in Figures 20 and 21. This text file model used in AVDS is included in Appendix C.

4.3 Real-Time Synthetic Vision

Although it may be somewhat beneficial to observe in real-time the attitude of the aircraft by using AVDS, it is best to also see the motion of the aircraft in relation to the surrounding terrain. AVDS allows terrain to be mapped using polygons written in a text file format. Because the linear interpolation scheme inherently works for all points, an estimate of the Area B terrain was readily available. A script was written to create a grid of evenly spaced points and the coordinates of each point were input into the linear interpolation algorithm along with the Area B terrain data in order to create a map of polygons like that depicted in Figure 22 with each point also containing an altitude coordinate. This data was then written to a text file that could be used in AVDS to combine the terrain of Area B with the display of the aircraft flying through it. By anchoring a known point of latitude and

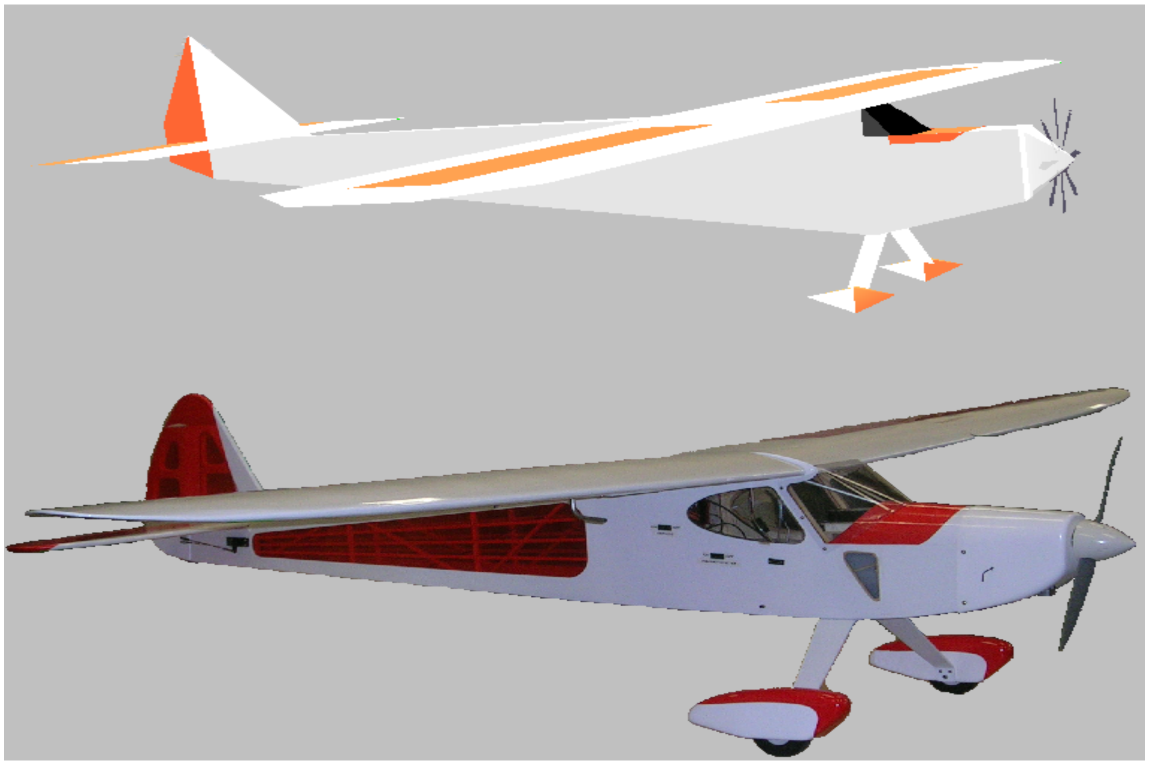


Figure 20 Comparison Between Side View of Actual and Simulated Aircraft

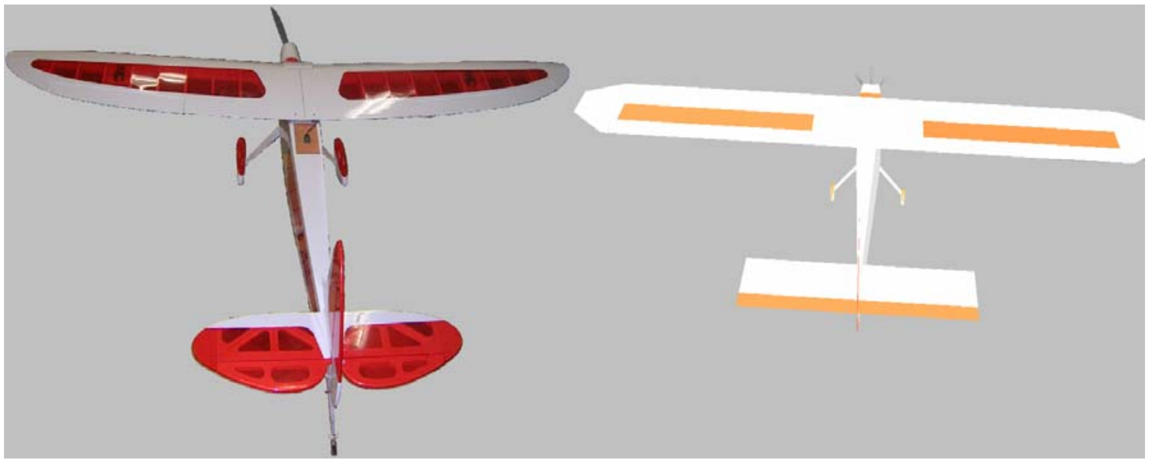


Figure 21 Comparison Between Top View of Actual and Simulated Aircraft

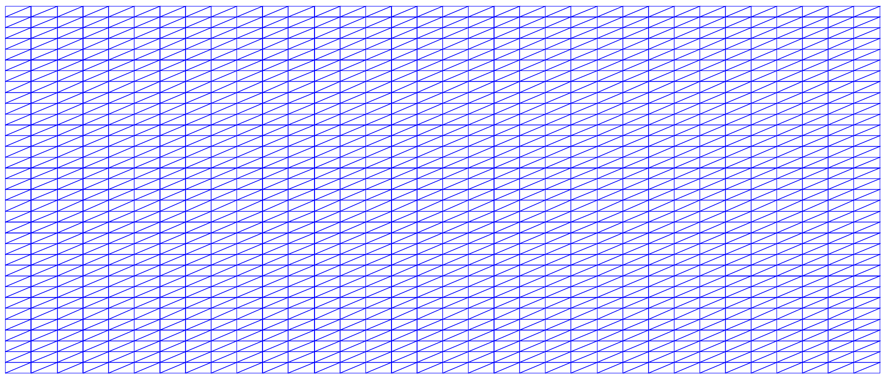


Figure 22   Example of Mapping Grid

longitude in the terrain, it was ensured that the synthetic vision truly estimated the terrain and buildings in the proper locations of Area B. Furthermore, a texture of the map of an Area B satellite image was placed on the terrain to further ensure AVDS showed where the aircraft actually was. A three-dimensional mapping of the estimated Area B terrain is shown in Figure 23. (Note: The buildings represent those in Area B; howeverm, the mountains in the background were included only to demonstrate the capability of the AVDS terrain feature.

4.4 Summary

Because the algorithms designed for maximum situational awareness made readily available the data needed to visual an aircraft relative to the surrounding terrain in a real-time manner, and because AVDS readily accepted such data when in its Network mode, it was a small step to take quantitative information and transform it into synthetic vision of the aircraft in flight.

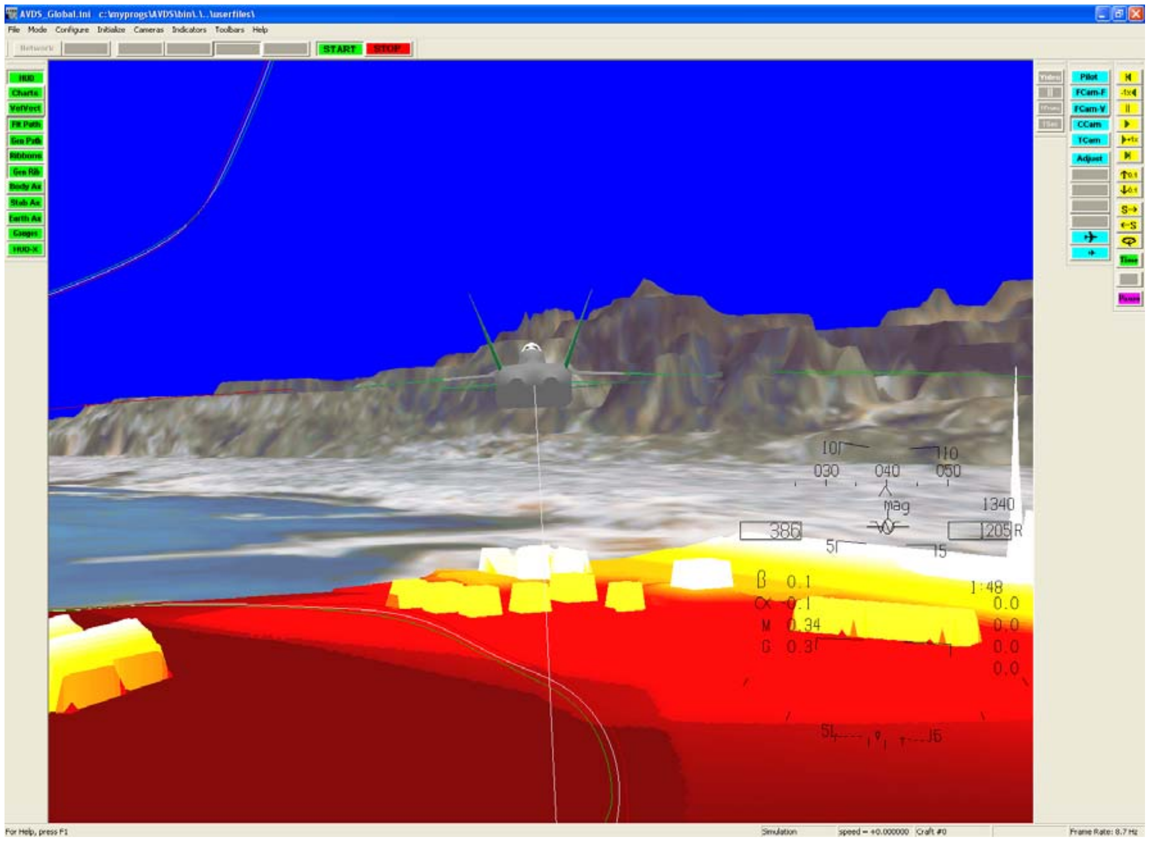


Figure 23   3-D Model of Area B

V. Simulation and Flight Testing

5.1 Overview

Due to the nature of the research, much of algorithm verification for SASVRT was done prior to running any simulations, and certainly before doing any flight testing. Many of the problems encountered and overcome were discovered early in the development phase. As the algorithms were written, many test cases were examined and led to the aforementioned exceptional circumstances, such as the rules for selecting the two closest points for the linear interpolation. Thus, when it came time to run simulations using the HITL Simulator, the results were primarily to affirm the effectiveness of situational awareness rather than to discover any circumstances in which the algorithms broke down.

5.2 SASVRT

SASVRT is the C++ code that incorporates all of the discussed algorithms and methods and it is listed in Appendix B. It is set up to run on the current Piccolo computer network in the ANT Center, while simulating or flight testing a Piccolo autopilot. In order to run SASVRT on a separate computer or network, the default IP address must be changed in the code. SASVRT allows the user several options. Figure 24 shows the initial screen of the program giving the user the options of:

- 'T' for Telemetry Data. This brings the user to a screen as shown in Figure 25. This serves best to give the user quantitative parameters to enhance situational awareness. These parameters include inside or outside of a hard boundary, distance to a hard boundary, time to a hard boundary, distance to a minimum altitude, and distance to another aircraft, as well as typical flight data such as altitude, airspeed, etc.

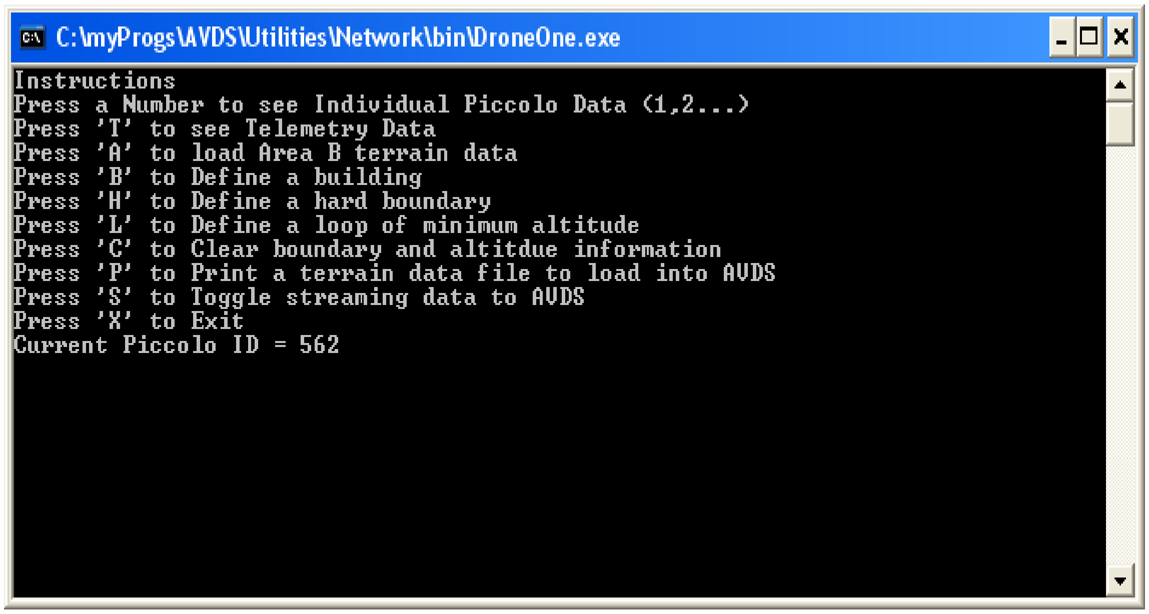

```
C:\myProgs\AVDS\Utilities\Network\bin\DroneOne.exe
Instructions
Press a Number to see Individual Piccolo Data (1,2...)
Press 'T' to see Telemetry Data
Press 'A' to load Area B terrain data
Press 'B' to Define a building
Press 'H' to Define a hard boundary
Press 'L' to Define a loop of minimum altitude
Press 'C' to Clear boundary and altitdue information
Press 'P' to Print a terrain data file to load into AVDS
Press 'S' to Toggle streaming data to AVDS
Press 'X' to Exit
Current Piccolo ID = 562
```

Figure 24 Startup Screen of the Situational Awareness Program

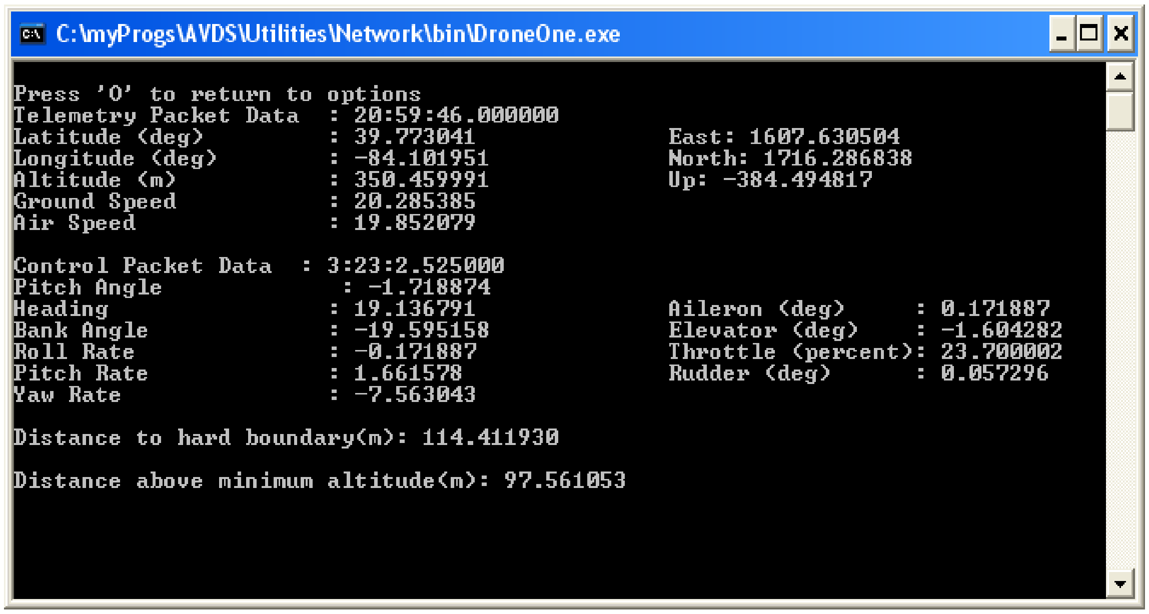

```
C:\myProgs\AVDS\Utilities\Network\bin\DroneOne.exe

Press 'O' to return to options
Telemetry Packet Data  : 20:59:46.000000
Latitude (deg)         : 39.773041            East: 1607.630504
Longitude (deg)        : -84.101951           North: 1716.286838
Altitude (m)           : 350.459991           Up: -384.494817
Ground Speed           : 20.285385
Air Speed              : 19.852079

Control Packet Data  : 3:23:2.525000
Pitch Angle             : -1.718874
Heading                : 19.136791            Aileron (deg)      : 0.171887
Bank Angle             : -19.595158           Elevator (deg)     : -1.604282
Roll Rate              : -0.171887            Throttle (percent): 23.700002
Pitch Rate             : 1.661578             Rudder (deg)       : 0.057296
Yaw Rate               : -7.563043

Distance to hard boundary(m): 114.411930

Distance above minimum altitude(m): 97.561053
```

Figure 25 Telemetry Data from the Situational Awareness Program

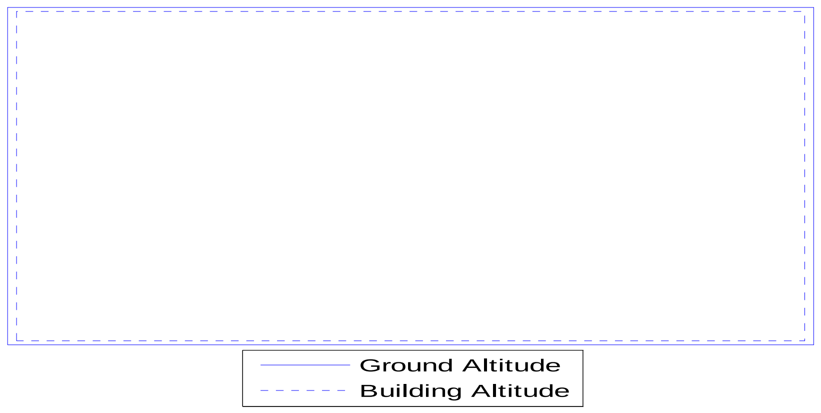


Figure 26   Example of Building Altitude Contours

- 'A' to load Area B data. To test SASVRT, the terrain of Area B was modelled using contours of constant altitude. This option loads in this data, allowing the program to display the distance to and time to the hard boundary of Area B, the distance to the estimated minimum altitude of Area B, as well as starting a log these parameters. The data loaded by this function is listed in Section A.2.
- 'B' to define a building. If the user chooses to define the environment on the fly, this option allows the user to define the location of a building by passing in the latitude and longitude of the corners of the building as well as the height. The program defines a contour of ground altitude around the building and a contour of building altitude just inside the ground altitude contour as seen in Figure 26.
- 'H' for hard boundary. This allows the user to pass in of latitude and longitude coordinates of a polygon in which the aircraft is designated to fly. SASVRT will then show pertinent data relating the aircraft and the time and distance to this defined hard boundary in the telemetry display.
- 'L' for a loop of altitude. This allows the user to define a contour of altitude to be interpolated by the minimum altitude algorithm by passing in the lati-

tude and longitude of the points making up the corners of a polygon and as well as the altitude of the contour. Based on this contour, minimum altitude information will be presented in the telemetry display.

- 'C' to clear data. If the user wishes to restart defining altitudes and hard boundaries, this function will clear the program of any existing data, and allow for new data to be incorporated.
- 'P' to print terrain file. Based on the existing hard boundary, building, and altitude contour data, this function will create a three-dimensional craft file that can be placed into AVDS in order to obtain synthetic vision of the aircraft with respect to its surrounding terrain. If the program is running the craft directory of AVDS and the craftcap.txt file has already been setup to enable this function, no further action is required. Otherwise, this terrain.txt file must be moved into the craft directory.
- 'S' to toggle streaming to AVDS. This activates a function that takes the telemetry data from the Piccolo and converts it to a six degree of freedom model that is sent into AVDS to visualize the aircraft in flight. This also toggles a log that can be used to play back a flight in AVDS for future reference.
- 'X' exits the programs and closes the telemetry logs.

5.3 Simulation

5.3.1 Hard Boundary. To test the Hard Boundary feature of SASVRT, the Area B hard boundary was coded into the program, and for a visual check, the Operator Interface map was altered to show this hard boundary as see in Figure 27.

A waypoint list was uploaded to the Piccolo as see in Figure 28 that included multiple crossings of the hard boundary to ensure that the program could detect when this occurred, as well as the distance, rate, and time until these crossings occurred.

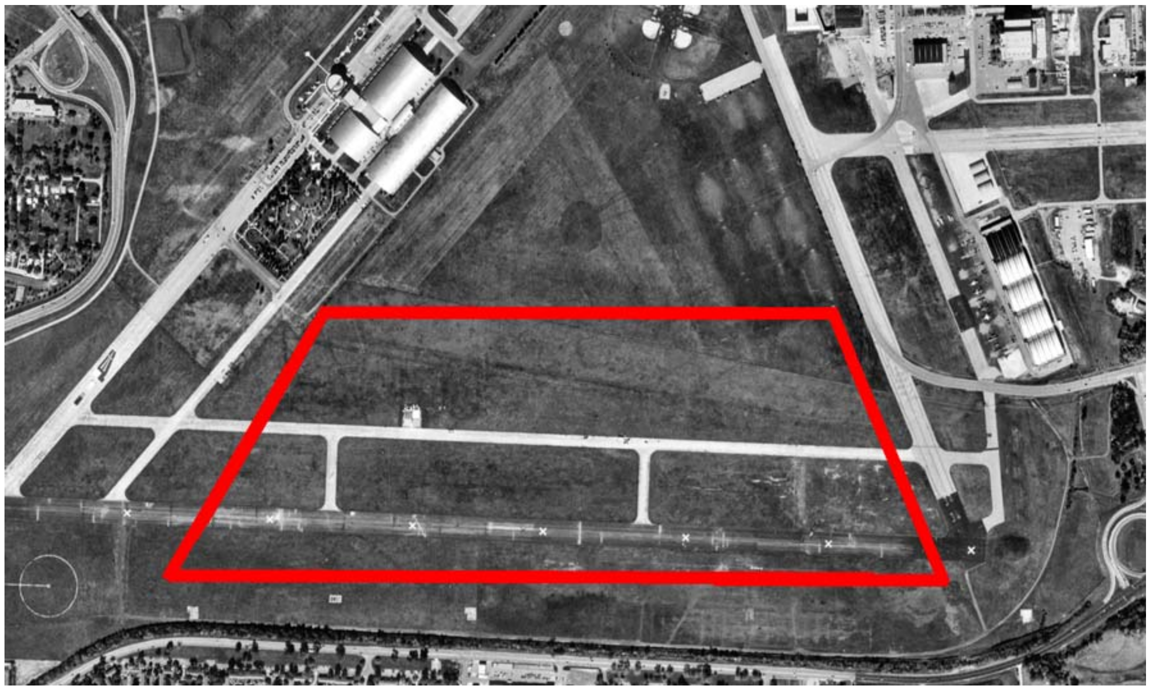
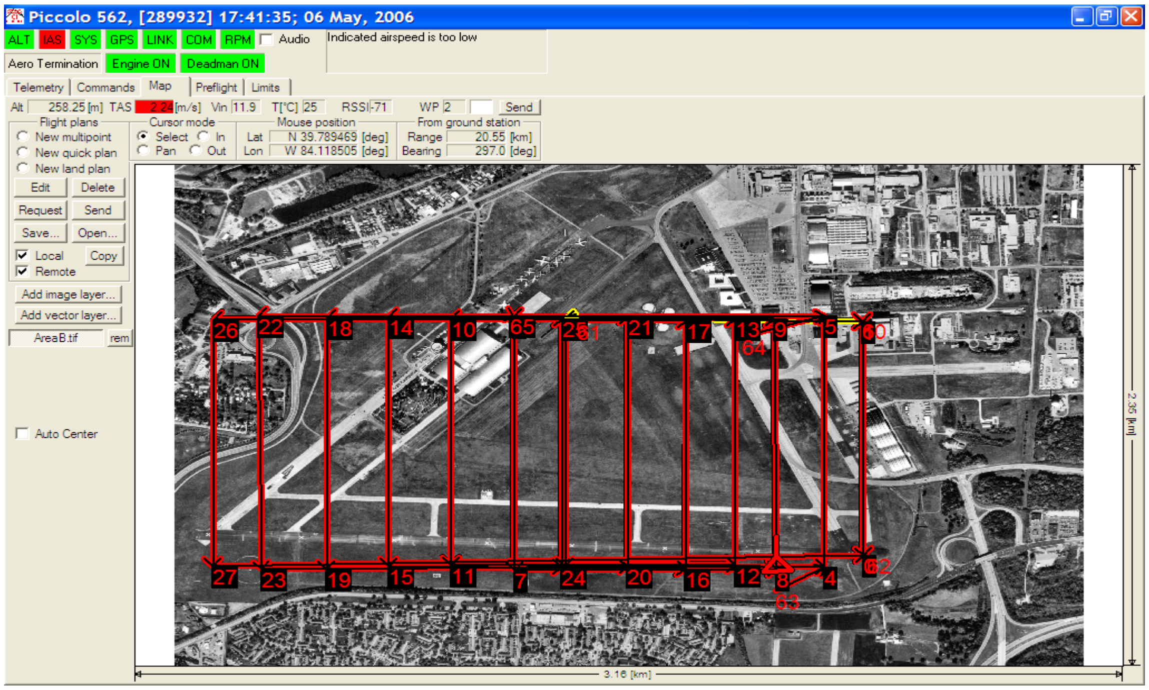


Figure 27 Hard Boundary Simulation

5.3.2 Minimum Altitude. A substantial benefit of HITL simulator over flight testing as far as monitoring minimum altitude is the ability to fly over and between buildings. In reality this would carry too much risk to property and bystanders. However, in the simulation, the Piccolo was commanded to fly over buildings as the minimum altitude was monitored and recorded. This enabled the user to confirm a rapid change in minimum altitude as the aircraft approached and passed over buildings.

