Development and Testing of an Advanced Terrain Awareness and Warning System

Version III

16.621

Spring 2003

Author: Shen Qu

Advisor: R. John Hansman, Jr. Partner: Chinwe Nyenke

12 May 2003

Table of Contents

LIST OF FIGURES 3

LIST OF TABLES 3

EXECUTIVE SUMMARY 4

1.0 INTRODUCTION 5

2.0 HYPOTHESIS, OBJECTIVE, AND SUCCESS CRITERION 7

3.0 LITERATURE REVIEW 7

4.0 TECHNICAL APPROACH 10

4.1 GENERAL EXPERIMENTAL OVERVIEW 10
4.2 APPARATUS 11
4.3 TEST ARTICLES AND SUBJECTS 12
4.4 TEST MATRICES AND VARIABLES 12
4.5 ERRORS 13

5.0 EXPERIMENTAL DESIGN 14

5.1 TAWS DESIGN 14
5.1.1 Baseline 15
5.1.2 Red Contour 15
5.2 SOFTWARE 15
5.3 TEST SCENARIOS 18
5.3 TEST ENVIRONMENT 18
5.4 EXPERIMENT TESTING 18
5.5 HUMAN EXPERIMENT PROTOCOL 19
5.6 OTHERS 20

6.0 DATA REDUCTION AND ANALYSIS 20

7.0 PROJECT PLANNING 20

7.1 BUDGET 20
7.2 SCHEDULE 21

8.0 LIST OF REFERENCES 22

APPENDIX 23

APPENDIX A: TEST PROTOCOLS DOCUMENT 23
APPENDIX B: TEST SCENARIOS 30
APPENDIX C: SOFTWARE DESIGN AND DEVELOPMENT DOCUMENT 32

APPENDIX D: COUHES 50

List of Figures

Figure 1: Red-contour display. 6
Figure 2: EGPWS. 7
Figure 3: Schematic of Apparatus. 11
Figure 4: Sketch of TAWS without terrain information 15
Figure 5: Software Architecture Flowchart 17

List of Tables

Table 1: Test Matrix 13
Table 2: Budget 20
Table 3: Schedule 21

Executive Summary

Controlled Flight into Terrain is one of the major source of accidents in aviation today. Studies have shown that for commercial aircrafts, the use Terrain Awareness and Warning Systems (TAWS) can decrease the number of such accidents. However, the TAWS available today are mostly geared towards commercial aircrafts. Their cost and size make them impractical for general aviation. Another problem with current TAWS is that TAWS mostly uses the same display window as other information such as weather, traffic, airport positions, and moving map programs. However, the amount of clutter the terrain information alone adds to the displays makes it difficult for other information to be perceived.

In this project document, we have designed the development and testing of a de-cluttered TAWS display format that will be geared towards general aviation aircrafts. The traditional color-coded contour terrain display is replaced with a single red contour indicating the projected points of impact of the aircraft with terrain within a 60-degree heading range. We believe that this method will provide the same terrain information for the users while dramatically improving the readability of other information on the display.

Both the traditional color-code terrain and this new red-contour display will be implemented in the language C using Open GL as the display program. Testing of the TAWS will be conducted using the basic flight displays in Microsoft Flightsim. Pre-programmed terrain and other information scenarios will appear on the TAWS, and the test subjects will attempt to navigate around these obstacles using controls linked to MSFS. Data will be taken regarding the subjects reaction time and judgment accuracy of the test subjects with regard to terrain and other information. Comparison of data obtained form the traditional and red-contour tests with 10 different subjects should allow us to draw certain conclusions towards the hypothesis. If a conclusion could not be drawn due to lack of statistical significance in the data, then it will be so noted.

1.0 Introduction

How does a disoriented pilot flying through a terrain rich environment covered in thick fog avoid an accident when all he has to his aid is a paper map? Often, s/he doesn't. That is one of the reasons why Controlled Flight into Terrain (CFIT) (any collision with ground or water while the pilot has control over the aircraft) is one of the leading causes of accidents in aviation. And that is why many research efforts have been made toward the development of Terrain Awareness and Warning Systems (TAWS). For this project a display format for a TAWS, cheaper and more effective then the existing designs, will be developed for the use of general aviation aircraft.

There are several types of TAWS available today. The Ground Proximity Warning System (GPWS) is a multifunctional display that shows terrain data along with weather, air traffic, and position information. It uses a lookdown sensor to determine the altitude and slope of the terrain below the aircraft and extrapolates the situation ahead from the sensor readings. This system has many shortcomings: limited warning time, incorrect terrain prediction, etc. The Enhanced Ground Proximity Warning System (EGPWS) is the successor of GPWS and uses a terrain database to predict the situation ahead. This system allows for longer warning periods and the accuracy of the warning is only limited by the quality of the database. The Ground Collision Avoidance System (GCAS) also uses a look-ahead terrain database; however, this display does not incorporate information such as weather and traffic. All three designs portray the terrain information from a top-down view and use different colors to indicate different hazard levels.

One of the concerns with the existing TAWS is that they are mostly geared towards larger, commercial aircraft where such a system is required by regulation. The size and cost of these TAWS make them impractical for smaller, general aviation aircraft. The few designs that are available for general aviation have serious drawbacks: either the terrain information is so poorly displayed that it is of little help to the pilots, or the terrain information is so cluttered that it compromises the pilots' ability to perceive other information in the display. In a world where 86% of all accidents occur with general aviation aircraft and 16% of fatalities in general aviation accidents are caused by CFIT,[1] it is imperative to develop a compact, affordable, and readable TAWS for general aviation usage.

This project will focus on the design, implementation, and testing of such a TAWS display. All information will be given from a top-down view. A red arc marking the points of impact within a 60 degrees range of the aircraft heading is shown in the display (Figure 1). As the course of an aircraft is altered, the red arc can either move towards the aircraft, away from it, or disappear altogether depending on the terrain. By eliminating the excess terrain information found in other TAWS, not only does the red-contour design reduce clutter and memory demands, but it also provides more intuitive information for the pilots by showing them only what they need to know. When completed, the performance of the red-contour display will be assessed through testing against a baseline display, standard EGPWS (Figure 2).

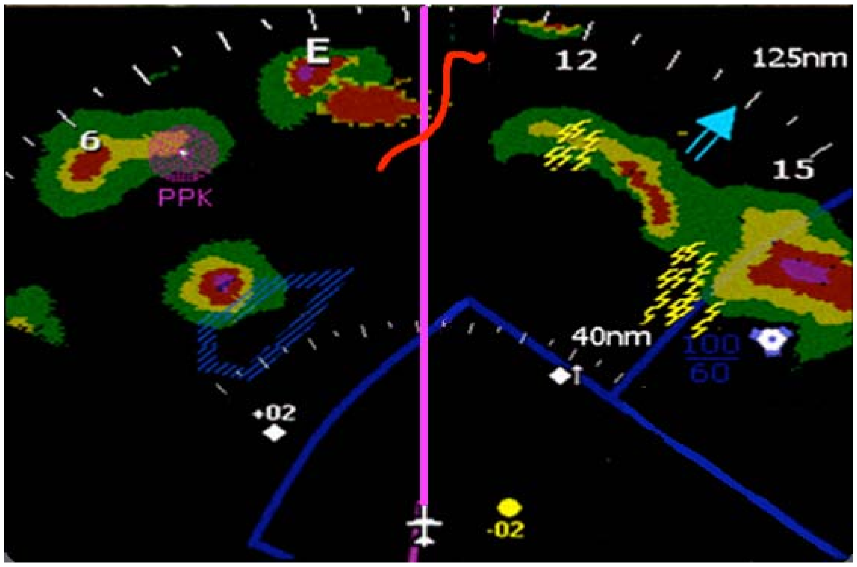


Figure 1: Red-contour display.

This is an example format of the red-contour display (see Figure 3 for a sketch of the final design). Notice red curve depicting the points of impact along the aircraft's course among the weather (color layers) and other information. This figure adapted from a slide in a Microsoft Power Point presentation by Hansman et al.[*1]

[*] Hansman, J. H., Davision, H., Vigeant-Langlois, L., Microsoft PowerPoint presentation, 2001.

This is a picture of the Honeywell KGP 560 EGPWS with the terrain feature in the display screen. Notice the amount of clutter in the terrain information alone and the buttons below the screen for the selection of other features such as weather and traffic.[2]

2.0 Hypothesis, Objective, and Success Criterion

Hypothesis: Reducing the clutter in the terrain awareness and warning systems will improve the overall readability of the display without having an adverse effect on the terrain awareness (i.e., number of false alarms and reaction time).
Objective: Use computer simulation to measure and compare pilots' performance with baseline and reduced clutter design.

Success Criterion: Obtain data to determine whether the performance of the reduced cluttered design is 1) better than the baseline's, 2) worse than the baseline's, or 3) indeterminate relative to the baseline's performance due to a lack of statistical significance in the level of difference

3.0 Literature Review

One of the key considerations in designing a TAWS is the optimal way of presenting the information to the pilots. In his paper, Glover discussed certain considerations in designing the terrain portion of the EGPWS.[3] Most of it consists of details specific to the EGPWS or the actual integration of the display system and aircraft, which is outside the scope of this project. However, he does note that "[e]xcept for some sound elementary principles, there are presently no detailed standards for flight deck displays which are accepted by all of the interested parties."[3] Thus, although general pilot intuition and requirements are important considerations for designing the red-contour display, standards specific to the flight deck is not an area of concern (since there are no standards).

