Air Force Institute of Technology

AFIT Scholar

Theses and Dissertations

Student Graduate Works

6-1999

Closely Supervised Reactive Control of an Uninhabited Aerial Vehicle

Roy G. Glassco

Follow this and additional works at: https://scholar.afit.edu/etd

Part of the Navigation, Guidance, Control and Dynamics Commons

Recommended Citation

Glassco, Roy G., "Closely Supervised Reactive Control of an Uninhabited Aerial Vehicle" (1999). *Theses and Dissertations*. 5161.
https://scholar.afit.edu/etd/5161

This Thesis is brought to you for free and open access by the Student Graduate Works at AFIT Scholar. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of AFIT Scholar. For more information, please contact AFIT.ENWL.Repository@us.af.mil.

AFIT/GAE/ENY/99J-01

CLOSELY SUPERVISED REACTIVE CONTROL OF AN UNINHABITED AERIAL VEHICLE

THESIS

Roy Glen Glassco, B.S.

Captain, USAF

AFIT/GAE/ENY/99J-01

19990625 012

Approved for public release; distribution unlimited

AFIT/GAE/ENY/99J-01

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

AFIT/GAE/ENY/99J-01

CLOSELY SUPERVISED REACTIVE CONTROL OF AN UNINHABITED AERIAL VEHICLE

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the Requirements for the

Degree of Masters in Aeronautical Engineering

Roy Glen Glassco, B.S.

Captain, USAF

Air Force Institute of Technology

Wright Patterson AFB, OH

June, 1999

Approved for public release; distribution unlimited

AFIT/GAE/ENY/99J-01

CLOSELY SUPERVISED REACTIVE CONTROL OF AN UNINHABITED AERIAL VEHICLE

Roy Glen Glassco, B.S.
Captain

Approved:

Curtis H. Spenny
Dr. Curtis H. Spenny
Air Force Institute of Technology
Committee Chairman

20 May 1999
date

Bradley S. Liebst
Dr. Bradley S. Liebst
Air Force Institute of Technology
Committee Member

20 May 1999
date

Kuldip S Rattan
Dr. Kuldip S. Rattan
Wright State University
Committee Member

07 May, 1999
date

Daniel W. Repperger
Dr Daniel W. Repperger
Air Force Research Laboratory,
Human Effectivness Directorate
Committee Member

14 May, 1999
date

Acknowledgments

I want to thank my wife and family for being there and toughing it out during this trying period. My daughter, Brianna Nicole was born midstream of my education and became my inspiration and strength. My son, Kyle Liam, bore the negative brunt of my schooling becoming second to my homework. And my wife, Michelle Marie, kept the household together and me on track. With great appreciation, I wish to thank Dr. Curtis H. Spenny, my thesis advisor. Together and after long hours of discussion, we have produced a most admirable product. I wish to apologize for the short time frame I have expressed upon my advisor and committee members. Dr Leibst, Dr Rattan, and Dr Reppereger, thanks for coming through for me in the end. I have enjoyed learning, discussing and working with you. I also want to say thank you to Dr. Paul King,my academic advisor, and Dr. Palazotto for being there to listen to my ideas. I especially want to thank Karen Dobbyn for all her hard work in helping me get this thesis completed in the proper format. I wish to express my sincerest thanks to Scott Isabella and Randy Green of the CAVE facility for all their hard work and help in accomplishing a working demonstration. Last but not least, I would like to thank all my friends, who have helped me along the way, especially JIM YEROVI for proof reading and editing my thesis.

Table of Contents

Page

Acknowledgments.....v

Table of Contents.....vi

List of Figures.....ix

List of Tables.....xi

Abstract.....xii

Chapter 1. Introduction.....1

1.1 Overview.....1

1.2 Background.....3

1.2.1 UAV Usage.....4

1.2.2 Navigation.....6

1.2.3 UAV Control.....8

1.3 Problem Statement.....9

1.4 Objectives.....10

1.5 Accomplishments.....11

1.6 Air Force Impacts.....13

Chapter 2. Closely Supervised Reactive Control Theory.....15

2.1 Frame of References.....15

2.2 Camera Angles.....20

2.2.1 Range Information.....21

2.2.2 Virtual Camera Angles.....23

2.2.3 Desired Camera Angles.....24

page

2.2.4 Delta Camera Deflections....27

2.3 Governing Equation for Aircraft Control....28

Chapter 3. CSRC's Automatic Flight-Control System....30

3.1 Waypoint Definition by Operator Mode....30

3.1.1 Operator Modes....30

3.1.1.1 Operator Mode One....30

3.1.1.2 Operator Mode Two....32

3.1.1.3 Operator Mode Three....33

3.1.1.4 Operator Mode Four....34

3.2 Aircraft Flight Modes....35

3.3 Velocity Hold....42

3.4 Altitude Hold....45

3.5 Automatic Control Theory....48

3.5.1 Classical Control Theory....48

3.5.2 Modern Control Theory....49

Chapter 4. Cave Simulation....50

4.1 CAVE Facility....50

4.2 Simulation vs. Real....54

4.3 Simulation Hierarchy....55

4.4 Source Code....57

4.5 Additional Code....58

4.6 Results....58

page

Chapter 5. Conclusion/Recommendations....65

Appendix A. Demonstration Source Code....68

Appendix B. Triangulation Method....116

Bibliography....124

Vita....125

List of Figures

Page

Figure 1. Notional Employment Profiles..5

Figure 2. Multiple Waypoint Navigation..7

Figure 3. Waypoint Bypassing..7

Figure 4. Predator Workstation..9

Figure 5 CSRC Waypoint Usage..13

Figure 6. Notation for Body Axes..16

Figure 7. Relationship between Body and Inertial Axes Systems........................17

Figure 8. Camera Angles with respect to Body Frame..19

Figure 9. Turn Range..36

Figure 10. Aircraft in Level Turn..39

Figure 11. Flight Mode One Roll Schedule..40

Figure 12. Scheduled Roll Rate Command 42

Figure 13. Thrust Command Flowchart..44

Figure 14. Scheduled Pitch Rate Command for Theta Error................................47

Figure 15. Haptic Joystick..51

Figure 16. Thrustmaster Front View..52

Figure 17. Thrustmaster Rear View..52

Figure 18. Simulation Structure..55

Figure 19. Mission Profile..59

Figure 20. Aircraft Commands..60

Page
Figure 21. Right Turn....61

Figure 22. Δε Camera Angles....62

Figure 23. Altitude Hold....63

Figure 24. Velocity Hold....64

Figure 25. Triangulation Overview....116

Figure 26. XZ Plane....117

List of Tables

Page

Table 1. UAV Control....................11

Table 2. Flight Modes....................11

Table 3. Operator Modes....................12

Table 4. Target Bubbles....................35

Table 5. Deflection Damping....................38

Table 6. Operator Inputs....................53

Table 7. Source Code Subroutines....................57

AFIT/GAE/ENY/99J-01

Abstract

Currently, control of an uninhabited aerial vehicle (UAV) in flight is accomplished by manual control or a prior prescription of waypoints. The use of waypoints requires knowledge of vehicle position from either an Internal Navigation System (INS) or by using the Global Positioning System (GPS). This thesis proposes an alternative control method that incorporates some of the beneficial aspect of both fully manual and fully autonomous operation. Utilizing an on-board camera, an operator can control an uninhabited aerial vehicle by manually choosing desired targets of interest. The flight path of the uninhabited vehicle is determined autonomously from the camera gimbal angles. Specifically, the camera azimuth angle and elevation angle are transformed by an autopilot, providing commands to the aircraft. In this shared control operation, the operator of the payload (i.e. camera), has close supervision of the aircraft. The aircraft using an on-board computer is given autonomous control of aircraft flight, reducing personnel requirements. The aircraft controls the operations to alter flight path to reorient the aircraft to fly towards a target and at a specified range, loiter over the target. In the most basic mode of operation, the camera operator must manually track the target providing continuous updates to the camera angles. In an advanced mode of operation with the use of an INS or GPS, the aircraft autonomously determines the camera angles from a single locked target position that the operator specifies. The camera angles autonomously determined are referred to as virtual camera angles and are used to control the aircraft in the same manner as real camera angles. With the use of the virtual camera angles, the operator is free to look for other targets or perform other tasks. As an added safe mode, in the event of data transmission loss, the

aircraft will fly straight and level in its current direction. This control method is being validated in the CAVE Automated Virtual Environment (CAVE) facility located at Wright Patterson AFB and owned and operated by Wright State University. The CAVE is a virtual environment with the projection of a 3-D scene onto the four walls and floor of a 3.1m x 3.1m room. The use of the CAVE facility provides a test bed to evaluate different modes of operator control and human interface factors associated with the control of an UAV. The aircraft flight simulator used is FLSIM with the flight characteristics of an F-16 aircraft.

CLOSELY SUPERVISED REACTIVE CONTROL OF AN UNINHABITED AERIAL VEHICLE

Chapter 1. Introduction

1.1 Overview

With the continuing advances in technology, more autonomous aircraft are becoming available, and as the level of autonomy increases, the level of manual control decreases. For UAV operation, the level of manual and autonomous operation to optimize mission effectiveness is one of the most significant developmental issues. This thesis is intended to demonstrate shared control, one in which the aircraft can navigate autonomously with the human providing close supervision [11]. The level of shared control used incorporates some of the beneficial aspects of both fully manual and fully autonomous operation. The 'man-in-the-loop' will manually provide positioning commands for the uninhabited aerial vehicle via an onboard camera. The aircraft will accept the commands and autonomously determine the required flight profile to reach the desired position. This configuration allows the human to concentrate on where the aircraft is and what information is collected. The human located at a master site controls the camera manually. In the basic mode of operation, manual control of the camera is used to provide continous updates to the camera angles. The human operator can switch the aircraft to an advanced mode of operation in which the aircraft will fly autonomously using virtual camera angles based upon the aircraft position and target position. The use of these modes allows the joint control of aircraft navigation. In both modes, the aircraft uses the camera angles, real or virtual, to determine the appropriate navigation commands.

In the terminology of telerobotics, this level of control falls within the category of teleoperation in which the human is at the master site and the aerial vehicle and payload is the slave [13]. The roles assigned apply well to the currently accepted definitions of supervisory control and shared control. The aircraft is given the role in which precision, repetition, rapid response, and a high level of understanding with little variation are needed (i.e. controlling aircraft flight). The human is given the tasks to monitor the ever-changing environment and make supervisory decisions for the aircraft [13]. For example, the human is better equipped to track an evasive target.

A reactive system is a system which is designed to react to changing events in the physical world in a predictable way [10]. The missions of tactical aircraft and reconaissance UAV's are examples of reactive systems for which reaction times are relatively short. The implication of close supervision is that the reaction of the system is of sufficient importance that the operator will be attentive to assure satisfactory performance. Attentiveness may take the form of: 1) monitoring automated operation and intervening as required or 2) manually performing aspects of the mission for which automated operation is impractical. The need for close supervision of automated operation results from concern with the ability of the system to react satisfactorily under circumstances that differ from those for which the system was designed. The challenge is to design the most effective operator interface for a closely supervised reactive system.

Using a virtual environment, the CAVE Automated Virtual Environment, this thesis will demonstrate the level of shared control instantiated. This demonstration uses the FLSIM flight simulator with the flight characteristics of an F-16 aircraft. The use of this particular F-16 simulator necessitated the development of an autonomous controller with inputs to the aircraft in terms of angular rates. When the CAVE visual scene is projected to represent the operator seated in the aircraft, the operator visually senses the aircraft's angular rates. This sensing promoted the use of angular rates as the input command as opposed to control surface deflections. Neither the case of the seat fixed to the camera nor fixed inertially (bird's eye view) was evaluated. This demonstration provides a testbed for evaluation of other levels of shared control and various human interface factors.

1.2 Background

A discussion of what is currently available is included to aid the understanding of the characteristics "closely supervised reactive control (CSRC) of an uninhabited aerial vehicle (UAV)" offers. The UAV reacts by steering the aircraft utilizing an automatic flight-control system (AFCS). Aircraft steering by an AFCS is accomplished via "en route navigation" or "terminal approach" navigation. In the course of developing a working demonstration, an AFCS was designed to include velocity control and altitude hold. This task proved difficult from the aspect that the only input commands available with FLSIM are stick commands. The stick commands that a human pilot does instinctively from visual and kinesthetic cues are not easily translated into control logic. However, the use of an F-16 as the simulator platform made this development easier since the inputs are angular rates rather than control surface deflections. In order, to evaluate the strengths and weaknesses of the AFCS logic developed to control velocity and altitude, a description of current control theories is discussed in the aircraft autopilot design section.

1.2.1 UAV Usage

Given the unique features that UAVs can provide to operational flexibility, the various methods of employment need to be determined. The questions, how, what, where and why must be answered to utilize UAVs effectivley. Figure 1 shows various possible employments of UAVs. Current applications of UAVs in the Air Force are high altitude and medium altitude reconaissance. Current levels of technology available in payload sensors like radar, ultraviolet, infrared and visual make reconnaisance at these altitudes possible. Combining this possibility with the low threat and minimally changing environment, make the use of autonomous UAVs a strong desireability. In the future, UAVs may be employed in combat missions such as standoff search and destroy or close combat support. These missions operate in a more aggressive environment and require UAV operational flexibility. The need for flexibility may necessitate some manual control of the UAV. The Air Force is currently addressing the level of autonomous control for tactical UAV. Figure 1 also shows a brief description of the communication and navigation systems currently in place to support Air Force requirements. Without this vast system, UAV operational deployments would not be possible or plausible. This system allows UAVs to navigate successfully and communicate effectively.