5.3.3 AVDS. In addition to simply monitoring the quantified minimum altitude during the simulation, AVDS allowed the user to visually compare the aircraft's altitude with that of the simulated terrain and buildings. The simulator also allowed the user to optimize the method of using AVDS without risk before taking it to the field for actual flight testing. This included determining what information to pass from the Piccolo to the AVDS program such as position or Euler angles as


Piccolo 562, [289932] 17:41:35; 06 May, 2006

Indicated airspeed is too low

Alt 258.25 [m] TAS 2.24 [m/s] Vin 11.9 T[°C] 25 RSSI -71 WP 2

Cursor mode: [✓ Select] In Pan Out

Mouse position: Lat N 39.789469 [deg] Lon W 84.118505 [deg]

From ground station: Range 20.55 [km] Bearing 297.0 [deg]

[✓ Local] [✓ Remote] AreaB.tif

Figure 28 Hard Boundary Simulation Racetrack

well as what terrain appeared most useful in the visualization. However, the most compelling reason to simulate using AVDS was to ensure the terrain file was properly aligned in the program, and the information being communicated was being properly applied. Because AVDS uses a different coordinate system, it is easy to confuse some directions, and the simulation allowed these errors to be discovered and eliminated. For example, the hard boundary of which the situational awareness program alerts the observer is entered into the program and is independent of the visualized hard boundary in AVDS. Consequently, a valuable check of the AVDS placement accuracy is observing the quantified boundary data as the aircraft visually crosses the boundary in AVDS. Also, by applying a satellite image texture to the AVDS terrain file, the relative aircraft position can be checked against where the Operator Interface shows the aircraft to currently be. These concepts are displayed in Figure 29, where the aircraft location is indicated by the green triangle.

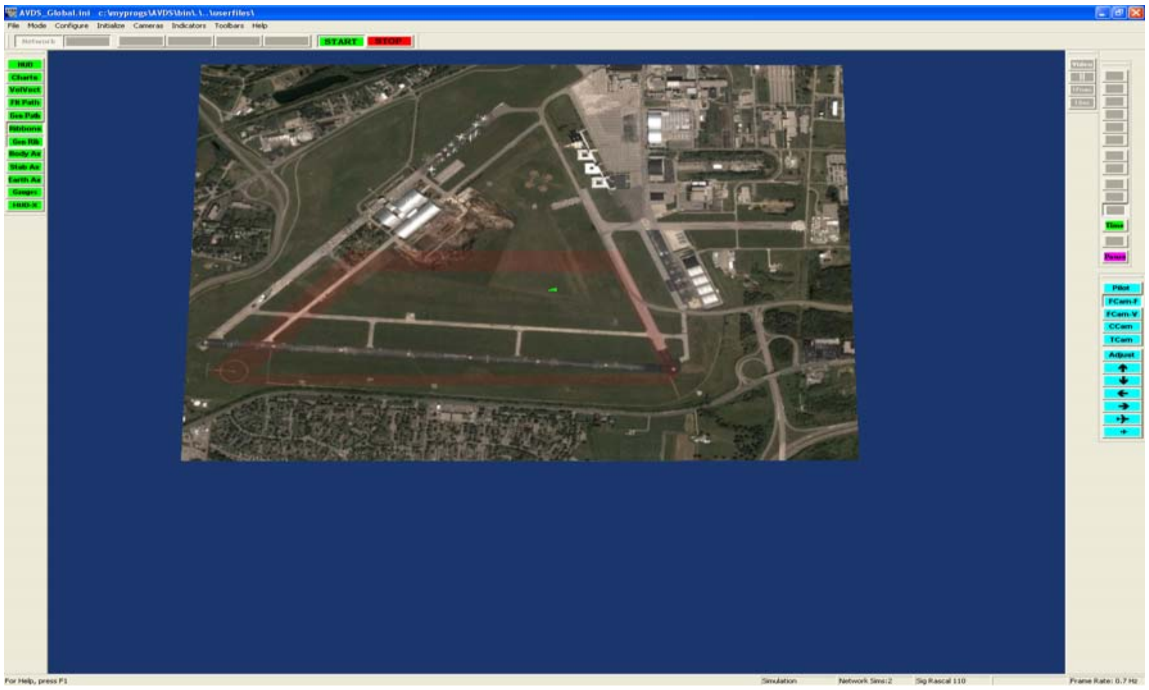


Figure 29 AVDS Screen Capture of Satellite View

5.4 Flight Testing

Flight testing could be conducted just as a simulated flight test with the exception of flying close to buildings or outside of the designated flying area. As such, the Area B model was modified to include a simulated hill as seen in Figure 30 and a much smaller designated flying area so that during a flight test, the algorithms could be examined for flaws. The ultimate goal of this research, however was to use this system as a reliable safety mechanism, and during a successful flight test in which the aircraft does not impact the terrain or cross out of the designated flying area, the results of this program should show that no alerts were triggered.

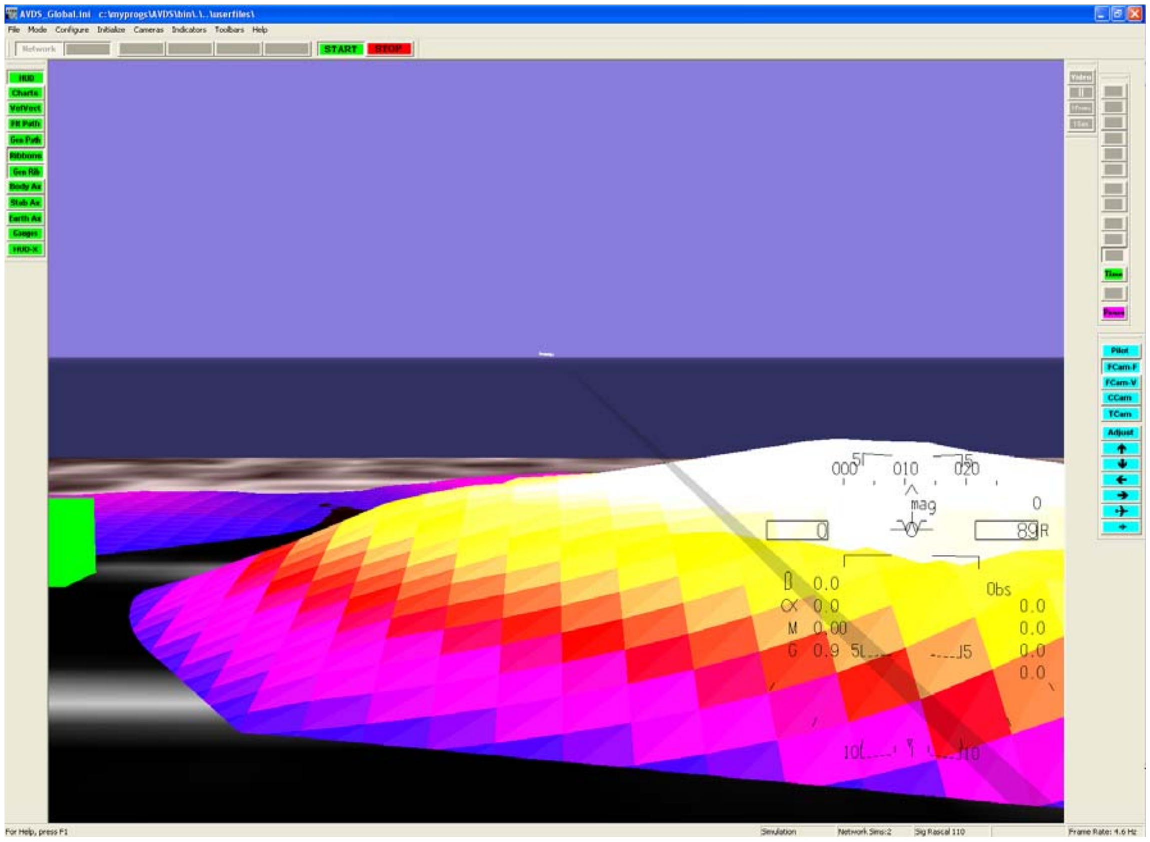

AVDS_Global.ini c:\myprogs\AVDS\bin\..\..\userfiles\
File Mode Configure Initialize Cameras Indicators Toolbars Help
START
STOP
HUD
Charts
Pilot
For Help, press F1
Frame Rate: 4.6 Hz

Figure 30 Simulated Hill in AVDS

VI. Results and Analysis

6.1 Overview

This chapter presents the results of the numerous simulations that were run to verify the algorithms developed in this research. These results are divided into information relative to hard boundaries, minimum altitude, multiple UAVs, and synthetic vision. These results are then analyzed to determine what conclusions or modifications can be made.

6.2 Simulation

6.2.1 Hard Boundaries.

6.2.1.1 Results. The data taken in the hard boundary simulation included distance, velocity, and rates to the boundary, and the instantaneous time until crossing a boundary. Data was taken with both a constant velocity vector towards the boundary and a changing velocity vector relative to the boundary. The results are plotted versus time in Figures 31 and 32 for the constant velocity vector case, and Figures 33 and 34 for the changing velocity vector case. Figure 35 shows an overhead view of the simulated flight path and indicates whether or not the aircraft was within the designated hard boundary.

6.2.1.2 Analysis. Safety is paramount in flight testing UAVs. As such, it is extremely important to the safety and security that the proximity of an aircraft relative to its surroundings and other aircraft is well known. Figure 35 clearly shows that the boundary warning system accurately determines at any given time whether or not the aircraft is within the simulated boundary. As important as this is however, the best information to have to ensure safety during flight testing is not simply whether or not the aircraft is within the boundary, but if the aircraft is on

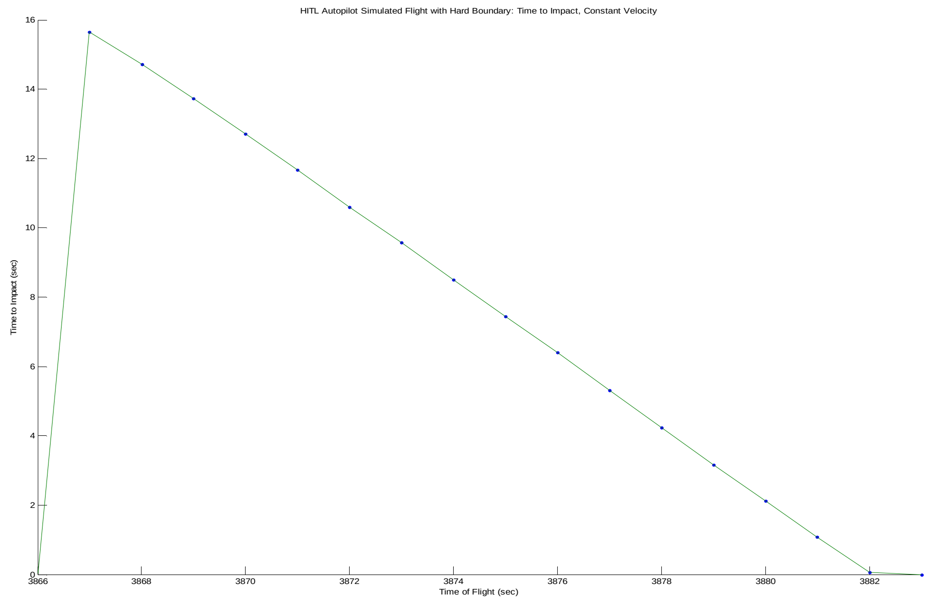


Figure 31 Time to Impact with Hard Boundary: Constant Velocity

track to violate the boundary, so as to be able to recover the aircraft before such violation occurs.

Figures 31 and 33 show clearly that the warning system is capable of displaying the dead reckoned time to crossing the boundary at a given instant in time. As expected for a constant velocity vector, Figure 31 shows a linear plot decreasing at a rate of $1\frac{sec}{sec}$, as the aircraft will cross out of the designated flying area at the time the line crosses the x-axis. Without changing the velocity vector, the dead reckoned time to crossing the boundary should be correct at every data point, and it is true, as the line initially indicates that the aircraft will cross the boundary in approximately 15 seconds, and 15 seconds later on the time of flight axis, the data indicates the aircraft crossed the boundary. Figure 33 is not quite as smooth as Figure 31 as at any instant the velocity vector is changing, and a new dead reckoned time to cross boundary is calculated. However, this type of data more accurately reflects what

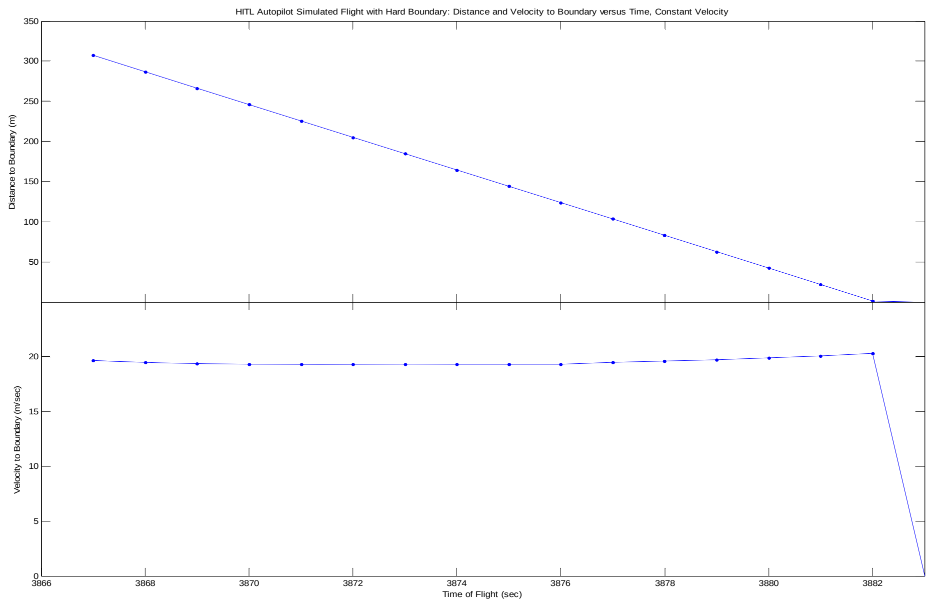


Figure 32 Distance and Velocity to Hard Boundary: Constant Velocity

might occur when using this program to monitor an actual flight test. The key results to take away from this plot is that the time to boundary is calculated accurately and relayed to the observer. This is vital information as there undoubtedly is a time when the pilot would want to take over control of the aircraft from the autopilot without crossing the hard boundary, and this information allows the pilot to do exactly that.

Figures 32 and 34 demonstrate most importantly that the information used to calculate time to boundary is reliable. Time to boundary is calculated by the equation,

$$t_B = d_B/v_B, \tag{19}$$

where t_B is the time to boundary, sec, d_B is the distance to boundary, calculated by aforementioned geometric methods, and v_B is the velocity towards boundary, $\frac{m}{sec}$,

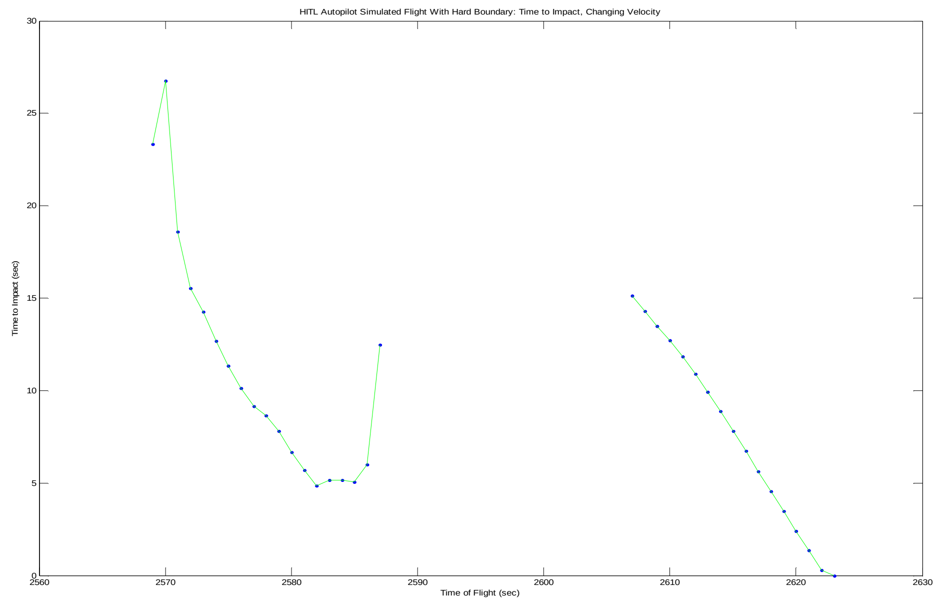


Figure 33 Time to Impact with Hard Boundary: Changing Velocity

calculated based on the aircrafts velocity and attitude. With a constant velocity vector as in Figure 32 the distance to the boundary d_B is constantly decreasing, thus its derivative, or negative derivative as was convention is this simulation, v_B should be constant, and this is clearly shown to be the case. This relationship also holds true for a given instant in Figure 34, where the cusps in the distance to boundary plot correspond to discontinuities in the velocity to boundary plot, and the smooth portions of the distance to boundary plot have corresponding derivative plots in the velocity to boundary plot.

6.2.2 Minimum Altitude.

6.2.2.1 Results.

In order to test the altitude proximity warning algorithm, tests points were run for a given range of x and y values, and the h_{est} was calculated at each point for several different sets of polygons of different height. The

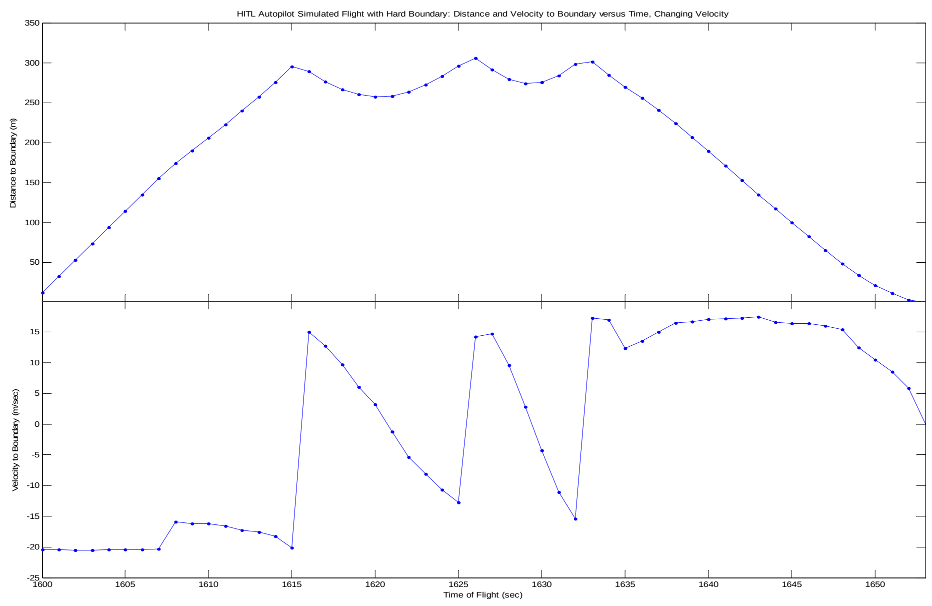


Figure 34 Distance and Velocity to Hard Boundary: Changing Velocity

data was then plotted and compared to what the plot should intuitively look like. Early on inappropriate spikes, or ground height values above the maximum given height quickly showed errors in the technique, but soon thereafter, the code was adjusted and the plots looked certainly reasonable for an interpolation of terrain.

Figure 36 shows the flight path of the simulated aircraft, which was flown over a simulated hill centered at a known point in the Area B flying area. This was done so that simulated results could later be compared with actual flight data without putting the aircraft in danger of hitting any terrain. As such Figures 36 and 37 show the three-dimensional and overhead flight paths along with the minimum altitude interpolated at any given point in the flight path. The red x's in the flight path indicates when the aircraft is below the minimum altitude. Figure 38 shows the same minimum interpolated altitude and flight path data but plotted versus time.

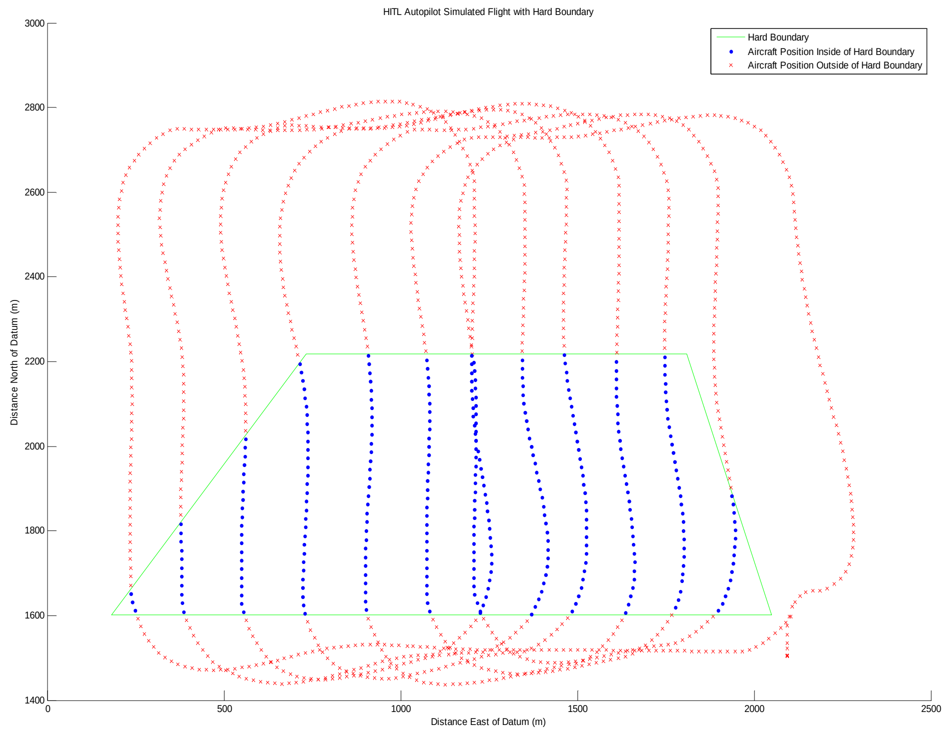


Figure 35 Overhead View of Simulated Flight Path with Respect to Hard Boundary

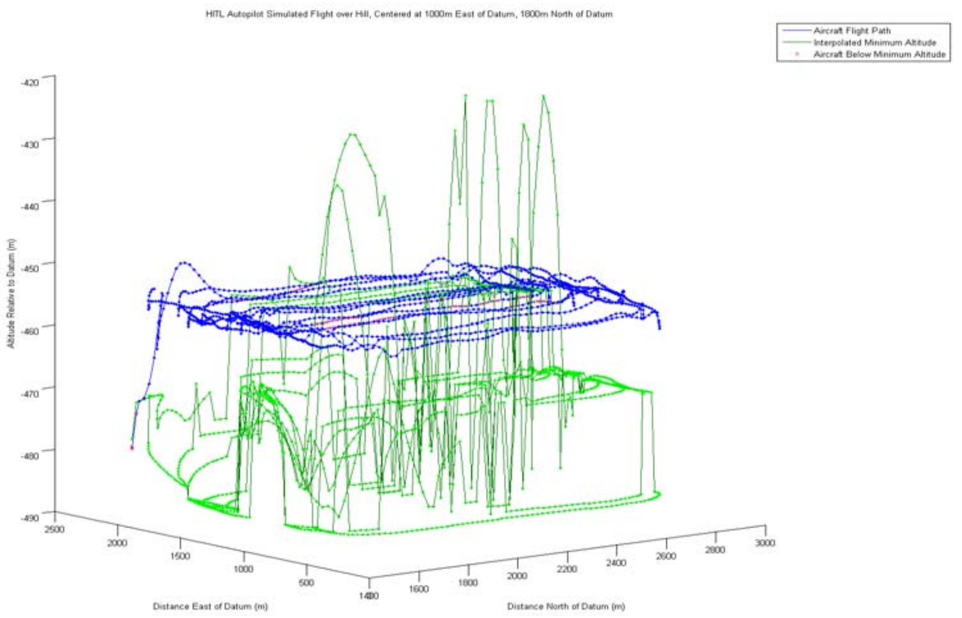


Figure 36 Simulated Flight Path with Respect to Minimum Altitude

6.2.2.2 Analysis. In the figures, any time the calculated minimum terrain is above the aircraft's altitude, the program correctly indicates this. Also Figure 36 shows that the simulated hill is correctly interpreted in the minimum altitude algorithm as it appears as the aircraft flies over it. The true vote of confidence of the minimum altitude calculation however, will be demonstrated in the AVDS craft model, as it shows the entire area of interpolated altitudes.