"Situation Awareness Information Requirements for Commercial Airline Pilots"[4] and *An Introduction to Human Factors Engineering*[5] both relate to the human factor aspect of display design. The former includes an extensive list of pilot situation awareness requirements. Details concerning weather, traffic, airport (runway), workload, etc. are helpful to both the display design and later the test scenarios. The latter address many considerations in human factors engineering with a section centered on display development. It considers the perception, attention, and memory factors of the display user(s) and provides helpful hints on effective display formats. Both of these sources will serve as references, not guidelines, for the design of the red-contour.

Testing is a major part of this project; and the first consideration in testing is the facility used. Sachs et al.[7] describes actual flight-testing of a GCAS. The objective of the test seems geared towards the accuracy of the display rather than its effectiveness; and the performance of the display under critical situations is not tested likely due to the fact that flight and crew could not be placed into danger. In testing the red-contour against the baseline, both systems provide accurate information (by definition of the experiment). The variable comes from the effectiveness in the display format. The testing of the systems will require many dangerous and complicated scenarios where accidents occur. Therefore, flight tests for this experiment will not only be impractical (due to resource constraints) but also ineffective.

Muynck and Khatwa used a flight simulator cockpit for the testing of a TAWS (similar to the EGPWS)[7] while Kuchar and Hansman tested three different display concepts (top-down "plan" view, left-to-right "profile" view, and front-back "perspective" view) using a part task simulator, a display system consisting of flight instruments and terrain warning system.[8] Both methods are valid and effective; and although the cockpit simulator may provide a more realistic portrayal of flight scenarios, the resources available for this project do not allow for the usage of such a facility. Furthermore, since the experiment centers on the comparison between two TAWS, the key lies in using the same simulator for both systems, the type of simulator used is a secondary concern.

Both experiments used experienced commercial aircraft pilots (or flight crews) as test subjects.[7,8] Although such expertise is desirable, there might be trouble committing such a group of subjects to this project. Also, as mentioned in Endsley et al., "[s]ituation awareness forms the critical input to—but is separate from—decision making." Even an experienced pilot may miss a piece of information on a display while an inexperienced subject may perceive the correct information but is unable to take the appropriate course of action.[4] This experiment aims to test the performance of the TAWS, not the piloting ability of the subjects. Thus, the flight course and accident rates will be noted and analyzed but will not be deterministic towards the TAWS performance, so subject experience is not as essential.

The testing procedures and scenarios used in Muynck and Khatwa are detailed and intensive. A set of standard operational procedures is provided with detailed guidelines towards actions during various situations (take-off, landing, TAWS warning, etc). The scenarios cover a wide range from normal, uneventful flights to navigating through terrain during an engine failure. While some of these may be referenced or even adapted for this project, others such as terrain database errors are not relevant.

The Kuchar and Hansman paper also designed various scenarios for experimentation. However, they utilized a terrain hazard button method where, every time a warning appeared, the subject is asked to press one of four buttons corresponding to none, low, moderate, and high hazards.[8] For the purpose of this project, such a method might not be appropriate because a terrain hazard could be visible through the display even though a warning is not given. The measurement of time between the appearance of the hazard and its identification will one of the essential parameters measured.

Finally, the Muynck and Khatwa paper found that systems using look-ahead databases produce better results then those with look-down sensors, and navigation displays with terrain information fare better then those without.[7] Kuchar and Hansman, on the other hand, found that the top-down view display is the most preferred by pilots.[8] These findings are encouraging in that they place our baseline terrain system (a top-down view terrain display and warning system with look-ahead database) as one of the most effective form of displays available. Thus it seems logical to improve upon this design to obtain even better results.

4.0 Technical Approach

4.1 General Experimental Overview

The two displays will be tested using Microsoft Flightsim 2000. A total of 10 subjects will be divided into 2 groups. One group will be tested first with the red-contour and then the baseline. The other group will be tested in reverse order. Before the simulation, each subject will be given an overview on simulator functions and simulation goals (see Appendix A for Test Protocols). Subjects will then be asked to run the simulations through designed flight scenarios along with a given flight plan (see Appendix B for Test Scenarios). The sets of scenarios used for the baseline and the red-contour will be similar, but not identical, to each other so that the same parameters can be tested without pilots realizing it. The scenarios in each case will be given in a random order.

All scenarios take place under zero visibility (no out-of-window view) where CFIT accidents are most likely to occur. If a deviation from the flight plan needs to be made due to perceived terrain hazard in the TAWS, the subject will press the hazard button on the controls. During the simulation, a series of pop-ups will appear on the display in regular intervals. These pop-ups will replace "other information" (i.e. weather, traffic, airports, etc.) in the display and serve the purpose of providing non-terrain distractions during flight. Each pop-up will have a letter and number associated with it. The letter is used to identify the pop-up while the number represents the pop-up's altitude. The subjects must press the corresponding letter on the keyboard for a pop-up if its altitude is above the level of the plane. This requires the subject to 1) recognize the presence of the pop-up, 2) decide on the altitude of the pop-up in-relation to the aircraft, and 3) distinguish and press the letter label associated with the pop-up.

Aircraft data along with reaction time and accuracy for terrain and pop-ups will be recorded by the data recorder. The subjects will also receive a pre-experiment survey concerning age, gender, flight experience, etc. and a post-experiment survey concerning their opinions towards the two displays (see Appendix A: Test Protocols).

4.2 Apparatus

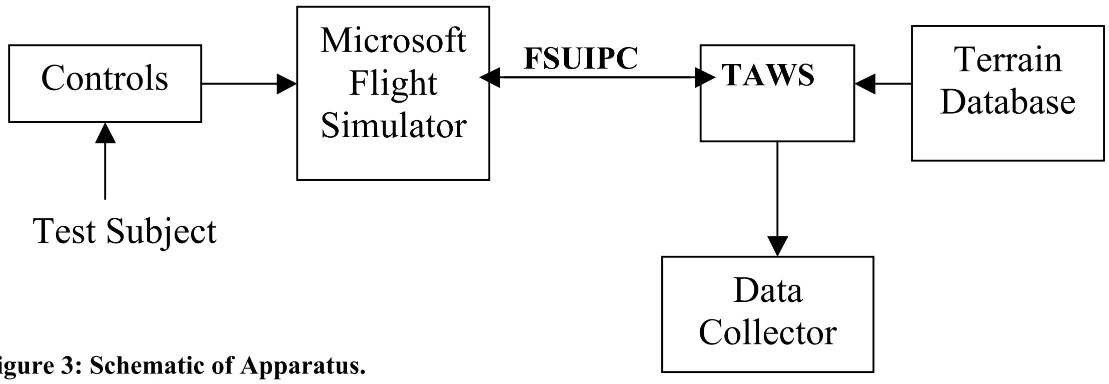


Figure 3: Schematic of Apparatus.

This is the internal setup of the flight simulator program with the TAWS.

Figure 3 shows the internal setup of the test apparatus. The TAWS and Data Collector will be programmed in the language C. The visual part of the TAWS will be displayed with the Open GL program for C. The FSUIPC is a communication program between Microsoft Flightsim and C. The Terrain Database will provide terrain information to the TAWS. The controls will include standard aircraft controls, a hazard button, and a standard keyboard. Human test subjects will utilize the controls and the displays in Flightsim and TAWS to run the simulations. All of the simulation data will be collected by the Data Collector (for more software details, refer to Appendix C).

4.3 Test Articles and Subjects

The articles tested will be the baseline and the red-contour display. The baseline will be similar to a standard version of EGPWS. Although there are many types of terrain designs, EGPWS is the one most commonly used and most suited for this experiment.

There will be 10 test subjects for this experiment. Licensed pilots are preferred however if there are difficulties in recruiting a sufficient number of pilots then experienced Microsoft Flightsim users are also acceptable (with the more experienced chosen over the less experienced). The Project Advisor has blanket authorization over human experimentations of this nature from the COUHES; therefore, no additional approval is required (See Appendix D for COUHES form).

4.4 Test Matrices and Variables

Our independent variables in this case are the two different types of displays and the terrain variations in the scenarios. The dependent variables are the pilot's reaction time to terrain and pop-ups (separately), the accuracy of recognition of terrain hazard and pop-ups (measured by the number of false alarms), and the number of accidents. Some of our control variables are the size, shape, color, and frequency of appearance of the pop-ups, the flight path that the subject takes during each scenario, and the type of terrain hazard given across the two TAWS (if a hazard is tested in one TAWS then it will also tested for the other). The position of the pop-up appearance will be randomized by noted.

During the experiment, the Data Collector will record the aircraft altitude, position, heading, flight path angle, and velocity, the position of the expected impact point (if one exists), and altitude, position, and identification of all the pop-ups present at the moment. Notes will be made in the data if 1) pop-up appears, 2) the hazard button is pressed, or 3) a key on the keyboard is pressed. Analysis of this data will reveal to us the time between the appearance of the hazard or pop-up on the display and the pressing of the corresponding button. It will also indicate incidences where the hazard button is pressed when there is no hazard present in the given course, the pop-up letter key is pressed when the pop-up is below the aircraft, and the wrong letter key for the pop-up is pressed.