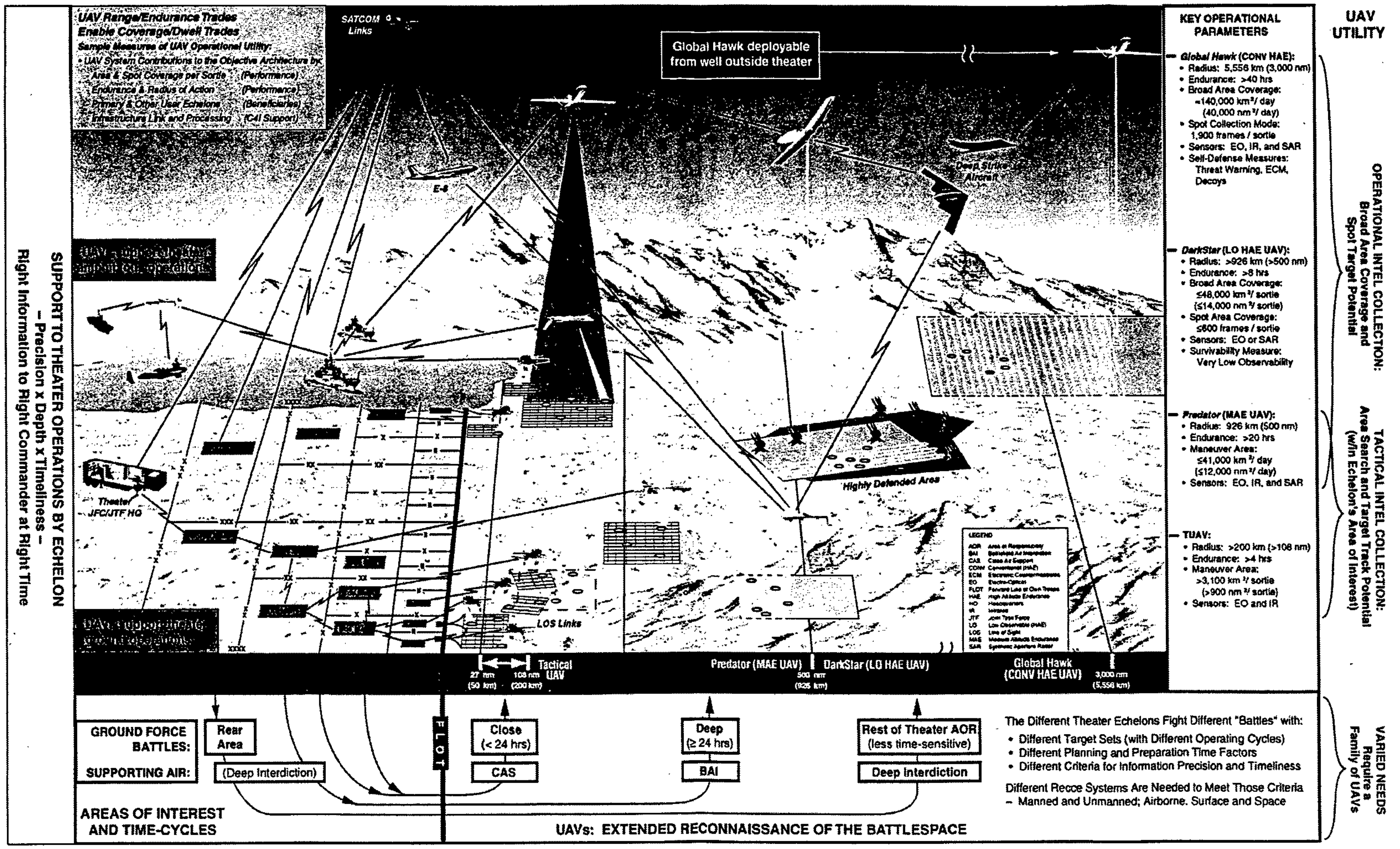

UAV Range/Endurance Trades
Enable Coverage/Dwell Trades
Sample Measures of UAV Operational Utility:
• UAV System Contributions to the Objective Architecture by:
Area & Spot Coverage per Sortie (Performance)
Endurance & Radius of Action (Performance)
Primary & Other User Echelons (Beneficiaries)
Infrastructure Link and Processing (C4I Support)
SATCOM Links
Global Hawk deployable from well outside theater
E-8
Deep Strike Aircraft
Theater JFC/JTF HQ
Highly Defended Area
LOS Links
LEGEND
AOR Area of Responsibility
BAI Battlefield Air Interdiction
CAS Close Air Support
CONV Conventional (HAE)
ECM Electronic Countermeasures
EO Electro-Optical
FLOT Forward Line of Own Troops
HAE High Altitude Endurance
HQ Headquarters
IR Infrared
JTF Joint Task Force
LO Low Observable (HAE)
LOS Line of Sight
MAE Medium Altitude Endurance
SAR Synthetic Aperture Radar
27 nm (50 km)
108 nm (200 km)
Tactical UAV
Predator (MAE UAV)
500 nm (926 km)
DarkStar (LO HAE UAV)
Global Hawk (CONV HAE UAV)
3,000 nm (5,556 km)
KEY OPERATIONAL PARAMETERS
Global Hawk (CONV HAE):
• Radius: 5,556 km (3,000 nm)
• Endurance: >40 hrs
• Broad Area Coverage: ≈140,000 km²/ day (40,000 nm²/ day)
• Spot Collection Mode: 1,900 frames / sortie
• Sensors: EO, IR, and SAR
• Self-Defense Measures: Threat Warning, ECM, Decoys
DarkStar (LO HAE UAV):
• Radius: >926 km (>500 nm)
• Endurance: >8 hrs
• Broad Area Coverage: ≤48,000 km²/ sortie (≤14,000 nm²/ sortie)
• Spot Area Coverage: ≤600 frames / sortie
• Sensors: EO or SAR
• Survivability Measure: Very Low Observability
Predator (MAE UAV):
• Radius: 926 km (500 nm)
• Endurance: >20 hrs
• Maneuver Area: ≤41,000 km²/ day (≤12,000 nm²/ day)
• Sensors: EO, IR, and SAR
TUAV:
• Radius: >200 km (>108 nm)
• Endurance: >4 hrs
• Maneuver Area: >3,100 km²/ sortie (>900 nm²/ sortie)
• Sensors: EO and IR
UAV UTILITY
OPERATIONAL INTEL COLLECTION: Broad Area Coverage and Spot Target Potential
TACTICAL INTEL COLLECTION: Area Search and Target Track Potential (w/in Echelon's Area of Interest)
VARIED NEEDS Require a Family of UAVs
GROUND FORCE BATTLES:
SUPPORTING AIR:
Rear Area
(Deep Interdiction)
FLOT
Close (< 24 hrs)
CAS
Deep (≥ 24 hrs)
BAI
Rest of Theater AOR (less time-sensitive)
Deep Interdiction
The Different Theater Echelons Fight Different "Battles" with:
• Different Target Sets (with Different Operating Cycles)
• Different Planning and Preparation Time Factors
• Different Criteria for Information Precision and Timeliness
Different Recce Systems Are Needed to Meet Those Criteria
– Manned and Unmanned; Airborne, Surface and Space
AREAS OF INTEREST AND TIME-CYCLES
UAVs: EXTENDED RECONNAISSANCE OF THE BATTLESPACE
SUPPORT TO THEATER OPERATIONS BY ECHELON
– Precision x Depth x Timeliness –
Right Information to Right Commander at Right Time

Figure 1. Notional Employment Profiles [13]

1.2.2 Navigation

Basic aircraft control requires steering, the determination of the change in vehicle motion required to obtain a desired result. Steering can be accomplished directly from a pilot applying stick commands or by an automatic flight-control system (AFCS) using steering errors [12]. AFCS's reduces operator workloads by automatically navigating an aircraft. This navigation can be accomplished by "en route" navigation or "terminal" navigation. It is en-route navigation that is related to the reconaissance task which this thesis addresses.

"En route" navigation operates in a "fly-to" mode, in which the aircraft navigates between two points. This can be done either radially or directly. In "en route" navigation, course changes are determined from the error in the aircraft position and a selected waypoint. This method of navigation is also known as "waypoint" navigation. The waypoints are defined as geographic points in terms of latitude and longitude or bearing and distance [12]. They can be combined to form a set of waypoints defining a mission profile. Each waypoint can have a change in course heading, speed, altitude or any other mission dependent parameter associated with it. As the aircraft reaches a waypoint, a single change or a combination of changes occurs. Figure 2 shows the use of multiple waypoint to change course headings along a mission profile. Figure 3 indicates the information necessary when transitioning to fly toward another waypoint.

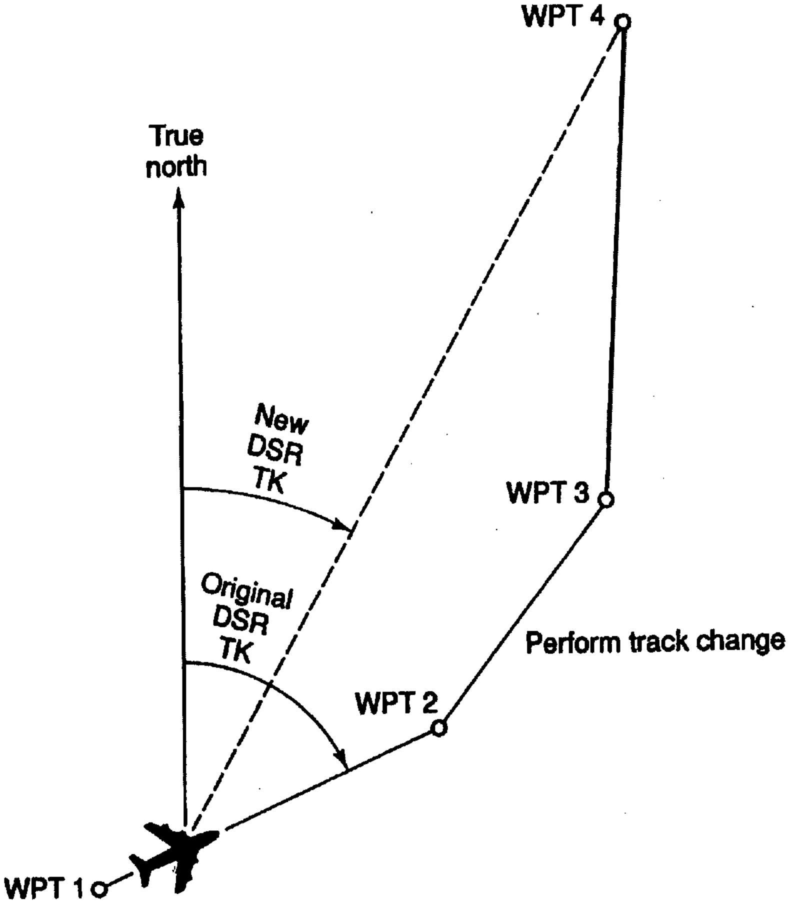

WPT 4
True
north
New
DSR
TK
WPT 3
Original
DSR
TK
Perform track change
WPT 2
WPT 1

Figure 2. Multiple Waypoint Navigation [12]

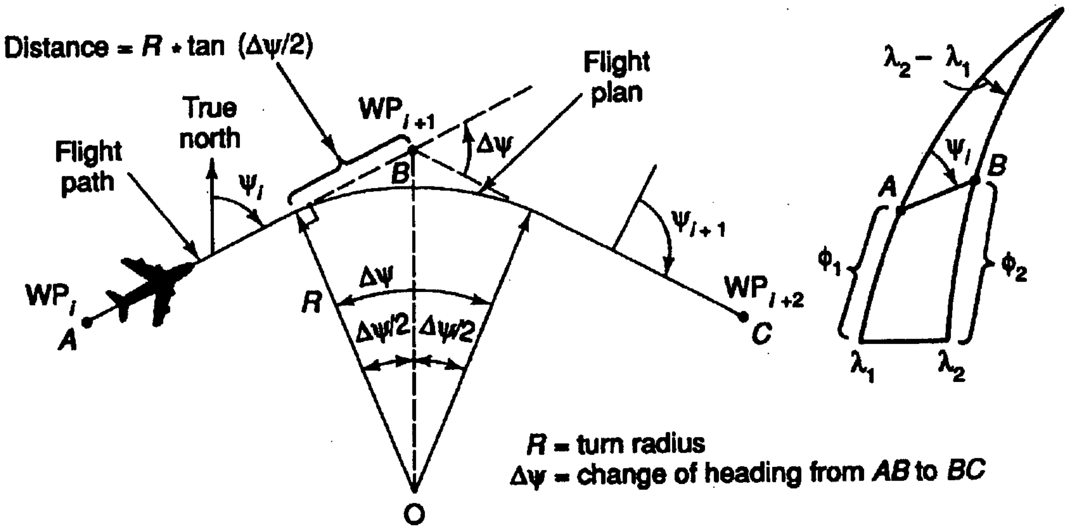

Distance = R * tan (Δψ/2)
Flight plan
True north
WP_{i+1}
Δψ
Flight path
ψ_i
B
ψ_{i+1}
WP_i
A
R
Δψ
Δψ/2
Δψ/2
WP_{i+2}
C
O
λ_2 − λ_1
ψ_i
A
B
φ_1
φ_2
λ_1
λ_2
R = turn radius
Δψ = change of heading from AB to BC

Figure 3. Waypoint Bypassing [12]

"Terminal" navigation provides relative aircraft position to a selected touchdown point [12]. This mode of navigation can be used for autonomous landing guidance or for missile guidance to target. With the use of radar, dead reckoning and/or forward-looking infrared (FLIR), terminal navigation to the endpoint is accomplished [12]. Course corrections are determined from the error in aircraft position and termination point. At great distances, the definition of the termination point does not have to be accurate. The definition can be updated as the termination point is approached increasing navigation accuracy.