Ideally, a time to impact with terrain would be the information passed to the pilot in order to judge when to take over for a failing autopilot. However, due to the vertical velocity constantly changing sign, and changing only slightly in magnitude, it is not as easy to obtain velocity towards terrain as it was to obtain velocity towards the hard boundary. Yet because it is much easier for an observer to judge the aircrafts altitude, this becomes less vital, and simply knowing the distance above terrain suffices for flight testing within visual range.

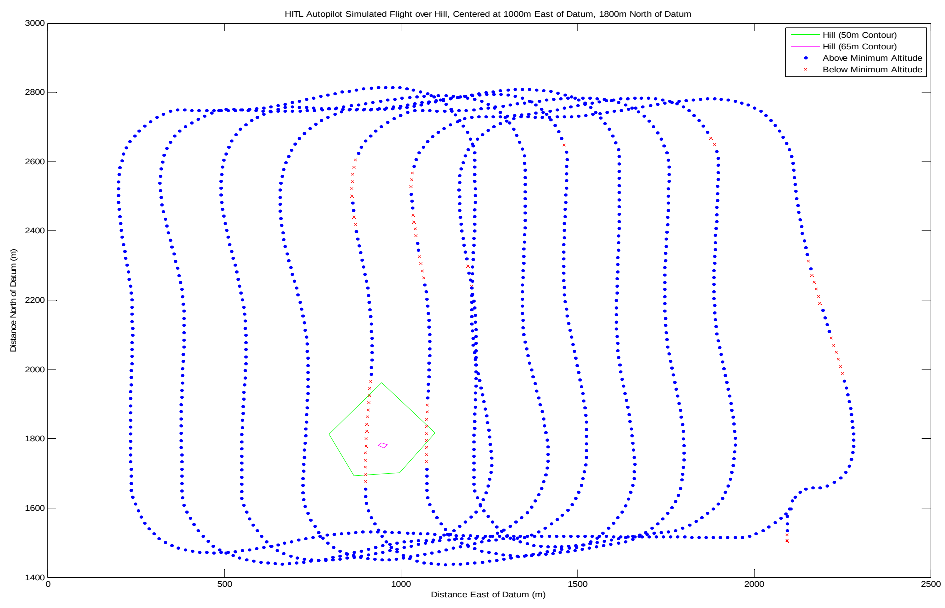


Figure 37 Overhead View of Simulated Flight Path

Finally the estimated ground height is going to have some inherent error as it is an interpolation in most cases. However, the more information input into the algorithm the more correct the approximations become. Also the urban canyon check provides a conservative approach to flight testing in simulated urban environments, which appears to be a big part of the future of UAVs, as urban environments tend to be overtly dangerous for large piloted vehicles.

6.2.3 Multiple UAVs.

6.2.3.1 Results. SASVRT was written to handle multiple UAVs flying simultaneously. As such an utterly important albeit easy calculation was distance between aircraft. A three-dimensional and overhead plot of the simulated flight paths of two UAVs are shown in Figure 39 and 40.

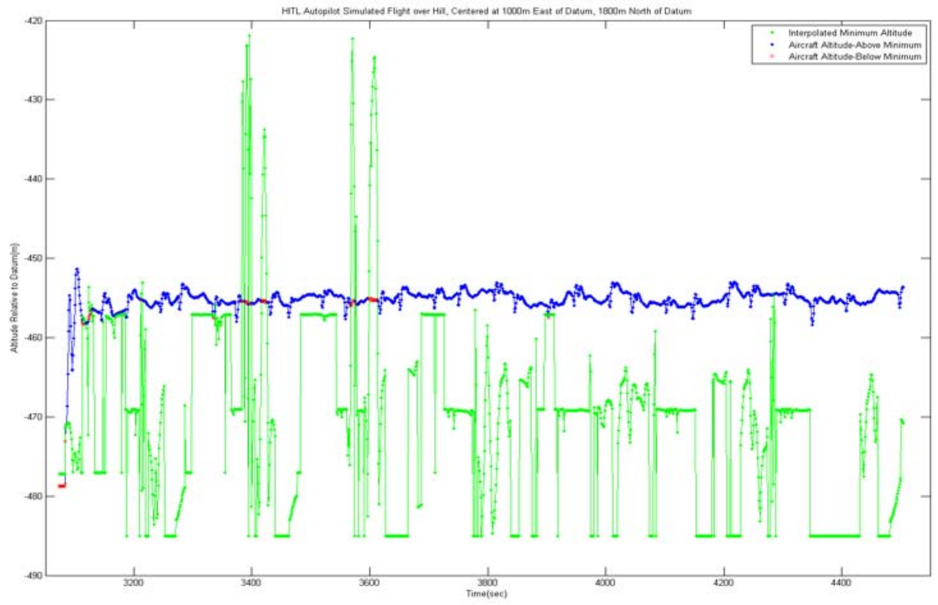


Figure 38 Minimum Altitude versus Time of Flight

6.2.3.2 Analysis. Distance between multiple UAVs is an extremely difficult parameter to gauge from afar. Consequently, to maximize situational awareness, this parameter needs to be quantified. Flight testing a single UAV is dangerous enough, in order to put two UAVs in the air, one must be utterly certain the UAVs will not collide. The simulated data in Figures 39 and 40 show that at the very least one can know the distance between the aircraft, and have the ability to knock off the flight test if the distance between the aircraft becomes too small.

6.2.4 AVDS.

6.2.4.1 Results. AVDS was used to provide synthetic vision to the observer. The aircraft and terrain were loaded into AVDS and showed up as expected during flight testing. Figure 41 shows a screen capture taken during the simulation from a point of view off the starboard wing of the aircraft, with the simulated

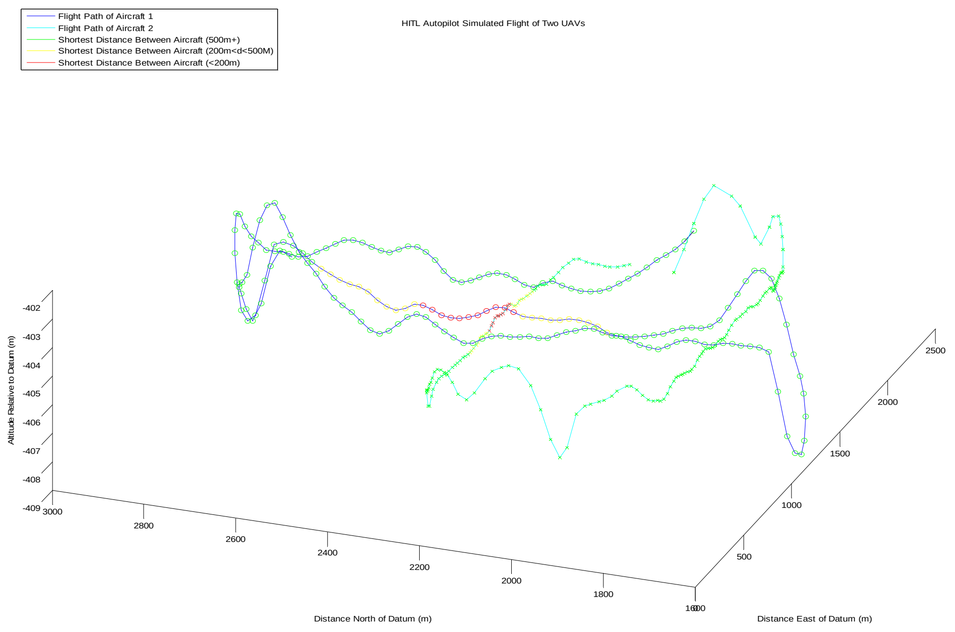


Figure 39 Simulated Flight Path of Multiple UAVs

buildings of the Air Force Museum in the background. The aircraft depicted is a rough model of the Sig Rascal 110 made to scale with the simulated surroundings of Area B. An overhead view of multiple UAVs flying is shown in Figure 42 which can be used to gauge distance between aircraft. The terrain is a result of a grid of north and east test points input into the minimum altitude algorithm to result in a three dimensional map of triangles. The coarseness of this grid can be adjusted to give a more or less accurate representation of the terrain, with the cost of a higher resolution being a larger data file and a slower frame rate in AVDS. The colors of the polygons in AVDS were experimented with to determine which color scheme most appropriately visualized the terrain. A color scheme loosely correlating the electromagnetic visual spectrum with the altitude of the polygons is shown in Figure 43. Hard coded waypoints were also placed in the synthetic terrain in order to better visualize them as shown in Figure 44. Also shown in this screen capture is a red sensor cone out the nose of the aircraft. Figure 45 shows the capability of the program to place multiple

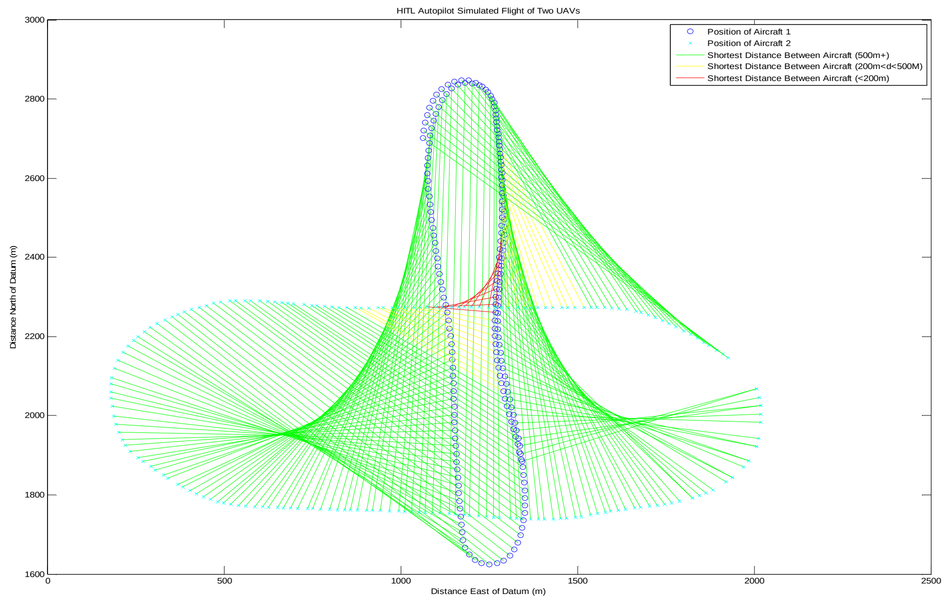


Figure 40 Overhead View of Simulated Flight Path of Multiple UAVs

aircraft in the simulated terrain, boosting the situational awareness. In addition to terrain texture, the screen captures also depict a hard boundary visualized by a set of four transparent red walls. When the aircraft approaches these walls, SASVRT can be monitored simultaneously to AVDS to ensure that when the program shows zero distance to boundary that AVDS shows the aircraft crossing the visual boundary. Similarly, using the Operator Interface as a control, visualizing the aircraft over known points in the satellite image as mentioned in discussing Figure 29, can be compared to the satellite image in the Operator Interface. The three-dimensional buildings in AVDS should line up with their footprints on the satellite image.

6.2.4.2 Analysis. The AVDS visualization worked as expected and provided a much better perspective in terms of relative distances to terrain than the observer had without AVDS. The ease of incorporating terrain and aircraft movement relative to terrain showed that it is an extremely useful and powerful tool during

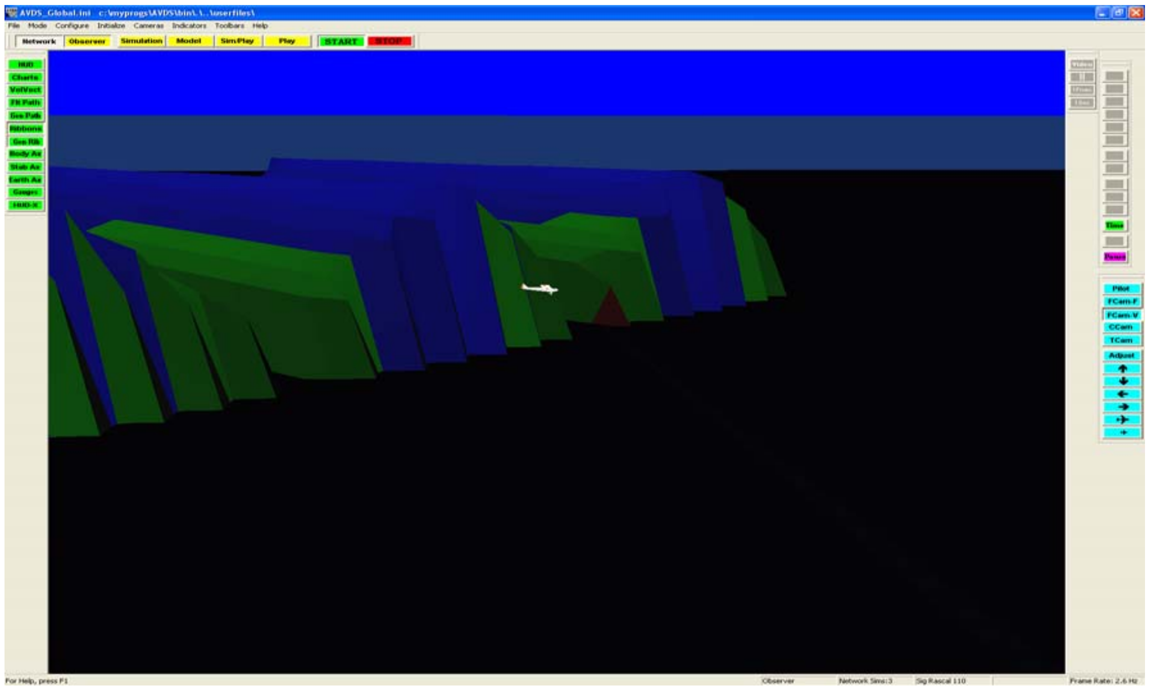

AVDS_Global.ini c:\myprogs\AVDS\bin\..\userfiles\
File Mode Configure Initialize Cameras Indicators Toolbars Help
Network Observer Simulation Model SimPlay Play START STOP
For Help, press F1
Observer
Network Sims:3
Sig Rascal 110
Frame Rate: 2.6 Hz

Figure 41 Simulated Aircraft with Air Force Museum

flight testing. In addition, it demonstrated that it can be relied upon to assist in navigation at night and in poor visibility.

The satellite image was quite useful in simulations, particularly on ensuring the aircraft was in the correct location with respect to the model as it was with respect to either the simulated or real environment. From an overhead view as seen in Figure 29, the AVDS simulation could be compared with that of the overhead view provided by the Operator Interface. Using the Operator Interface as a true location, the model in AVDS could be adjusted to ensure proper orientation and location within the synthetic environment.

Furthermore, from an operational standpoint, a satellite overview of an area can be an excellent tool while monitoring a UAV. With simply a camera on board, the user can see only what the UAV is flying over, whereas with synthetic vision

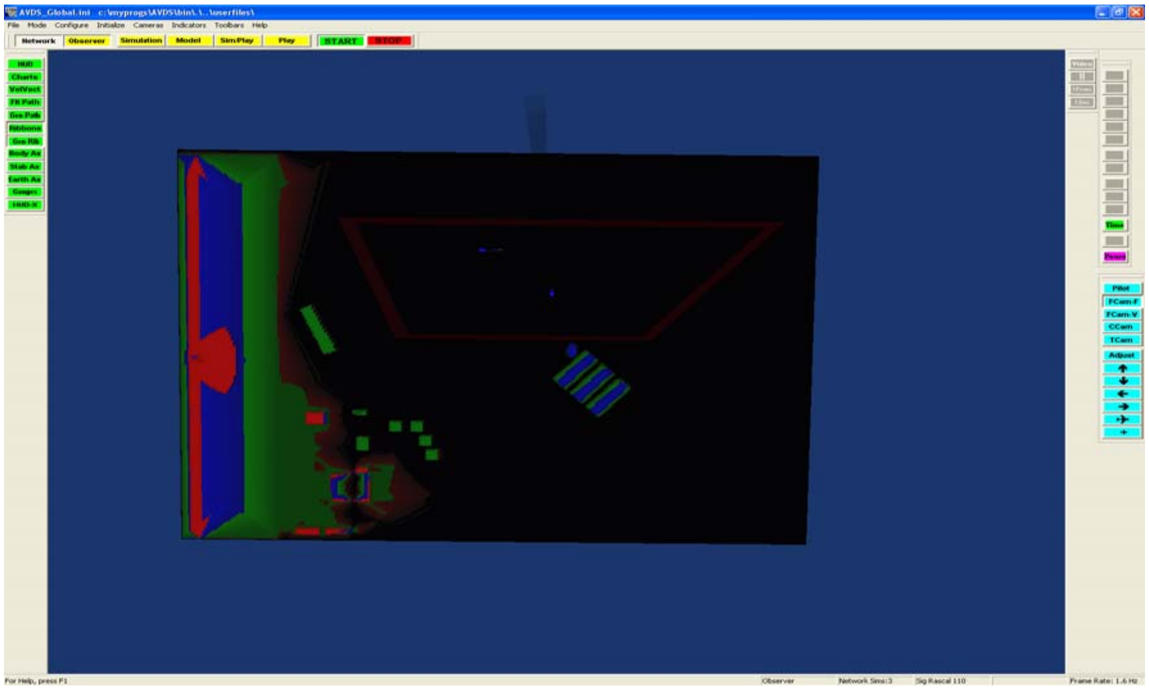


Figure 42 Simulated Multiple UAVs from Above

from above, the user can compare what a camera might pick up to what a satellite has previously captured on a larger scale.

The aforementioned researched concurrently performed by Robinson sought to keep a sensor fixed on a waypoint amidst wind which would cause the aircraft to crab (Robinson, 2006). Placing the waypoints in AVDS along with a sensor cone out the nose of the aircraft allowed Robinson to visualize the results of his research by checking if the sensor cone remained on the synthetic waypoints amidst simulated wind.

Perhaps the most powerful application of the AVDS was the incorporation of multiple UAVs. As it stood before, the operator interface would only show the position of a single aircraft per window, and gauging the distance between aircraft became a guessing game. However, with AVDS, particularly from an overhead view, the relative location of each aircraft was clearly shown. This also had benefits in

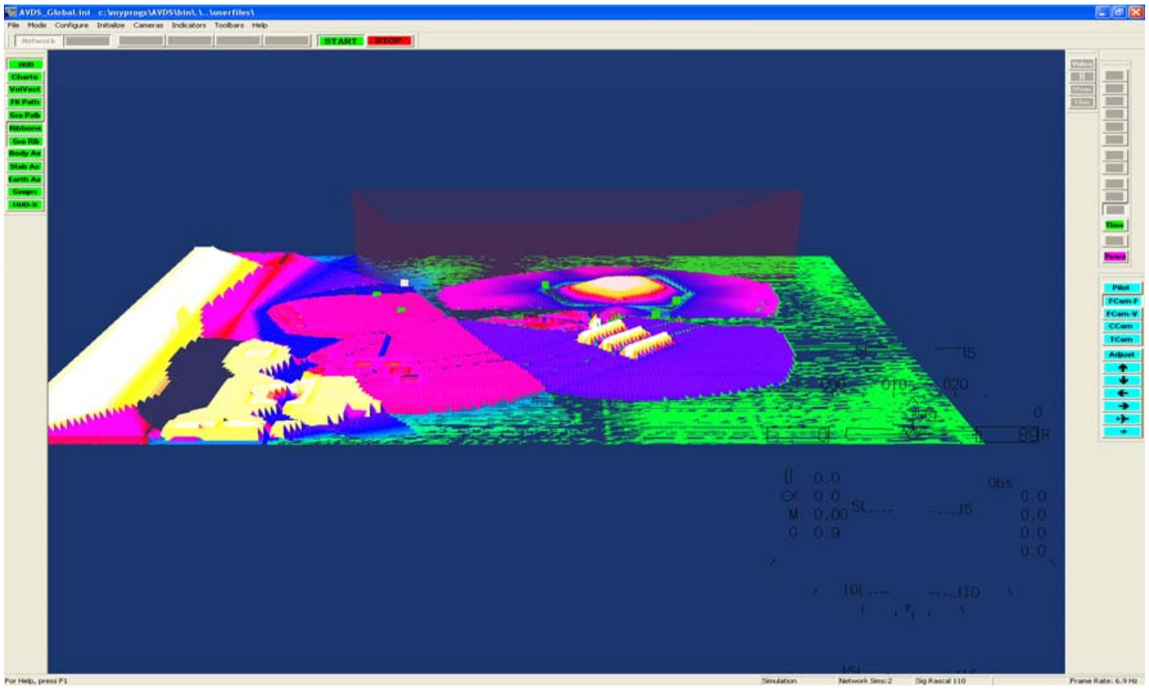


Figure 43 Simulated Alternate Color Scheme

the research of McCarthy, who sought to fly a UAV based on a relative position to a lead aircraft (McCarthy, 2006). As it is quite dangerous to actually flight test two aircraft simultaneously, especially basing ones position of the other, AVDS allowed one aircraft to be simulated and another be flight tested, while placing both aircraft in the same synthetic terrain.

Beyond the Area B data loaded into AVDS for the simulations, SASVRT allows for an option of entering topographic data on any location, and will at the command of the user print a three-dimensional terrain file to be loaded into AVDS. This allows for synthetic vision to take place at any location so long as the terrain data is known.

6.3 Flight Test

Due to factors outside of the author's control, actual flight tests planned to demonstrate the SASVRT were delayed and scheduled to occur post thesis defense.