The test matrix will look like the table below. Each number in the table will show the mean and standard deviation of all subjects. If significant discrepancy occurs between the 10 subjects, separate analysis will be made and the result will be displayed in a manner most fitting.

Table 1: Test Matrix

Scenarios	**Baseline**	**Red Contour**
1		
2		
…		

4.5 Errors

Sources for error in this experiment are subject variability, learning effect, fatigue, and bias towards the TAWS. The variability in the subject is an expected error in all human testing; subjects differ by experience, reaction time, and general decision making tendencies. Subject experience will be controlled to a certain extend to limit this source of error. Subjects many perform better as the experiment progresses because they have gained more experience at using the system; however, they might also perform progressively worse due to fatigue from simulation. Both of these effects will be balanced by switching the order in which the display and scenarios are tested. There will be a visual screening for serious fatigue and other subject inadequacies prior to testing. The bias towards the TAWS happens when subject pays more attention to the TAWS displays because they know that they are the articles tested. This effect will be minimized by giving subjects secondary tasks to focus on: telling the subjects that the experiment tests his/her ability to stay on course, recognize hazards, etc. In large subject pools the result of all of these errors will be normally distributed. In the relatively small group of 10 subjects, the errors will still be modeled as normal for analysis purposes. Most of these errors fall about the true mean and only add uncertainty to a data point. The bias towards TAWS, if not carefully treated, will offset the mean of the distributions.

5.0 Experimental Design

5.1 TAWS design

Both TAWS displays will be 3 inches by 3 inches in dimensions (the standard size of Electrical Horizontal Situations Indicator in general aviation aircraft: instrument used to display weather and traffic). The background of the TAWS will be black. Aside from terrain information, they will also have several other features on display (Figure 4).

The aircraft will be represented as a white triangle on the lower bottom of the display. The base will take up 4% of the display width and the height will be 5% of the display height. These dimensions seemed reasonable in the display of the prototype and are not unique (aircraft size may be varied in small amounts). Three pale blue lines and 2 pale blue curves will help orient the user. One line runs along the current heading of the aircraft while the other two lines form 60-degree angles on either side of the heading. The two curves are circular arcs marking equidistant points away from the aircraft. At the moment, the only number on the TAWS is the time to impact in seconds. This number is normally displayed in white on the bottom right corner of the window. The number will turn red when the time to impact reaches below 60 seconds and will start flashing when the time to impact drops below 30 seconds. Other numbers may be added to the display if they seem necessary during the preliminary tests. The pop-up will be a white circle with its letter reference on top and altitude on bottom. The size of the circle will be decided in the preliminary tests.

Figure 4: Sketch of TAWS without terrain information

5.1.1 Baseline

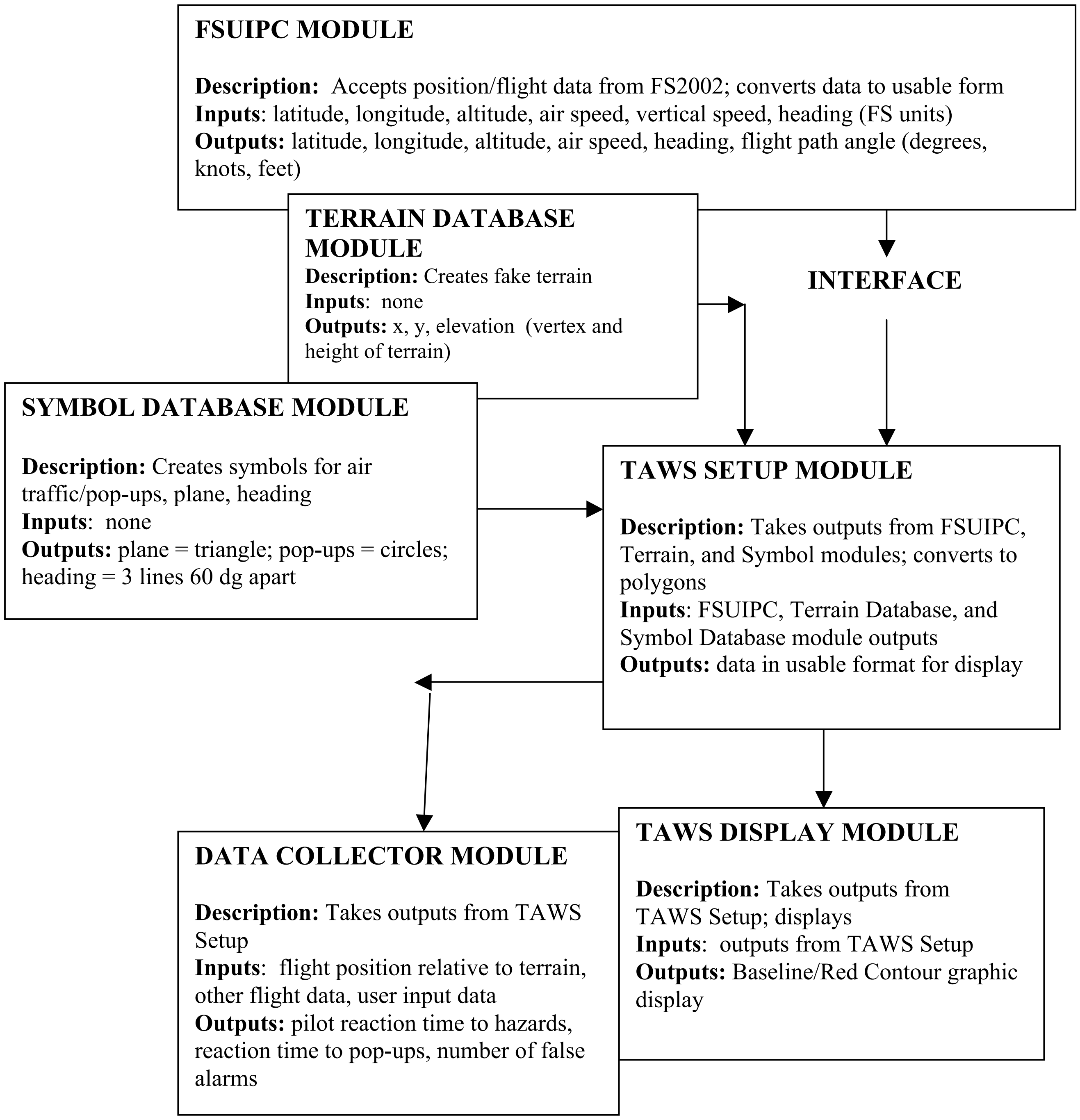
The baseline terrain information will be displayed in color-coded contours with the different colors representing the altitude of a display region relative to the aircraft's altitude: red, 2000+feet above; yellow, 1000-2000 above; light yellow, 500 below-1000 above; dark green, 1000-500 below; light green; 2000-100 below; black, 2000+ below. These color schemes are fairly standard for terrain color-coding.

5.1.2 Red Contour

The terrain information will be presented in the form of a red contour representing the points of impact of the aircraft with in a 60-degree range of its current heading. The aircraft flight path angle will be accounted for in the calculation of the red contour. The line thickness of the red contour will be 5 pixels. This value seemed reasonable in the prototype display, but may change upon further testing.

5.2 Software

The software consists of the following main components: FSUIPC code, FSUIPC and system interface, database reading functions, baseline setup, baseline display, red contour setup, red contour display, and the data collector (Figure 5). The FSUIPC code takes parameters from MSFS and converts them to appropriate units. The interface takes parameters from FSUIPC code, shifts them to the World coordinates and sends them to the TAWS codes. For details of the different coordinate systems used in the code, refer to the Software Architecture section in Appendix C. Database reading functions basically reads in the terrain and pop-up files. The setup for baseline and red-contour takes database and aircraft parameter information and outputs the appropriate values to be displayed. The display section for baseline and red-contour basically displays the given information into the window. The setup and display system updates every tenth of a second. The data collector takes relevant values from the system and outputs them to file. For more detail of the software design and development see Appendix C.



FSUIPC MODULE
Description: Accepts position/flight data from FS2002; converts data to usable form
Inputs: latitude, longitude, altitude, air speed, vertical speed, heading (FS units)
Outputs: latitude, longitude, altitude, air speed, heading, flight path angle (degrees, knots, feet)
TERRAIN DATABASE MODULE
Description: Creates fake terrain
Inputs: none
Outputs: x, y, elevation (vertex and height of terrain)
INTERFACE
SYMBOL DATABASE MODULE
Description: Creates symbols for air traffic/pop-ups, plane, heading
Inputs: none
Outputs: plane = triangle; pop-ups = circles; heading = 3 lines 60 dg apart
TAWS SETUP MODULE
Description: Takes outputs from FSUIPC, Terrain, and Symbol modules; converts to polygons
Inputs: FSUIPC, Terrain Database, and Symbol Database module outputs
Outputs: data in usable format for display
DATA COLLECTOR MODULE
Description: Takes outputs from TAWS Setup
Inputs: flight position relative to terrain, other flight data, user input data
Outputs: pilot reaction time to hazards, reaction time to pop-ups, number of false alarms
TAWS DISPLAY MODULE
Description: Takes outputs from TAWS Setup; displays
Inputs: outputs from TAWS Setup
Outputs: Baseline/Red Contour graphic display

Figure 5: Software Architecture Flowchart

5.3 Test Scenarios

Each test scenario has a terrain database and pop-up database associated with it. The simulation starts at the bottom middle of the terrain database World and moves according to the navigation of the pilots. If the pilot attempt to move outside of the World. The aircraft will crash.