1.2.3 UAV Control

Due to the level of currently available unclassified/non-proprietary information available, the discussion on current UAV control is kept to a top level. In the past and in current operations, the majority of the UAV systems available require a pilot operator to manually control the aircraft. Though the aircraft is uninhabited, its flight is not autonomous, for the aircraft itself does not determine the flight path. This manner of operation can be found in systems like the Predator, Dragon Drone, Eagle Eye and NASA's extreme altitude and long endurance (ERAST) platforms [5]. Figure 4 shows the workstation used by the Predator program. This system requires two operators. One operator controls the aircraft, while the other controls the payload. Each operator station has a set of aircraft controls, stick, throttle and rudder pedals. Each station's screen has the ability to display any aircraft instruments. This allows redundancy in aircraft control. In standard operation, the left console is for the payload operator while the right is for the aircraft operator.

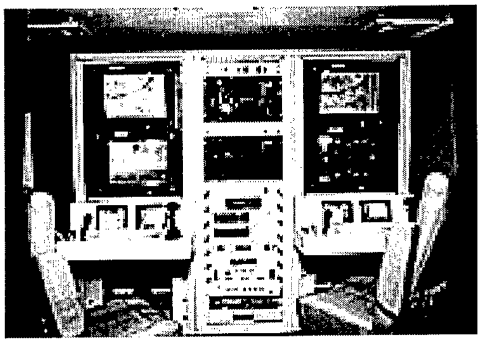


Figure 4. Predator Workstation [2]

Autonomous flight, via waypoints is used for some UAVs. A mission profile containing waypoints is developed prior to flight and the UAV flies autonomously from waypoint to waypoint [13]. This method is comparable to the waypoint navigation used by many commercial autopilots and can be found on platforms such as Cypher and Global Hawk and the recently cancelled Darkstar [4, 13]. A major drawback to this method is the difficulty with changing waypoints and the high accuracy of aircraft position required. This drawback, however, can be minimized with simplifications in waypoint defining and improvements in the navigational system. As technological advances in INS and GPS continue, the error associated with aircraft position diminishes, leaving only the degree of difficulty with changing waypoints as the only significant drawback.

1.3 Problem Statement

The overall issue is what is the best combination of manual and autonomous control to obtain the best UAV system performance. This thesis does not attempt to answer this issue but rather provides a concept for evaluation. CSRC gives the camera operator indirect control of the aircraft and gives the aircraft autonomous control of the flying.

1.4 Objectives

The objectives were to design a control theory in which the uninhabited aerial vehicle's flight is automated, while the human still retains indirect control and situational awareness of the UAV's position and to develop a demonstration for evaluating system performance. In the Predator platform, the pilot operates the aircraft based on the needs of the camera operator. CSRC is designed to reduce the Predator manpower requirements from two operators to one. In CSRC, aircraft operation is still subject to the camera operator's requirements. The role of aircraft flight, however, is removed from the pilot operator and given to the aircraft. Based upon the camera operator's requirements, the aircraft flies autonomously to a designated location. The demonstration is designed to show operator interface and aircraft operation.

In the presence of well-refined guidance systems that are able to autonomously track a target, the use of manual tracking may seem unnecessary. A desired role to be performed by CSRC is the tracking of evasive targets such as tanks. Current autonomous tracking algorithms are unable to effectively track hidden or multiple targets. This scenario requires a human to make objective decisions based upon situational awareness. To handle all possible situations the control of tracking targets was left to the human.

CSRC has characteristics of both manual control and autonomous control. In the manual mode of flight, CSRC operates like a remotely piloted UAV in direct response to the camera operator. In the shared mode, it effectively operates by creating and navigating to operator selected waypoints, leaving the operator free to scan with the camera in search of other targets. Table 1 shows a comparison of who is in control in the Predator program, Global Hawk, and CSRC.

Table 1 UAV Control

	Predator	Global Hawk	CSRC Manual Mode	CSRC Shared Mode
AirCraft Navigation	**Pilot Operator**	**Autonomous**	**Camera Operator**	**Autonomous**
Payload Control	**Camera Operator**	**Autonomous**	**Camera Operator**	**Camera Operator**

CSRC was not developed with any specific employment criteria. An objective of CSRC was that it be able to be incorporated into any mission platform. However, emphasis was placed on the role of tactical UAVs where the role of search and target tracking is the mission.

1.5 Accomplishments

In the development of this thesis, a working demonstration of the theory was accomplished. The demonstration shows that the theory can be applied to virtually any current UAV/aircraft, and various modes of operation can be employed. The modes of operation are broken into two categories: aircraft flight modes, determined autonomously by the aircraft and operator control modes, determined manually through the operator interface. The aircraft autonomously selects one of two modes in which to operate as shown in table 2.. In flight mode one, the aircraft turns and flies a straight path at a constant altitude toward a selected target. In flight mode two, the aircraft will circle a selected target at a determined standoff range and constant altitude.

Table 2 Flight Modes

Flight Mode	Description
Flight Mode One	**Turn and fly towards target**
Flight Mode Two	**Circle Target**

The decision to switch between flight modes one and two is done autonomously by the aircraft based upon distance from target. The operator interface allows four modes of controls. Operator mode 1 (Free fly mode), the aircraft responds to all camera inputs, as the operator manually manuevers the camera the aircraft will follow. In operator mode two (Target locked mode), the aircraft flies towards a specified target and once within turn range of the target, the aircraft will circle the specified target. In this mode, the target is fixed and the camera is free to move. In operator mode three (Straight and level), the aircraft flies straight and level on the current heading while the camera is free to move. In operator mode four (Level turn), the aircraft flies a level turn at a fixed bank determined by the operator, while the camera is free to move. Table 3 shows the flight modes and operator modes and their relationships.

Table 3 Operator Modes

	Flight Mode One	Flight Mode Two	Camera Angles
Free Fly	✓	✓*	**Real**
Target Locked	✓	✓	**Virtual**
Straight and Level	✓	**NO**	**Virtual**
Level Turn	✓	**NO**	**Virtual**

* This operation is not included in the demonstration

All modes of operation have been successfully demonstrated in the CAVE facility. In addition, the theory is demonstrated to be robust enough to handle re-targeting at any point in flight operations. This ability allows the camera operator to track a moving target or a new target without the aircraft responding abnormally. In the course of developing the demonstration, it was necessary to develop a method for controlling altitude and velocity using only standard throttle and flight stick commands.

Both have been accomplished successfully with an adequate level of accuracy and stability.

Steering of the UAV by CSRC was accomplished via "en-route" or waypoint navigation. Figure 5 demonstrates how waypoints are used by CSRC. Instead of altering heading towards a new waypoint, when the waypoint is reached the aircraft enters a circling pattern around the waypoint until a new waypoint is defined.

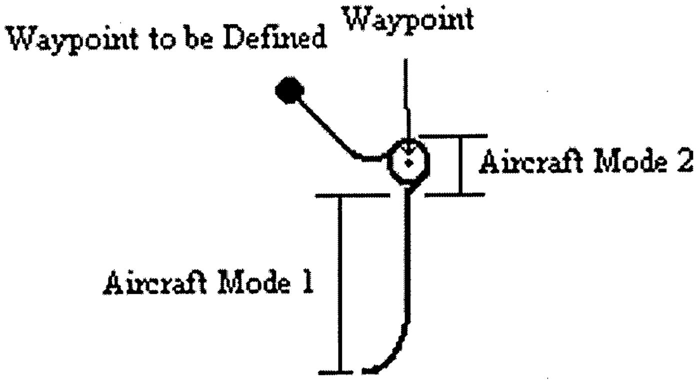


Figure 5. CSRC Waypoint Usage

Waypoints are established by the operator using a joystick to position a "target bubble" on the desired target. In order to determine the waypoints in a global coordinate system, aircraft position must be known. This thesis assumes that this information will always be available from either an INS or GPS. In operator modes two, three and four the camera angles are determined virtually from the aircraft position and the waypoint position, both in global coordinates.

1.6 Air Force Impact

This thesis addresses an alternative way of controlling UAV's from what is currently used by the Air Force. The value of this method, CSRC, is not addressed but left for future assessment. Of more important interest than CSRC as a whole are the components of the control method developed. In the development of a working demonstration, different algorithms were required to obtain the necessary information to operate correctly. The various aspects, taken individually, could have significant impacts on different Air Force programs. As a minimum, the employment of this control method as a whole could reduce the operator requirements for programs such as the Predator. Individually, the algorithm for waypoint definition could be employed by Global Hawk reducing operator workload and time requirements. Additionally, modifying the algorithm for waypoint definition leads to a method capable of triangulating onto a target using only two aircraft or one aircraft taking two separated measurements and obtaining the same level of accuracy as three.

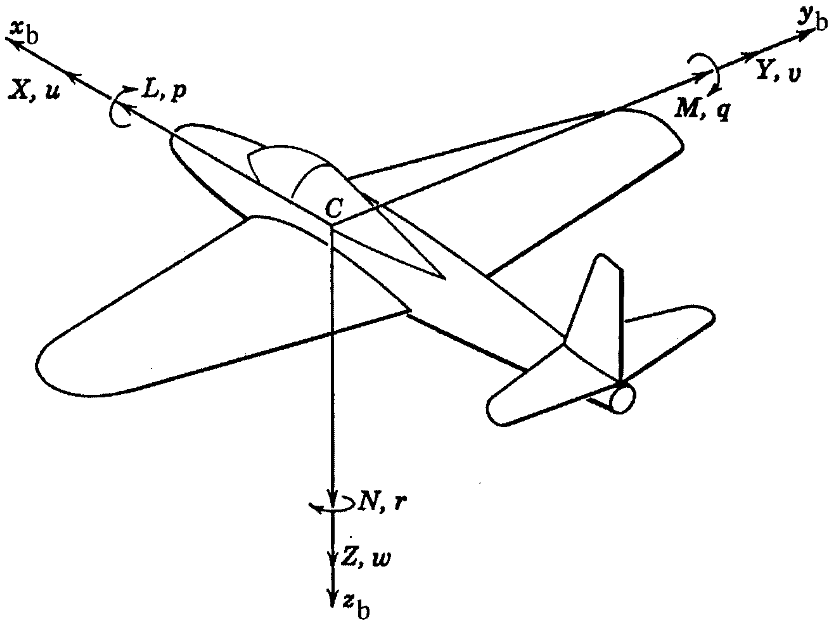
Chapter 2. Closely Supervised Reactive Control Theory

2.1 Frame of Reference

In order to understand CSRC, it is necessary to understand the various frames of reference used. The first frame of reference is a frame fixed to the Earth where North is represented by the x axis, East is by the y axis, and z is positive in downward vertical direction toward the center of the earth [6]. This frame has it's origin at 0.0 degrees latitude and 0.0 degrees longitude and is referred to as the Earth fixed frame. The Earth fixed frame allows the determination of aircraft position and target position in terms of global coordinates.

A second frame with the same axes as the Earth fixed frame is attached to the center of gravity of the aircraft. It is important to accept that this axes sytsem is fixed to the center of gravity of the aircraft with no rotation allowed. This frame is only allowed to translate from the Earth fixed frame. This frame will be referred to as the vehicle carried vertical frame or fixed inertial frame [6].

In the development of the equations of motion for an aircraft, an axis system fixed to the aircraft is used, known as the body frame, b. A discussion of the equations of motion is limited to defining the moments and the angular rates and the axes about which they rotate. The first moment and rate is the rolling moment (L) and roll rate (p). This set rotates about the x axis; the axis out the nose of the aircraft. The second set is the pitching moment (M) and rate of pitch (q), which rotate about the y axis; the axis out the right wing of the aircraft. The third moment and rate is the yawing moment (N) and yaw rate (r). This moment and angular rate rotate about the z axis; the axis straight down from the aircraft [7]. All axes are assumed to originate at the center of gravity. Figure 6 shows a graphical representation the fixed body frame.



L = rolling moment p = rate of roll
M = pitching moment q = rate of pitch
N = yawing moment r = rate of yaw
$[X, Y, Z]$ = components of resultant aerodynamic force
$[u, v, w]$ = components of velocity of C relative to atmosphere

Figure 6. Notation for Body Axes [7]

The orientation of any reference frame can be described by three consecutive rotations whose order is important [9]. The angles associated with these rotations are known as the Euler angles. The first rotation is about the z_b axis and is known as the yaw angle, ψ. The second rotation is about the y_1 axis and is referred to as the pitch angle, θ. The final rotation is about the x_2 axis and is called the roll angle, ϕ. Figure 7 shows the three rotations and the relationship between the body (b) and inertial (f) axes systems.