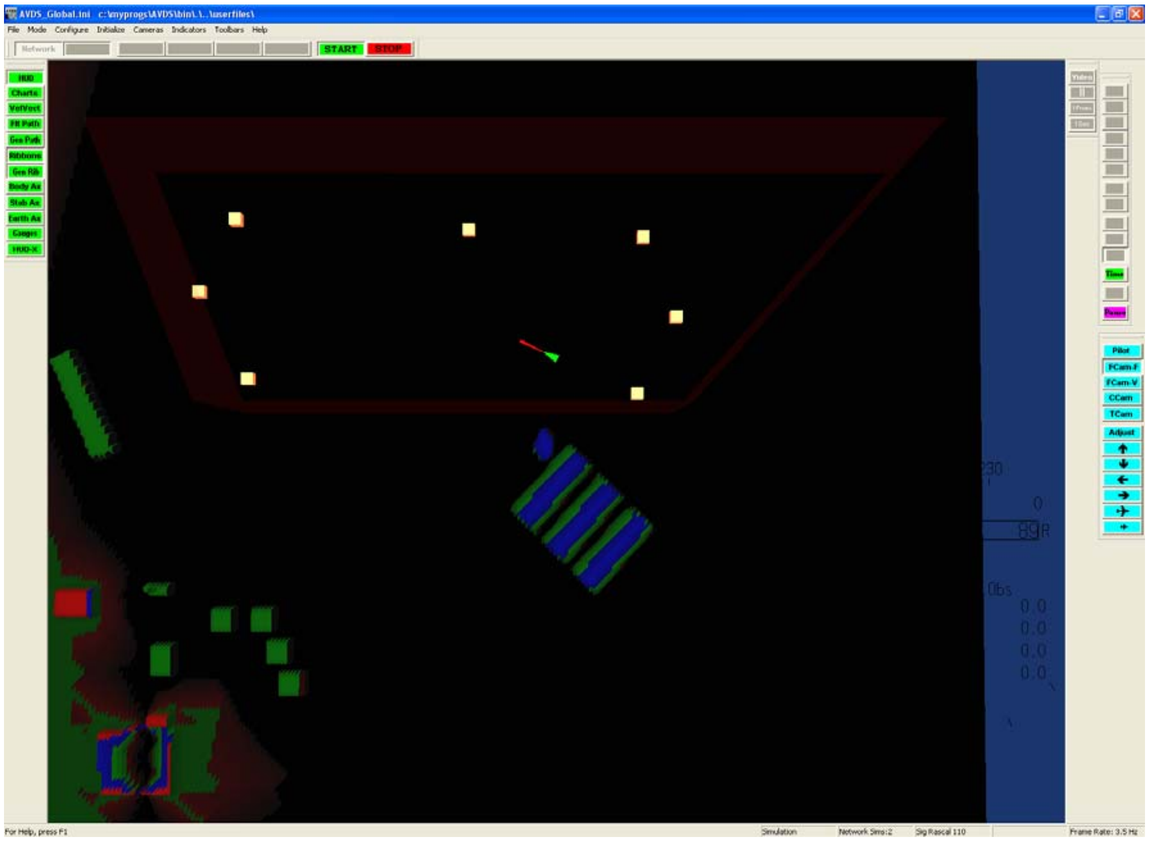

AVDS_Global.ini c:\myprogs\AVDS\bin\.\..\userfiles\
File Mode Configure Initialize Cameras Indicators Toolbars Help
START
STOP
For Help, press F1
Simulation
Network Sims:2
Frame Rate: 3.5 Hz

Figure 44 Simulated Waypoints

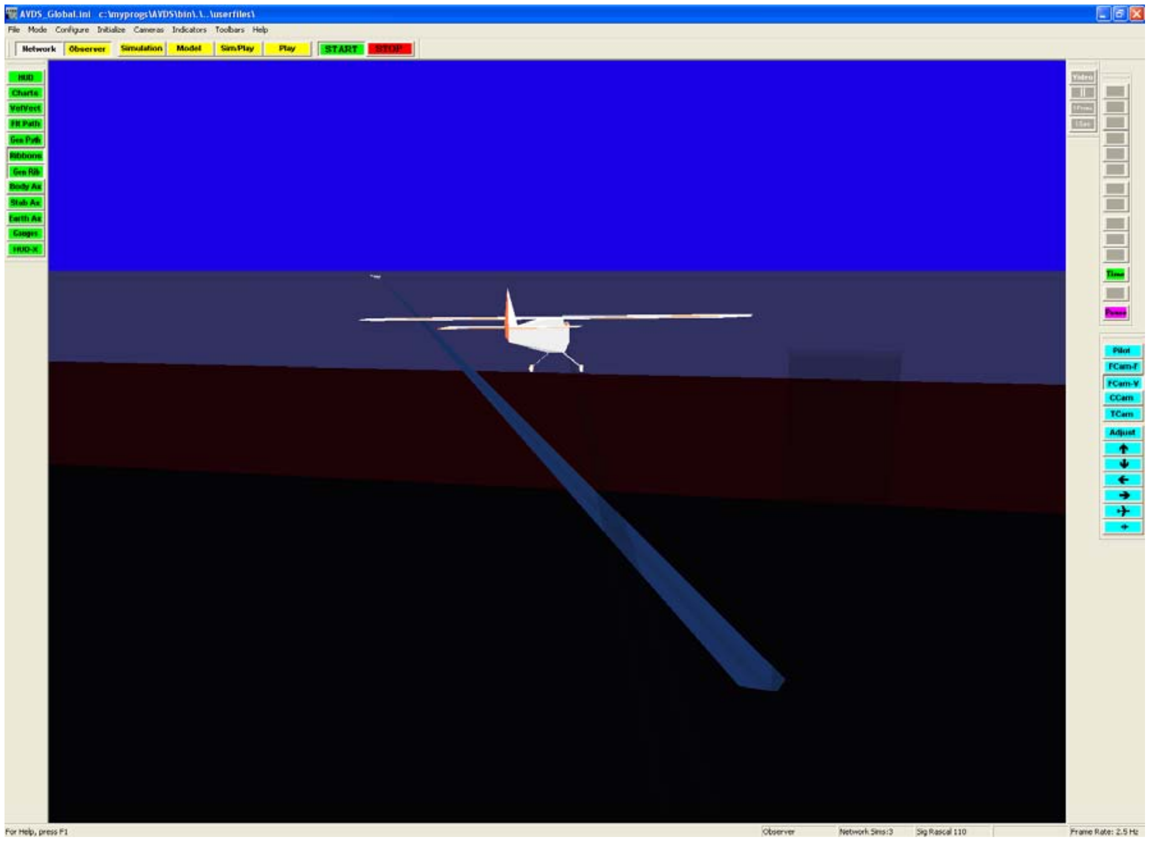


Figure 45 Simulated AVDS Screen Capture of Multple UAVs

However, due to the passive nature of the software, there would be no reason other than the potential loss of signal from the aircraft to expect any different results between simulation and flight testing. In both cases the aircraft is sending telemetry wirelessly data to the software, which then uses it with a synthetic model of the environment in order to render the most pertinent data to the user.

VII. Conclusions and Recommendations

7.1 Overview

This chapter draws conclusions based on the results and analysis of this research. Based on these conclusions, several recommendations are made for future work in the area.

7.2 Conclusions

The research has accomplished several key objectives:

- Provides the user with the ability to define a hard boundary that encompasses a designated flying area.
- Provides a real-time proximity warning system that givees the user continuous feedback regarding the position of the aircraft relative to boundaries.
- Relays to the user the distance and dead reckoned time until the aircraft travels outside of the designated flying area.
- Allows the user to input contours of ground altitude for a designated flying area.
- Uses the altitude information provided by the user to interpolate a safe flying minimum altitude at any point within the flying area.
- Relays to the user the distance above the safe flying altitude the aircraft currently is.
- Uses the altitude information to construct a three-dimensional terrain file.
- Places the aircraft within the three-dimensional terrain to give the user synthetic vision.

With this type of real-time situational awareness and synthetic vision, the user is able to better know the current circumstances of the aircraft, it also enables a pilot to have a better idea of when it becomes necessary to relieve a failing autopilot. The software is not quite as user friendly as it could be, as some parameters still needed to be hard coded and compiled in order to adjust them, and this is a problem that could have been solved giving more time. Detailed recommendations follow in the next section.

7.3 Recommendations

The following recommendations are made for future work in this area:

- Implement a way of retrieving the coordinates of all waypoints in a Piccolo sent list, such that the waypoint list sending can be used as a medium for defining contours of altitude, hard boundaries, and known buildings which could then be read into the synthetic environment.
- Attempt to model a different flight area, as Area B was the only real site modelled. This involves defining the altitude contours and adjusting the place craft methods in order to have the aircraft place with respect the terrain as it is in reality.
- Develop a way to use fewer polygons to define large flat areas, in order to minimize the craft file size and consequently improve frame rate in AVDS.
- Rather than use one large texture to visualize the satellite image in AVDS, use several small texture images to optimize frame rate in AVDS.

Appendix A. MATLAB® *Program Code*

The following MATLAB® script files was used to find the minimum altitude based on the terrain and building of Area B for safe flight and write a text file to be loaded into AVDS. The program sequence is as follows:

1. Create a text file and write in necessary craft information to be read by AVDS
2. Load the terrain data of Area B as an array of x and y coordinates where each loop corresponds to a loop of constant altitude
3. Determine the resolution of the model based on dx and dy
4. For each test point (xp, yp)
   (a) Determine the 2-D coordinates of two right triangles forming a rectangle measuring dx by dy, the upper corner of which is (xp, yp)
   (b) Interpolate the minimum altitude of the coordinates of the triangles based on provided terrain information
   (c) Write to a text file the 3-D coordinates of the right triangles in a format readable by the AVDS program
   (d) Loop through all x values in a row by increments of dx then increment by dy and repeat
5. Apply a texture to the craft file in the image of a satellite photograph of Area B

These same files were used to compare real time location data of the aircraft with the interpolated minimum altitude data for the Area B model.

A.1 Altitude_Data_to_AVDS_Polygon.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%The following was written by ENS Joe Dugan under the  %
```

```
%  advisory of Major Paul Blue for his thesis work on  %
%                   AFIT/GAE/ENY/05-J08.               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


close all
clear all
clc
%Opens a .txt file to be written in a format that AVDS can read as a craft
%file
fid = fopen('areaB.txt', 'wt');
%Informs AVDS of characteristics of the craft file areaB.txt
fprintf(fid,'areaB|Area B:\\\n\n# exhaust position\n    :ep# 5100.00  0.00...
  0.10:\\\n#ID number\n:ID#23 45 21 22 45 77 222 33:\\\n#\n#\n#\n#...
 Distance > 0 ft\n  :d0:\\\n#\n#double sided\n  :DS#1:\\\n\n    :ca=fuselage...
:\\\n   :co# 0.620  0.659  0.824:\\\n# center of mass\n :cm# 5100.00...
   3700.0   10.00:\\\n# starboard wing tip\n    :Sw# 10200.00  3700.00...
  0.39:\\\n#  port wing tip\n :Pw# 0.00  3700.00  0.39:\\\n\n');
%loads topographic data of Area B in terms of x,y coordinates in feet along
%with total number of loops n, number of points in each loop m, and...
 height of each loop h
[areaBx areaBy n m h]=areaB;
%dx,dy determine the resolution of the mapping of Area B, smaller values
%equate to higher resolution, but larger .txt files, which may bog down AVDS
dx=30;
dy=30;
for yp=0:dy:7684.34 %in feet this loop breaks surface into 7684.3/dy columns
    for xp=0:dx:10171.8 %each row is broken into 10171.8/dx pairs of right...
 triangles
        x(1)=xp; %these are the x and y coordinates of right triangles
        x(2)=xp+dx;
        x(3)=xp;
        y(1)=yp;
        y(2)=yp;
        y(3)=yp-dy;
        %based on relative position to the terrain of Area B,
        %3DLinearInterpolate determines the ground height at the given x and...
 y values
        z(1)=3DLinearInterpolate(areaBx,areaBy,n,m,h,x(1),y(1));
        z(2)=3DLinearInterpolate(areaBx,areaBy,n,m,h,x(2),y(2));
        z(3)=3DLinearInterpolate(areaBx,areaBy,n,m,h,x(3),y(3));
        if (max(z)/350)<(1/3) %these statements change the color of the polygon...
 based on its elevation
            co1=3*(max(z)/350);
            co2=0;
            co3=0;
        elseif (max(z)/350)<=(2/3) && (max(z)/350)>(1/3)
            co1=1;
            co2=3*((max(z)/350)-1/3);
            co3=0;
        else
            co1=1;
            co2=1;
            co3=3*((max(z)/350)-(2/3));
        end
        %here the polygon coordinates are written to the .txt file along
        %with information about its color
        fprintf(fid,'\n :co# %6.2f  %6.2f  %6.2f:\\\n   :pt#  %6.2f  %6.2f  %6.2f :pt#...
  %6.2f  %6.2f  %6.2f:\\ \n   :pt#  %6.2f  %6.2f  %6.2f :cl:\\ ',co1,co2,co3,x(1),...
y(1),z(1),x(2),y(2),z(2),x(3),y(3),z(3));
        %the above is repeated for the complimentary right triangle in the
        %current row and column
        x(1)=xp;
        x(2)=xp+dx;
        x(3)=xp+dx;
        y(1)=yp-dy;
        y(2)=yp;
        y(3)=yp-dy;
        z(1)=3DLinearInterpolate(areaBx,areaBy,n,m,h,x(1),y(1));
        z(2)=3DLinearInterpolate(areaBx,areaBy,n,m,h,x(2),y(2));
        z(3)=3DLinearInterpolate(areaBx,areaBy,n,m,h,x(3),y(3));
        if (max(z)/350)<(1/3)
            co1=3*(max(z)/350);
            co2=0;
            co3=0;
        elseif (max(z)/350)<=(2/3) && (max(z)/350)>(1/3)
            co1=1;
            co2=3*((max(z)/350)-1/3);
            co3=0;
        else
            co1=1;
            co2=1;
            co3=3*((max(z)/350)-(2/3));
        end
        fprintf(fid,'\n :co# %6.2f  %6.2f  %6.2f:\\\n   :pt#  %6.2f  %6.2f  %6.2f :pt#...
  %6.2f  %6.2f  %6.2f:\\ \n   :pt#  %6.2f  %6.2f  %6.2f :cl:\\ ',co1,co2,co3,x(1),...
y(1),z(1),x(2),y(2),z(2),x(3),y(3),z(3));
    end
end
%the below line maps a satellite picture of Area B onto the surface for
%enhanced awareness of the aircrafts relative position
%fprintf(fid,'\n#select texture file\n:XI#0:\\\n#texture coordinates\n :XC#...
1.0 0.0 0.0 :XC#1.0 1.0 0.0 :XC#0.0 1.0 0.0 :XC#0.0 0.0 0.0:\\\n# (left)\n...
   :pt# 2976.00  0.00  0.00:pt# 2976.00  2246.00 10.00:\\\n    :pt# 0.00...
   2246.00 10.00:pt# 0.00   0.00  -10.00:cl:');
fclose(fid);%closes the .txt file
```

A.2 areaB.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%The following was written by ENS Joe Dugan under the  %
%  advisory of Major Paul Blue for his thesis work on  %
%                   AFIT/GAE/ENY/05-J08.                %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
function [x,y,n,k,h]=areaB
%this is an estimate of the terrain and building of Area B based on a
%satellite image. Its 2-Dimensional accuracy of the buildings is as good as
%a 2246x2976 pixel satellite image can provide, but in 3-Dimensions should
%not be relied upon, as the height of the terrain was merely estimated to
%check other algorithms. The coordinates are entered as the pixel location
%of the buildings and estimated terrain and then can be converted to
%lat/lon/alt or east/north/up in feet based on the "for" loops below.

%x and y are arrays of n x m size, where each row represents a loop of
%constant altitude and each column with the row is a coordinate of a corner
%of the loop. h is a vector of the altitude of each loop and k is a vector
%containing the number of points in each loop. NOTE: the number of points
%in each loop is one less than the number of columns in each row, because
%the row represents a closed loop, i.e. the first and last coordinates are
%the same.

%outside box
x(1,:)=[0 2976 2976 0 0 0 0];
y(1,:)=[0 0 2246 2246 0 0 0];
k(1)=4;
h(1)=1;

%museum A
x(2,:)=[844 998 1067 914 844 0 0];
y(2,:)=[909 735 802 973 909 0 0];
h(2)=1;
k(2)=4;
x(14,:)=[848 913 1066 999 848 0 0];
y(14,:)=[911 971.5 801.1 736 911 0 0];
h(14)=5;
k(14)=4;
x(3,:)=[858 1009 1049 895 858 0 0];
```

```
y(3,:)=[920 749 786 954 920 0 0];
h(3)=7;
k(3)=4;
x(4,:)=[876 1024 1034 880 876 0 0];
y(4,:)=[936 764 775 939 936 0 0];
k(4)=4;
h(4)=9;

%museum B
x(5,:)=x(2,:)+78;
y(5,:)=y(2,:)+80;
k(5)=4;
h(5)=1;
x(15,:)=x(14,:)+78;
y(15,:)=y(14,:)+80;
h(15)=5;
k(15)=4;
x(6,:)=x(3,:)+78;;
y(6,:)=y(3,:)+80;
k(6)=4;
h(6)=7;
x(7,:)=x(4,:)+78;
y(7,:)=y(4,:)+80;
k(7)=4;
h(7)=9;

%museum C
x(8,:)=x(5,:)+78;
y(8,:)=y(5,:)+80;
k(8)=4;
h(8)=1;
x(16,:)=x(15,:)+78;
y(16,:)=y(15,:)+80;
h(16)=5;
```

```
k(16)=4;
x(9,:)=x(6,:)+78;
y(9,:)=y(6,:)+80;
k(9)=4;
h(9)=7;
x(10,:)=x(7,:)+78;
y(10,:)=y(7,:)+80;
k(10)=4;
h(10)=9;

%hill GL
x(11,:)=[2218 2466 2146 2158 2976 2976 2218];
y(11,:)=[2234 1306 722 5 0 2246 2234];
k(11)=6;
h(11)=2;

%hill 5
x(12,:)=[2514 2514 2970 2970 2514 0 0];
y(12,:)=[2214 10 10 2214 2214 0 0];
k(12)=4;
h(12)=5;

%hill 10
x(13,:)=[2870 2870 2970 2970 2870 0 0];
y(13,:)=[2214 10 10 2214 2214 0 0];
k(13)=4;
h(13)=10;

%tower 20
x(17,:)=[2870 2882 2894 2894 2882 2870 2870];
y(17,:)=[1062 1054 1062 1076 1079 1076 1062];
k(17)=6;
h(17)=10;
```

```
x(18,:)=[2871 2882 2893 2893 2882 2871 2871];
y(18,:)=[1062 1055 1062 1075 1078 1075 1062];
k(18)=6;
h(18)=20;

%tower 50
x(19,:)=[2861 2866 2866 2861 2861 0 0];
y(19,:)=[1047 1047 1053 1053 1047 0 0];
k(19)=4;
h(19)=10;

x(20,:)=[2862 2865 2865 2862 2862 0 0];
y(20,:)=[1048 1048 1052 1052 1048 0 0];
k(20)=4;
h(20)=50;

%building
x(21,:)=[2278 2372 2372 2278 2278 0 0];
y(21,:)=[690 690 764 764 690 0 0];
k(21)=4;
h(21)=5;

x(22,:)=[2279 2371 2371 2279 2279 0 0];
y(22,:)=[691 691 763 763 691 0 0];
k(22)=4;
h(22)=10;

%building
x(23,:)=[1747 1810 1810 1747 1747 0 0];
y(23,:)=[485 485 542 542 485 0 0];
k(23)=4;
h(23)=2;

x(24,:)=[1748 1809 1809 1748 1748 0 0];
```

```
y(24,:)=[486 486 541 541 486 0 0];
k(24)=4;
h(24)=7;

%building
x(25,:)=x(23,:)+38;
y(25,:)=y(23,:)+82;
k(25)=4;
h(25)=2;

x(26,:)=x(24,:)+38;
y(26,:)=y(24,:)+82;
k(26)=4;
h(26)=7;

%building
x(27,:)=x(25,:)+38;
y(27,:)=y(25,:)+82;
k(27)=4;
h(27)=2;

x(28,:)=x(26,:)+38;
y(28,:)=y(26,:)+82;
k(28)=4;
h(28)=7;

%building
x(29,:)=x(27,:)+103;
y(29,:)=y(27,:);
k(29)=4;
h(29)=2;

x(30,:)=x(28,:)+103;
y(30,:)=y(28,:);
```

```
k(30)=4;
h(30)=7;

%building
x(31,:)=[2229 2295 2409 2350 2229 0 0];
y(31,:)=[1113 1084 1350 1374 1113 0 0];
k(31)=4;
h(31)=2;

x(32,:)=[2230 2294 2408 2351 2230 0 0];
y(32,:)=[1114 1085 1349 1373 1114 0 0];
k(32)=4;
h(32)=7;

%building
x(33,:)=[2090 2159 2159 2090 2090 0 0];
y(33,:)=[734 734 772 772 734 0 0];
k(33)=4;
h(33)=2;

x(34,:)=[2091 2158 2158 2091 2091 0 0];
y(34,:)=[735 735 771 771 735 0 0];
k(34)=4;
h(34)=7;

%building
x(35,:)=[2079 2146 2146 2079 2079 0 0];
y(35,:)=[538 538 621 621 538 0 0];
k(35)=4;
h(35)=2;

x(36,:)=[2080 2145 2145 2080 2080 0 0];
y(36,:)=[539 539 620 620 539 0 0];
k(36)=4;
```

```
h(36)=7;

%building
x(37,:)=[2071 2255 2266 2144 2142 2075 2071];
y(37,:)=[241 240 401 411 439 441 241];
k(37)=6;
h(37)=5;

x(38,:)=[2072 2254 2265 2143 2141 2076 2072];
y(38,:)=[242 241 400 410 438 440 242];
k(38)=6;
h(38)=10;

%building
x(39,:)=[2155 2285 2287 2155 2155 0 0];
y(39,:)=[52 47 81 89 52 0 0];
k(39)=4;
h(39)=5;

x(40,:)=[2156 2284 2286 2156 2156 0 0];
y(40,:)=[53 48 80 88 53 0 0];
k(40)=4;
h(40)=10;

%building
x(41,:)=[2306 2431 2430 2306 2306 0 0];
y(41,:)=[48 40 84 85 48 0 0];
k(41)=4;
h(41)=5;

x(42,:)=[2307 2430 2429 2307 2307 0 0];
y(42,:)=[49 41 83 84 49 0 0];
k(42)=4;
h(42)=10;
```

```
%building
x(43,:)=[3550 3580 3680 3730 3680 3580 3550]/3.2105198786266;
y(43,:)=[3600 3750 3750 3600 3480 3480 3600]/3.2105198786266;
k(43)=6;
h(43)=1;

x(44,:)=[3566 3630 3704 3704 3630 3566 3566]/3.2105198786266;
y(44,:)=[3675 3749 3675 3550 3481 3550 3675]/3.2105198786266;
k(44)=6;
h(44)=9;

%hard boundary
x(101,:)=[802 1903 2149 240 802 0 0];
y(101,:)=[1209 1209 1838 1838 1209 0 0];
k(101)=4;
h(101)=25;

for j=1:max(k)+1 %mirrors and shifts image to adjust for pixels counting down
    for i=1:n
        y(i,j)=-y(i,j)+2246;
    end
    y(101,j)=-y(101,j)+2246;
end
% for j=1:max(k)+1 %converts to lat/long
%     for i=1:n
%         x(i,j)=x(i,j)*.033937/2976-84.121328;
%         y(i,j)=(-y(i,j)+2246)*.021078/2246+39.768439;
%     end
% end
for j=1:max(k)+1
    for i=1:n
        x(i,j)=x(i,j)*3.2105198786266*.3048-52.790896; %#of meters/feet per pixel in avds
        y(i,j)=y(i,j)*3.2105198786266*.3048+1202.619793; %based on differenece in east
```

```
% coordinates at corners divided by .3048 /# of pixels (2976)
    end
    x(101,j)=x(101,j)*3.2105198786266*.3048-52.790896;
    y(101,j)=y(101,j)*3.2105198786266*.3048+1202.619793;
end
for j=1:n
    h(j)=(h(j)-.99)*8-485.1; %estimated height in enu
end

%simulated hill to test program, located at thrid 'x' from left of runway
x(45,1)=946.987432-200;
x(45,2)=946.987432+200;
x(45,3)=946.987432+300;
x(45,4)=946.987432+200;
x(45,5)=946.987432-200;
x(45,6)=946.987432-300;
x(45,7)=946.987432-200;

y(45,1)=1782.186816+250;
y(45,2)=1782.186816+200;
y(45,3)=1782.186816;
y(45,4)=1782.186816-100;
y(45,5)=1782.186816-200;
y(45,6)=1782.186816+50;
y(45,7)=1782.186816+250;
k(45)=6;
h(45)=-485.020000;

x(46,1)=946.987432-150;
x(46,2)=946.987432;
x(46,3)=946.987432+150;
x(46,4)=946.987432+50;
x(46,5)=946.987432-80;
x(46,6)=946.987432-150;
```

```
x(46,7)=946.987432;

y(46,1)=1782.186816+30;
y(46,2)=1782.186816+180;
y(46,3)=1782.186816+35;
y(46,4)=1782.186816-80;
y(46,5)=1782.186816-90;
y(46,6)=1782.186816+30;
y(46,7)=1782.186816;
k(46)=5;
h(46)=-435.020000;

x(47,1)=946.987432;
x(47,2)=946.987432+15;
x(47,3)=946.987432+5;
x(47,4)=946.987432-10;
x(47,5)=946.987432;
x(47,6)=946.987432;
x(47,7)=946.987432;

y(47,1)=1782.186816+5;
y(47,2)=1782.186816;
y(47,3)=1782.186816-8;
y(47,4)=1782.186816-2;
y(47,5)=1782.186816+5;
y(47,6)=1782.186816;
y(47,7)=1782.186816;
k(47)=4;
h(47)=-420;

n=47;

fid = fopen('areaB.txt', 'wt'); %this prints to a text file the equivalent
%information but in c++ code rather than matlab.
```

```
for i=1:n
    for j=1:7
        fprintf(fid,'\nx_e[%i][%i]=%f;',i-1,j-1,x(i,j));
    end
     fprintf(fid,'\n');
    for j=1:7
        fprintf(fid,'\ny_n[%i][%i]=%f;',i-1,j-1,y(i,j));
    end
    fprintf(fid,'\nm[%i]=%i;',i-1,k(i));
    fprintf(fid,'\nh[%i]=%f;\n\n',i-1,h(i));
end
%hard boundary
for j=1:6
    fprintf(fid,'\nx_eH[%i]=%f;',j-1,x(101,j));
end
for j=1:6
    fprintf(fid,'\ny_nH[%i]=%f;',j-1,y(101,j));
end
fprintf(fid,'\nmH=%i;',k(101));
fclose(fid);

for i=1:n
plot(x(i,1:k(i)+1),y(i,1:k(i)+1))
hold on
end
plot(x(101,1:k(101)+1),y(101,1:k(101)+1));
```