For a list of test scenarios see Appendix B.

5.3 Test Environment

The testing of the TAWS will take place in the Break Out Room in the basement of building 33 below the aero-astro lounge. The set up will include either 1 or 2 computer(s) and monitor(s), a desk upon which the monitors can be placed, and a chair for the test subjects. At the moment, we plan on using one computer and monitor with two windows showing the MSFS cockpit displays and TAWS due to the issues speed in the link between 2 computers. However, we may choose a different system if the speed issue can be resolved (especially if the display format affects performance in the preliminary testing). Although we plan on using the Break Out room at the moment, the exact area of testing will not be important so long as the area is isolated and not extremely noisy. Although it would be desirable to stay in one testing environment, we may switch rooms during the middle of tests (from one subject to the next) without have any foreseeable effect on the experimental outcome.

5.4 Experiment Testing

After the completion of all software and data files and prior to the testing of the TAWS, we will conduct several simulation runs to test our experiment: first with Chinwe and myself and then with a few volunteers. These simulations will be conducted in the same manner as the actual experiments. The purpose of these runs is to test the validity and effectiveness of our experiment. Here is a list of specific design questions that we hope answer from these simulations (there may be others).

1) What is the optimal size and frequency of the pop-ups. We aim to find a size that will not be blatantly obvious but will also not be impossible to see. The frequency needs to be such that the workload for the pilots will be challenging but will not be impossible to keep up with.
2) Are the colors and sizes of symbols used too hard to distinguish from one another? Are they so big that they interfere with other information in the display?
3) Are the instructions clear?
4) Is the system easy/hard to use?
5) What will be a good skill level for the subjects coming into the experiment?
6) What will be an appropriate training time for the subjects?
7) What is the approximate time needed the experiment and each of its parts

Procedures and software can be changed pending the result from these tests. If significant changes are made, these tests will be re-run for the new system.

5.5 Human Experiment Protocol

In Appendix A is the Test Protocol Document for this experiment. This document contains 7 parts. The first is a email that we plan to sent out at the beginning of the semester to recruit test subjects for our experiment and next is an email reminder that we plan on sending individual subjects the night before the come in for testing. There may also be other communications between these emails, which would differ on a case-by-case basis. The simulation procedures cover a step-by-step process of exactly how the experiment will be conducted. The subject screening will help us ascertain the experience of the subjects. We expect participants to be truthful regarding their piloting and/or MSFS abilities, but this will be a way to provide formal demonstration of those abilities. We will also visually screen the subjects. Subjects will be disqualified or asked to return on a later date if their physical condition is obviously not suitable for participating in the experiment (ex. if they are sleeping on their feet). This visual screening process is not mentioned in the document because there is no formal procedure associated with it. Next the Pre-Simulation survey asks general information from the subject and some specifics concerning their flight experience and conditions on the day of the test; and the test instructions contains four sections of written instructions what will be read to the subject. This ensures that all subjects are given the same information during the simulations. Finally, the Post-Simulation Survey asks detailed questions concerning the subjects' opinions on the two specific displays. The sections are given in the order by which they will be presented to the subjects. (See Appendix D for the COUHES approval for human experiment.)

5.6 Others

There are no planned equipment purchases or foreseeable safety concerns at the moment

6.0 Data Reduction and Analysis

Reducing the data involves taking all the files generate by the data collector and tabulating them to get the following parameters for each test subject during each scenario: reaction time to terrain (if exists), reaction time to the pop-ups, number of false alarms in relation to the terrain, number of false alarms in relation to the pop-ups, the number incorrect key inputs for the pop-ups, and the number of crashes. A Paired T test will be performed between the two displays for each of the variables. The difference in performance between the two displays must achieve a standard 95% confidence level for the results to be considered statistically significant.

7.0 Project Planning

7.1 Budget

Table 2: Budget

ITEM	AVAILABILITY	ACTUAL COST	16.62X COST
Microsoft Flight Simulator 2002	MIT	\$49.99	\$0
FSUIPC	Internet Download	\$0	\$0

Computers	MIT	$499.97	$0
Monitors	MIT	$249.97	$0
Test Subjects	MIT (tentative)	~$750	$0
Testing Facility Rental	MIT	~$900	$0
Joystick	MIT	$59.99	$0
Software Modules	N/A	~$2000	$0
Microsoft Visual C++	MIT	$112.09	$0
OpenGL	Internet Download	$0	$0
TOTAL		$4622.01	$0

7.2 Schedule

Table 3: Schedule

SEPT. OCT. NOV. DEC.

WEEK / TASK	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
IMPLEMENT PHASE	■	■	■	■	■								
Write/Revise Software	▨	▨	▨										
Test Software		▨	▨										
Take Preliminary Data				▨									
Revisions					▨								
OPERATE PHASE						■	■	■	■	■			
Collect/Reduce Data						▨	▨	▨	▨				
REPORT PHASE									■	■	■	■	■

8.0 List of References

1. Bud, M. J., Mengert, P., Ransom, S., Stearns, M.D., et al, "General Aviation Accidents, 1983-1994: Identification of Factors Related to Controlled-Flight-Into-Terrain (CFIT) Accidents. Final Report DOT-VNTSC-FAA-97-8 (DOT/FAA/AAR-100-97-2)." Volpe National Transportation Systems Center, http://www.hf.faa.gov/docs/508/docs/volpe/volpe_972.pdf, 1997

2. Bendix, and King, "KGP-560 GA-EGPWS: Enhanced Ground Proximity and Warning System (TAWS) for General Aviation." http://www.egpws.com/general_information/brochures/KGP560.pdf.

3. Glover, H., "Terrain Displays for Enhanced Ground Prozimity Warning System," AIAA and SEA, 1997 World Aviation Congress, AIAA 97-5557, Anaheim, CA, Oct. 13-16, 1997.

4. Endsley, M. R., Farley, T. C., Jones, W. M., Midkiff, A. H., and Hansman, R. J., "Situation Awareness Information Requirements for Commercila Airline Pilots," MIT International Center for Air Transportation, ICAT-98-1, Cambridge, MA, Sep. 1998.

5. Wickens C. D., Gordon, S. E., and Liu, Y., *An Introduction to Human Factors Engineering*, Addison Wesley Longman, New York, 1998.

6. Sachs, G., Sennes, U., and Dudek, H.-L., "Design and Flight Tests of a Ground Collision Avoidance System," AIAA Modeling and Simulation Technoloies Conference and Exhibit, AIAA 99-4037, Portland, OR, Aug. 9-11, 1999.

7. Muynck, R. J., and Khatwa, R., "Flight Simulator Evaluation of the safety benefits of Terrain Awareness and Warning Systems," AIAA Guidance, Navigation, and Control Conference and Exhibit, AIAA 99-3965, Portland, OR, Aug. 9-11, 1999.

8. Kuchar, J. K., and Hansman, R. J., "Part-Task Simulation Study of Candidate Terrain Alerting Displays," MIT Aeronautical Systems Laboratory, ASL-93-6, Cambridge, MA, Jun. 1993.

Appendix

Appendix A: Test Protocols Document

Obtaining Test Subjects Email

(via email and poster)

Have you ever piloted a plane …
… either in reality OR on Microsoft Flight Simulator?

If so, you are just the subject we are seeking for a Flight Simulator experiment. If you have some actual piloting experience OR are familiar with Microsoft Flight Simulator, we need YOU to reply to this email/contact us at __________. We only require ____ minutes/hours of your time, and all you need to do is play several Flight Simulator games we have designed!!!! The dates we will run the experiment are as follows:

Looking forward to hearing from you!!!

Subject Qualification: in addition to having pilot/FlightSim experience, subject MUST NOT be colorblind and must have vision correctable to 20/20.

Pre-testing Reminder email

Hi <name>,

This is a reminder you have a Microsoft Flightsim simulation appointment with us tomorrow at <time>.

We look forward to seeing you. Thank you for your time.

Sincerely

Shen Qu and Chinwe Nyenke

Simulation Procedures

The same procedure will apply to all subjects. Only one subject will be in the simulation room at any given time. Subject will be asked if s/he is ready before the beginning of each scenario.

1) Give subject 5-minute skills test; if the subject passes, go to step 2.
2) Ask subjects to complete pre-simulation survey

3) Seat subjects in front of simulator
4) Start simulator with baseline TAWS (or red contour)
5) Bring up learning scenario
6) Read simulation instructions section 1
7) Start learning scenario when subjects are ready
8) If subject crashes within 15 min, restart the scenario
9) End scenario at 15 min
10) Bring up Scenario 1
11) Read simulation instructions section 2.
12) Start scenario
13) Repeat steps 10 through 12 for scenarios 2-6. No step 11
14) 10-minute break
15) Bring up the other TAWS display
16) Repeat steps 5-12. Step 6 reads simulation instruction section 3. No step 11
17) Ask subject to complete post-simulation survey
18) Read simulation instruction section 4

Subject Screening

Pilots will be asked to present pilot license.