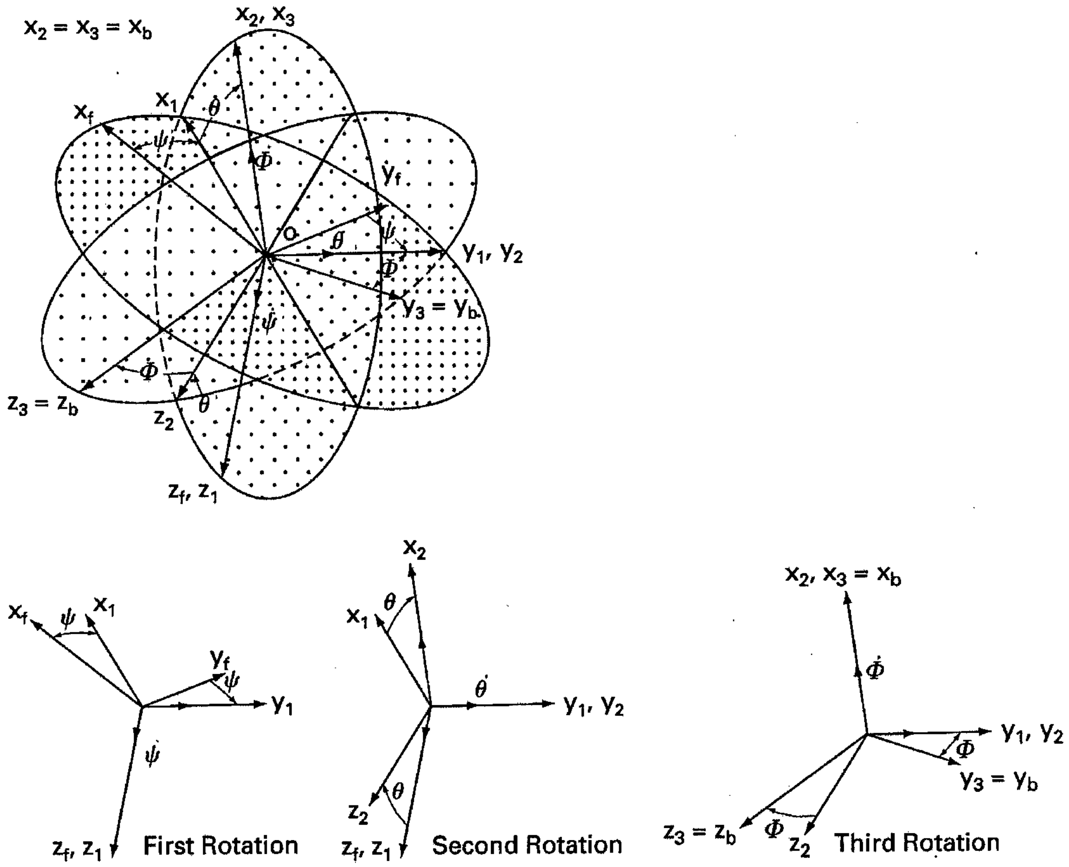

$x_2 = x_3 = x_b$
x_2, x_3
x_f
x_1
θ
ψ
$\dot{\Phi}$
y_f
O
$\dot{\theta}$
$\dot{\psi}$
y_1, y_2
Φ
$y_3 = y_b$
$\dot{\psi}$
Φ
θ
$z_3 = z_b$
z_2
z_f, z_1
x_f
ψ
x_1
y_f
ψ
y_1
$\dot{\psi}$
z_f, z_1
First Rotation
x_2
x_1
θ
$\dot{\theta}$
y_1, y_2
θ
z_2
z_f, z_1
Second Rotation
$x_2, x_3 = x_b$
$\dot{\Phi}$
y_1, y_2
Φ
$y_3 = y_b$
$z_3 = z_b$
Φ
z_2
Third Rotation

Figure 7. Relationship between Body and Inertial Axes Systems [9]

With the Euler angles, the flight velocity components in the fixed inertial frame $\left(\frac{dx}{dt}, \frac{dy}{dt}, \frac{dz}{dt}\right)$, can be determined from the velocity components in the body frame (u, v, w), using matrix algebra. This is accomplished by multiplying the three rotational matrices to obtain a transformation matrix as shown below [3].

$$\begin{Bmatrix} \frac{dx}{dt} \\ \frac{dy}{dt} \\ \frac{dz}{dt} \end{Bmatrix} = \left[L^{BI}\right] \begin{Bmatrix} u \\ v \\ w \end{Bmatrix} \tag{1}$$

where

$$L^{BI} = \begin{bmatrix} \cos\Theta\cos\Psi & \sin\Phi\sin\Theta\cos\Psi - \cos\Phi\sin\Psi & \cos\Phi\sin\Theta\cos\Psi + \sin\Phi\sin\Psi \\ \cos\Theta\sin\Psi & \sin\Phi\sin\Theta\sin\Psi + \cos\Phi\cos\Psi & \cos\Phi\sin\Theta\sin\Psi - \sin\Phi\cos\Psi \\ -\sin\Theta & \sin\Phi\cos\Theta & \cos\Phi\cos\Theta \end{bmatrix} \quad (2)$$

The superscript, BI, denotes that this rotational matrix determines components in the body frame from the components in the inertial frame. Conversely, the inverse matrix L^{IB} would indicate components in the inertial frame in terms of those in the body frame. This annotation will be used consistently throughout this thesis.

Similar to the relationship between the axes velocities, a relationship between the angular velocities and the Euler rates, $\left(\dot{\Phi} \quad \dot{\Theta} \quad \dot{\Psi}\right)$, is obtained using a jacobian matrix.

$$\begin{Bmatrix} p \\ q \\ r \end{Bmatrix} = \begin{bmatrix} 1 & 0 & -\sin\Theta \\ 0 & \cos\Phi & \cos\Theta\sin\Phi \\ 0 & -\sin\Phi & \cos\Theta\cos\Phi \end{bmatrix} \begin{Bmatrix} \dot{\Phi} \\ \dot{\Theta} \\ \dot{\Psi} \end{Bmatrix} \equiv \left[J^{BI}\right] \begin{Bmatrix} \dot{\Phi} \\ \dot{\Theta} \\ \dot{\Psi} \end{Bmatrix} \quad (3)$$

Conversely, we can determine the Euler rates in terms of the angular rate through the following inverse jacobian matrix.

$$\begin{Bmatrix} \dot{\Phi} \\ \dot{\Theta} \\ \dot{\Psi} \end{Bmatrix} = \begin{bmatrix} 1 & \sin\Phi\tan\Theta & \cos\Phi\tan\Theta \\ 0 & \cos\Phi & -\sin\Phi \\ 0 & \sin\Phi\sec\Theta & \cos\Phi\sec\Theta \end{bmatrix} \begin{Bmatrix} p \\ q \\ r \end{Bmatrix} \equiv \left[J^{-BI}\right] \begin{Bmatrix} p \\ q \\ r \end{Bmatrix} = \left[J^{IB}\right] \begin{Bmatrix} p \\ q \\ r \end{Bmatrix} \quad (4)$$

Lastly, a fourth reference frame is required. This frame is fixed to the camera that rotates with respect to the aircraft body frame. Two angles are required to locate the camera frame with respect to the body frame. The first is the azimuth angle defined as ε and is about the z_b axis. The second is the elevation angle defined as λ and is about the carried x_b axis. Rotation about the y_c axis, the line-of-sight direction of the camera is unnecessary to solve the problem. It therefore is not included in the rotation matrix. Figure 8 shows the sensor angles with respect to the body axis.

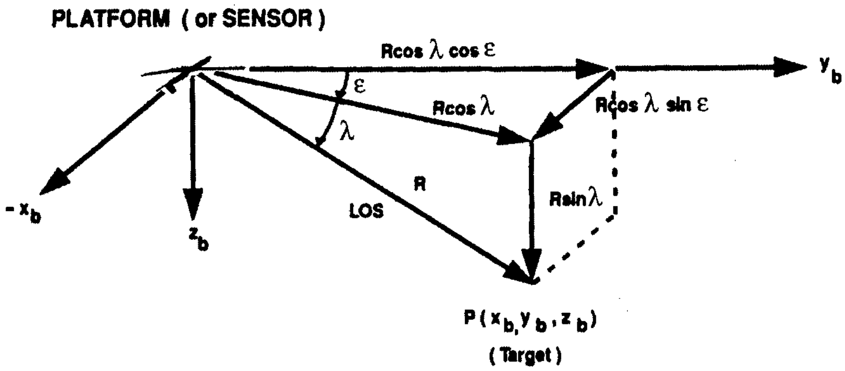


Figure 8. Camera Angles with respect to the Body Frame [12]

With the definition of the camera angles, a rotational matrix relating components in the camera frame with those in the body frame is defined as L^{CB}, where:

$$L^{CB} = \begin{bmatrix} \cos\varepsilon & \sin\varepsilon & 0 \\ -\cos\lambda\sin\varepsilon & \cos\lambda\cos\varepsilon & \sin\lambda \\ \sin\lambda\sin\varepsilon & -\sin\lambda\cos\varepsilon & \cos\lambda \end{bmatrix} \tag{5}$$

Using a jacobian matrix, the angular rates of the camera can be determined in the aircraft body frame as shown below.

$$\begin{Bmatrix} \omega_x^C \\ \omega_y^C \\ \omega_z^C \end{Bmatrix}_B = \begin{bmatrix} -\sin\varepsilon & 0 \\ \cos\varepsilon & 0 \\ 0 & 1 \end{bmatrix} \begin{Bmatrix} \dot{\lambda} \\ \dot{\varepsilon} \end{Bmatrix} \tag{6}$$

CSRC places the origin of all frames of reference, except for the Earth fixed frame, at the center of gravity of the aircraft. This assumption relieves the need to account for any translation, requiring only the rotations.

2.2 Camera Angles

CSRC requires the steady state or nominal camera angles to be determined in order to operate correctly. These camera angles are referred to as the desired camera angles and are a function of aircraft orientation and flight mode. To correctly calculate the camera angles certain information is required. In the first flight mode (turn and fly towards the target), the range to the target is needed besides the three Euler angles and the two camera angles. This range can be obtained either from an onboard sensor or calculated with the five required angles and the height above target. When calculated from the height above target, the range accuracy is affected by the assumption that the target is at sea level and the height above target is the aircraft's true altitude, which may not be true. With the use of radar to measure height above terrain providing the absolute altitude, range accuracy can be increased but is still subject to the assumption that the target is at the same terrain height below the aircraft. In the three operator modes, (free fly, straight and level, and coordinated level turn), accuracy of range is not important. This is because range is only used to calculate the desired angles and the calculations will produce the same results regardless of range as long as there is similarity. The similarity required is discussed in the section on desired camera angles. Note that all three of these modes correspond with flight mode one (turn and fly towards the target). Operator mode two, in which the UAV flies to and circles the target, does require a higher level of accuracy in order to calculate the range at which to start circling. Along with the determination of entry into flight mode two (circling), continued operation in this mode requires accurate updates of range in order to function correctly. Operator mode two as simulated requires an additional piece of information. In order to fix the target in the

Earth Fixed frame, it is necessary to know the global coordinates of the aircraft's position. This can be obtained from an onboard inertial navigation system (INS) or from GPS. The source of the information is immaterial, only the accuracy. For continued operation, updates to aircraft position are necessary. It will be shown that from a locked target position and from aircraft position, camera angles can be determined as if the camera was continously tracking the target. These camera angles will be referred to as virtual camera angles. The case in which aircraft position was not available is not demonstrated and will be addressed in the simulation chapter.