A.3 3DLinearInterpolate.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%The following was written by ENS Joe Dugan under the  %
%  advisory of Major Paul Blue for his thesis work on  %
%                   AFIT/GAE/ENY/05-J08.                 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
function[hp]=3DLinearInterpolate(x,y,n,m,h,xp,yp)
%Interpolates a minimum altitude, hp, based on terrain elevation
%information provided by x, y, and h

%x=x-coordinates, y=y-coordinates, n=#rows of x, m=#columns in a particular
%row of x, h=height of particular line, xp=test point x, yp=test point y,
%NOTE WELL: the arrays x and y must be rows of CLOSED loops, meaning the
%first and last coordinates in the row must be the same and m(i) is the
%number of POINTS (NOT including the ending point, which is the same as the
%starting point) in ith row.
count=1;
%loops through all rows (i.e. all topographic loops) and within each row,
%all points from 1 to m(i) and forms a vector of the distances between the
%test point and the closest points on the boundary
for i=1:n
    for j=1:m(i)
        %the following if statement provides xperp,yperp, the point on the
        %boundary closest to the test point (xp,yp)
        if abs(x(i,j+1)-x(i,j))<0.01 %vertical boundary check
            xperp(i,j)=x(i,j);
            yperp(i,j)=yp;
        elseif abs(y(i,j+1)-y(i,j))<0.01 %horizontal boundary check
            xperp(i,j)=xp;
            yperp(i,j)=y(i,j);
        else
            mline=(y(i,j+1)-y(i,j))/(x(i,j+1)-x(i,j)); %oblique boundary
            b=y(i,j)-mline*x(i,j);
            mperp=-1/mline;
            bperp=yp-xp*mperp;
            xperp(i,j)=(b-bperp)/(mperp-mline);
            yperp(i,j)=mperp*xperp(i,j)+bperp;
        end
        %distance from current corner of boundary to the next corner
        dist12=sqrt((x(i,j+1)-x(i,j))^2+(y(i,j+1)-y(i,j))^2);
        %distance from current corner to the point on the boundary closest
        %to (xp,yp)
        dist1p=sqrt((x(i,j)-xperp(i,j))^2+(y(i,j)-yperp(i,j))^2);
        %distance from next corner to the point on the boundary closest to
        %(xp,yp)
        dist2p=sqrt((x(i,j+1)-xperp(i,j))^2+(y(i,j+1)-yperp(i,j))^2);
        %if the closest point to the line containing the boundary is
        %between the corners, dist1p+dist2p=dist12
        if abs((dist1p+dist2p)-(dist12))>0.01
            %if the closest point to the line is not on the boundary, do
            %not record in the closest points vector
            count=count;
        else
            %distance from test point to point on boundary
            dista(count,:)=[i sqrt((xp-xperp(i,j))^2+(yp-yperp(i,j))^2)];
            %xx,yy,hh matrices contain x,y,h of the point plus what line it
            %is on in the array
            xx(count,:)=[i xperp(i,j)];
            yy(count,:)=[i yperp(i,j)];
            hh(count,:)=[i h(i)];
            %this is the maximum rate of climb determined by the performace
            %of the aircraft
            maxslope=35;
            %this defines a second minimum altitude determined not by the
            %terrain, but by the performance of the aircraft
            hmin(count)=h(i)-dista(count,2)*maxslope;
            count=count+1; %increments the closest points vector index
        end
    end
    for j=1:m(i)
        %checks the corners for the closest points
        dista(count,:)=[i sqrt((x(i,j)-xp)^2+(y(i,j)-yp)^2)];
        xx(count,:)=[i x(i,j)];
        yy(count,:)=[i y(i,j)];
        hh(count,:)=[i h(i)];
        maxslope=35;
        %again, defines minimum altitude based on a/c performance, referred
        %to as the urban canyon check
        hmin(count)=h(i)-dista(count,2)*maxslope;
        count=count+1;
    end
end
%dist1(2) is the smallest distance between (xp,yp) and a any boundary or
%corner, index1(2) is the row index for this point
[dist1,index1]=min(dista);
%temporary variable to store the largest distance between the test point
%and any boundary/corner
temp=max(dista);
for i=1:length(dista)
    %this changes all of the points on the loop containing the closest
    %point to the largest distance to ensure the second closest point comes...
 from a different loop
    if dista(i,1)==dista(index1(2),1)
        %changes the smallest distance to the largest in order to find
        %second smallest
        dista(i,2)=temp(2);
    end
end
%Checks if test point is within the loop containing the closest point
b=Check_In_or_Out(x(dista(index1(2)),1:m(dista(index1(2)))+1),y(dista(index1(2)),...
1:m(dista(index1(2)))+1),xp,yp);
bcount=0;
bbb=0;
%below is a method to determine if any other of the closest points to the
%test point are within the loop containing the closest point
if b==1
    for i=1:length(dista)
        if xx(i,1)~=xx(index1(2),1)
```

```
            bb=Check_In_or_Out(x(dista(index1(2)),1:m(dista(index1(2)))+1),y(dista...
(index1(2)),1:m(dista(index1(2)))+1),xx(i,2),yy(i,2));
            if bb==1
                bbb=bb;%number of points inside closest boundary\
            end
        end
    end
end
%if the point is within the loop containing the closest point AND there are
%no more points within boundary
if bbb==0 && b==1
    hp=h(dista(index1(2)));
else
    for i=1:length(dista) %checks for 2nd closest point to the test point
        [dist2,index2]=min(dista); %finds 2nd smallest distance
        %if the 2nd closest is inside other bound
        insidebound=Check_In_or_Out(x(dista(index1(2)),1:m(dista(index1(2)))+1),...
y(dista(index1(2)),1:m(dista(index1(2)))+1),xx(index2(2),2),yy(index2(2),2));
%If the test point is inside the loop containing the closest point, then
%the second closest point must also come from within the loop. If the test
%point is outside the loop containing the closest point, then
%the second closest point must also come from outside the loop.
        if insidebound~=b
            temp=max(dista);
            dista(index2(2),2)=temp(2);
        else
            break
        end
    end
    %Now the two closest valid points are stored and minimum altitude can
    %be interpolated
    dist12=sqrt((xx(index1(2),2)-xx(index2(2),2))^2+(yy(index1(2),2)-yy(index2(2),2))^2);
    %the following method determines the equation of a line between the two
    %closest points of known altitude, then determines where the closest
```

```
%point is on that line to the test point, and uses that point to
%interpolate between the points
if abs(xx(index1(2),2)-xx(index2(2),2))<0.01 %vertical line check
    dd1=abs(yp-yy(index1(2),2));
    dd2=abs(yp-yy(index2(2),2));
elseif abs(yy(index1(2),2)-yy(index2(2),2))<0.01 %horizontal line check
    dd1=abs(xp-xx(index1(2),2));
    dd2=abs(xp-xx(index2(2),2));
else %oblique line
    mm=(yy(index1(2),2)-yy(index2(2),2))/(xx(index1(2),2)-xx(index2(2),2));
    bb=yy(index1(2),2)-mm*xx(index1(2),2);
    mmperp=-1/mm;
    bbperp=yp-xp*mmperp;
    xxperp=(bb-bbperp)/(mmperp-mm); %xperp=(b12-bpostoperp)/(mpostoperp-m12)
    yyperp=mmperp*xxperp+bbperp;
    dd1=sqrt((xxperp-xx(index1(2),2))^2+(yyperp-yy(index1(2),2))^2);
    dd2=sqrt((xxperp-xx(index2(2),2))^2+(yyperp-yy(index2(2),2))^2);
end
%in the case where the closest point on the interpolation line is not
%between the two closest points, use a best guess to interpolate, in
%order to not go above the highest altitude nor below the lowest altitude
if dd1+dd2-dist12>.01
    dist12=dd1+dd2;
end
%if two loops happen to intersect at the closest points, use that known
%altitude as to not have the interpolation go unbounded
if dist12<.0000001
    hp=hh(index1(2),2);
else
    hp=dd1/dist12*hh(index2(2),2)+dd2/dist12*hh(index1(2),2);
end
%the following can be commented out to turn off the urban canyon safety
%check, but it provides somewhat of a circus tent effect over tall
%buildings so as to ensure the aircraft can overcome the building
```

```
    %based on its performance capabilities
   if hp<max(hmin)
       hp=max(hmin);
   end
end
```

A.4 Check_In_or_Out.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%The following was written by ENS Joe Dugan under the  %
%  advisory of Major Paul Blue for his thesis work on  %
%                   AFIT/GAE/ENY/05-J08.                %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [b]=Check_In_or_Out(x,y,xp,yp)
%takes testpoint (xp,yp) and determines if it's within boundary
%defined by vector of coords x and y. NOTE WELL: the boundary must be
%CLOSED, that is, the first and last coordinates must be the same
kk=length(x)-1;
%kk is the number of points on the boundary, NOT including the last point
%which would be the same as the first for a closed boundary
count=0;
%the method assumes a horizontal line extending from the test point (xp,yp)
%in the positive x direction and counts the number of boundaries the line
%crosses. If it crosses an even number of boundaries, the test point is
%outside the closed polygon. If it crosses an odd number of boundaries, the
%test point is within the bounds of the polygon.
for i=1:kk
    %dist12 is the distance from the current corner [x(i),y(i)], to the
    %next corner [x(i+1),y(i+1)], dist1p is the distance from the current
    %corner to the test point, and dist2p is the distance from the next
    %corner, to the test point. xI is the x-coordinate of the intersection
    %the line containing the boundary and the horizontal line extending to
    %infinity from the test point. Because the line is horizontal, the
    %y-coordinate of the intersection point is always equat to yp
    if x(i+1)-x(i)==0; %horizontal boundary check
        dist12=abs(y(i+1)-y(i));
        dist2p=abs(y(i+1)-yp);
        dist1p=abs(yp-y(i));
        xI=x(i);
    elseif y(i+1)-y(i)==0; %vertical boundary check
        dist12=0;
        dist2p=1;
        dist1p=1;
        xI=1;
    else
        m=(y(i+1)-y(i))/(x(i+1)-x(i));
        b=y(i)-m*x(i);
        xI=(yp-b)/(m);
        dist12=sqrt((x(i+1)-x(i))^2+(y(i+1)-y(i))^2);
        dist1p=sqrt((x(i)-xI)^2+(y(i)-yp)^2);
        dist2p=sqrt((x(i+1)-xI)^2+(y(i+1)-yp)^2);
    end
    %this claims that if the intersection point is on the boundary, and it
    %lies to the right of the test point, then it counts as an intersection
    %with a boundary
    if ((dist1p+dist2p)-(dist12))<.000001 && xI>xp
        count=count+1;
    end
end
%after looping, if the number of crosses of the is even or odd then...
if mod(count,2)==0
    b=0; %if it's even, the test point is outside or...
else
    b=1; %if it's odd, it is inside the bounds of the polygon.
end
```

Appendix B. C++® *Program Code*

The following C++® code was written, compiled, and linked in order to produce the SASVRT executable file. Note: to avoid excessive redundant information, the AreaB funciton of this program was omitted. So when the program calls the AreaB function, it is simply loading the information output by the areaB.m script in Appendix A converted to ENU coordinates and written in C++® . For example, in the MATLAB® script the first Air Force Museum building appears as:

```
%museum A
x(2,:)=[844 998 1067 914 844 0 0];
y(2,:)=[909 735 802 973 909 0 0];
h(2)=1;
k(2)=4;
x(14,:)=[848 913 1066 999 848 0 0];
y(14,:)=[911 971.5 801.1 736 911 0 0];
h(14)=5;
k(14)=4;
x(3,:)=[858 1009 1049 895 858 0 0];
y(3,:)=[920 749 786 954 920 0 0];
h(3)=7;
k(3)=4;
x(4,:)=[876 1024 1034 880 876 0 0];
y(4,:)=[936 764 775 939 936 0 0];
k(4)=4;
h(4)=9;
```

and after its coordinates are transformed and written into C++® it appears as:

```
x_e[1][0]=773.119195;
```

```
x_e[1][1]=923.818430;
x_e[1][2]=991.339516;
x_e[1][3]=841.618848;
x_e[1][4]=773.119195;
x_e[1][5]=-52.790896;
x_e[1][6]=-52.790896;

y_n[1][0]=2510.963149;
y_n[1][1]=2681.233713;
y_n[1][2]=2615.669760;
y_n[1][3]=2448.334895;
y_n[1][4]=2510.963149;
y_n[1][5]=3400.480060;
y_n[1][6]=3400.480060;
m[1]=4;
h[1]=-485.020000;


x_e[2][0]=786.819126;
x_e[2][1]=934.582661;
x_e[2][2]=973.725319;
x_e[2][3]=823.026085;
x_e[2][4]=786.819126;
x_e[2][5]=-52.790896;
x_e[2][6]=-52.790896;

y_n[2][0]=2500.198918;
y_n[2][1]=2667.533782;
```

```
y_n[2][2]=2631.326823;
y_n[2][3]=2466.927658;
y_n[2][4]=2500.198918;
y_n[2][5]=3400.480060;
y_n[2][6]=3400.480060;
m[2]=4;
h[2]=-437.020000;


x_e[3][0]=804.433322;
x_e[3][1]=949.261158;
x_e[3][2]=959.046823;
x_e[3][3]=808.347588;
x_e[3][4]=804.433322;
x_e[3][5]=-52.790896;
x_e[3][6]=-52.790896;

y_n[3][0]=2484.541854;
y_n[3][1]=2652.855285;
y_n[3][2]=2642.091054;
y_n[3][3]=2481.606155;
y_n[3][4]=2484.541854;
y_n[3][5]=3400.480060;
y_n[3][6]=3400.480060;
m[3]=4;
h[3]=-421.020000;
```