MSFS users will be asked to fly a 5-minute simulation that begins and ends at cruising altitude and speed. They must be able to complete the planned flight course without crashing.

Pre-Simulation Survey

Name:
Age:
Gender:

1. Have you ever piloted an aircraft before? Yes/No

2. If you answered *No* to the question 1, skip to question 3.

a) Do you have a piloting license? Yes/No

b) How long has it been since you've gotten your license (years):

c) What type of aircraft have you flown:

d) How many piloting hours do you have logged:

e) When was the last time that you've flown (please give approximate
month/day/year):

f) How would you describe your skill as a pilot (1 = poor, 7 = excellent):

1 2 3 4 5 6 7

3. How many hours of experience do you have on Microsoft Flightsim (MSFS):

4. How often do you use MSFS (everyday, once a week/month/year):

5. When was the last time that you used MSFS:

6. How would you describe your skill in using MSFS (1 = poor, 7 = excellent):

1 2 3 4 5 6 7

7. How many hours of sleep did you get last night:

8. How would you describe your level of alertness today:

1 2 3 4 5 6 7

9. Do you have any comments relating to the experiment (general conditions, experiences, concerns, etc.)?

<u>Test Instructions (to be read to the subject)</u>

General Welcome:

Hi <name>. Thank you for coming. This experiment will run no longer than X hours. You will be asked to complete several flight simulations using Microsoft Flight Simulator. We will first have you undergo a skill test before we go on any further. *(If subject recruited as a pilot, request pilot license at this point.)* Any questions at the moment?

Section 1 – Trial Simulation:

The trial simulation will last 15 minutes and consist of zero out-of-window visibility. Your primary task is to maneuver around any dangerous terrain and press this button *(point to HAZARD button)* anytime a hazard appears on your flight path. You must also stay within the simulation environment boundaries. Crashing into terrain or failing to stay within the environment will lead to an end in the simulation. If this end occurs before the 15-minute period, you will restart the simulation until you complete a full trial run. The simulation will start at cruising speed and altitude.

In order to navigate around the environment, you will have to use the FlightSim cockpit panels and this terrain display as a guide.
- *(Pull out terrain display diagram.)*
- *Description of display, HAZARD button, and pop-ups:*

If using BASELINE display first:
The red-yellow-green regions represent different terrain elevations – red being 2000+ feet above the plane, yellow being within 2000 feet above the aircraft and 500 feet below the plane, and green being 500-2000 feet below the plane. Thus, you must attempt to avoid the red regions and be cautious of the yellow areas on the display. Press this HAZARD button whenever you notice a red region on your flight path.
The blue line on the display represents your flight course, which you must do your best to follow in the actual simulation. This will not be important in the trial run.
Pop-ups such as these *(point to display diagram)* may occur at random intervals, and you must note them by pressing the corresponding key *(point to keyboard).* For example, if aircraft A pops up and it is within 2000feet relative to your altitude, you must press the letter "A" on the keyboard immediately.
So, in this trial simulation, you will need to: discern hazardous terrain by pushing the HAZARD button, note pop-ups by pushing the corresponding key, and do your best to stay within the environment boundaries. Any questions?

If using RED CONTOUR display first:

A red contour will appear on the screen only if there is imminent terrain danger within 60 degrees of your flight path. Thus, you must attempt to avoid the red contour or you will crash.
The blue line on the display represents your flight course, which you must do your best to follow in the actual simulation. This will not be important in the trial run.
Pop-ups such as these *(point to display diagram)* may occur at random intervals, and you must note them by pressing the corresponding key *(point to keyboard).* For example, if aircraft A pops up and it is within 2000feet relative to your altitude, you must press the letter "A" on the keyboard immediately.
So, in this trial simulation, you will need to: discern hazardous terrain by pushing the HAZARD button, note pop-ups by pushing the corresponding key, and do your best to stay within the environment boundaries. Any questions?

Section 2 – Scenario Description:

Before each scenario, tell/reinforce to the subject: Your primary task is to fly from waypoint A to B (and then to C – *depending on the scenario*). You will receive X points for staying on the designated flight course, Y points for recognizing hazardous terrain, and Z points for noting relevant pop-ups. As was the case in the trial simulation, you will need to push the HAZARD button when you see dangerous terrain on your flight path. Also, you will again have to press the appropriate key on the keyboard when a pop-up within 2000feet relative to your altitude appears. The values of Y and Z depend on how fast you react to the hazards and pop-ups. Any questions?

Section 3 – Trial Simulation:

You will now perform another 15-minute trial simulation using a different terrain display. The same rules apply: press this HAZARD button when necessary, stay within the simulation environment boundaries, and press the appropriate key on the keyboard corresponding to any pop-ups. Once again, crashing into terrain or failing to stay within the environment will lead to an end in the simulation, and if this end occurs before the 15-minute period, you will restart the simulation.

Here is the terrain display you will have to use to navigate in the environment.
- *(Pull out terrain display diagram.)*
- *Description of display, HAZARD situation:*

If using BASELINE display now:
The red-yellow-green regions represent different terrain elevations – red being within 2000+ feet above the plane, yellow being within 2000 feet above the aircraft and 500 feet below the plane, and green being 500-2000 feet below the plane. Thus, you must attempt to avoid the red regions and be cautious of the yellow areas on the display. Press this HAZARD button whenever you notice a red region on your flight path.
The blue line on the display represents your flight course, which you must do your best to follow in the actual simulation. This will not be important in the trial run.

Any questions?

If using RED CONTOUR display now:
A red contour will appear on the screen only if there is imminent terrain danger within 10 degrees of your flight path. Thus, you must attempt to avoid the red contour or you will crash.
The blue line on the display represents your flight course, which you must do your best to follow in the actual simulation. This will not be important in the trial run.
Any questions?

Section 4 – Post-Experiment Survey:

Now we ask that you fill out one more short survey concerning your experience during the flight simulation. Please be honest and give plenty of feedback. Feel free to ask us any questions.
After subject finishes survey: Thank you for your participation. We ask that you do not discuss the nature and specifics of these tests to others as there are other subjects who have not yet preformed these tasks.

Post-Simulation Survey

Name:

On scale of 1to 5 (1 being the *red-contour*, 5 being the *multi-colored design*), rate the display that provided you with BETTER ABILITY TO:

View waypoints:

1	2	3	4	5
red-contour		same		multi-colored

Recognize and respond to pop-ups:

1	2	3	4	5
red-contour		same		multi-colored

Follow the planned flight course:

1	2	3	4	5
red-contour		same		multi-colored

Discern hazardous terrain:

1	2	3	4	5
red-contour		same		multi-colored

Read information other than terrain data:

1	2	3	4	5
red-contour		same		multi-colored

Other

Which display do you prefer? Why?

Any comments:

Thank you for your participation!!!

Appendix B: Test Scenarios

Subject Skill Test:

Subjects will fly a 5-minute simulation that begins and ends at cruising altitude and speed. Completion of the planned flight course is necessary. Those who crash or stop before the 5-minute period ends will not move on to the experiment.

Learning Scenario:

This is a test scenario designed to help subjects learn our simulation set up. The simulation world is a terrain-rich area several times larger then the TAWS display area. Clear boundaries will be given. Subject will crash if he/she tries to fly outside these boundaries. The learning time is tentatively 15 min. This time may change pending our evaluation of the difficulty of the simulation setup.

The simulation will start with the aircraft at cruising speed and altitude. Subject will learn by flying around the simulation world, avoiding terrain with the TAWS display. If the subject crashes before 15min, the scenario will be restarted. Time between restarts will not count towards the 15min.

Data will be collected during this scenario as it would any other scenario (data collection methods and parameters described in the body of Version III). Data from these scenarios will be analyzed differently then all other scenarios. (Analysis method described in the text.)

Below is a list of actual test scenarios. All scenarios start with the aircraft at cruising speed and altitude. Scenarios 2-6 will end a few seconds after the subject recognizes a terrain hazard or crashes. Scenario 1 will end when the subject is close to point B. Pop-ups will appear once every second in all scenarios. Each scenario will be programmed in two alternative forms for the two TAWS (same idea, different surrounding factors). The number of crashes in each case will be noted. The planned flight course routing will be updated on the display screen after each waypoint (if applicable).

Scenario 1: No Hazard

Flight from point A to point B. Flat terrain; no hazard appears. Simulation ends when the subject is within 10 seconds of reaching point B or crashes. Scenario 1 addresses the effect of the display on the subject's recognition of pop-ups.

Scenario 2: Real Hazard

Flight from point A to point B. Terrain hazard appears midway through course. Simulation ends after subject recognizes hazard or crashes into terrain. Scenario 2 addresses the effect of the display on the subject's recognition of pop-ups. It also addresses the effect of the pop-up discerning task on the subject's terrain awareness.