2.2.1 Range Information

In this section, it will be shown how range can be obtained from only the height above the target and the five angles. Recalling the two rotational matrices, L^{CB} and L^{BI}, equations (5) and (2), the rotational matrix, L^{CI}, is developed through the matrix multiplication of these two matrices.

$$\left[L^{CI}\right]=\left[L^{CB}\right]\left[L^{BI}\right]=\begin{bmatrix} L_{11} & L_{12} & L_{13} \\ L_{21} & L_{22} & L_{23} \\ L_{31} & L_{32} & L_{33} \end{bmatrix} \tag{7}$$

where

$$
\begin{aligned}
&L_{11} = \cos\varepsilon\cos\Theta\cos\Psi + \sin\varepsilon(\sin\Phi\sin\Theta\cos\Psi - \cos\Theta\sin\Psi)\\
&L_{12} = \cos\varepsilon\cos\Theta\sin\Psi + \sin\varepsilon(\sin\Phi\sin\Theta\sin\Psi + \cos\Theta\cos\Psi)\\
&L_{13} = -\cos\varepsilon\sin\Theta + \sin\varepsilon\sin\Phi\cos\Theta\\
&L_{21} = -\cos\lambda\sin\varepsilon\cos\Theta\cos\Psi + \cos\lambda\cos\varepsilon(\sin\Phi\sin\Theta\cos\Psi - \cos\Phi\sin\Psi)\\
&+\sin\lambda(\cos\Phi\sin\Theta\cos\Psi + \sin\Phi\sin\Psi)\\
&L_{22} = -\cos\lambda\sin\varepsilon\cos\Theta\sin\Psi + \cos\lambda\cos\varepsilon(\sin\Phi\sin\Theta\sin\Psi + \cos\Phi\cos\Psi)\\
&+\sin\lambda(\cos\Phi\sin\Theta\sin\Psi - \sin\Phi\cos\Psi)\\
&L_{23} = \cos\lambda\sin\varepsilon c\sin\Theta + \cos\lambda\cos\varepsilon\sin\Phi\cos\Theta + \sin\lambda\cos\Phi\cos\Theta\\
&L_{31} = \sin\lambda\sin\varepsilon\cos\Theta\cos\Psi - \sin\lambda\cos\varepsilon(\sin\Phi\sin\Theta\cos\Psi - \cos\Phi\sin\Psi)\\
&+\cos\lambda(\cos\Phi\sin\Theta\cos\Psi + \sin\Phi\sin\Psi)\\
&L_{32} = \sin\lambda\sin\varepsilon\cos\Theta\sin\Psi - \sin\lambda\cos\varepsilon(\sin\Phi\sin\Theta\sin\Psi + \cos\Phi\cos\Psi)\\
&+\cos\lambda(\cos\Phi\sin\Theta\sin\Psi - \sin\Phi\cos\Psi)\\
&L_{33} = -\sin\lambda\sin\varepsilon\sin\Theta - \sin\lambda\cos\varepsilon\sin\Phi\cos\Theta + \cos\lambda\cos\Phi\cos\Theta
\end{aligned}
\tag{8}
$$

This matrix allows the determination of position vector in the camera frame relative to that in the fixed inertial frame.

$$
\begin{Bmatrix} 0 \\ R \\ 0 \end{Bmatrix}_C = \left[L^{CI}\right] \begin{Bmatrix} Dx \\ Dy \\ Dz \end{Bmatrix}_I \tag{9}
$$

where

$$
R_c = \sqrt{Dx_I^2 + Dy_I^2 + Dz_I^2} \tag{10}
$$

Dx_I, Dy_I, and Dz_I are the inertial frame components of the target position with respect to the UAV. Because the direction cosine matrix, L^{CI}, is an orthogonal matrix the range in the inertial frame is equal to the range in the camera frame. Solving equation (9) with a known Dz_I, the following is obtained:

$$
\begin{aligned}
Dx_I &= \left(\frac{L_{32}L_{13} - L_{33}L_{12}}{L_{31}L_{12} - L_{11}L_{32}}\right) * Dz_I\\
Dy_I &= \left(\frac{L_{11}L_{33} - L_{31}L_{13}}{L_{31}L_{12} - L_{11}L_{32}}\right) * Dz_I
\end{aligned}
\tag{11}
$$

As shown, the range can be determined from just Dz_I and the five rotation angles. Using the method previously discussed to determine X, Y, and Z positions in the global coordinates, a more accurate method to triangulate onto a target using two data points is addressed in Appendix B.

2.2.2 Virtual Camera Angles

CSRC can operate in a shared mode in which the camera operator is free to move the camera without impacting flight. This mode is implemented through the determination of virtual camera angles. Virtual camera angles are the angles the camera would have if the camera continued to track the target. A method to calculate these camera angles is necessary. The calculations below are the ones used by the simulator to determine not only the virtual camera angles but, for simulation purposes, the real camera angles as well and will be discussed in chapter 3. Using equation (2), the rotational matrix relating the inertial frame to the body frame, the distances to the target in the body frame are found from the Euler angles and the distances in the inertial frame:

$$\begin{Bmatrix} Dx \\ Dy \\ Dz \end{Bmatrix}_B = \left[L^{BI}\right]\begin{Bmatrix} Dx \\ Dy \\ Dz \end{Bmatrix}_I \tag{12}$$

where, Dx, Dy, and Dz are the distances from the aircraft to the target in the respective subscripted frames. The solution to equation (12), the distances in the body frame, is then incorporated into the following equation:

$$\begin{Bmatrix} 0 \\ R \\ 0 \end{Bmatrix}_C = \left[L^{CB}\right]\begin{Bmatrix} Dx \\ Dy \\ Dz \end{Bmatrix}_B \tag{13}$$

where

$$R_C = \sqrt{Dx_B^2 + Dy_B^2 + Dz_B^2} \tag{14}$$

R_C could have also been found using equation (10), since the range is the same in all frames. Solving equation (13), the virtual camera angles are found to be the following:

$$\begin{aligned} \tan\varepsilon &= -\frac{Dx_B}{Dy_B} \\ \tan\lambda &= -\frac{Dz_B}{(Dx_B \sin\varepsilon - Dy_B \cos\varepsilon)} \end{aligned} \tag{15}$$

To insure that we are using the most accurate and up to date virtual angles, the distance to the target is constantly being recomputed with navigation information for the UAV.

2.2.3 Desired Camera Angles

Calculating the desired camera angles is accomplished with the virtual camera angle equations, equations (12, 13, 14, and 15), with certain conditions placed upon the determination of the body frame distances. In essence, the desired camera angles are themselves virtual camera angles. The desired camera angles are the nominal camera angles that would be present if the aircraft were in a steady state condition. The desired camera angles are broken up into sets associated with the different flight modes.

The first set of desired camera angles is the one representative of level flight. This set of desired camera angles helps determine target location for correct entry into the circling turn associated with flight mode two (circling) and determine if and when roll-over should occur. (Roll-over is a turning maneuver used during the entry into the circling turn, as discussed in section 3.2). To calculate the level flight camera angles, the virtual camera angle equations are reapplied with L^{BI} being modified. L^{BI} is simply recalculated with θ and ϕ set equal to zero. Having done this, the angles calculated will be those associated with no roll or pitch, ie. level flight.

For the correct operation of flight mode one (turn and fly towards the target), another set of desired camera angles is required. The need for such a set may not be obvious. It could be assumed that the desired azimuth angle, ε, will always be off the nose of the aircraft which corresponds to -90 degrees in level flight. It could also be assumed that the desired elevation angle, λ, should always be 180 degrees corresponding to a target on the same plane as the aircraft. These assumptions, however, are incorrect. To understand why they are incorrect, two cases are considered. The first case is the aircraft in straight and level flight, no pitch or roll. The second is the aircraft with a bank angle, ϕ equal to 90 degrees and a pitch angle, θ equal to zero degrees. In both cases the target is considered to be located 45 degrees to the right of the nose and 45 degrees below. In the first case, the azimuth angle, ε, would signify the change in heading while the elevation angle, λ, would reflect the range to the target. Now in the second case, ε would not represent the change in heading but rather the range to the target, and λ would signify the change in heading. This second case demonstrates that the desired camera angles are dependent upon aircraft orientation. In order to account for aircraft orientation, a set of desired camera angles needs to be calculated as if the target is directly in front of the aircraft with the current aircraft orientation. This requires modifying the Dx and Dy of the target in the inertial frame. This is done by taking the sine and cosine of the current heading and translating the distances to the target to a position in front of the aircraft.

$$\begin{aligned} Dx_1 &= \cos(heading) * t\arg et\_range \\ Dy_1 &= -\sin(heading) * t\arg et\_range \end{aligned} \tag{16}$$

where target_range is defined as:

$$t\arg et\_range = \sqrt{Dx_I + Dy_I} \tag{17}$$

After modifying these values, the virtual camera angle equations are used to determine the flight mode one desired camera angles. Since, the modifications are made using the same range information given from the camera angles, real or virtual, all angles will be proportional to each other. This proportionality insures that similarity is met and confirms that in flight mode one (turn and fly towards the target), range accuracy is not important. Similarity allows inaccurate range information to be used without any impact upon the flight qualities of flight mode one operation.

For flight mode two (circling), two additional sets of desired camera angles are needed, representing the steady state camera angles in a left turn and in a right turn. These sets do not use the target range information as in the determination of the previous set but rather the turn radius determined from the current velocity and the user specified load factor. Using the turn radius, the desired camera angles represent not only the aircraft orientation to the target but also the standoff range. This allows the desired camera angles to determine the range to the target without checking the actual range. Using the turn radius and current heading, new Dx and Dy's are determined for each turn direction. For a right turn, the current heading is modified by subtracting 90 degrees. This modification places the target off the right wing.

$$Dx_{2R} = \cos(heading - 90^o) * turn - radius$$
$$Dy_{2R} = -\sin(heading - 90^o) * turn - radius \tag{18}$$

The turn radius is defined as:

$$turn\_radius = \frac{V^2}{g\sqrt{n^2 - 1}} \tag{19}$$

where V is the velocity, g is the acceleration of gravity, and n is the load factor, more commonly known as the number of g's. Subtracting 90 degrees is required because of the coordinate system used by the scene generator. Unlike conventional heading angles with 90 degrees being east and 270 degrees being west, the scene generator associates 90 degrees with west. For a left turn, equation (18) is used with 90 degrees added to the heading rather than subtracted, placing the target of the left wing.

As before, after the modification, the virtual camera angle equations are reapplied to obtain the desired camera angles. In flight mode two, since the desired camera angles are not calculated using the target range information, range accuracy does not effect the calculation of the correct desired camera angles. However, in order to calculate the correct delta camera deflection from virtual camera angles range accuracy is important. Any inaccuracy will be reflected in incorrect virtual camera angles.

2.2.4 Delta Camera Angles

With the desired camera angles and the actual camera angles, the deflection required to return the aircraft to a nominal position is determined.. This deflection will be used by the governing equation for aircraft flight control. Prior to determining the delta camera angles, $\Delta\varepsilon$ and $\Delta\lambda$, all camera angles are required to be within ± 180 degrees. If an angle is greater than 180 degrees then 360 degrees will be subtracted from it.

Conversely, if an angle is less than –180 degrees then 360 degrees will be added to it. For example, an ε of 270 degrees will be modified to –90 degrees. The delta camera deflection are then calculated as follows:

$$\begin{aligned} \Delta\varepsilon &= \varepsilon_{desired} - \varepsilon \\ \Delta\lambda &= \lambda - \lambda_{desired} \end{aligned} \tag{20}$$

The reason why the $\Delta\lambda$ equation is different from the $\Delta\varepsilon$ equation is the coordinate system used. All elevation angles, λ, downward from the aircraft to the target are negative. This is opposite from the frames of reference used, where the z_c axis is positive in the downward direction. As with the camera angles, all delta camera deflections are modified to be within ± 180 degrees.

2.3 Governing Equation for Aircraft Flight Control

Using vector algebra, the angular rates of the camera can be defined in the inertial frame through the following equation.

$$\omega^{CI} = \omega^{CB} + \omega^{BI} \tag{21}$$

This equation implies that the angular rates of the camera in the inertial frame, ω^{CI}, is the sum of the angular rates of the camera in the body frame, ω^{CB}, and the angular rates of the body in the inertial frame, ω^{BI}. By manipulating equation (21), ω^{BI} is obtained as shown below. This set of angular rates is commonly expressed in aircraft body frame components as p, q, and r, and as shown in equation (4) using the inverse Jacobian, the Euler rates can be determined from the angular rates.

$$\omega^{BI} = \omega^{CI} - \omega^{CB} = \begin{Bmatrix} p \\ q \\ r \end{Bmatrix} = J^{BI} \begin{Bmatrix} \dot{\Phi} \\ \dot{\Theta} \\ \dot{\Psi} \end{Bmatrix} \tag{22}$$

By multiplying the angular rates by infinitesimal time, dt, a relationship between infinitesimal angles is obtained:

$$\begin{Bmatrix}\Delta\Phi\\ \Delta\Theta\\ \Delta\Psi\end{Bmatrix} \equiv \begin{bmatrix}1 & \sin\Phi\tan\Theta & \cos\Phi\tan\Theta\\ 0 & \cos\Phi & -\sin\Phi\\ 0 & \sin\Phi\sec\Theta & \cos\Phi\sec\Theta\end{bmatrix} \times \{A\} = J^{-BI} \times \left\{dt * \left(\omega^{CI} - \omega^{CB}\right)\right\}$$

where (23)

$$\{A\} \equiv \left\{ dt * \begin{bmatrix}1 & 0 & -\sin\Theta\\ 0 & \cos\Phi & \cos\Theta\sin\Phi\\ 0 & -\sin\Phi & \cos\Theta\cos\Phi\end{bmatrix} \times \begin{Bmatrix}0\\ 0\\ \omega_z^c\end{Bmatrix}_I - \begin{bmatrix}-\sin\varepsilon & 0\\ \cos\varepsilon & 0\\ 0 & 1\end{bmatrix} \times \begin{Bmatrix}\Delta\lambda\\ \Delta\varepsilon\end{Bmatrix} \right\}$$

$\omega_z^c{}_{\mathrm{I}}$ is the camera angular rate about the z axis of the inertial frame. The remaining camera angular rates are nominally zero for the two flight modes demonstrated. With the governing equation, the inputs to an automatic flight-control system, AFCS, can be determined via camera angles by using the delta camera deflections required to orientate the aircraft to the desired position. In the case of desired constant altitude, the governing equation is further reduced to just $\Delta\psi$, since the remaining two $\Delta\angle$'s are not needed to control aircraft position in the xy plane. The remaining two, $\Delta\theta$ and $\Delta\phi$ would reorient the aircraft to obtain the desired deflection, thus conflicting with the desired change in position and causing a direct change in altitude. Therefore, the changes in pitch and roll are determined from the desired change in yaw, $\Delta\psi$, by the AFCS. Thus pitch and roll are not directly a function of the camera angles but rather indirectly through $\Delta\psi$, which is a function of $\Delta\lambda$ and $\Delta\varepsilon$. With $\Delta\psi$, an AFCS is used to steer the aircraft to the desired heading.