The rest of the C++ code is as follows:

```
/*******************************************************************\
%The following was written by ENS Joe Dugan under the  %
%  advisory of Major Paul Blue for his thesis work on  %
%                   AFIT/GAE/ENY/05-J08.                %
\***************************************************************/

#include "windows.h"
//#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#include <UserNetwork.h>

//Piccolo Includes
#include<iostream.h>
#include<conio.h>
#include "CommManager.h"
#include "Win32Serial.h"
#include"lla2enu.h"
#include"my_types.h"

//Basic Variables/Arrays vital to all parts of code
CCommManager* m_pComm = NULL; //initialize Communications Manager m_pComm
Queue_t* pQ = NULL; //used to see if autopilot packets exist
ENUCoord PosENU; //East-North-Up coordinate used for telemetry and avoiding obstacles
telemetry current_telemetry[10];//holds decoded telemetry packet data for up to
// 10 networks
control current_control[10]; //holds decoded control packet data for up to
//10 networks

FILE * pFile1;
FILE * pFile2;
FILE * pFile3;
FILE * cFile1;
FILE * acFile1;
FILE * pbFile1;
FILE * pbFile2;

UInt8 Waypoint_cmd[10]; //holds the index of the waypoint each network is currently
// heading towards - up to 10 networks

float deg_to_rad = 3.14159/180.0;

void send_telemetry(float east, float north, float up, int i);
void send_control(float phi, float theta, float psi, int i);
void init_AC(float lat, float lon, float alt, int i);
int Mindex(double *Numbers, int Count);
double Min(double *Numbers, int Count);
int Maxi(int *Numbers, int Count);
double Max(double *Numbers, int Count);
bool CheckInside(double *x_et, double *y_nt, double cte, double ctn, int count);
void hardbound(double *x_et, double *y_nt, int count, int i); //for hard bounds
double MinH(double ctn, double cte);
void printTerrain();
void addLoop();
void addHardBound();
void addBuilding();
void printTerrain();
int RetrieveLoop();
void clrscr();
void displayTelemetry(int i);
void displayData(int i, int numnets);
void areaB();
void clearData();
void Waypoints();


Aircraft ac[10];
Aircraft ground;

//Basepoint to use for all ENU coordinates...calculated by doing lla2ecef transformation
// in matlab at a lat/lon/alt near AFIT
//Note that the further the basepoint from actual position, the more error
const double Base_X = 503000;
const double Base_Y = -4884700;
const double Base_Z = 4057800;
double x_e[100][10], y_n[100][10], x_eH[10], y_nH[10], h[100],x_o[5],y_o[5], ctu_old=0,...
 time_old, timeH[10], distH, VH, distS, timeS, VS,xtemp_e[10],ytemp_n[10],htemp,...
xstart, ystart, zstart, xground, yground, zground, PBtime=.01,timeStart=100000;
int m[100], n=-1, mH, mo, data=8;
bool pboo=false, wpoo=false,stoo[10]={false,false,false,false,false,false,false,false,...
false,false}, soo=false, hio,sio, hoo=false, firsttime[10]={false,false,false,false,false,...
false,false,false,false,false},goo=false,sioo=false;


void hardbound(double *x_eH, double *y_nH, int mH, int i)
{
double xperp, yperp, mline, b, mperp, bperp, dist12, dist1p, dist2p, dista[20],x[20],y[20];
int count=0,indexa[20];
double ctn,cte;
int j;
int NumNets = m_pComm->GetNumNets();
//for(int i=0; i<NumNets; i++)

{
ctn = current_telemetry[i].North;
cte = current_telemetry[i].East;
//hio = CheckInside(x_eH,y_nH,cte,ctn,count);
//if (hio != 0)
{
for (int k=0; k<mH; k++) //checks closed loop for closest boundary
{
```

```
if (fabs(x_eH[k+1]-x_eH[k])<0.01)
{
xperp=x_eH[k];
yperp=ctn;
}
else if (fabs(y_nH[k+1]-y_nH[k])<0.01)
{
xperp=cte;
yperp=y_nH[k];
}
else
{
mline=(y_nH[k+1]-y_nH[k])/(x_eH[k+1]-x_eH[k]);
b=y_nH[k]-mline*x_eH[k];
mperp=-1/mline;
bperp=ctn-cte*mperp; //ypos-xpos*mpostoperp=bpostoperp
    xperp=(b-bperp)/(mperp-mline);
yperp=mperp*xperp+bperp;
}
dist12=sqrt((x_eH[k+1]-x_eH[k])*(x_eH[k+1]-x_eH[k])+(y_nH[k+1]-y_nH[k])*(y_nH[k+1]...
-y_nH[k])); //closest distance from test point to boundary line
    dist1p=sqrt((x_eH[k]-xperp)*(x_eH[k]-xperp)+(y_nH[k]-yperp)*(y_nH[k]-yperp));
dist2p=sqrt((x_eH[k+1]-xperp)*(x_eH[k+1]-xperp)+(y_nH[k+1]-yperp)*(y_nH[k+1]-yperp));
if (fabs((dist1p+dist2p)-(dist12))<0.01) //makes sure intersection point is on boundary
{
dista[count]=sqrt((xperp-cte)*(xperp-cte)+(yperp-ctn)*(yperp-ctn));
x[count]=xperp;
y[count]=yperp;
indexa[count]=count;
count=count+1;
}
dista[count]=sqrt((x_eH[k]-cte)*(x_eH[k]-cte)+(y_nH[k]-ctn)*(y_nH[k]-ctn));
y[count]=x_eH[k];
y[count]=y_nH[k];
```

```
indexa[count]=count;
count=count+1;
}
distH = Min(dista, count); //closest border
//int index = Mindex(dista, count);
for (int index=0; index<mH; index++){
VH=current_telemetry[i].Velocity * cos(current_control[i].psi*deg_to_rad)*...
(y[index]-ctn)/(sqrt((y[index]-ctn)*(y[index]-ctn)+(x[index]-cte)*(x[index]-cte)))+...
current_telemetry[i].Velocity * sin(current_control[i].psi*deg_to_rad)*(x[index]-cte)/...
(sqrt((y[index]-ctn)*(y[index]-ctn)+(x[index]-cte)*(x[index]-cte)));
if (VH>0.01)
timeH[index] = distH/VH;
else
timeH[index] =-1;
}
}
}
}

int Mindex(double *Numbers, int Count) //finds the index of the minimum in an
// array of doubles
{
double Minimum = Numbers[0];
int Index=0;

for(int k = 0; k < Count; k++)
if( Minimum > Numbers[k] )
{
Minimum = Numbers[k];
Index=k;
}

return Index;
}
```

```
double Min(double *Numbers, int Count) //finds the minimum value in an array of doubles
{     double Minimum = Numbers[0];


for(int k = 0; k < Count; k++){
if( Minimum > Numbers[k] )
Minimum = Numbers[k];
}

return Minimum;
}
int Maxi(int *Numbers, int Count) //finds the minimum value in an array of integers
{    int Maximum = Numbers[0];


for(int k = 0; k < Count; k++){
if( Maximum < Numbers[k] )
Maximum = Numbers[k];
}

return Maximum;
}

double Max(double *Numbers, int Count)//finds the maximum value in an array of doubles
{
double Maximum = Numbers[0];


for(int k = 0; k < Count; k++){
if( Maximum < Numbers[k] )
Maximum = Numbers[k];
}
```

```
return Maximum;
}

bool CheckInside(double *x_et, double *y_nt, double cte, double ctn, int count)
//pass in CLOSED LOOPS described by VECTORS x_et, y_nt, along
// with ctn, cte to determine if it is within the bounds
{
double dist12, dist2p, dist1p, xI, m, b;
int condition=0;
for(int counter=0; counter<count; counter++) //loops through all points,
// doesn't include first point twice
{
if (fabs(x_et[counter+1]-x_et[counter])<.01)
{
    dist12=fabs(y_nt[counter+1]-y_nt[counter]);
   dist2p=fabs(y_nt[counter+1]-ctn);
dist1p=fabs(y_nt[counter]-ctn);
xI=y_nt[counter];
}
else if (fabs(y_nt[counter+1]-y_nt[counter])<.01)
{
        dist12=0;
       dist2p=1;
dist1p=1;
xI=1;
}
else
{
m=(y_nt[counter+1]-y_nt[counter])/(x_et[counter+1]-x_et[counter]);
b=y_nt[counter]-x_et[counter]*m;
xI=(ctn-b)/m;
dist12=sqrt((x_et[counter+1]-x_et[counter])*(x_et[counter+1]-x_et[counter])+...
(y_nt[counter+1]-y_nt[counter])*(y_nt[counter+1]-y_nt[counter]));
dist1p=sqrt((xI-x_et[counter])*(xI-x_et[counter])+(ctn-y_nt[counter])...
*(ctn-y_nt[counter]));
dist2p=sqrt((x_et[counter+1]-xI)*(x_et[counter+1]-xI)+(y_nt[counter+1]-ctn)...
*(y_nt[counter+1]-ctn));
}
if (dist1p+dist2p-dist12<0.01 && xI>cte) {condition=condition+1;
}
else condition=condition;
}
if (condition%2==0)
return false;
else
return true;
}


double MinH(double ctn, double cte) //given a system of n closed topographic loops
// defined by x_e, y_n, h_u, and point cte,ctn, determines min alt. total length is
// all points times 2.
{
double xperp, yperp, mline, b, mperp, bperp, dist12, dist1p, dist2p, dista[1000],...
xx[1000],yy[1000],hh[1000],minalt,x[100][10],y[100][10];
int count=0, indexa[1000],ma[1000],j,k,Index2;
int Total=sizeof(m)/sizeof(int);
int maxm = Maxi(m, n);
for (k=0; k<100; k++)
{
dista[k]=0;
indexa[k]=0;
xx[k]=0;
yy[k]=0;
hh[k]=0;
}

for (k=0; k<n; k++){
```

```
for (j=0; j<maxm; j++)
{
x[k][j]=0;
y[k][j]=0;
}
}

for (k=0; k<n; k++){ //reconstructs x and y matrices
for (j=0; j<10; j++)
{
x[k][j]=x_e[k][j];//x_e[k*10+j]; //the 1+maxm accounts for the first and last values
// of columns are the same closing the
y[k][j]=y_n[k][j];//y_n[k*10+j]; //loop and making the rows one column longer than
// the number of points maxm
}
}
for (k=0; k<n; k++) //these loops form a vector of the distances between the test
// point and the closest points on the boundary
{

for (j=0; j<m[k]; j++)
{

if (fabs(x[k][j+1]-x[k][j])<0.01)
{
xperp=x[k][j];
yperp=ctn;
}
else if (fabs(y[k][j+1]-y[k][j])<0.01)
{
xperp=cte;
yperp=y[k][j];
}
else
```

```
{
mline=(y[k][j+1]-y[k][j])/(x[k][j+1]-x[k][j]);
b=y[k][j]-mline*x[k][j];
mperp=-1/mline;
bperp=ctn-cte*mperp; //ypos-xpos*mpostoperp=bpostoperp
    xperp=(b-bperp)/(mperp-mline);
yperp=mperp*xperp+bperp;
}
        dist12=sqrt((x[k][j+1]-x[k][j])*(x[k][j+1]-x[k][j])+(y[k][j+1]-y[k][j])*...
(y[k][j+1]-y[k][j])); //closest distance from test point to
// boundary line
    dist1p=sqrt((x[k][j]-xperp)*(x[k][j]-xperp)+(y[k][j]-yperp)*(y[k][j]-yperp));
dist2p=sqrt((x[k][j+1]-xperp)*(x[k][j+1]-xperp)+(y[k][j+1]-yperp)*(y[k][j+1]-yperp));
if (fabs((dist1p+dist2p)-(dist12))<0.01) //makes sure intersection point is on boundary
{
dista[count]=sqrt((xperp-cte)*(xperp-cte)+(yperp-ctn)*(yperp-ctn));
xx[count]=xperp;
yy[count]=yperp;
hh[count]=h[k];
indexa[count]=k; //indexa is the LOOP NUMBER
ma[count]=m[k];
count=count+1;
}

}
for (j=0; j<m[k]; j++) //checks corners for distance
{
dista[count]=sqrt((x[k][j]-cte)*(x[k][j]-cte)+(y[k][j]-ctn)*(y[k][j]-ctn));
xx[count]=x[k][j];
yy[count]=y[k][j];
hh[count]=h[k];
indexa[count]=k;
ma[count]=m[k];
count=count+1;
```

```
}
}

double Min1 = Min(dista, count); //count should be the number of elements, starts
//at zero, goes to one above highest index
int Index1 = Mindex(dista, count); //index1 is the index number of the minimum
//distance from ct point and boundary: indexa[Index1]=min dist loop #
double xindex[50];
double yindex[50];
for (k=0; k<(1+ma[Index1]); k++); //ma[index1]+1 refers to the length of the CLOSED loop
// describing the closest boundary
{
xindex[k]=x[Index1][k]; //writes a vector for the x,y coords for the loop
yindex[k]=y[Index1][k];
}
int temp=sizeof(xindex)/sizeof(double);
xindex[temp]=xindex[0];
yindex[temp]=yindex[0]; //closes loop
float maxdist = Max(dista, count);
for (k=0; k<count; k++)
{
if (indexa[k]==indexa[Index1]) //this ensures that the second closest point comes
// from a different loop
dista[k]=maxdist; //changes the smallest distance to the largest in order to find
// second smallest
}
bool inorout=CheckInside(xindex, yindex, cte, ctn, ma[Index1]+1); //if test point
//is within closest boundary
int bcount=0;
bool bbb=false;
if (inorout==true)
{
for (k=1; k<count; k++)
{
```

```
if (xx[k]!=xx[Index1])
{
bool inorout2=CheckInside(xindex, yindex, xx[Index1], yy[Index1], ma[Index1]+1); //*?
if (inorout2==true)
bbb=inorout2; //there exists other boundaries within closest boundary
           }
        }
    }
if (bbb==false && inorout==true) //if the point is within the boundary of the closest
// point and there are no more points within boundary
{
minalt=hh[Index1];
return minalt;
}
else
{
for (int k=0; k<ma[Index1]; k++)
{
double Min2 = Min(dista, count);//find 2nd smallest distance
Index2 = Mindex(dista, count); //and index

bool inorout3=CheckInside(xindex, yindex, xx[Index2], yy[Index2], ma[Index1]+1);
 //this determines if the second min distance is within first bound
            if (indexa[Index1]==indexa[Index2]) //if they're on the same line
dista[Index2]=maxdist;
else if (inorout3!=inorout) //if the point is inside the bound, use the next closest
// point inside the bound
                dista[Index2]=maxdist; //OR if the point is outside th bound, use the
//next closest point outside the bound
}
}
    // at this point should have two closest points(within bounds) and the distance two the points
double dd1,dd2,mm,bb,mmperp,bbperp,xxperp,yyperp;
    dist12=sqrt((xx[Index1]-xx[Index2])*(xx[Index1]-xx[Index2])+(yy[Index1]-...
yy[Index2])*(yy[Index1]-yy[Index2]));
    if (fabs(xx[Index1]-xx[Index2])<0.01)
{
        dd1=fabs(ctn-yy[Index1]);
        dd2=fabs(ctn-yy[Index2]);
}
    else if (fabs(yy[Index1]-yy[Index2])<0.01)
{
        dd1=fabs(cte-xx[Index1]);
        dd2=fabs(cte-xx[Index2]);
}
    else
{
        mm=(yy[Index1]-yy[Index2])/(xx[Index1]-xx[Index2]);
        bb=yy[Index1]-mm*xx[Index1];
        mmperp=-1/mm;
        bbperp=ctn-cte*mmperp;
        xxperp=(bb-bbperp)/(mmperp-mm); //xperp=(b12-bpostoperp)/(mpostoperp-m12)
        yyperp=mmperp*xxperp+bbperp;
        dd1=sqrt((xxperp-xx[Index1])*(xxperp-xx[Index1])+(yyperp-yy[Index1])*...
(yyperp-yy[Index1]));
        dd2=sqrt((xxperp-xx[Index2])*(xxperp-xx[Index2])+(yyperp-yy[Index2])*...
(yyperp-yy[Index2]));
    }
    if ((dd1+dd2-dist12)>.01)
        dist12=dd1+dd2;
if (dist12<.0000001)
        minalt=hh[Index1];
    else
        minalt=dd1/dist12*hh[Index2]+dd2/dist12*hh[Index1];
//printf("\nminalt: %f",minalt);
//Sleep(10000);
return minalt;
}
```

```
//Displays Instructions
void displayData(int i, int NumNets) {

switch(data){
case(0):{
displayTelemetry(i);
break;
}
case(1):{
areaB();
break;
}
case(2):{
soo=true;
addBuilding();
break;
}
case(3):{
hoo=true;
addHardBound();
break;
}
case(4):{
soo=true;
addLoop();
break;
}
case(5):{
clearData();
data=8;
break;
}
```

```
case(6):{
printf("\nPrinting Terrain File");
printTerrain();
//printf("\nback in loop");
data=0;
break;
}
case(7):{
if(stoo[i]==true){
stoo[i]=false;
printf("\nNot streaming data to AVDS");
Sleep(1000);
}
else{
stoo[i]=true;
printf("\nStreaming data to AVDS");
Sleep(1000);
}
data=8;
break;
case(10):{
if(pboo==false){
pboo=true;
pbFile1 = fopen ("terrain.save.txt","w");
pbFile2 = fopen ("sig.save.txt","w");
fprintf(pbFile1,"%%LAT %f\n%%LONG %f\n\%%time xpos ypos Alt xrot yrot zrot craft type"...
,current_telemetry[i].Latitude,current_telemetry[i].Longitude);
fprintf(pbFile2,"%%LAT %f\n%%LONG %f\n\%%time xpos ypos Alt xrot yrot zrot craft type...
 rud el ail",current_telemetry[i].Latitude,current_telemetry[i].Longitude);
printf("\nPlayback Recording On");
}
else{
pboo=false;
fclose (pbFile1);
fclose (pbFile2);
printf("\nPlayback Recording Off");
}

Sleep(1000);
data=8;
break;
}
case(11):{
if(sioo==false){
sioo=true;
printf("\nSatellite image on");
}
else{
sioo=false;
printf("\nSatellite image off");
}
Sleep(1000);
data=8;
break;
}
case(8):{
clrscr();
printf("Instructions");
printf("\nPress a Number to see Individual Piccolo Data (1,2...)");
printf("\nPress 'T' to see Telemetry Data");
printf("\nPress 'A' to load Area B terrain data");
printf("\nPress 'B' to Define a building");
printf("\nPress 'H' to Define a hard boundary");
printf("\nPress 'L' to Define a loop of minimum altitude");
printf("\nPress 'C' to Clear boundary and altitdue information");
printf("\nPress 'P' to Print a terrain data file to load into AVDS");
printf("\nPress 'S' to Toggle streaming data to AVDS");
printf("\nPress 'R' to Toggle Playback Recorder");
printf("\nPress 'X' to Exit");

printf("\nCurrent Piccolo ID = %i", m_pComm->GetIDFromIndex(i));
break;
}
}
}

}
//displayData

void addBuilding(){ //adds a building to the synthetic terrain
n=n+1;
m[n]=RetrieveLoop();
double xt=xtemp_e[0],xcent[2]={0,0},ycent[2]={0,0};
double yt=ytemp_n[0];
int i, temp;

h[n]=MinH(xt, yt);
for (i=0; i<m[n]; i++){
x_e[n][i]=xtemp_e[i];
y_n[n][i]=ytemp_n[i];
}
for (i=0; i<m[n]; i++){
xcent[0] += x_e[n][i];
ycent[0] += y_n[n][i];
}
xcent[0]=xcent[0]/m[n];
ycent[0]=ycent[0]/m[n];
n=n+1;
h[n]=htemp;
m[n]=m[n-1];
for (i=0; i<m[n]; i++){
x_e[n][i]=0.99*xtemp_e[i];
```

```
y_n[n][i]=0.99*ytemp_n[i];
}
for (i=0; i<m[n]; i++){
xcent[1] += x_e[n][i];
ycent[1] += y_n[n][i];
}
xcent[1]=xcent[1]/m[n];
ycent[1]=ycent[1]/m[n];
for (i=0; i<m[n]; i++){
x_e[n][i]=x_e[n][i]+xcent[0]-xcent[1];
y_n[n][i]=y_n[n][i]+ycent[0]-ycent[1];
}
data=0;

} //Add Building

void addLoop(){ //adds loops of constant altitude
n=n+1;
m[n]=RetrieveLoop();
h[n]=htemp;
for (int i=0; i<m[n]; i++){
x_e[n][i]=xtemp_e[i];
y_n[n][i]=ytemp_n[i];
}
x_e[n][m[n]]=xtemp_e[0];
y_n[n][m[n]]=ytemp_n[0];
data=0;
}

void addHardBound(){ //adds loops of constant altitude
mH=RetrieveLoop();
for (int i=0; i<mH; i++){
x_eH[i]=xtemp_e[i];
y_nH[i]=ytemp_n[i];
```

```
}
x_eH[mH]=xtemp_e[0];
y_nH[mH]=ytemp_n[0];
data=0;
}


void clearData(){
n=0;
soo=false;
hoo=false;
wpoo=false;
printf("\nData Cleared");
Sleep(1000);
}

//Displays Network Telemetry and Control Information
void displayTelemetry(int i) {
clrscr();
UInt32 NumNets;
NumNets = m_pComm->GetNumNets();
printf("\nPress '0' to return to options");
if(data==8)
displayData(i,NumNets);
printf("\nTelemetry Packet Data  : %i", current_telemetry[i].Hours);
printf(":%i", current_telemetry[i].Minutes);
printf(":%f", current_telemetry[i].Seconds);
printf("\nLatitude (deg)         : %f", current_telemetry[i].Latitude);
printf(" East: %f", current_telemetry[i].East);
printf("\nLongitude (deg)        : %f", current_telemetry[i].Longitude);
printf(" North: %f", current_telemetry[i].North);
printf("\nAltitude (m)           : %f", current_telemetry[i].Altitude);
printf(" Up: %f", current_telemetry[i].Up);
printf("\nGround Speed           : %f", current_telemetry[i].Velocity);
printf("\nAir Speed             : %f", current_control[i].AirSpeed);
//print current control data
printf("\n\nControl Packet Data  : %i", current_control[i].Hours);
printf(":%i", current_control[i].Minutes);
printf(":%f", current_control[i].Seconds);
printf("\nPitch Angle            : %f", current_control[i].theta);
printf("\nHeading               : %f", current_control[i].psi);
printf(" Aileron (deg)     : %f", current_control[i].Aileron);
printf("\nBank Angle            : %f", current_control[i].phi);
printf(" Elevator (deg)    : %f", current_control[i].Elevator);
printf("\nRoll Rate             : %f", current_control[i].RollRate);
printf(" Throttle (percent): %f", current_control[i].Throttle*100);
printf("\nPitch Rate            : %f", current_control[i].PitchRate);
printf(" Rudder (deg)      : %f", current_control[i].Rudder);
printf("\nYaw Rate              : %f", current_control[i].YawRate);


//Boundary Information
double ctn = current_telemetry[i].North;
double cte = current_telemetry[i].East;
double ctu = current_telemetry[i].Up;

if (hoo==false)
printf("\n\nHard boundary not defined");
else{
hio=CheckInside(x_eH, y_nH, cte, ctn, mH);
hardbound(x_eH,y_nH,mH, i);
switch(hio)
{
case false:{
printf("\n\nOUTSIDE OF HARD BOUNDARY");
printf("\n");
fprintf(pFile2,"\n%6.4f %6.2f %6.2f %6.2f 0   0   0   0",(current_telemetry[i].Minutes...
+current_telemetry[i].Seconds/60),ctn,cte,ctu, hio);
```

```
break;
}
case true:{
printf("\n\nDistance to hard boundary(m): %f", distH);
for (int index=0; index<mH; index++){
if (timeH>0){
printf("\nTime until impact with hard boundary [%i](m): %f", index, timeH[index]);
fprintf(pFile2,"\n%6.4f %6.2f %6.2f %6.2f 1   %6.2f %6.2f %6.2f",(...
current_telemetry[i].Minutes+current_telemetry[i].Seconds/60),ctn,cte,...
ctu,distH,VH,timeH[index]);
}
else{
printf("\n");
fprintf(pFile2,"\n%6.4f %6.2f %6.2f %6.2f 1   %6.2f  %6.2f 0",(...
current_telemetry[i].Minutes+current_telemetry[i].Seconds/60),...
ctn,cte,ctu,distH,VH);
}
}
break;
}
}
}

switch(soo)
{
case false:{
printf("\nNo minimum altitude information present");
break;
   }
case true:{

double tempmin=MinH(ctn, cte);
distS=ctu-tempmin;
if (distS<0){
```

```
printf("\nBelow minimum altitdude, PULL UP!");
fprintf(pFile3,"\n%6.4f %6.2f %6.2f %6.2f 0   %6.2f",(current_telemetry[i].Minutes...
+current_telemetry[i].Seconds/60),ctn,cte,ctu,tempmin);
}

else{
printf("\nDistance above minimum altitude(m): %f", distS);;
fprintf(pFile3,"\n%6.4f %6.2f %6.2f %6.2f 1 %6.2f",(current_telemetry[i].Minutes...
+current_telemetry[i].Seconds/60),ctn,cte,ctu,tempmin);
}
break;
   }
}

for(int j=0; j<NumNets-1; j++){
if (m_pComm->GetIDFromIndex(j)!=m_pComm->GetIDFromIndex(i)){
double ctn2 = current_telemetry[j].North;
double cte2 = current_telemetry[j].East;
double ctu2 = current_telemetry[j].Up;
double distance = sqrt((ctn-ctn2)*(ctn-ctn2)+(cte-cte2)*(cte-cte2)+(ctu-ctu2)*(ctu-ctu2));
printf("\nDistance to aircraft %i: %f (m)", m_pComm->GetIDFromIndex(j),distance);
fprintf(acFile1,"\n%6.4f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f",(...
current_telemetry[i].Minutes+current_telemetry[i].Seconds/60),ctn,cte,ctu,ctn2,...
cte2,ctu2,distance);
}
}
if (stoo[i]==1)
printf("\nStreaming data to AVDS");
}
//displayTelemetry

void printTerrain(){
pFile1 = fopen ("terrain.txt","w");
fprintf(pFile1,"Synthetic Terrain|terrain:\\\n\n\n# center of mass\n :cm#...
 -1782.187376 -946.987728  -473.92:\\\n\n# Distance > 0 ft\n :d0:\\\n\n\n:DS#1:\\\n
:ca=fuselage:\\\n :ct# 1.0 0.0 0.0:\\\n\n# (back)\n :pt#  %f %f 0.00:pt# %f %f -600.0:\\
\n :pt#  %f %f -600.0:pt#  %f %f  0.00:cl:\\\n\n# (front)\n :pt#  %f %f  0.00:pt#  %f %f...
 -600.0:\\\n :pt# %f %f -600.0:pt# %f %f  0.00:cl:\n\n# (left)\n :pt#  %f %f  0.00:pt#...
  %f %f -600.0:\\\n :pt# %f %f -600.0:pt#  %f %f  0.00:cl:\\\n# (right)\n :pt# %f %f...
 0.00:pt# %f %f -600.0:\\\n :pt# %f %f -600.0:pt# %f %f  0.00:cl:\n :co# 0.700 0.700...
 0.700:\\\n\n\n#texture files\n:XF=AreaBsmall.bmp:\\\n:XF=AreaBsmall.bmp:\\\n\n\n\n#...
 (back)\n :pt# 2859.4   1202.6  -473.92:pt# 2859.4   1202.6 -476.92:\\\n ...
:pt#  -52.8   1202.6  -476.92:pt#  -52.8  1202.6  -473.92:cl:\\\n\n\n\n#select texture file...
\n:XI#1:\\\n#texture coordinates \n:XC#1.0 0.0 0.0 :XC#1.0 1.0 0.0 :XC#0.0 1.0 0.0 :...
XC#0.0 0.0 0.0:\\\n# (bottom)\n :pt#  -52.8   1202.6  -473.92:pt#  -52.8 3400.5...
  -473.92:\\\n :pt# 2859.4  3400.5  -473.92:pt# 2859.4   1202.6  -473.92:cl:\\\n\n\n#...
 (front)\n :pt# -52.8  3400.5  -473.92:pt#  -52.8  3400.5 -476.92:\\\n :pt#...
 2859.4  3400.5 -476.92:pt# 2859.4  3400.5  -473.92:cl:\\\n\n#select texture file\n:...
XI#1:\\\n#texture coordinates \n:XC#1.0 0.0 0.0 :XC#1.0 1.0 0.0 :XC#0.0 1.0 0.0 :XC#...
0.0 0.0 0.0:\\\n# (top)\n :pt# 2859.4   1202.6 -476.92:pt# 2859.4  3400.5 -476.92...
:\\\n :pt#  -52.8  3400.5 -476.92:pt# -52.8 1202.6 -476.92:cl:\\\n\n\n\n :co# 0.000...
 0.700 0.000:\\\n# (left)\n :pt#  -52.8   1202.6  -473.92:pt#  -52.8  1202.6 -476.92:...
\\\n :pt#  -52.8  3400.5 -476.92:pt# -52.8 3400.5  -473.92:cl:\\\n\n#select texture file...
\n:XI#0:\\\n#texture coordinates \n:XC#1.0 0.0 0.0 :XC#1.0 1.0 0.0 :XC#0.0 1.0 0.0 :...
XC#0.0 0.0 0.0:\\\n# (right):pt# 2859.4  3400.5  -473.92:pt# 2859.4  3400.5 -476.92:\\\...
 n\n :pt# 2859.4  1202.6 -476.92:pt# 2859.4   1202.6  -473.92:cl:\n",x_eH[3], y_nH[3...
],x_eH[3],y_nH[3],x_eH[2],y_nH[2],x_eH[2],y_nH[2],x_eH[4] ,y_nH[4],x_eH[4],y_nH[4]...
, x_eH[1],y_nH[1],x_eH[1],y_nH[1],x_eH[3],y_nH[3],x_eH[3],y_nH[3], x_eH[4],y_nH[4]...
,x_eH[4], y_nH[4], x_eH[2], y_nH[2],x_eH[2], y_nH[2],x_eH[1],y_nH[1],x_eH[1],y_nH[1]);


//fprintf(pFile1,"Synthetic Terrain|terrain:\\\n\n\n# center of mass\n :cm# -1782.187376...
 -946.987728  -473.92:\\\n\n# Distance > 0 ft\n :d0:\\\n\n\n:DS#1:\\\n :ca=...
fuselage:\\\n :ct# 1.0 0.0 0.0:\\\n\n# (back)\n :pt#  182.07 1601.87  0.00:...
pt# 182.07 1601.87 -600.0:\\\n :pt#  2050.15 1601.87 -600.0:pt#  2050.15...
  1601.87  0.00:cl:\\\n\n# (front)\n :pt#  732.01  2217.39  0.00:pt#  732.01 2217.39...
 -600.0:\\\n :pt# 1809.42  2217.39 -600.0:pt# 1809.42 2217.39  0.00:cl:\n\n#...
 (left)\n :pt#  182.07   1601.87  0.00:pt#  182.07  1601.87 -600.0:\\\n :pt#  732.01...
```

```
  2217.39 -600.0:pt#  732.01  2217.39  0.00:cl:\\\n# (right)\n :pt# 2050.15  1601.87...
  0.00:pt# 2050.15  1601.87 -600.0:\\\n :pt# 1809.42   2217.39 -600.0:pt#...
 1809.42   2217.39  0.00:cl:\n :co# 0.700 0.700 0.700:\\\n\n");
float minX=Min(*x_e,n);
float minY=Min(*y_n,n);
float maxX=Max(*x_e,n);
float maxY=Max(*y_n,n);

float x[3],y[3],z[3],co1,co2,co3;
double xp=-52.8,yp=1202.6;
float zz[200][200];
float dx=(2859.4+52.8)/200;//(maxX-minX)/300;
float dy=(3400.5-1202.6)/200;//(maxY-minY)/300;
int i=0,j=0;
for (i=0; i<200; i++){
        for (j=0; j<200; j++){
yp+=dy;
zz[i][j]=MinH(yp,xp);/*
if (fabs(zz[i][j])>490)
zz[i][j]=-yp/10;
if (fabs(zz[i][j])<290)
zz[i][j]=-xp/10;*/
//fprintf(pFile1,"\n%6.2f %6.2f %6.2f ",xp,yp,zz[i][j]);
}
yp=1202.6;
xp+=dx;
//fprintf(pFile1,"\n");
}
float minZ=-486;//Min(*zz,10000);
float maxZ=-420.02;//Max(*zz,10000);
xp=-52.8;
yp=1202.6;
float h;
for (i=0; i<199; i++){
        for (j=0; j<199; j++){
x[0]=xp;
x[1]=xp+dx;
x[2]=xp+dx;
y[0]=yp;
y[1]=yp;
y[2]=yp+dy;
z[0]=zz[i][j];
z[1]=zz[i+1][j];
z[2]=zz[i+1][j+1];
h=fabs(z[0]-minZ)/fabs(maxZ-minZ);
if (h<0.125){
co1=0.0;
co2=1-8*h;
co3=8*h;
}
if (h<0.25 && h>0.125){
co1=2.0*(h-0.125);
co2=0.0;
co3=1.0-2.0*(h-0.125);
}
if (h>0.25 && h<0.375){
co1=0.25+6.0*(h-0.25);
co2=0.0;
co3=0.75+2.0*(h-0.25);
}
if (h<0.5 && h>0.375){
co1=1.0;
co2=0.0;
co3=1.0-8.0*(h-0.375);
}
if (h>0.5 && h<0.625){
co1=1.0;
co2=4.0*(h-0.5);
```

```
co3=2.0*(h-0.5);
}
if (h<0.75 && h>0.625){
co1=1.0;
co2=0.5+4.0*(h-0.625);
co3=0.25-2.0*(h-0.625);
}
if (h>0.75 && h<0.875){
co1=1.0;
co2=1.0;
co3=8.0*(h-0.75);
}
if (h>0.875){
co1=1.0;
co2=1.0;
co3=1.0;
}
fprintf(pFile1,"\n :co# %6.2f  %6.2f  %6.2f:\\\n :pt#  %6.2f  %6.2f  %6.2f :pt#...
  %6.2f  %6.2f  %6.2f:\\ \n :pt#  %6.2f  %6.2f  %6.2f :cl:\\ ",co1,co2,co3,x[0],...
y[0],z[0],x[1],y[1],z[1],x[2],y[2],z[2]);
x[0]=xp;
x[1]=xp;
x[2]=xp+dx;
y[0]=yp;
y[1]=yp+dy;
y[2]=yp+dy;
z[0]=zz[i][j];
z[1]=zz[i][j+1];
z[2]=zz[i+1][j+1];
h=fabs(z[0]-minZ)/fabs(maxZ-minZ);
if (h<0.125){
co1=0.0;
co2=1-8*h;
co3=8*h;
```

```
}
if (h<0.25 && h>0.125){
co1=2.0*(h-0.125);
co2=0.0;
co3=1.0-2.0*(h-0.125);
}
if (h>0.25 && h<0.375){
co1=0.25+6.0*(h-0.25);
co2=0.0;
co3=0.75+2.0*(h-0.25);
}
if (h<0.5 && h>0.375){
co1=1.0;
co2=0.0;
co3=1.0-8.0*(h-0.375);
}
if (h>0.5 && h<0.625){
co1=1.0;
co2=4.0*(h-0.5);
co3=2.0*(h-0.5);
}
if (h<0.75 && h>0.625){
co1=1.0;
co2=0.5+4.0*(h-0.625);
co3=0.25-2.0*(h-0.625);
}
if (h>0.75 && h<0.875){
co1=1.0;
co2=1.0;
co3=8.0*(h-0.75);
}
if (h>0.875){
co1=1.0;
co2=1.0;
```

```
co3=1.0;
}
fprintf(pFile1,"\n :co# %6.2f  %6.2f  %6.2f:\\\n :pt#  %6.2f  %6.2f  %6.2f :pt#...
  %6.2f  %6.2f  %6.2f:\\ \n :pt#  %6.2f  %6.2f  %6.2f :cl:\\ ",co1,co2,co3,...
x[0],y[0],z[0],x[1],y[1],z[1],x[2],y[2],z[2]);
yp+=dy;
}
yp=1202.6;
xp+=dx;
}

//brent's data
float wpLat[7],wpLon[7],wpAlt=260;
wpLat[0]=39.776000;
wpLat[1]=39.776000;
wpLat[2]=39.776000;

wpLat[3]=39.776000;
wpLat[4]=39.776000;
wpLat[5]=39.776000;
wpLat[6]=39.776000;

wpLon[0]=-84.117796;
wpLon[1]=-84.103704;
wpLon[2]=-84.090613;

wpLon[3]=-84.117796;
wpLon[4]=-84.117796;
wpLon[5]=-84.117796;
wpLon[6]=-84.117796;
ENUCoord WAYPOINT[7];
for(i=0; i<7; i++){
WAYPOINT[i].lla2enu(wpLat[i]*deg_to_rad, wpLon[i]*deg_to_rad, wpAlt, Base_X,...
 Base_Y, Base_Z);
```

```
}
double ybase=WAYPOINT[0].GetNorth();
double xbase=WAYPOINT[0].GetEast();
double zbase=WAYPOINT[0].GetUp();
fprintf(pFile1,"\n :co# 1.0 0.5 0.25:\\");//ORANGE
for(i=0;i<7;i++){
double ybase=WAYPOINT[i].GetNorth();
double xbase=WAYPOINT[i].GetEast();
double zbase=WAYPOINT[i].GetUp();
fprintf(pFile1,"\n :pt#  %6.2f %6.2f  %6.2f:pt#  %6.2f %6.2f  %6.2f:\\\n :pt#...
  %6.2f %6.2f  %6.2f:pt#  %6.2f %6.2f  %6.2f:cl:\\",xbase,ybase,zbase,xbase,...
ybase,zbase+30,xbase+30,ybase,zbase+30,xbase+30,ybase,zbase);//right
fprintf(pFile1,"\n :pt#  %6.2f %6.2f  %6.2f:pt#  %6.2f %6.2f  %6.2f:\\\n :pt#...
  %6.2f %6.2f  %6.2f:pt#  %6.2f %6.2f  %6.2f:cl:\\",xbase,ybase,zbase,xbase,...
ybase-30,zbase,xbase,ybase-30,zbase+30,xbase,ybase,zbase+30);//back
fprintf(pFile1,"\n :pt#  %6.2f %6.2f  %6.2f:pt#  %6.2f %6.2f  %6.2f:\\\n :pt#...
  %6.2f %6.2f  %6.2f:pt#  %6.2f %6.2f  %6.2f:cl:\\",xbase,ybase-30,zbase,xbase,...
ybase-30,zbase+30,xbase+30,ybase-30,zbase+30,xbase+30,ybase-30,zbase);//left
fprintf(pFile1,"\n :pt#  %6.2f %6.2f  %6.2f:pt#  %6.2f %6.2f  %6.2f:\\\n :pt#...
  %6.2f %6.2f  %6.2f:pt#  %6.2f %6.2f  %6.2f:cl:\\",xbase+30,ybase-30,zbase,xbase...
+30,ybase-30,zbase+30,xbase+30,ybase,zbase+30,xbase+30,ybase,zbase);//front
fprintf(pFile1,"\n :pt#  %6.2f %6.2f  %6.2f:pt#  %6.2f %6.2f  %6.2f:\\\n :pt#...
  %6.2f %6.2f  %6.2f:pt#  %6.2f %6.2f  %6.2f:cl:\\",xbase,ybase,zbase+30,xbase,...
ybase-30,zbase+30,xbase+30,ybase-30,zbase+30,xbase+30,ybase,zbase+30);//top
fprintf(pFile1,"\n :pt#  %6.2f %6.2f  %6.2f:pt#  %6.2f %6.2f  %6.2f:\\\n :pt#...
  %6.2f %6.2f  %6.2f:pt#  %6.2f %6.2f  %6.2f:cl:\\\n",xbase,ybase,zbase,xbase,...
ybase-30,zbase,xbase+30,ybase-30,zbase,xbase+30,ybase,zbase);//bottom
}
fclose (pFile1);
printf("\nterrain.txt file printed successfully");
Sleep(2000);
}//printTerrain

int RetrieveLoop(){
```

```
double x_lon[10], y_lat[10], h_alt;
//Figure out how to decode waypoint list packet, store it in vectors x_lon, y_lat,
// and altitude h
//int mtemp = sizeof(//decoded x loop)/sizeof(double);  //# of points in loop
int mtemp=6,i,j;
printf("\nNumber of Points:");
while (1>0){
if (i==13)
break;
if (kbhit()){
i=getche();
if (i!=13){
mtemp=(i-48);
}
}
}
i=0;
printf("\nAlt(100m<alt<999m):",j+1);
double mult=100,alt=0;
while (1>0){
if (i==13)
break;
if (kbhit()){
i=getche();
if (i!=13){
alt+=mult*(i-48);
mult=mult/10;
if (mult==0.01)
printf(".");
}
}
}
h_alt=alt;
i=0;
```

```
for(j=0; j<mtemp; j++){
mult=10;
double lat=0,lon=0,alt=0;
printf("\nLAT(%i):",j+1);
while (1>0){
if (i==13)
break;
if (kbhit()){
i=getche();
if (i!=13){
lat+=mult*(i-48);
mult=mult/10;
if (mult==0.1)
printf(".");
}
}
}
i=0;
mult=10;
printf("\nLON(%i):-",j+1);
while (1>0){
if (i==13)
break;
if (kbhit()){
i=getche();
if (i!=13){
lon+=mult*(i-48);
mult=mult/10;
if (mult==0.1)
printf(".");
}
}
}
i=0;
y_lat[j]=lat;
x_lon[j]=-lon;
}
ENUCoord LoopENU[10];
for (i=0; i<mtemp; i++){
LoopENU[i].lla2enu(y_lat[i]*3.1415926/180, x_lon[i]*3.1415926/180, h_alt, ...
Base_X, Base_Y, Base_Z);
}
for(i=0; i<mtemp; i++){
xtemp_e[i]=LoopENU[i].GetEast();
ytemp_n[i]=LoopENU[i].GetNorth();
}
htemp=LoopENU[0].GetUp();

return mtemp;
}

void NewNetwork(UInt16 NetworkID, void* Parameter) {

}
//Needed to Initialize Networks



//Looks for and gleans data from an autopilot packet sent from a network
void LookForAutopilotData(QType* pQ, int whosData)
{
    static AutopilotPkt_t APPkts[10];
static AutopilotCmd_t Cmd[10];
double mins, hours;
UInt32 i, NumNets;
SInt32 ID;

//look at how many networks m_pComm can see
NumNets = m_pComm->GetNumNets();
```

```
for(i = 0; i < NumNets; i++)
{
// Don't display past 10 networks since we didn't include the space
if(i >= 10) break;

ID = m_pComm->GetIDFromIndex(i);

pQ = m_pComm->GetStreamRxBuffer((UInt16)ID, AUTOPILOT_STREAM);

if(!pQ) continue;

// Now check to see if a packet exists.  Note!!! The raw packet
//   structure MUST persist between calls, and it MUST be unique to this
//   network.
if(LookForAutopilotPacket(pQ, &(APPkts[i])))
{
switch(APPkts[i].PktType)
{
case TELEMETRY:
UserData_t telemData;
DecodeTelemetryPacket(&(APPkts[i]), &(telemData));
//update telemtry struct
current_telemetry[i].Longitude = telemData.GPS.Longitude * 180.0 / 3.1415926;
current_telemetry[i].Latitude = telemData.GPS.Latitude * 180.0 / 3.1415926;
//printf("\ndx %f",current_telemetry[i].Latitude);
current_telemetry[i].Altitude = telemData.GPS.Altitude;
current_telemetry[i].Velocity = telemData.GPS.Speed;

//convert lla data to enu
PosENU.lla2enu(current_telemetry[i].Latitude *3.1415926/180,
   current_telemetry[i].Longitude *3.1415926/180,
   current_telemetry[i].Altitude,
   Base_X, Base_Y, Base_Z);
```

```
current_telemetry[i].East = PosENU.GetEast();
current_telemetry[i].North = PosENU.GetNorth();
//ctu_old=current_telemetry[i].Up;
current_telemetry[i].Up = PosENU.GetUp();
current_telemetry[i].Hours = telemData.GPS.hours;
current_telemetry[i].Minutes = telemData.GPS.minutes;
current_telemetry[i].Seconds = telemData.GPS.seconds;
if (stoo[i]==true){
if (firsttime[i]==false){
init_AC(current_telemetry[i].Latitude,current_telemetry[i].Longitude,...
current_telemetry[i].Altitude, i);
}
send_telemetry(current_telemetry[i].East,current_telemetry[i].North,...
current_telemetry[i].Up, i);
}
//display the data
displayData(whosData, NumNets);
break;
case CONTROL_DATA:
UserData_t controlData;
float gyroBias[3], controls[10];
DecodeControlDataPacket(&(APPkts[i]), &(controlData), gyroBias, controls);
//update telemetry struct
current_control[i].phi = controlData.Euler[0] * 180/3.1415926;
current_control[i].theta = controlData.Euler[1] * 180/3.1415926;
current_control[i].psi = controlData.Euler[2] * 180/3.1415926;
current_control[i].RollRate = controlData.Gyro[0] * 180/3.1415926;
current_control[i].PitchRate = controlData.Gyro[1] * 180/3.1415926;
current_control[i].YawRate = controlData.Gyro[2] * 180/3.1415926;
current_control[i].AirSpeed = controlData.TAS;
current_control[i].Pdynamic = controlData.Pdynamic;
current_control[i].MagHeading = controlData.MagHeading*180/3.1415926;

current_control[i].Aileron = controls[0] * 180/3.1415926;
```

```
current_control[i].Elevator = controls[1] * 180/3.1415926;
current_control[i].Throttle = controls[2];
current_control[i].Rudder = controls[3] * 180/3.1415926;
//convert GPS seconds into hours, minutes, and seconds
hours = controlData.SystemTime / 3600000.0;
current_control[i].Hours = hours;
mins = (hours - (double)current_control[i].Hours) * 60;
current_control[i].Minutes = mins;
time_old=current_telemetry[i].Seconds;
current_control[i].Seconds = (mins - (double)current_control[i].Minutes) * 60;
if (stoo[i]==true){
if (firsttime[i]==true)
send_control(current_control[i].phi,current_control[i].theta,current_control[i].psi, i);
}
displayData(whosData, NumNets);
break;
case WAYPOINT:
break;
case WAYPOINT_LIST:/*
printf("\nppppppppppppp:");
Sleep(10000);
FPMask_t wplistMask;
DecodeWaypointListPacket(&(APPkts[i]), wplistMask);
printf("\nppppppppppppp:%i",wplistMask);
Sleep(10000);*/
break;
case TRACK:
break;
case AUTOPILOT_COMMAND:
AutopilotCmd_t Cmd[3];
Waypoint_cmd[i] = DecodeAutopilotControlPacket(&(APPkts[i]), &Cmd[i]);
//This returns Waypoint_cmd[i] as the waypoint the Piccolo is currently heading towards
//displayData(whosData, NumNets);
break;
```

```
}
}
}
}
// LookForAutopilotData

void main()
{
int i = 0, whosData = 0;
pFile2 = fopen ("hbdata.txt","w");
pFile3 = fopen ("altdata.txt","w");
acFile1 = fopen ("multi.txt","w");
m_pComm = new CCommManager(0, 57600, "1.1.1.3:2000", 0);

if(m_pComm->GetLastError() != 0){
printf("%s", m_pComm->GetLastError());
printf("\n");
i = 1;
}

m_pComm->SetNewNetworkCallBack(NewNetwork, m_pComm);
char keypress;

while(m_pComm && i == 0)
{
m_pComm->RunNetwork();
int NumNets=m_pComm->GetNumNets();
LookForAutopilotData(pQ, whosData);

if (kbhit()){

keypress = getch(); //get commands via keypress

switch(keypress)
```

```
{
case 'x':
i = 1;
printf("\n");
fclose (pFile2);
fclose (pFile3);
fclose (acFile1);
break;
case 't': //print telemetry data for selected network
data=0;
break;
case 'a': //print telemetry data for selected network
data=1;
break;
case 'b': //building
data=2;
case 'h': //hard boundary
data=3;
break;
case 'l': //min altitude loop
data=4;
break;
case 'c': //clear information->n=0;
data=5;
break;
case 'p': //print terrain info for AVDS
data=6;
break;
case 's': //toggle streaming to AVDS
data=7;
break;
case 'o': //return to options
data=8;
break;
```

```
case ’w’: //turn on waypoints
data=9;
break;
case ’r’: //turn on recording
data=10;
break;
case ’i’: //turn on image
data=11;
break;

case ’1’: //print telemetry data for first Network
whosData = 0;
break;
case ’2’: //print telemetry data for second Network
whosData = 1;
break;
case ’3’: //print telemetry data for third Network
whosData = 2;
break;
case ’4’: //print telemetry data for fourth Network
whosData = 3;
break;
case ’5’: //print telemetry data for fifth Network
whosData = 4;
break;
case ’6’: //print telemetry data for sixth Network
whosData = 5;
break;
case ’7’: //print telemetry data for seventh Network
whosData = 6;
break;
case ’8’: //print telemetry data for eighth Network
whosData = 7;
break;
```

```
case '9': //print telemetry data for ninth Network
whosData = 8;
break;
case '0': //print telemetry data for tenth Network
whosData = 9;
break;
}
}
//Sleep(10);
if (pboo==true){
if (PBtime<current_telemetry[i].Minutes*60+current_telemetry[i].Seconds){
if (timeStart>PBtime && PBtime>0.02){
timeStart=PBtime;
}
double tnew=PBtime-timeStart;
if (tnew<0){
tnew=0;
}
PBtime=current_telemetry[i].Minutes*60+current_telemetry[i].Seconds;
fprintf(pbFile1,"\n%f %f %f %f %f %f %f 17",tnew, ground.pos_X,ground.pos_Y,...
ground.pos_Z+500,ground.rot_X,ground.rot_Y,ground.rot_Z); //terrain
fprintf(pbFile2,"\n%f %f %f %f %f %f %f 18 %f %f %f %f",tnew, ac[0].pos_X,...
ac[0].pos_Y,ac[0].pos_Z+500,ac[0].rot_X,ac[0].rot_Y,ac[0].rot_Z,ac[0].rud,ac[0].elev,ac[0].ail,-ac[0
//terrain
}
}
}

CloseAVDSNetwork();

}
//Main
```

```
void init_AC(float initlat, float initlon, float initalt, int i){
float fLon,fLat,fAlt,acLon,acLat,acAlt;
int iCount = 1;
char* strAddress[1];
strAddress[0] = strdup("224.0.5.20:2267");/* this is the default AVDS multicast address*/
OpenAVDSNetwork(iCount,strAddress);
fLon=-84.109666;fLat=39.773633;fAlt=350.0;
acLon=-84.109666; acLat=39.773633; acAlt=0.0;
int idnumb[10]={1,2,3,4,5,6,7,8,9,10};
Init_Aircraft(&ac[i],"Sig Rascal 110",1,AIRCRAFT_TYPE_17,&acLon,&acLat,&acAlt);
Init_Aircraft(&ground,"Terrain",2,AIRCRAFT_TYPE_16,&fLon,&fLat,&fAlt);
ac[i].rot_X = 0.0;
ac[i].rot_Y = 0.0;
ac[i].rot_Z = 0.0;
ground.rot_X = 0.0;
ground.rot_Y = 0.0;
ground.rot_Z = 180.0;

ENUCoord GroundENU;
GroundENU.lla2enu(fLat*deg_to_rad, fLon*deg_to_rad, fAlt, Base_X, Base_Y, Base_Z);
yground=GroundENU.GetEast();
xground=GroundENU.GetNorth();
zground=GroundENU.GetUp();
ground.pos_X = xground;
ground.pos_Y = yground;
ground.pos_Z = zground;

ENUCoord StartENU;
StartENU.lla2enu(acLat*deg_to_rad, acLon*deg_to_rad, acAlt, Base_X, Base_Y, Base_Z);
ystart=StartENU.GetEast();
xstart=StartENU.GetNorth();
zstart=StartENU.GetUp();

int idnumbs[10];
```

```
idnumbs[0] = 1234;
idnumbs[1] = 5423;
idnumbs[2] = 9821;
idnumbs[3] = 6243;
idnumbs[4] = 5678;
idnumbs[5] = 3498;
idnumbs[6] = 1235;
idnumbs[7] = 5424;
idnumbs[8] = 9822;
idnumbs[9] = 6244;
ac[i].packet.craftid = idnumbs[i]; // must be unique for each aircraft
ac[i].packet.hostid = idnumbs[i];


ground.packet.craftid = 7125; // must be unique for each aircraft
ground.packet.hostid = 7125;
ac[i].dt = 1.0;
ground.dt = 1.0;
firsttime[i]=true;
}

void send_telemetry(float east, float north, float up, int i)
{
ac[i].pos_X =-east;//-north; //converts from delta_N (m) to delta X (ft)
ac[i].pos_Y =-north;//east;
ac[i].pos_Z =up+474.840818;//up;
Put_Aircraft(&ac[i]);
Put_Aircraft(&ground);

}
void send_control(float phi, float theta, float psi, int i)
{
ac[i].rot_X = phi;//theta;
ac[i].rot_Y = theta;//phi;
```

```
ac[i].rot_Z = current_control[i].MagHeading+90;
ac[i].ail = current_control[0].Aileron*3.1415926/20;
ac[i].elev = current_control[0].Elevator*3.1415926/20;
ac[i].rud = current_control[0].Rudder*3.1415926/20;
Put_Aircraft(&ac[i]);
Put_Aircraft(&ground);
}
```

```
//clears the screen
void clrscr() {
  HANDLE hStdOut = GetStdHandle(STD_OUTPUT_HANDLE);
  COORD coord = {0, 0};
  DWORD count;
  CONSOLE_SCREEN_BUFFER_INFO csbi;
  GetConsoleScreenBufferInfo(hStdOut, &csbi);
  FillConsoleOutputCharacter(hStdOut, ' ', csbi.dwSize.X * csbi.dwSize.Y, coord, &count);
  SetConsoleCursorPosition(hStdOut, coord);
}
//clears the screen
```