Scenario 3: False Hazard

Flight from point A to point B to point C. Terrain hazard appears after the turn at point B such that if the course continues in a straight line from A to B and beyond, the aircraft runs into the terrain. This hazard is false because it does not appear on the planned course but several kilometers off of it. Simulation ends after subject recognizes hazard or crashes into terrain. It also addresses the effect of the display on the subject's recognition of pop-ups as well as the effect of the pop-up discerning task on the subject's terrain awareness. This is also used to balance out Scenario 4 so that the subject does not expect a real hazard every time he/she makes a turn. *Note: in this scenario, since the hazard is false and on the line past A to B, there is no hazard present on the course from B to C*

Scenario 4: Turn into Hazard

Flight from point A to point B. Terrain hazard appears several seconds after the ATC (Air Traffic Controller) asks the subject to maneuver West. Simulation ends after subject recognizes hazard or crashes into terrain. Scenario 4 addresses a possible design issue in the reduced clutter display. While the terrain hazard is readily visible on the baseline display, it does not appear onto the screen of the test display until the turn is in progress. This scenario also addresses the effect of the display on the subject's recognition of pop-ups and the effect of the pop-up discerning task on the subject's terrain awareness.

Appendix C: Software Design and Development Document

1.0 Databases

1.1 Terrain Database

A terrain database will be artificially made up for each test scenario. The file will contain a list of rectangles represented by the x and y positions of it's 4 vertices. The positions are given in a relative scale to the simulation world normalized from 0 to 1. Each rectangle has an altitude value attached to it. Thus the file will be in matrix like format with each row containing 9 values representing the rectangle and altitude and a number of different rectangles along the columns. The rectangles are sorted along the columns by its altitude with the lower altitudes appearing first in the file. When the file is read by either the baseline or the red contour, the lower altitude values will be read in first. Thus, if a higher altitude rectangle occupies the same space, it will over ride the lower altitude one at the areas of intersection since it will be read in later. A detailed list of such rectangles will create realistic looking terrains.

The simulation world is the area that the aircraft is allowed to travel in for each simulation. This is a rectangular world with its four corners normalized to (0, 0), (0, 1), (1, 0), and (1, 1). All internal position parameters and headings will be in reference to this world.

The terrain data can also be constructed using other geometric shapes (circles, points, etc.), there are certain mathematical functions that can be used to generate random looking terrain, and we have a terrain database of the Eastern part of the US. None of these three terrain formats are used at the moment, however, they do serve as possible backups.

1.2 Pop-up Database

There will also be a pop-up database made for each scenario. Each pop-up is represented by x variables: x position, y position, letter reference, altitude value, time of appearance (time from start of simulation), and duration of appearance. The database will be a list of pop-ups each with these parameters.

2.0 Prototype

The prototype for our software is an extremely simplified version of the final software program. It contains three main pieces: C code using FSUIPC to read in MSFS parameters and convert them into standard units, a <u>very</u> simple terrain database file, and C code using Open GL to run a moving display of the terrain database in both baseline and red contour format. It is important to note that the FSUIPC and Open GL codes are not integrated. For the purpose of this prototype, an aircraft parameter file is used to provide AC parameters to the TAWS. Such a file will not exist for the final software. AC parameters will be obtained from MSFS real time.

The main purpose of this prototype is to demonstrate that we can obtain appropriate parameters from MSFS, and we can achieve baseline and red contour displays using the database and Open GL. The terrain and AC parameters provided are not realistic. And the specific colors, size, and position of the display may not be the same as the final design.

2.1 MSFS and FSUIPC

FSUIPC is a package in C with functions that allows for communication with MSFS. The code responsible for this communication and information retrieval will here after be referred to as the FSUIPC code. Below is an overview of what the prototype version of this code accomplishes.

1) checks if MSFS is running. If not, nothing happens
2) If so: gets latitude, longitude, heading, vertical speed, altitude, and air speed from MSFS
3) Converts data from MSFS units to real/usable units (degrees, knots, feet) using conversion factors from MSFS
4) generates flight path angle
5) displays 15 example data to screen
6) ends

2.2 Aircraft(AC)Parameters File

The AC parameter files provides 6 AC parameters in the database "World" coordinate system: x position, y position, altitude, heading, flight path angle, and velocity. There are 2 AC parameter files provided (see appendix). Each file contains 10 sets of parameters. Only the y position is updated to move the terrain display area along the World. One file has constant altitude while the other has variable altitude levels.

2.3 Terrain Database

The terrain database in this case is a very simple file containing 3 rectangles at altitudes 1000, 2000, and 3000 ft respectively. In the terrain display if forms a "mountain" with 3 contour layers.

2.4 Baseline

The baseline prototype takes in terrain and AC data and displays an example baseline that update once every second for 10 updates. Red represents terrain > 500ft above the aircraft, green represents terrain with in 500ft of the aircraft, and blue represents terrain over 500ft below the aircraft. This is not the final color code used for the baseline. For the correct color-coding refer to the text. When reading in the AC data file 2 with different altitudes the color scheme changes accordingly for each update.

Main() opens a TAWS window with the appropriate size and location. It also calls init() and display(). Init() is just an initialization function. The main display function display() is responsible for displaying and updating the appropriate terrain. Find_taws_position() finds the corners and center of the TAWS in the World reference frame.

This prototype does not allow for rotation of TAWS with respect to the World.

2.5 Red Contour

The red contour prototype takes in terrain and AC data and displays an example red contour that update once every second for 10 updates. It does so by dividing the TAWS area into a 100 x 100 grid area with each grid represented by a single point. The finds the altitude of the terrain at each point and also the lowest altitude at which the aircraft will impact with terrain at each of these points. The latter is calculated using aircraft flight path angle. If the impact altitude is lower then the terrain value then the aircraft will crash if it hit that point. Finally, an algorithm is designed to search for the impact points closest to the aircraft along each heading and display it as part of the red contour.

Main() opens a TAWS window with the appropriate size and location. It also calls init() and display(). Init() is just an initialization function. The main display function display() is responsible for displaying and updating the appropriate terrain. Find_taws_position() finds the corners and center of the TAWS in the World reference frame. Unlike the baseline prototype, it also has functions clear_terrain_alt() that resets all elements in the global variable terrain_alt to 0, find_terrain_alt that finds the altitude value of at each point, and find_impact_alt() that find all of the impact altitude for each point.

This prototype does not allow for rotation of TAWS with respect to the World.

3.0 Software Architecture

There are 3 coordinate systems used through the entire code. First is the system with longitude, latitude, heading, and other values given by MSFS. Next is the reference coordinate for the simulation World with each axis normalized from 0 to 1 and the heading set at 0 degrees along the y direction. This reference frame is used for all internal calculations. Finally, there is the coordinate system used for displaying the TAWS. This is also a normalized reference frame with the TAWS center at (0, 0) with x and y both ranging from –1 to 1 at the four corners. This knowledge is needed to understand part of the discussion below.

The sections describe the changes and additions needed for the prototype before the software can be finalized. All of these changes can be easily made by person with adequate programming experience in the language C and a basic understanding of FSUIPC and Open GL.

3.1 FSUIPC

1) set aircraft at cruising altitude and airspeed at start of simulation
2) transfer data to TAWS module
3) record time and other necessary info for when user hits hazard button and keyboard
4) end when MSFS ends

3.2 Baseline and Red Contour

1) Interface with FSUIPC
2) Be able to move data from World coord. to TAWS coord.
   (Allow TAWS to rotate with respect to the World.)
3) Read pop-up display
4) Display pop-ups. (The ideas are the same as terrain read and display)
5) Display all symbols and colors according to design parameters.
   (see text.)

1.5 Interface

1) Change MSFS parameters into World coord
   (Must consider initial values and rotation factors)
2) Set the first set of MSFS variables as reference values for the rest
3) Provide parameters to TAWS code real time

1.6 Data Collector

1) Extract relevant data from TAWS code
2) Write parameters to file

4.0 Code

4.1 Terrain Database

```
0.4     0.2     0.6     0.2     0.6     0.8     0.4     0.8     1000.0
0.45    0.5     0.55    0.5     0.55    0.7     0.45    0.7     2000.0
0.475   0.55    0.525   0.55    0.525   0.65    0.475   0.65    3000.0
```

4.2 AC Parameter File 1

```
0.5     0.0     2000    0       0       100
0.5     0.1     2000    0       0       100
0.5     0.2     2000    0       0       100
```

0.5	0.3	2000	0	0	100
0.5	0.4	2000	0	0	100
0.5	0.5	2000	0	0	100
0.5	0.6	2000	0	0	100
0.5	0.7	2000	0	0	100
0.5	0.8	2000	0	0	100
0.5	0.9	2000	0	0	100

4.3 AC Parameter File 2

0.5	0.0	1000	0	0	100
0.5	0.1	2000	0	0	100
0.5	0.2	2000	0	0	100
0.5	0.3	3000	0	0	100
0.5	0.4	3000	0	0	100
0.5	0.5	2000	0	0	100
0.5	0.6	2000	0	0	100
0.5	0.7	2000	0	0	100
0.5	0.8	1000	0	0	100
0.5	0.9	2000	0	0	100