To further simplify the governing equation, the infinitesimal time, dt, is assumed to be a unit of one.

Chapter 3. CSRC's Automatic Flight-Control System

3.1 Waypoint Definition by Operator Mode

CSRC's automatic flight-control system (AFCS) controls the UAV flight using waypoints. The waypoints are determined by the camera operator. The method by which the waypoints are set varies with the different operator modes. The operator controls the position of the waypoints using a haptic joystick that controls camera position. The mode of operation is selected by a switch located on the joystick (shown in figure 11).

3.1.1 Operator Modes

As stated, the operator can toggle through the mode of operations by a switch located on the joystick. The first mode of operation is free fly (operator mode one). By depressing the switch on the joystick, the mode of operation will change to target locked (operator mode two). If the aircraft is in flight mode two (circling) each additional pressing of the switch will lock in a new target. When the aircraft is in flight mode one (turn and fly towards target) a pressing of the switch will change the mode of operation to straight and level (operator mode three). A third depression will enter level turn (operator mode four). Any additional depresses will cycle from operator mode one through operator mode four.

3.1.1.1 Operator Mode One

In operator mode one the waypoint is determined from the line of sight of the camera. The waypoint established by a bubble projected on the operator's screen where the camera bore sight intersects the terrain is constantly changing with camera movement. The changing of the waypoint causes the AFCS to react to camera position. The distance to the waypoint in the inertial frame is determined from the range to the waypoint, the three Euler angles and the two camera angles by modifying equation (9) as follows:

$$\begin{Bmatrix} Dx \\ Dy \\ Dz \end{Bmatrix}_I = \left[L^{-CI}\right] \begin{Bmatrix} 0 \\ R \\ 0 \end{Bmatrix}_C \tag{24}$$

where, L^{-CI} is the inverse rotational matrix relating the camera components in terms of the inertial frame. R is the range to the target. Since the desired camera angles for both flight modes associated with operator mode one use the distance from the aircraft to waypoint in the determination, the range to the target does not require high accuracy. The desired camera angles will be proportional to the real camera angles insuring that similarity is met.

In operator mode one, conditions are placed on the range to the target in the xy plane. To avoid the UAV from instantaneously entering flight mode two, a minimum range is set to the turn radius. This requires the operator to select a target outside the turn radius of the aircraft, promoting stable aircraft operation in response to operator inputs.

$$\begin{gathered} R\min = turn\_radius \\ and \\ t\arg et\_range \geq R\min \end{gathered} \tag{25}$$

Since it is inconceivable for the operator to see over the horizon, a maximum range is established. This maximum range is set to the distance the camera can see five degrees below the nose of the aircraft at the current true altitude.

$$\begin{gathered} R\max = true\_altitude * \tan(85^\circ) \\ and \\ t\arg et\_range \leq R\max \end{gathered} \tag{26}$$

$$\begin{aligned}&therefore\\&R\min \le t\arg et\_range \le R\max\end{aligned} \tag{27}$$

In all modes, a green targeting bubble is used to represent the line of sight of the camera. Each operator mode has a colored bubble associated with it representing the position of the waypoint. Different colored bubbles are used as an indicator to the operator of which mode of operation the UAV is using. In the free fly mode, there is only the green bubble.

3.1.1.2 Operator Mode Two

When the operator toggles the mode of operation into operator mode two, the current camera position is used to lock a waypoint. In order to lock the waypoint to the correct desired position, the range information must be accurate. Any inaccuracy will result in the distance to the waypoint not being correctly determined. In addition to the information used to calculate the distance to the waypoint, the position of the aircraft in the inertial frame is required to determine the waypoint position. The position of the waypoint is determined from the aircraft position plus the distance to the target in the inertial frame.

$$\begin{Bmatrix} X_T \\ Y_T \\ Z_T \end{Bmatrix}_I = \begin{Bmatrix} X_{AC} \\ Y_{AC} \\ Z_{AC} \end{Bmatrix}_I + \begin{Bmatrix} Dx \\ Dy \\ Dz \end{Bmatrix}_I \tag{28}$$

Any inaccuracy in the aircraft position will also affect the accuracy of the waypoint position. An incorrect waypoint location will not affect the operation of flight mode one but will affect when the transition from flight mode one to flight mode two occurs. This transition may occur earlier or later than desired depending on the position of the waypoint. The proper standoff range while in flight mode two will also be affected. These effects are a result from the virtual camera angles used in operator mode two being calculated from the waypoint position. Incorrect waypoint position means incorrect virtual camera angles. In operator mode two, CSRC's AFCS uses the governing equation with the virtual camera angles rather than the actual camera angles to navigate the UAV. The use of virtual camera angles allows the operator to maneuver the camera without influencing UAV flight. Once a target is locked into a waypoint, the operator is free to search for other targets.

In operator mode two, a red bubble will be placed over the waypoint. This red bubble indicates the target position and will remain with the target. In addition to the red bubble, the green bubble is present representing the camera's current line of sight. In operator mode two, the UAV turns and flys to the locked waypoint (flight mode one) and once there circles the target (flight mode two). After the UAV establishes itself in a circling turn around the target, the operator may reselect the locked position using the green bubble and pressing the switch on the joystick. This will cause the red bubble to move to the new position and update the waypoint. The UAV will remain in flight mode two until a position outside twice the turn radius is selected. When a position outside this range is selected, the aircraft will leave flight mode two and enter flight mode one. When in flight mode one, an additional press of the switch will toggle to operator mode three.

3.1.1.3 Operator Mode Three

Operator mode three (straight and level) is the most simple of the four operator modes. When the switch is toggled to this mode, the current heading is locked. The distance to the waypoint in the inertial frame is then determined using the current heading and the turn radius as the range. Since the virtual camera angles and desired camera angles use the same distance information, similarity is met. For this reason, the turn radius can be used as the range without any impact upon flight performance. As with operator mode two, in operator mode three, CSRC's AFCS uses virtual camera angles instead of the real camera angles. Operator mode three uses a blue targeting bubble. This targeting bubble is always directly ahead of the aircraft at a distance equal to the turn radius and moves with the aircraft.

3.1.1.4 Operator Mode Four

Operator mode four (level turn) is very similar to operator mode three. In operator mode three the target angle was zero, thus the angle to the desired target in the global coordinate system was simply the current heading. In operator mode four, the target angle is set by the operator. When the switch on the joystick is toggled to operator mode four, the current offset of the green bead to the nose of the aircraft is locked as the target angle. The distance to the waypoint in the inertial frame is then determined using the current heading plus the target angle and the turn radius as the range.

$$\begin{aligned} Dx_I &= \sin(current\_heading + target\_angle) * turn\_radius \\ Dy_I &= \cos(current\_heading + target\_angle) * turn\_radius \end{aligned} \tag{29}$$

This causes the aircraft to constantly alter heading in an attempt to reach the offset target. The angle of bank for this level turn is set equal to the offset angle. For example, selecting the target to be 10 degrees to the right of the nose of the aircraft will result in the aircraft performing a 10 degrees banked level turn to the right. A one to one ratio was used to promote the greatest range of turn angles. Both operator modes three and four are considered "carrot" modes. An achieveable but unobtainable goal is placed in front of the aircraft that moves with the aircraft. Thus, the virtual camera angles and the desired camera angles are held constant and since they are calculated using the same distance information, similarity is met. Therefore, the use of the turn radius as the range has no impact upon flight performance. As with operator mode two and operator mode three, CSRC's AFCS uses virtual camera angles instead of the real camera angles in operator mode four. Operator mode four has a yellow targeting bubble. This targeting bubble moves with the aircraft at the selected offset.

Table 4. Targeting Bubbles

Color	Description
Green	**Line of Sight of the Camera**
Red	**Locked Target (Waypoint)**
Blue	**Straight and Level**
Yellow	**Level Turn**

3.2 Aircraft Flight Modes

The aircraft has two flight modes as previously mentioned: 1) Turn and fly towards a waypoint and 2) Circle a target (waypoint). In order to determine when to transition between flight modes, a turn range based upon turn radius is calculated. This range is calculated as if the aircraft is approaching straight at the target and a turn into the circling mode turn, meets the turn radius tangentially. This can be seen graphically by figure 9.

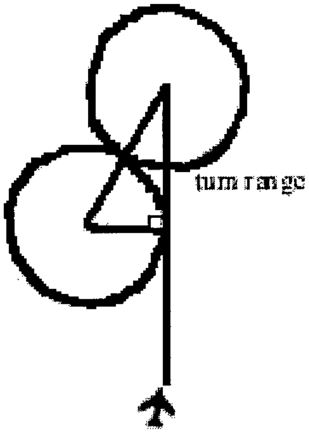


Figure 9. Turn Range

The length of the tangent is equal to the turn range by the equation below:

$$turn\_range = \sqrt{3 * turn\_radius^2} \tag{30}$$

This turn range is checked against the range to the target to determine the entry into flight mode two (circling). The turn rate associated with the calculated turn radius, equation (19) is calculated using the following equation.

$$\omega = \frac{V}{turn\_radius} \tag{31}$$

where ω is the turn rate and V is the UAV velocity. Entry into flight mode two is then determined by the following conditions: 1) target range is less than or equal to the turn range and 2) aircraft is currently in flight mode one (turn and fly towards the target).

$$target\_range \le turn\_range \tag{32}$$

If these conditions are met, the UAV enters flight mode two. If the conditions are not met, the UAV remains in flight mode one. If the aircraft is already in flight mode two, it will remain in this mode until the target range is greater than or equal to two times the turn radius.

$$target\_range \geq 2 * turn\_radius \qquad (33)$$

This condition allows the variation in range from the target to be one turn radius without changing flight modes. This allows flexibility in re-targeting while in flight mode two.

Prior to turn entry, it is necessary to determine whether the UAV will be turning in a left or right circle. This is accomplished with the level desired camera epsilon angle. This camera angle indicates the location of the target with respect to the aircraft. If the target is located directly in front of the aircraft or to the right, a right circling pattern is desired. Obviously, if the target is to the left, a left circling pattern is desired. It is also necessary to determine which direction the UAV should turn to enter the turning circle. If the target is in front and to the left or behind and to the right, the entry turn will be to the right. Conversely, if the target is in front and to the right or behind and to the left, the entry turn will be to the left. After the turn direction is determined by the automatic flight control system, the turn rate and aircraft mode two delta camera deflections are set for the desired direction. If a right turn is desired, the turn rate will be positive and the delta camera deflection will be the set of camera deflections known as 'mode two R'. A left turn will have a negative turn rate and delta camera deflection set known as 'mode two L'.

Prior to determining the $\Delta\psi$ from the governing equation, equation (23), the delta camera deflections will be damped against oscillations caused by small fluctuations. The flight mode one deflections will be checked for target lock on. If the deflections are within the range ± .01 degrees for $\Delta\lambda$ and ± .03 degrees for $\Delta\varepsilon$, the UAV is considered to have completely turned towards the target. When this occurs, the allowable variations are increased to ± 1.0 degrees for Δλ and ± 3.0 degrees for Δε. If these variations fail then lock on is considered lost. In flight mode two, Δλ and Δε are both allowed to vary ± 1.0 degrees. An additional constraint is place upon the use of Δε in flight mode two. If the range to the target is greater than 100.0 meters, Δε will be set to zero giving Δλ sole control of determining Δψ. This constraint allows a quicker response to correct target range. Any time the deflections are within the allowed variations they will be set to zero.

Table 5. Deflection Damping

	Flight Mode One	Target Locked On	Flight Mode Two
Δε	**± 0.03**	**± 3.0**	**± 1.0 ***
Δλ	**± 0.01**	**± 1.0**	**± 1.0**

* Only when target range is within 100.0 meters

Once flight mode determination and deflection oscillations are accounted, Δψ is determined. In flight mode one, equation (23) is simplified by setting $\{\omega_z^C\}_I = 0.0$ since the camera has no turn rate about the target. In flight mode two, prior to using equation (23), two turn entry flight maneuvers are accomplished. The first maneuver is the initial turn entry. This maneuver is accomplished in the direction previously determined and at the calculated turn rate. The second maneuver is accomplished when the UAV achieves the target off the desired wing, and is called the roll over maneuver. The roll over manuever simply rolls the aircraft from its current bank angle through zero degrees of bank to the correct bank angle. These two maneuvers are required due to the sign changes that occur during turn entry and roll over. These two maneuvers also prevent a singularity in the Δε calculation from occuring when the target transitions directly below the aircraft body. The sign changes, if used directly in equation (23) would cause oscillations and greatly degrade the quality of the entry into the desired turn. After both maneuvers are accomplished, equation (23) is used to determine $\Delta\psi$. In flight mode two, $\{\omega_z^C\}_I$ will be set to the turn rate previously calculated.