Appendix C. Sample Craft File

```
Sig| Sig Rascal 110:\



# angle between views
:av#30.0:\

# control type
:cn=pilot:\
# pilot’s position
:pp# 0.00  0.0  0.0:\

#  weight - used for crash detection
:we#500.0:\
#  Ixx - used for crash detection
:ix#122200.0:\
#  Iyy - used for crash detection
:iy#25000.0:\
#  Izz - used for crash detection
:iz#139800.0:\

# center of mass
:cm# 0.00   0.0   0.00:\
# exhaust position
# :ep# 0.00  0.00  0.0:\



# Distance > 0 ft
:d0:\
:DS#1:\



#texture files
```

```
:XF=Sunset.bmp:\
:XF=Sunset.bmp:\

# port wing tip
# :Pw# -14.00  -8.33  0.39:\
# starboard wing tip
# :Sw# 14.00  -8.33  0.39:\

#
#

#from above
:ca=fuselage:\
:co# 0.0  1.0  0.0:\
:pt#  -20.0  -10.0   100.0:pt#   -20.0   10.0   100.0:\
:pt#  20.0   0.0   100.0:cl:\



# wing
:ca=fuselage:\
:co# 1.0  1.0 1.0:\
:pt#  -0.4  -0.50   0.30:pt#   -0.4   0.5   0.30:\
:pt#  -0.47   0.5   0.30:pt#  -0.47  -0.50   0.30:cl:\
#select texture file
:XI#1:\
#texture coordinates
:XC#1.0 0.0 0.0 :XC#1.0 -1.0 0.0 :XC#0.0 -1.0 0.0 :XC#0.0 0.0 0.0:\
:pt#  0.0  0   0.30:pt#   0.0   1.4   0.30:\
:pt#  -0.1   1.4   0.35:pt#  -0.1  0   0.35:cl:\
#select texture file
:XI#1:\
#texture coordinates
:XC#1.0 0.0 0.0 :XC#1.0 -1.0 0.0 :XC#0.0 -1.0 0.0 :XC#0.0 0.0 0.0:\
:pt#  0.0  0   0.30:pt#   0.0   -1.4   0.30:\
:pt#  -0.1   -1.4   0.35:pt#  -0.1  0   0.35:cl:\
#select texture file
:XI#1:\
#texture coordinates
:XC#1.0 0.0 0.0 :XC#1.0 -1.0 0.0 :XC#0.0 -1.0 0.0 :XC#0.0 0.0 0.0:\
:pt#  -0.4  0.0   0.30:pt#   -0.4   1.4   0.30:\
:pt#  -0.1   1.4   0.35:pt#  -0.1  0.0   0.35:cl:\
#select texture file
:XI#1:\
#texture coordinates
:XC#1.0 0.0 0.0 :XC#1.0 -1.0 0.0 :XC#0.0 -1.0 0.0 :XC#0.0 0.0 0.0:\
:pt#  -0.4  0.0   0.30:pt#   -0.4   -1.4   0.30:\
:pt#  -0.1   -1.4   0.35:pt#  -0.1  0.0   0.35:cl:\


:co# 0.9  0.9 0.9:\
#select texture file
:XI#1:\
#texture coordinates
:XC#1.0 0.0 0.0 :XC#1.0 1.0 0.0 :XC#0.0 1.0 0.0 :XC#0.0 0.0 0.0:\
:pt#  0.0  -1.40   0.30:pt#   0.0   0.0  0.30:\
:pt#  -0.4   0.0   0.30:pt#  -0.4  -1.40   0.30:cl:\


#select texture file
:XI#1:\
#texture coordinates
:XC#1.0 0.0 0.0 :XC#1.0 1.0 0.0 :XC#0.0 1.0 0.0 :XC#0.0 0.0 0.0:\
:pt#  0.0  1.40   0.30:pt#   0.0   0.0   0.30:\
:pt#  -0.4   0.0   0.30:pt#  -0.4  1.40   0.30:cl:\


:co# 0.8  0.8 0.8:\
:pt#  0.0  1.4   0.30:pt#   -0.1   1.4   0.35:\
:pt#  -0.1   1.6   0.3:cl:\
:pt#  -0.3  1.6   0.30:pt#   -0.1   1.4   0.35:\
:pt#  -0.1   1.6   0.3:cl:\
:pt#  -0.3  1.6   0.30:pt#   -0.1   1.4   0.35:\
:pt#  -0.47   1.4   0.3:cl:\


:pt#  0.0  -1.4   0.30:pt#   -0.1   -1.4   0.35:\
:pt#  -0.1   -1.6   0.3:cl:\
:pt#  -0.3  -1.6   0.30:pt#   -0.1   -1.4   0.35:\
:pt#  -0.1   -1.6   0.3:cl:\
:pt#  -0.3  -1.6   0.30:pt#   -0.1   -1.4   0.35:\
:pt#  -0.47   -1.4   0.3:cl:\

:co# 1.0  1.0 1.0:\

:pt#  0.0  1.40   0.30:pt#   -0.1   1.6   0.30:\
:pt#  -0.3   1.6   0.30:pt#  -0.47  1.40   0.30:cl:\

:pt#  0.0  -1.40   0.30:pt#   -0.1   -1.6   0.30:\
:pt#  -0.3   -1.6   0.30:pt#  -0.47  -1.40   0.30:cl:\




# ailerons
:ca=articulated-surface-06:\
:ra#  -.4   0.5  0.3 -.4 1.4 0.3:\
:co# 1.0  1.0  1.0:\
:pt#  -0.47  1.40   0.30:pt#   -0.47   0.5   0.30:\
:pt#  -0.4   0.5   0.30:pt#  -0.4  1.40   0.30:cl:\

:ca=articulated-surface-03:\
:ra#  -.4   0.5  0.3 -.4 1.4 0.3:\
:pt#  -0.47  -1.40   0.30:pt#   -0.47   -0.5   0.30:\
:pt#  -0.4   -0.5   0.30:pt#  -0.4  -1.40   0.30:cl:\



# horizontal tail
:ca=fuselage:\
:co# 1.0  1.0  1.0:\
:pt#  -1.30  -0.50   0.30:pt#   -1.5   -0.5   0.30:\
:pt#  -1.5   0.5   0.30:pt#  -1.3  0.50   0.30:cl:\
:co# 0.0  1.0  0.0:\
:pt#  -1.3  0.48  0.299:pt#   -1.5  0.48   0.299:\
:pt#  -1.5 0.44   0.299:pt#  -1.3  0.44  0.299:cl:\



# elevator
:ca=articulated-surface-02:\
:ra#   -1.5   0.5   0.30  -1.5  -0.50   0.30:\
:co# 1.0  0.4  0.20:\
:pt#  -1.57  -0.50   0.30:pt#   -1.57   0.5   0.30:\
:pt#  -1.5   0.5   0.30:pt#  -1.5  -0.50   0.30:cl:\

# vertical tail
:ca=fuselage:\
:co# 1.0  1.0  1.0:\
:pt#  -1.24  0.0   0.30:pt#   -1.5   0.0   0.58:\
:pt#  -1.5   0.0   0.3:cl:\

# rudder
:ca=articulated-surface-01:\
:ra# -1.5  0.0   0.2  -1.5   0.0   0.58 :\
:co# 1.0  0.4  0.20:\
:pt#  -1.5  0.0   0.2:pt#   -1.6   0.0   0.25:\
:pt#   -1.6   0.0   0.35:pt#  -1.5   0.0   0.58:cl:\

# fuselage
:ca=fuselage:\
:co# 0.8  0.8  0.8:\

:pt#  0.0  -0.07   0.0:pt#   0.0 0.07 0.0:\
:pt#  -1.5   0.0   0.2:cl:\
:co# 1.0  1.0  1.0:\
:pt#  0.0  -0.07   0.3:pt#   0.0 0.07 0.3:\
:pt#  -1.5   0.0   0.3:cl:\

:co# 0.9  0.9  0.9:\

#select texture file
:XI#0:\
#texture coordinates
:XC#0.0 1.0 0.0 :XC#1.0 1.0 0.0 :XC#1.0 0.0 0.0 :XC#0.0 0.0 0.0:\
:pt#  0.0  -0.07   0.0:pt#   0.0 -0.07 0.3:\
:pt#  -1.5   0.0   0.3:pt#  -1.5  0.0  0.2:cl:\
#select texture file
:XI#0:\
#texture coordinates
:XC#0.0 1.0 0.0 :XC#1.0 1.0 0.0 :XC#1.0 0.0 0.0 :XC#0.0 0.0 0.0:\
:pt#  0.0   0.07   0.0:pt#   0.0 0.07 0.3:\
:pt#  -1.5   0.0   0.3:pt#  -1.5  0.0  0.2:cl:\



:co# 0.8  0.8  0.8:\
:pt#  0.0  -0.07   0.0:pt#   0.3 -0.05 0.065:\
:pt#  0.3   0.05 0.065:pt#  0.0  0.07  0.0:cl:\
:pt#  0.4  0.0   0.15:pt#   0.3 -0.05 0.065:\
:pt#  0.3  0.05 0.065:cl:\
:co# 1.0  1.0  1.0:\
:pt#  0.1  -0.07   0.235:pt#   0.3 -0.05 0.235:\
:pt#  0.3   0.05 0.235:pt#  0.1  0.07  0.235:cl:\
:pt#  0.4  0.0   0.15:pt#   0.3 -0.05 0.235:\
:pt#  0.3  0.05 0.235:cl:\


:co# 0.9  0.9  0.9:\
#select texture file
:XI#0:\
#texture coordinates
:XC#-1.0 0.0 0.0 :XC#-1.0 -1.0 0.0 :XC#0.0 -1.0 0.0 :XC#0.0 0.0 0.0:\
:pt#  0.0  -0.07   0.0:pt#   0.3 -0.05 0.065:\
:pt#  0.3  -0.05 0.235:pt#  0.0  -0.07  0.235:cl:\
#select texture file
:XI#0:\
#texture coordinates
:XC#-1.0 0.0 0.0 :XC#-1.0 -1.0 0.0 :XC#0.0 -1.0 0.0 :XC#0.0 0.0 0.0:\
:pt#  0.0   0.07   0.0:pt#   0.3 0.05 0.065:\
:pt#  0.3   0.05 0.235:pt#  0.0  0.07  0.235:cl:\

:pt#  0.4  0.0   0.15:pt#   0.3 -0.05 0.065:\
:pt#  0.3  -0.05 0.235:cl:\
:pt#  0.4  0.0   0.15:pt#   0.3 0.05 0.065:\
:pt#  0.3  0.05 0.235:cl:\



# Gear
:ca=fuselage:\


:co# 1.0 0.5 0.25:\
:pt#  0.00 0.2 -0.1:pt# 0.00 0.22 -0.1:\
:pt#  0.08 0.21 -0.12:cl:\
:pt#  0.00 0.2 -0.16:pt# 0.00 0.22 -0.16:\
:pt#  0.08 0.21 -0.12:cl:\
:pt#  0.00 0.22 -0.1:pt# 0.00 0.22 -0.16:\
:pt#  0.08 0.21 -0.12:cl:\
:pt#  0.00 0.20 -0.1:pt# 0.00 0.20 -0.16:\
:pt#  0.08 0.21 -0.12:cl:\
:pt#  0.00 0.2 -0.1:pt# 0.00 0.22 -0.1:\
:pt#  -0.1 0.21 -0.12:cl:\

:pt#  0.00 -0.2 -0.1:pt# 0.00 -0.22 -0.1:\
:pt#  0.08 -0.21 -0.12:cl:\
:pt#  0.00 -0.2 -0.16:pt# 0.00 -0.22 -0.16:\
:pt#  0.08 -0.21 -0.12:cl:\
:pt#  0.00 -0.22 -0.1:pt# 0.00 -0.22 -0.16:\
:pt#  0.08 -0.21 -0.12:cl:\
:pt#  0.00 -0.20 -0.1:pt# 0.00 -0.20 -0.16:\
:pt#  0.08 -0.21 -0.12:cl:\
:pt#  0.00 -0.2 -0.1:pt# 0.00 -0.22 -0.1:\
:pt#  -0.1 -0.21 -0.12:cl:\

:co# 1.0 1.0 1.0:\
:pt#  0.00 0.2 -0.16:pt# 0.00 0.22 -0.16:\
:pt#  -0.1 0.21 -0.12:cl:\
:pt#  0.00 0.22 -0.1:pt# 0.00 0.22 -0.16:\
:pt#  -0.1 0.21 -0.12:cl:\
:pt#  0.00 0.2 -0.1:pt# 0.00 0.2 -0.16:\
:pt#  -0.1 0.21 -0.12:cl:\

:pt#  0.00 -0.2 -0.16:pt# 0.00 -0.22 -0.16:\
:pt#  -0.1 -0.21 -0.12:cl:\
:pt#  0.00 -0.22 -0.1:pt# 0.00 -0.22 -0.16:\
:pt#  -0.1 -0.21 -0.12:cl:\
:pt#  0.00 -0.2 -0.1:pt# 0.00 -0.2 -0.16:\
:pt#  -0.1 -0.21 -0.12:cl:\




#windshield

:co# 0.0 0.0 0.0:\
:pt#  0.0   0.07   0.3:pt#   0.0 -0.07 0.3:\
:pt#  0.1   -0.07 0.235:pt#  0.1   0.07 0.235:cl:\
:pt#  0.0   0.07   0.3:pt#   0.1  0.07 0.235:\
:pt#  0.0  0.07 0.235:cl:\
:pt#  0.0   -0.07   0.3:pt#   0.1  -0.07 0.235:\
:pt#  0.0  -0.07 0.235:cl:\
```