4.4 FSUIPC Code

```
/*   AUTHOR: Chinwe Nyenke (part of code borrowed from FSUIPC Demo Code)
     DATE LAST ALTERED:  May 2, 2003
          TASK: This code uses FSUIPC to obtain Microsoft Flight Simulator data
(aircraft's
     position, speed, vertical speed, and heading). First it verifies that the simulator is
          running, exiting if the sim is not running; then it runs a loop, outputting
     the position data, speed, heading, and flight path angle to screen  */


#include <stdio.h>
#include <math.h>
#include "IPCuser.h"

int main () {

  unsigned int counter = 0;

  /* used to hold a result code from any of the FSUIPC functions. */
  DWORD dwResult;

  /* the memory offsets for the altitude, latitude and slew mode control */
  DWORD altitudeoffset = 0x0574;
  DWORD headingoffset = 0x0580;  /* heading */
  DWORD latitudeoffset = 0x0560;
  DWORD longitudeoffset = 0x0568;   /* longitude */
  DWORD speedoffset = 0x02B8;  /* true air speed */
  DWORD v_speedoffset = 0x02C8;     /* vertical speed */



  /* the size of altitude, latitude, and slew mode control in FS2002. */
  DWORD altitudesize = 4;
  DWORD headingsize = 4;
  DWORD latitudesize = 8;
  DWORD longitudesize = 8;
  DWORD speedsize = 4;
  DWORD v_speedsize = 4;


  /* used to hold the altitude and latitude in real (not FS) units. */
  double altitude = 0;
  double heading = 0;
  double latitude = 0;
  double longitude = 0;
  double speed = 0;
  double v_speed = 0;
  double path_angle;


  /* used to hold the altitude, latitude, and slew mode status in
     FS2002's format. */
  INT32 fs_altitude = 0;
  INT32 fs_heading = 0;
  INT64 fs_latitude = 0;
  INT64 fs_longitude = 0;
  INT32 fs_speed = 0;
  INT32 fs_vspeed = 0;

  /* conversion factors */
  double alt_conv = 1/3.28084;  /* multiply by this -- feet */
  INT64 head_conv = 11930465;  /*divide by this -- degrees */
  INT64 lat_conv = 477302101697422;   /*divide by this -- degrees (+N, -S)*/
  INT64 long_conv = 51240955760300000;   /*divide by this -- degrees (+E, -W)*/
  double speed_conv = 1/128; /* multiply by this -- knots */
  double vspeed_conv = 1.944/256; /* multiply by this -- knots */

  /* open a link to flight sim. (FSUIPC Demo)*/
  if (!FSUIPC_Open(SIM_ANY, &dwResult))
    {
      printf("Cannot start the FSUIPC client.\n");
      printf("Flight Simulator is probably not running.\n");
      return -1;
    }

  /* read the current altitude and latitude, storing the
     values in fs_altitudevalue and fs_latitudevalue */
  if (!FSUIPC_Read(altitudeoffset, altitudesize, &fs_altitude, &dwResult))
    {
      printf("FSUIPC_Read (altitude) failed.\n");
      FSUIPC_Close();
      return -1;
    }

  if (!FSUIPC_Read(headingoffset, headingsize, &fs_heading, &dwResult))
    {
      printf("FSUIPC_Read (heading) failed.\n");
      FSUIPC_Close();
      return -1;
    }

  if (!FSUIPC_Read(latitudeoffset, latitudesize, &fs_latitude, &dwResult))
    {
      printf("FSUIPC_Read (latitude) failed.\n");
      FSUIPC_Close();
      return -1;
    }

  if (!FSUIPC_Read(longitudeoffset, longitudesize, &fs_longitude, &dwResult))
    {
      printf("FSUIPC_Read (longitude) failed.\n");
      FSUIPC_Close();
      return -1;
    }

  if (!FSUIPC_Read(speedoffset, speedsize, &fs_speed, &dwResult))
    {
      printf("FSUIPC_Read (speed) failed.\n");
      FSUIPC_Close();
      return -1;
    }

  if (!FSUIPC_Read(v_speedoffset, v_speedsize, &fs_vspeed, &dwResult))
    {
      printf("FSUIPC_Read (vertical speed) failed.\n");
      FSUIPC_Close();
```

```
     return -1;
    }


  /* process all pending requests. (FSUIPC Demo)*/
  if (!FSUIPC_Process(&dwResult))
    {
     printf("FSUIPC_Process (data read) failed.\n");
     FSUIPC_Close();
     return -1;
    }

  /* LOOP to obtain data.  Each time convert the data from fs units into
    real ones, storing the result in the "<>value" variables.
    .....*/


  printf("ANGLE ALTITUDE  HEADING  LATITUDE  LONGITUDE  SPEED\n");

  for (counter = 0; counter < 15; counter++)
          {
     /* convert from FS2002 units into real ones */
     altitude = (double) fs_altitude / alt_conv;     /*feet*/
           heading = (double) fs_heading / (double) head_conv + 349;    /*degrees*/
     latitude = (double) fs_latitude / (double) lat_conv;   /*degrees*/
           longitude = (double) fs_longitude / (double) long_conv;   /*degrees*/
           speed = (double) fs_speed *1/128;   /*knots*/
           v_speed = (double) fs_vspeed *vspeed_conv;   /*knots*/
           path_angle = asin(v_speed/speed)*180/3.1416;   /*calculation of flight path
angle  */

     /* print altitude and latitude values to screen. */
           printf("%.1f    %.1f     %.1f   %.2f    %.2f    %.2f          %.1f\n", path_angle,
altitude, heading, latitude, longitude, speed, v_speed);

    }

  /* close link to FS2002 */
  FSUIPC_Close();

  return 1;
}
```

4.5 Baseline Code

```
#include <GL/glut.h>
#include <time.h>
#include <stdio.h>

void init(void);
void display(void);
void read_data (void);
void find_taws_position (float ac_x, float ac_y);

//global variables

float ACParamList[10][6] = {0};
float TerrainDataList[3][9] = {0};

float world_AC_x_Pos = 0.5;
float world_AC_y_Pos = 0.0;

const float x_scale = 2.0;
const float y_scale = 2.0;

float taws_pos[5][2]={0.0};
const float taws_AC_x_pos = 0.5;
const float taws_AC_y_pos = 0.0;

int main(int argc, char **argv)
{
        read_data();
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
        glutInitWindowSize (400, 400);
        glutInitWindowPosition (600, 250);

        glutCreateWindow("TAWS");

        init();
        glutDisplayFunc(display);


        glutMainLoop();
        return 0;
}

void init(void)
{
  glClearColor(0.0, 0.0, 0.0, 0.0);
```

```
}

void display(void)

{
        float start_time=0.0;
        float rel_alt = 0.0;
        int i = 0;
        int j = 0;

        for (i=0;i<10;i++)
        {
                find_taws_position (ACParamList[i][0], ACParamList[i][1]);

                glClear(GL_COLOR_BUFFER_BIT) ;
                start_time=clock();
                //glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);

                for(j=0;j<3;j++)
                {
                        rel_alt = TerrainDataList[j][8]-ACParamList[i][2];

                        glColor3f(1.0, 1.0, 1.0);

                        if(rel_alt>-500.0 && rel_alt<=500.0)
                                glColor3f(0.0, 0.0, 1.0);

                        if (rel_alt>500.0)
                                glColor3f(1.0, 0.0, .0);

                        if (rel_alt <= -500.0)
                                glColor3f(0.0, 1.0, 0.0);

                        glBegin(GL_QUADS);
                                glVertex2f(2*x_scale*(TerrainDataList[j][0]-
taws_pos[4][0]), 2*y_scale*(TerrainDataList[j][1]-taws_pos[4][1]));
                                glVertex2f(2*x_scale*(TerrainDataList[j][2]-
taws_pos[4][0]), 2*y_scale*(TerrainDataList[j][3]-taws_pos[4][1]));
                                glVertex2f(2*x_scale*(TerrainDataList[j][4]-
taws_pos[4][0]), 2*y_scale*(TerrainDataList[j][5]-taws_pos[4][1]));
                                glVertex2f(2*x_scale*(TerrainDataList[j][6]-
taws_pos[4][0]), 2*y_scale*(TerrainDataList[j][7]-taws_pos[4][1]));


                        glEnd();
                }
                glColor3f(1.0, 1.0, 1.0);

                glBegin(GL_TRIANGLES);
                        glVertex2f(-0.04,-0.99);
                        glVertex2f(0.04, -0.99);
                        glVertex2f(0.0, -0.90);
                glEnd();

                glFlush();
                glutSwapBuffers() ;
                while(((clock() - start_time)/CLOCKS_PER_SEC)<1.0 ) {}
        }
}

void read_data (void)
{
        FILE *ACParamFile; //format x(rel) y(rel) alt(ft) heading(deg) FPAngle(deg)
vel(ft/s)
        FILE *TerrainDataFile; //format vert1         vert2    vert3    vert4    alt
                                                      //each vert has x y vals
        int i=0;

        ACParamFile = fopen("AC_param2.dat","r");
        TerrainDataFile = fopen("terrain.dat","r");

        while(!feof (TerrainDataFile))
        {
                fscanf(TerrainDataFile, "%f", &TerrainDataList[i][0]);
                fscanf(TerrainDataFile, "%f", &TerrainDataList[i][1]);
                fscanf(TerrainDataFile, "%f", &TerrainDataList[i][2]);
                fscanf(TerrainDataFile, "%f", &TerrainDataList[i][3]);
                fscanf(TerrainDataFile, "%f", &TerrainDataList[i][4]);
                fscanf(TerrainDataFile, "%f", &TerrainDataList[i][5]);
                fscanf(TerrainDataFile, "%f", &TerrainDataList[i][6]);
                fscanf(TerrainDataFile, "%f", &TerrainDataList[i][7]);
                fscanf(TerrainDataFile, "%f", &TerrainDataList[i][8]);

                i++;
        }

        i=0;

        while(!feof (ACParamFile))
        {
                fscanf(ACParamFile, "%f", &ACParamList[i][0]);
```

```
		fscanf(ACParamFile, "%f", &ACParamList[i][1]);
		fscanf(ACParamFile, "%f", &ACParamList[i][2]);
		fscanf(ACParamFile, "%f", &ACParamList[i][3]);
		fscanf(ACParamFile, "%f", &ACParamList[i][4]);
		fscanf(ACParamFile, "%f", &ACParamList[i][5]);

		i++;
	}

	fclose(ACParamFile);
	fclose(TerrainDataFile);

	return;

}

void find_taws_position (float ac_x, float ac_y)
{

	//taws_pos[1] to [4] are the 4 corners of the taws, taws_pos[5] is the center of
taws
	taws_pos[0][0] = ac_x - 1/(2*x_scale);
	taws_pos[0][1] = ac_y;
	taws_pos[1][0] = ac_x + 1/(2*x_scale);
	taws_pos[1][1] = ac_y;
	taws_pos[2][0] = ac_x - 1/(2*x_scale);
	taws_pos[2][1] = ac_y + 1/y_scale;
	taws_pos[3][0] = ac_x + 1/(2*x_scale);
	taws_pos[3][1] = ac_y + 1/y_scale;
	taws_pos[4][0] = ac_x;
	taws_pos[4][1] = ac_y+1/(y_scale);

	return;

}
```