Before the roll desired is determined from $\Delta\psi$, $\Delta\psi$ will be modified to be within ± 180 degrees. $\Delta\psi$ is also damped in flight mode two from small fluctuations in camera deflections. In order to determine the roll desired, the bank required for level flight and the desired number of g's must be determined. This bank angle, ϕ, and the lift, L, are such that the component of the lift in the vertical direction must equal the weight as shown in figure 10 [1].

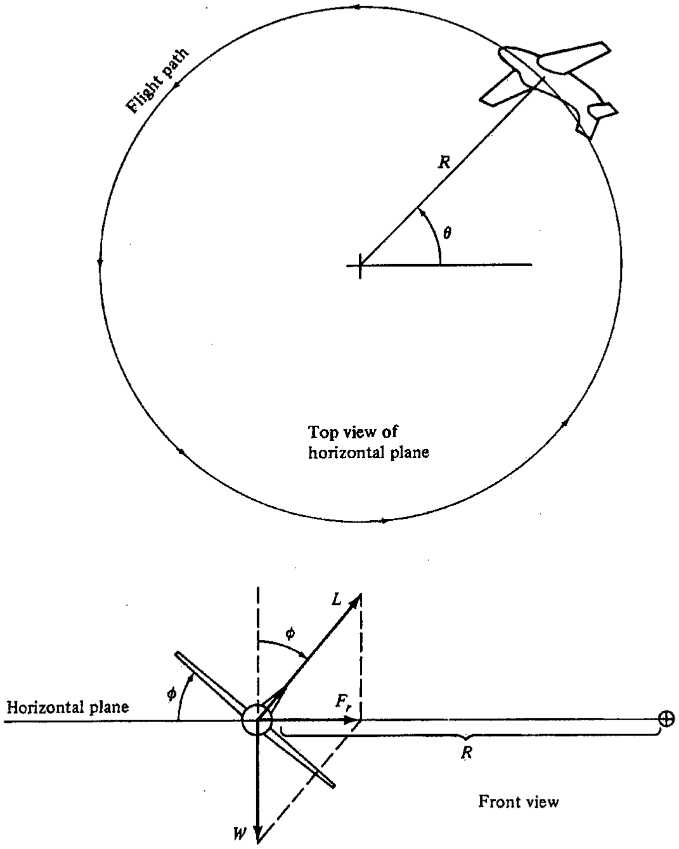


Figure 10. Aircraft in a Level Turn [1]

The bank angle is determined by solving the following equation:

$$L\cos\phi = W \quad \text{where} \quad \frac{L}{W} \equiv n \tag{34}$$

solving for ϕ

$$\phi = \cos^{-1}\left(\frac{1}{n}\right) \tag{35}$$

Where L is the lift of the UAV, W is the weight, n is the g_load and ϕ is the desired bank angle. In flight mode one, any change in heading, $\Delta\psi$, greater than 3.0 degrees will result in a desired roll equal to the desired bank angle calculated by equation (35). Any change less than 3.0 degrees will be linearly reduced by the change divided by three, $\Delta\psi$ / 3.0, until zero change is reached. The use of 3.0 degrees allows for a controlled roll out onto target. Roll-outs faster than this result in the aircraft unrolling faster than changes in heading. This causes periodic pauses in the roll out. Figure 11 graphically depicts the scheduled roll angles for a change in heading, $\Delta\psi$.

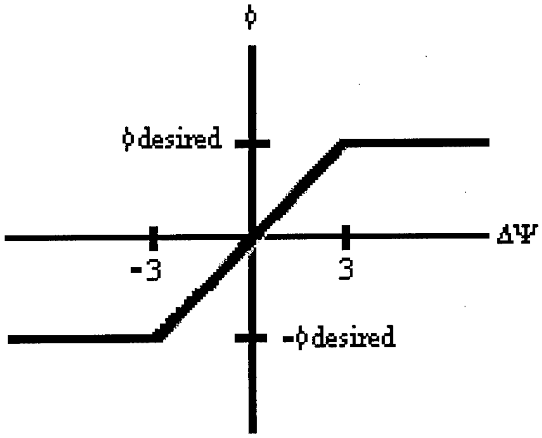


Figure 11. Flight Mode One Roll Schedule

In flight mode two, the desired roll is determined by the ratio of the desired change in heading over the turn rate, ω_z^c I. The ratio is used based upon the fact that if the aircraft is in a perfect turn around the target where the delta camera deflections are zero, the change in heading would equal the turn rate. If the aircraft is outside the desired target range or drifting away from the target, the $\Delta\psi$ calculated by equation (23) is greater than the turn rate. Thus, the ratio is greater than one and the roll desired is greater than the desired bank resulting in a tighter turn correcting the delta deflections. An aircraft inside the desired target range or drifting toward the target results in a roll desired less than the bank desired. The variation of the roll desired from the bank desired is limited to a maximum variation determined by the variance from the turn radius times 5.0 degrees as seen in the following equation:

$$Max\_variance = \left(\frac{target\_range}{turn\_radius}\right) * 5.0 \tag{36}$$

This limitation allows for greater changes as the distance from the desired target range increases. By limiting the roll desired, stability in flight mode two is greatly increased and oscillations are reduced.

With the roll desired determined, the roll-rate command to FLSIM is calculated. The roll rate command is determined by the error between the roll desired and the actual roll of the aircraft, ϕ. If the roll error is greater than 60.0 degrees (value determined by bank angle of a 2g turn), the roll rate command is set to 1.0 or the maximum roll. As the error becomes less than 60 degrees, the roll rate is linearly dampened by 1.0/60.0 until zero error is achieved. Figure 12 shows the scheduled roll rate command for roll error.

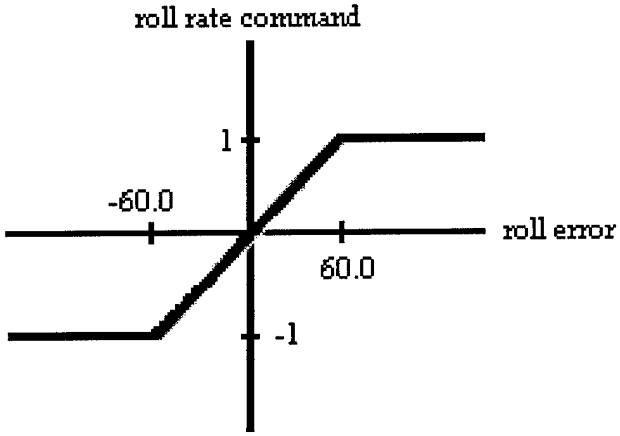


Figure 12. Scheduled Roll Rate Command

3.3 Velocity Hold

Though all the equations are capable of handling any value of velocity, it is desired to hold velocity to a constant. In FLSIM, velocity is controlled via thrust commands. Since velocity is a function of thrust, altitude and aircraft orientation, the thrust command must change with continuing changes in flight. This is accomplished by using an error in the velocity from desired velocity. The desired velocity is initially set equal to the current velocity when the transition between standard aircraft operation and UAV control occurs. After the transition, the desired velocity may be increased or decreased by a switch on the thrust master joystick. When a change in the desired velocity occurs, a counter is reset allowing a new zero lift, angle of attack to be determined for the new velocity.

A change in the thrust command is accomplished using the error between the desired velocity and the actual velocity and comparing the actual velocity with the previous velocity. The actual change in the thrust command is determined using these parameters with the following set of conditions: 1) If the velocity error is positive and the change in velocity is not increasing, the thrust command is increased by 0.001. 2) Conversely, if the error is negative and the change in velocity increases, the thrust command decreases by 0.001. To provide increased stability and to prevent significant overshooting of the desired velocity, the gains are changed when the error of the velocity is within 1.0 m/s. Therefore, conditions 3) and 4) modify the thrust command by a factor of 0.001 times the error when the error is within 1.0 m/s. Figure 13. shows the decision making process for determining thrust command changes.

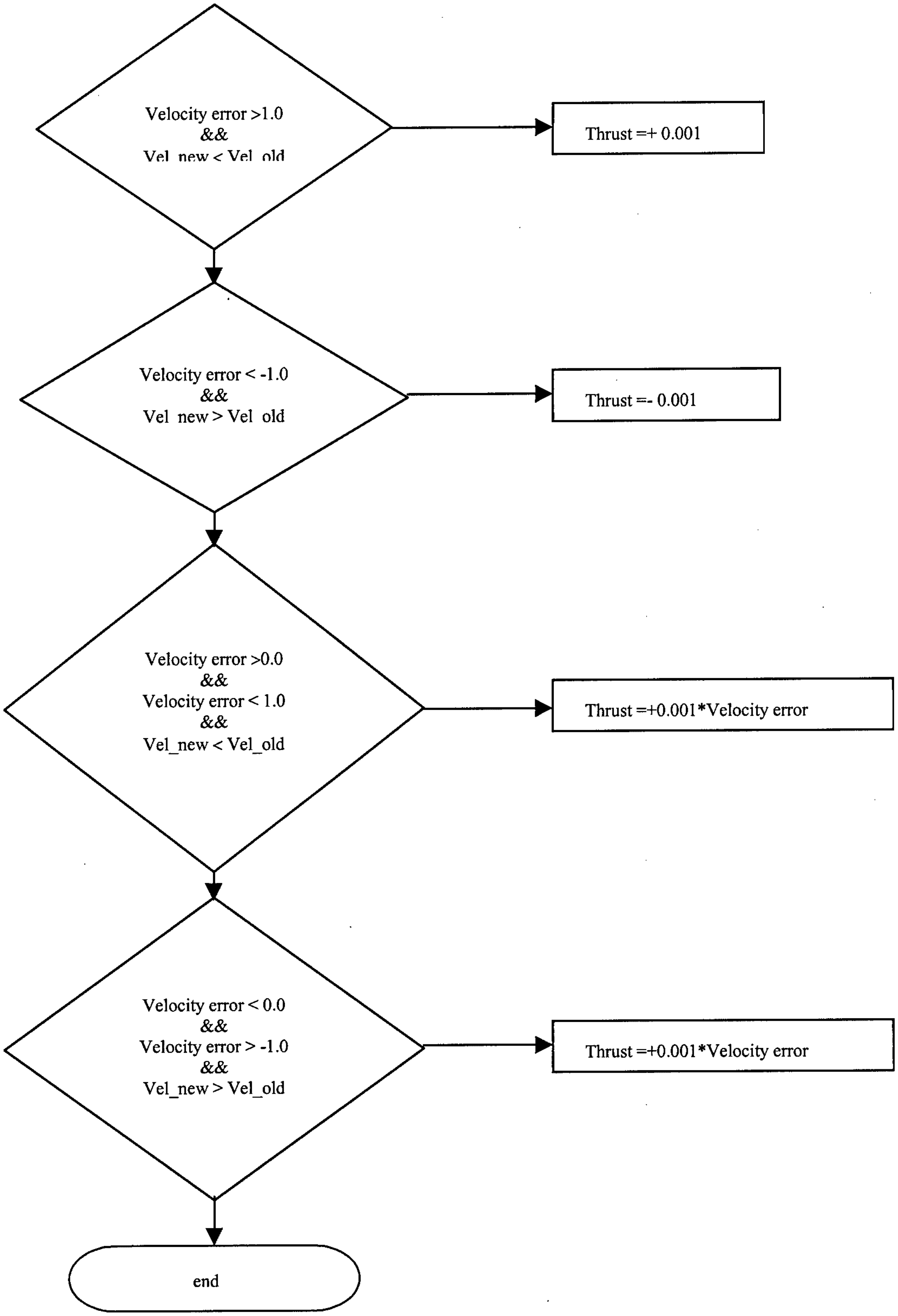

Velocity error >1.0
&&
Vel_new < Vel_old
Thrust =+ 0.001
Velocity error < -1.0
&&
Vel_new > Vel_old
Thrust =- 0.001
Velocity error >0.0
&&
Velocity error < 1.0
&&
Vel_new < Vel_old
Thrust =+0.001*Velocity error
Velocity error < 0.0
&&
Velocity error > -1.0
&&
Vel_new > Vel_old
Thrust =+0.001*Velocity error
end

Figure 13. Thrust Command Flowchart

3.4 Altitude Hold

Similar to velocity, all equations are capable of handling any value of altitude. However, it is strongly desired to have a constant altitude. In FLSIM, altitude is controlled via the pitch rate command. The pitch rate command must be continuously updated since the pitch required to hold a constant altitude is a function of altitude, velocity and bank of the aircraft. Changes to the pitch rate command are accomplished by using an error in the pitch, θ, and the desired pitch. Similar to the desired velocity, the desired altitude is initially set equal to the current altitude when the transition between standard aircraft operation and UAV control occurs. After the transition, the desired altitude is increased or decreased by a switch on the thrust master joystick.

Controlling altitude is a two-fold process. As previously stated, since pitch is a function of altitude, velocity and bank of the aircraft, the desired pitch must be continuously updated from the error in altitude. The condition set for controlling altitude is more complicated than the condition set for controlling velocity and is explained in the order in which the conditions occur. First, every time the aircraft rolls out of a turn, the theta desired is reset to the theta for zero lift. This condition allows for a quick return to the angle of attack required for zero lift without using the modification algorithm and increases the stability of holding altitude. Without this condition, the response time to hold a constant altitude becomes sluggish and big altitude changes may occur.