Appendix D. Flight Test Results

D.1 Overview

Due to unavoidable circumstances, SASVRT was not used while actually flying an air vehicle until after this thesis was written and defended. However, the results and conclusions of this flight test are hereby appended.

D.2 Flight Test

SASVRT was both tested and used as it was intended during the flight test. In order to test the system without flying outside the SRB prescribed hard boundary, a simulated hard boundary was placed within the actualy hard boundary. Similarly, a hill was simulated in order to test the minimum altitude calculation. In addition, the WPAFB Area B data was used to monitor aircraft position relative to the SRB prescribed hard boundary and estimated terrain. The quantitative information was relayed to the pilot in control in order to heighten situational awareness. Synthetic vision was used in order to visualize the aircraft during flight as well as place a simulated trail aircraft that followed the actual aircraft's position.

D.3 Results

Figures 46 and 47 depict the data taken regarding the aircraft position relative to simulated and SRB prescribed hard boundaries, respectively. The blue position markers correspond to when SASVRT indicated the aircraft was within the bounds defined by the green line. The red x's correspond to when SASVRT warned that the aircraft was outside of the bounds.

Figures 48 and 49 depict the data taking regarding the aircraft altitude relative to the estimated terrain both with and without a simulated hill in the testing area, respectively. The markers line indicates the aircraft position at a point in time, and the green markers indicate the estimated terrain at that point in time. The red x's

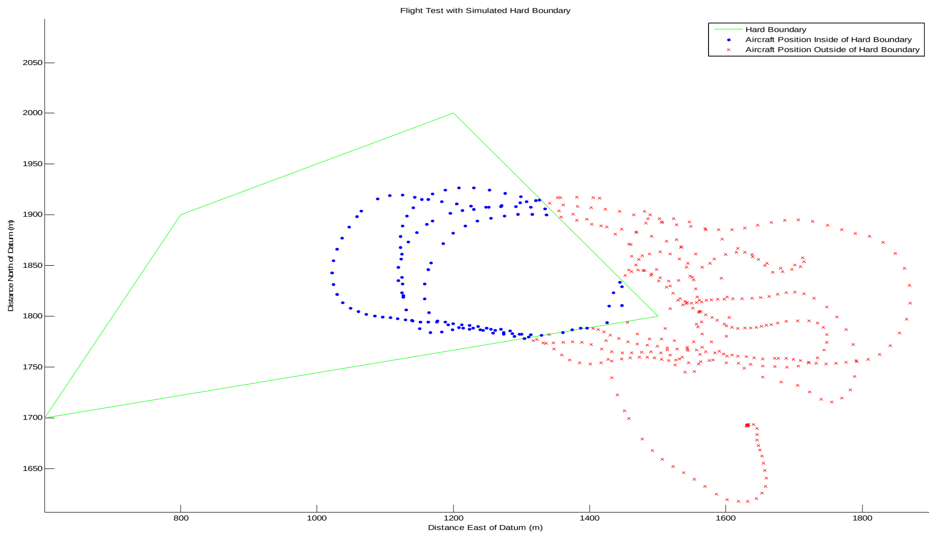


Figure 46 Flight Test of a Simulated Hard Boundary

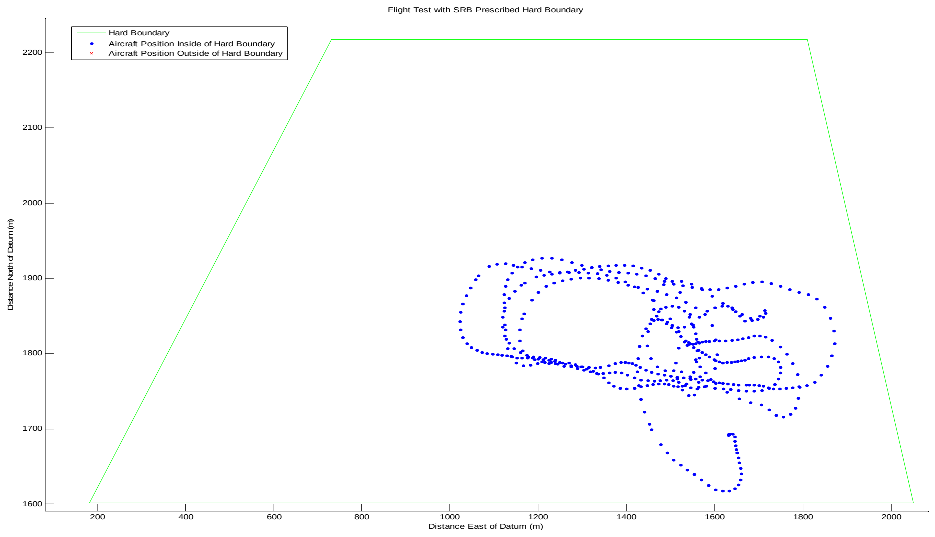


Figure 47 Flight Test of a SRB Prescribed Hard Boundary

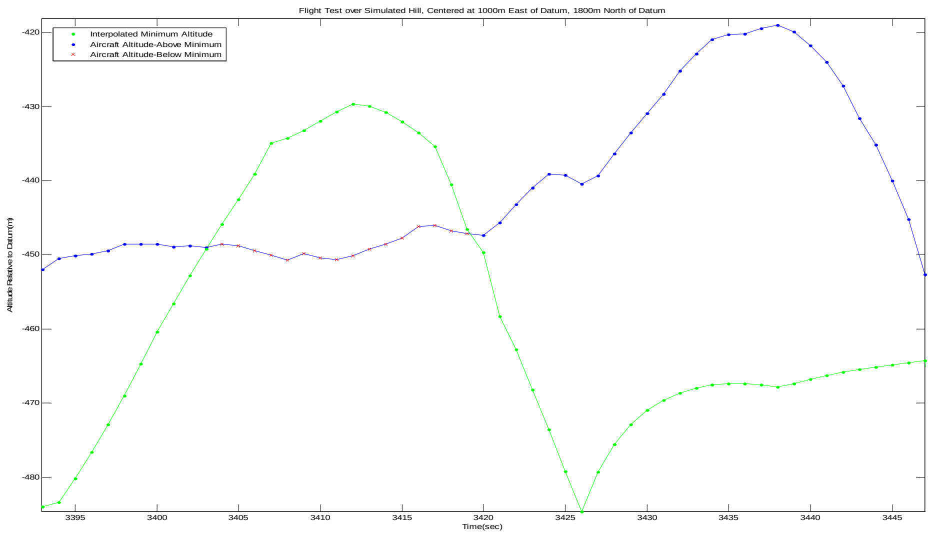


Figure 48 Clip of Aircraft Altitude Relative to Estimated Ground with Simulated Hill

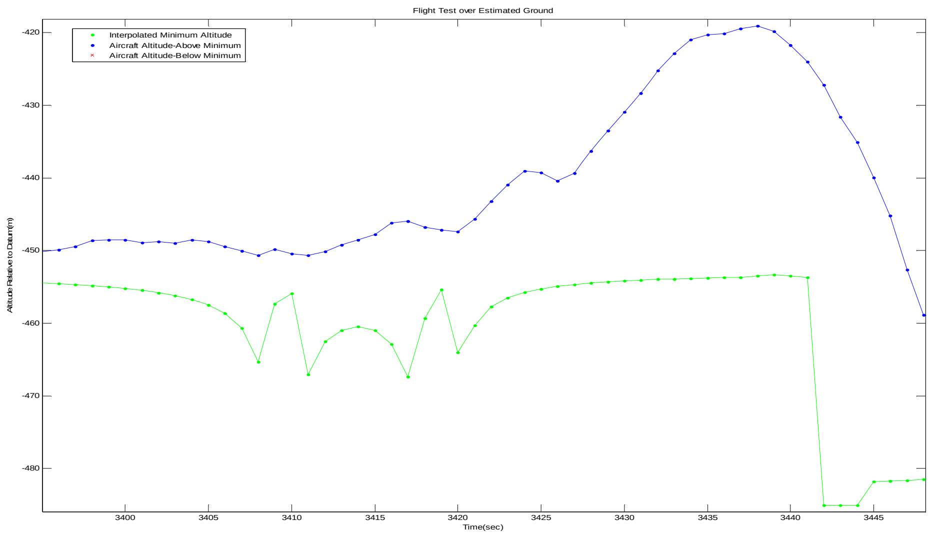


Figure 49 Clip of Aircraft Altitude Relative to Estimated Ground

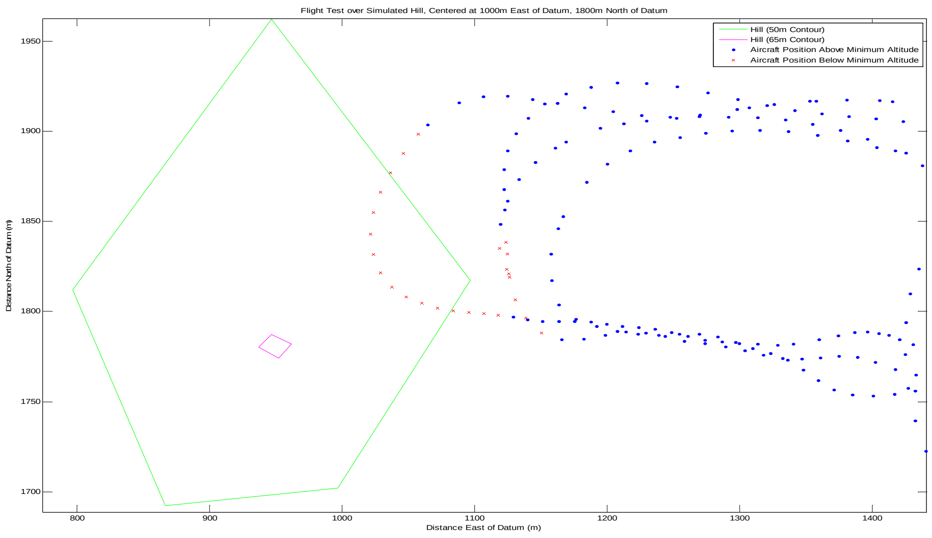


Figure 50 Aircraft Position Relative to Simulated Hill

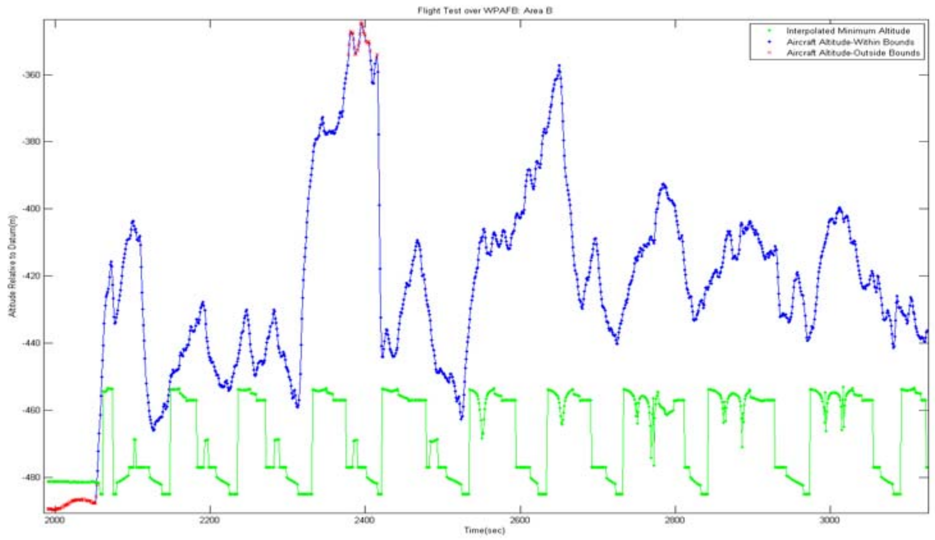


Figure 51 Aircraft Altitude Relative to Estimated Ground

correspond to when SASVRT calculated the minimum altitude to be higher than the altitude of the aircraft. The points in time when the aircraft is below the minimum altitude correspond to the points in Figure 50 when the position of the aircraft is withing the outer contour defining the simulated hill. Figure 48 is just a portion of the Figre 51 which shows the altitude of the aircraft relative to the estimated terrain over a longer duration, in which the aircraft crosses both the the minimum altitude during takeoff, and the flight ceiling.

D.4 Analysis

Figures 46 and 47 confirm that SASVRT can accurately determine whether or not the aircraft is within a closed boundary. Regarding safety, Figure 47 clearly shows that the aircraft remained within the SRB prescribed hard boundary during the flight.

Figures 48 and 49 demonstrate that at a given point in time, SASVRT can use the most pertinent altitude contour information to estimate a ground height and determine the altitude of the aircraft relative to the ground. This also shows that the estimate need not be solely based on physical constraints. Figure 50 further confirms that as expected flying over a simulated hill without increasing altitude will result in a below minimum altitude alram. As important as it is to stay above the minimum altitude, it is relatively easy for the pilot to judge how far above the ground the aircraft is. However, the maximum altitude allowed for flight testing is 400 ft, which is much harder to judge for the pilot. Figure 51 shows that SASVRT can accurately assess whether or not the aircraft is above this maximum altitude.

D.5 Conclusions

The results of the flight tests both verify SASVRT as an accurate judge of distance to boundaries and estimations of ground height, and demonstrate its applicability to the safe operations of UAVs. During flight, the pilot was unaware the aircraft had exceeded its flight ceiling until SASVRT relayed this information to him. The distance to boundary information allowed the pilot to know when to turn the aircraft around in order to not cross the SRB prescribed boundary. The synthetic vision allowed for the observers to view the aircraft during flight as well as visualize a simulated trail aircraft with respect to the lead aircraft. The heightened situational awareness was clearly evident during flight testing and the immediate impact of SASVRT on AFIT's ANT Center UAV program was undoubtedly demonstrated.

Bibliography

1. Bailey, R.E., Kramer, L.J, and Prinzel, L.J. III, *Crew and Display Concepts Evaluation for Synthetic/Enhanced Vision Systems.* NASA Langley Research Center: Meeting Presentation, 2006-05-03.

2. Breen, Barry, C. “Controlled Flight Into Terrain and the Enhanced Ground Proximity Warning System.” AlliedSignal Commerical Avionics Systems, Redmond, Washington, USA.

3. Frew, E., Xaio, X., Spry, S., McGee, T., Kim, Z., Tisdale, J., Sengupta, R., Hendric, K.J. ”Flight Demonstrations of Self-directed Collaborative Navigation of Small Unmanned Aircraft.” *Proceedings of the 2004 IEEE Aerospace Sciences Meeting and Exhibit.* Big Sky, MT, March 2004.

4. Jodeh, Nidal, M. *Development of Autonomous Unmanned Aerial Vehicle Research Platform: Modeling, Simulating, and Flight Testing.* MS Thesis, AFIT/GAE/ENY/06-M18. School of Engineering and Management, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH. March 2005.

5. Kramer, L.J, Arthur, J.J., Prinzel, L.J. III, and Williams, S.P., *Synthetic Vision Enhanced Surface Operations and Flight Procedures Rehearsal Tool.* NASA Langley Research Center: Meeting Presentation, 2006-05-04.

6. McCarthy, Patrick, A. *Characterization of UAV Performance Evaluation and Development of a Formation Flight Controller for Multiple Small UAVs.* MS Thesis, AFIT/GAE/ENY/06-J13. School of Engineering and Management, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH. June 2006.

7. Rasmussen, Steve. *AVDS User's Manual.* Rasmussen Simulation Technologies, LTD., February 2005.

8. Robinson, Brent, K. *An Investigation Into Robust Wind Correction Algorithms for Off the Shelf Unmanned Aerial Vehicle Autopilots.* MS Thesis, AFIT/GAE/ENY/06-J13. School of Engineering and Management, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH. June 2006.

9. Baleri, Giri. *Datum Transformations of NAV420 Reference Frames.* Crossbow Technology, Inc. http://www.xbow.com.

10. Vaglienti, B., Hoag, R., Niculescu, M. *Piccolo System Users Guide.* Hood River OR. Coud Cap Technology, 18 April 2005.

Vita

Ensign Joseph M. Dugan was raised in Western Michigan and graduated from Forest Hills Northern High School in 2001. He entered undergraduate studies at the University of Notre Dame where he graduated with a Bachelor of Science degree in Aerospace Engineering in May 2005. He was subsequently commissioned through the Naval ROTC Unit at the University of Notre Dame.

In June 2005, he entered the Graduate School of Engineering and Management, Air Force Institute of Technology. Upon graduation, he will be assigned to the Naval Aviation School as a student pilot in Pensacola, Florida.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)* 13–06–2006	**2. REPORT TYPE** Master's Thesis	**3. DATES COVERED** *(From — To)* Jun 2005 – Jun 2006

4. TITLE AND SUBTITLE SITUATIONAL AWARENESS AND SYNTHETIC VISION FOR UNMANNED AERIAL VEHICLE FLIGHT TESTING	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Dugan, Joseph M., Ensign, USN	**5d. PROJECT NUMBER**
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology Graduate School of Engineering and Management 2950 P Street, Building 640 WPAFB OH 45433-7765	**8. PERFORMING ORGANIZATION REPORT NUMBER** AFIT/GAE/ENY/06-J03
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)	**10. SPONSOR/MONITOR'S ACRONYM(S)**
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

APPROVAL FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The research is this thesis describes the development of a program to enhance situational awareness and provide synthetic vision during UAV flight testing in order to provide observers with the most pertinent information regarding operational safety. The program developed and tested by this research provides users with information regarding hard boundaries, interpolated terrain, and other aerial vehicles present. The hard boundary is a user defined area designated by a Safety Review Board within which the UAV is to stay at all times. Pertinent data to this boundary is both distance and time until crossing, and is provided to the user during flight by the developed program. The interpolated terrain part of the program allows the user to input contours of altitude and based on all present information, interpolates a minimum safe altitude for flight. The program relays telemetry data from the aircraft to the Aviator Visual Display Simulator, a program that provides a real-time visualization of the aircraft's position and attitude relative to a synthetic terrain.

15. SUBJECT TERMS

Synthetic Vision; Situational Awareness; Unmanned Aerial Vehicles; Aviator Visual Display Simulator; Piccolol; Autopilot

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Paul Blue, Maj, USAF (ENY)
U	U	U	UU	166	**19b. TELEPHONE NUMBER** *(include area code)* (937) 255–6565 x4714

Standard Form 298 (Rev. 8–98)
Prescribed by ANSI Std. Z39.18