4.6 Red Contour Code

```
#include <GL/glut.h>
#include <time.h>
#include <stdio.h>
#include <math.h>

void init(void);
void display(void);
void read_data (void);
void find_taws_position (float ac_x, float ac_y);
void clear_terrain_alt(void);
void find_impact_alt(float alt, float FPangle);
void find_terrain_alt(void);

//global variables

const float x_scale = 2.0;
const float y_scale = 2.0;

const float x_dist = 32808.3989; // ft, for TAWS
const float y_dist = 32808.3989; // ft, for TAWS

const float taws_AC_x_pos = 0.0;
const float taws_AC_y_pos = -1.0;

float taws_pos[5][2]={0.0};

float world_AC_x_Pos = 0.5;
float world_AC_y_Pos = 0.0;

float ACParamList[10][6] = {0};
float TerrainDataList[3][9] = {0};

float impact_alt[100][100] = {0};
float terrain_alt [100][100]= {0};

int main(int argc, char **argv)
{
        read_data();
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
        glutInitWindowSize (400, 400);
        glutInitWindowPosition (600, 250);

        glutCreateWindow("Red Contour");

        init();
        glutDisplayFunc(display);


        glutMainLoop();
        return 0;
}

void init(void)
```

```
{
  glClearColor(0.0, 0.0, 0.0, 0.0);

}

void display(void)

{
        float start_time=0.0;
        float rel_alt = 0.0;
        int i=0;
        int j=0;
        int k=0;

        for (i=0;i<10;i++)
        {
                find_taws_position (ACParamList[i][0], ACParamList[i][1]);
                clear_terrain_alt();
                find_terrain_alt();
                find_impact_alt(ACParamList[i][2], ACParamList[i][4]);

                glClear(GL_COLOR_BUFFER_BIT) ;
                start_time=clock();
                //glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);

                glColor3f(1.0, 0.0, 0.0);

                glPointSize(5.0);
                glBegin(GL_POINTS);
                for(j=0;j<100;j++)
                {
                        for(k=0; k<100; k++)
                        {
                                if(terrain_alt[j][k]>=impact_alt[j][k])
                                {
                                        glVertex2f((j/50.0)-1.0, (k/50.0)-1.0);
                                        break;
                                }
                        }
                }
                glEnd();

                glColor3f(1.0, 1.0, 1.0);

                glBegin(GL_TRIANGLES);
                        glVertex2f(-0.04,-0.99);
                glVertex2f(0.04, -0.99);
                glVertex2f(0.0, -0.90);
            glEnd();

            glFlush();
            glutSwapBuffers() ;
            while(((clock() - start_time)/CLOCKS_PER_SEC)<1.0 ) {}
        }
}

void read_data (void)
{
        FILE *ACParamFile; //format x(rel) y(rel) alt(ft) heading(deg) FPAngle(deg)
vel(ft/s)
        FILE *TerrainDataFile; //format vert1      vert2   vert3   vert4   alt
                                                   //each vert has x y vals
        int i=0;

        ACParamFile = fopen("AC_param2.dat","r");
        TerrainDataFile = fopen("terrain.dat","r");

        i=0;

        while(!feof (TerrainDataFile))
        {
            fscanf(TerrainDataFile, "%f", &TerrainDataList[i][0]);
            fscanf(TerrainDataFile, "%f", &TerrainDataList[i][1]);
            fscanf(TerrainDataFile, "%f", &TerrainDataList[i][2]);
            fscanf(TerrainDataFile, "%f", &TerrainDataList[i][3]);
            fscanf(TerrainDataFile, "%f", &TerrainDataList[i][4]);
            fscanf(TerrainDataFile, "%f", &TerrainDataList[i][5]);
            fscanf(TerrainDataFile, "%f", &TerrainDataList[i][6]);
            fscanf(TerrainDataFile, "%f", &TerrainDataList[i][7]);
            fscanf(TerrainDataFile, "%f", &TerrainDataList[i][8]);

            i++;
        }

        i=0;

        while(!feof (ACParamFile))
        {
            fscanf(ACParamFile, "%f", &ACParamList[i][0]);
            fscanf(ACParamFile, "%f", &ACParamList[i][1]);
            fscanf(ACParamFile, "%f", &ACParamList[i][2]);
            fscanf(ACParamFile, "%f", &ACParamList[i][3]);
```

```
		fscanf(ACParamFile, "%f", &ACParamList[i][4]);
		fscanf(ACParamFile, "%f", &ACParamList[i][5]);

		i++;
	}
	fclose(ACParamFile);
	fclose(TerrainDataFile);

	return;

}

void find_taws_position (float ac_x, float ac_y)
{

	//taws_pos[1] to [4] are the 4 corners of the taws, taws_pos[5] is the center of
taws
	taws_pos[0][0] = ac_x - 1/(2*x_scale);
	taws_pos[0][1] = ac_y;
	taws_pos[1][0] = ac_x + 1/(2*x_scale);
	taws_pos[1][1] = ac_y;
	taws_pos[2][0] = ac_x - 1/(2*x_scale);
	taws_pos[2][1] = ac_y + 1/y_scale;
	taws_pos[3][0] = ac_x + 1/(2*x_scale);
	taws_pos[3][1] = ac_y + 1/y_scale;
	taws_pos[4][0] = ac_x;
	taws_pos[4][1] = ac_y+1/(y_scale);

	return;

}

void clear_terrain_alt(void)
{
	int i=0;
	int j=0;

	for(i=0;i<100;i++)
	{
		for(j=0;j<100;j++)
		{
			terrain_alt[i][j] = 0.0;
		}
	}
	return;
}
```

```
void find_impact_alt(float alt, float FPangle)
{
        float dist = 0.0;
        int i=0;
        int j=0;

        for(i=0;i<100;i++)
        {
                for(j=0;j<100;j++)
                {
                        dist = sqrt(pow((x_dist*(i-50)/100), 2)+pow((y_dist*i/100), 2));
                        impact_alt[i][j] = alt- sin(FPangle*3.14/180)*dist;
                }
        }
        return;
}

void find_terrain_alt(void)
{

        float xMin=0.;
        float xMax=0.;
        float yMin=0.;
        float yMax=0.;

        int i=0;
        int j=0;
        int k=0;

        for(k=0;k<3;k++)
        {
                xMin = x_scale*(TerrainDataList[k][0]-taws_pos[0][0]);
                xMax = x_scale*(TerrainDataList[k][4]-taws_pos[0][0]);
                yMin = y_scale*(TerrainDataList[k][1]-taws_pos[0][1]);
                yMax = y_scale*(TerrainDataList[k][5]-taws_pos[0][1]);


                if(xMin<0.)
                        xMin=0.;
                if(xMax>1)
                        xMax=1.0;
                if(yMin<0)
                        yMin=0.0;
                if(yMax>1)
                        yMax=1.0;
```

```
				for(i= (int) (xMin*100); i < (100 * xMax) && xMax>0;i++)
				{
					for (j= (int) (yMin*100); j<(100*yMax) && yMax>0; j++)
					{
						terrain_alt[i][j] = TerrainDataList[k][8];
					}
				}
			}

			return;

		}
```

Appendix D: COUHES