Next, the desired theta is modified based upon the altitude error. If the change in altitude, as measured by the vertical velocity indicator (VVI), is greater than the error in altitude divided by 15.0, and is changing in the wrong direction, theta desired is modified by 0.005 times the difference in the desired VVI and the actual VVI. The condition upon the maximum change rate is used to help stabilize the correction to the desired altitude and prevent significant overshooting. The small gain is used to prevent rapid changes in the desired theta when there is a theta error greater than 0.25 degrees. If the theta error is within 0.25 degrees, the gain is increased to 0.01. Next, if the error in altitude is less than 0.5 meters, theta desired is set to theta for zero lift. Because, theta for zero lift is initially set to zero and may not be the correct theta, modifications to this value are accomplished. Initially, the first time the error in altitude is within 0.5 meters, theta for zero lift is set to the current desired theta. After the initial setting, theta for zero lift is modified by 0.0001 whenever the altitude error is less than 0.5 meters and the VVI is changing in the wrong direction.

The use of changing in the wrong direction is important to the success of this algorithm scheme. The control of altitude is highly subject to overshooting. If modifications were accomplished any time the VVI was in the wrong direction, an uncontrolled oscillation would occur. The use of changing in the wrong direction dampens these oscillations and allows zeroing in on the desired altitude.

After determining the desired theta, the error in theta is calculated by the difference in the actual pitch and the desired pitch. This error is used to determine the pitch rate command for altitude. Whenever this error is greater than 5.0 degrees, the pitch rate is set to 1.0 or the maximum pitch rate. If the error is between 1.0 and 5.0 degrees, the pitch rate is set to the error divided by 10.0. If this error is less than 1.0 degrees, the gain is set to 5.0/6.75. These gains were determined by experimentation. Figure 14 shows the scheduled changes to pitch rate for theta error.

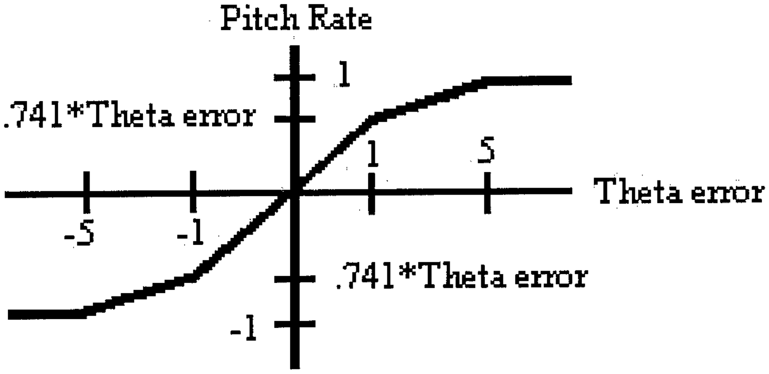


Figure 14. Scheduled Pitch Rate Command for Theta error

Whenever the aircraft is in level flight, the pitch rate command as determined above is the only input into the total pitch rate commanded to FLSIM. Whenever the aircraft is in a banked turn, the pitch rate required to hold altitude is not enough. In this situation, an additional pitch rate is added to the total pitch rate command. This pitch rate addition is initially set to the value of the desired roll angle (in degrees) divided by 100.0. This pitch rate addition is necessary, since to turn an aircraft, two requirements must be met. An aircraft must have bank and pitch. Since the pitch rate initially added may or may not hold the aircraft to a constant altitude, this pitch rate is also modified. The modification to the pitch rate command in a turn is accomplished whenever the aircraft is climbing or descending, the change in pitch rate is less than 0.0009 and the aircraft altitude is changing in the wrong direction. The requirement of the change in pitch rate to a small error prevents this pitch rate from being arbitrarily modified simultaneously when the altitude hold algorithm is making the needed changes. When a modification to this pitch rate is required, it will be modified by 0.001.

Lastly, pitch rates are added to produce a total pitch rate. This total pitch rate is subject to the maximum of 1.0. Any time the pitch rate is greater than 1.0, it will be set to 1.0. With the commanded pitch rate and the previously determined roll rate, FLSIM is provided the stick commands in terms of these rates.

3.5 Automatic Control Theory

Auto-pilots require an extensive knowledge of the system being flown and some method of controlling it. Two theories are currently accepted for accomplishing this. The first method, classical or conventional control theory can be dated back to the 1930's. The second method more commonly known as modern control theory has been around since the 1960's [9]. Both methods have their strengths and weaknesses and both are widely used. Both systems use a closed-loop scheme with some form of feedback to obtain an error signal. These systems are discussed to help the reader understand the value of the previously discussed AFCS design.

3.5.1 Classical Control Theory

The classical control theory is based on frequency response. It uses root-locus technique, transfer functions and Laplace transforms to determine aircraft response to desired inputs. In classical control theory, a transfer function is defined in terms of the Laplace transform of the output over the Laplace transform of the input [9]. Using a root-locus technique, the roots of the transfer function are obtained and the stability of the system is determined. From the roots and the transfer function, the gains required in the closed-loop feedback system are determined to provide stable aircraft's response to the input in order to obtain the desired output. A strength of this method is that compared to its counter part, it is relatively simple and the calculations are readily accomplished without the use of a computer. This method can also be applied to steady state and time domain problems. A weakness of this system is that it is normally limited to second-order, while many systems are usually of a higher order. However, a good approximation can be obtained by representing a higher order system be a second-order. In essence, the strength of this method becomes its weakness as complexity increase.

3.5.2 Modern Control Theory

Modern control theory applies a more systematic approach to control system design. With modern control theory, the system is represented by a system of first-order differential equations [9]. Due to the complexity of this method, a computer is required to solve the problem. In modern control theory, the equations of motion are written in the state-space form. The physical system is reduced to a set of differential equations and the state variables and equation are used to describe the system. From the system of equations, the eigenvalues are determined and the feedback gains are calculated. Using the feedback gains, the system is returned quickly to the desired equilibrium state. A strong advantage of the modern control theory is that with the use of optimization techniques, optimal control systems can be designed. Another strength is that modern control theory is suitable for control systems with multiple inputs and can be used to determine the optimum set of control surface deflections.

Chapter 4. Cave Simulation

4.1 CAVE Facility

The CAVE is 3.1 x 3.1 x 3.1-meter, with four rear-projected walls and a down-projected floor (Pyramid Video). Imagery is created by an SGI Onyx with InfiniteReality graphics. Images are displayed by CRT projectors (M8500, Electrohome). A magnetic tracking system is employed to monitor the position and orientation of the user's head and hand (Flock of Birds, Ascension Technology), while stereo images are created by use of LCD shutter glasses (CrystalEyes, Stereographics). In addition, the user's finger-pinch gestures are sensed to provide a means to interact with the virtual environment (PinchGloves, Fakespace). The PinchGlove was not used by CSRC but is required to enter the demonstration. The addition of the fourth wall provides complete immersion in the azimuthal dimension (the only non-imaged area is directly overhead), allowing the operator to monitor and interact with the environment to the rear as well as to the sides and the front.

The VERITAS CAVE emphasizes multisensory displays. The off-the-body projection system of the CAVE limits the ability to integrate haptic stimulation (since haptic stimulators are likely to be visible to the user). Nevertheless, a force-feedback manual control stick (IE-2000, Immersion Corp.) is integrated for manipulating objects and controlling the virtual environment. CSRC uses this haptic joystick without the force-feedback features and as shown in figure 15, has two programmable switches built into it.

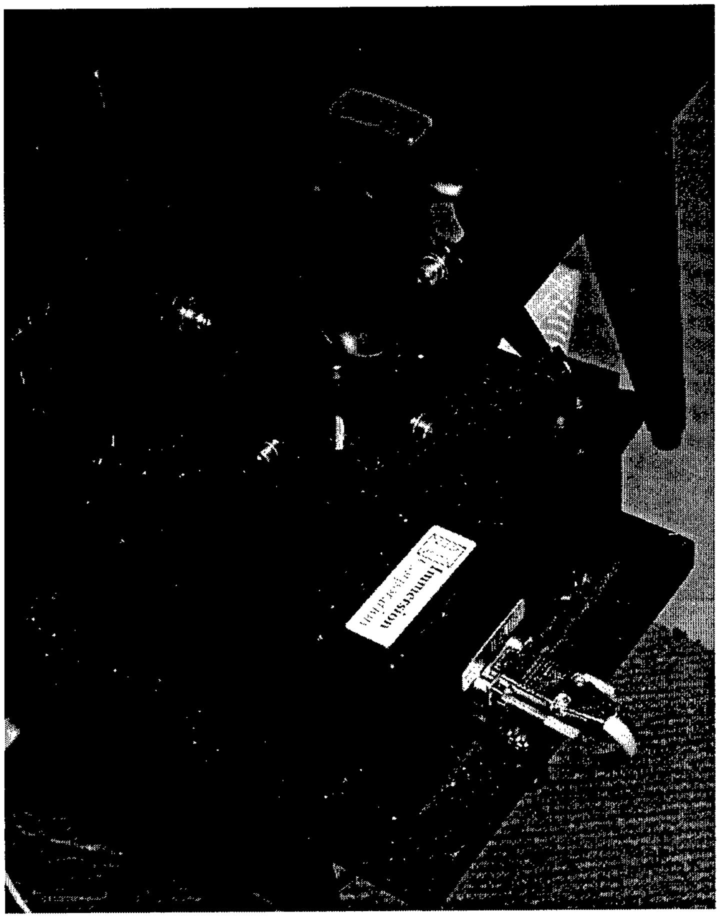


Figure 15. Haptic Joystick

Also used in conjunction with the haptic joystick is a thrustmaster joystick shown in figures 16 and 17. The thrustmaster joystick has a one dimension sliding position along with two radial switches, one four position spring toggle, one three position toggle, one two position toggle, and two press switches, all whose functions are programmable. Table 6 addresses the various operator inputs used and their functions.

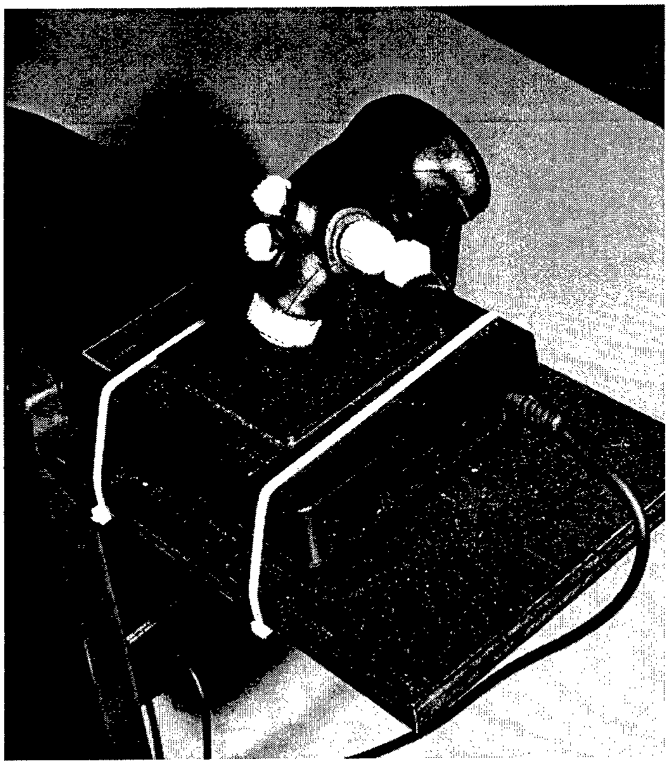


Figure 16. Thrustmaster Front View

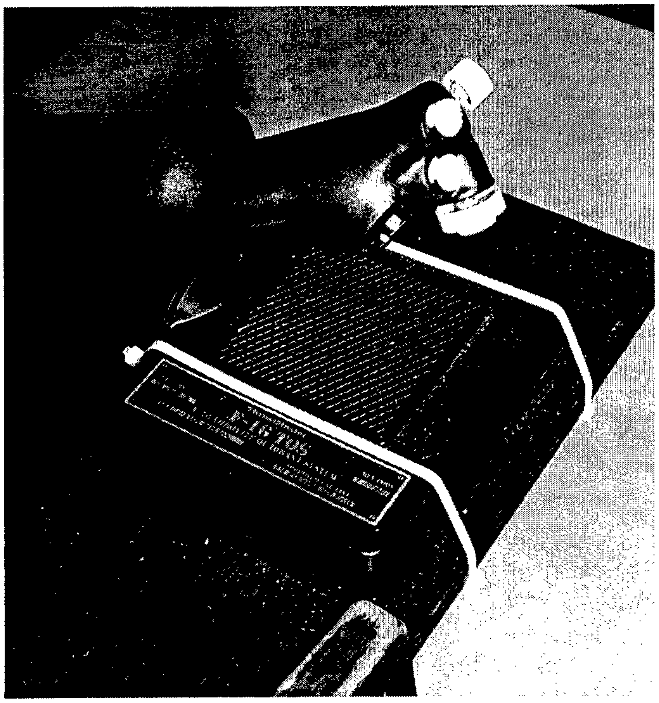


Figure 17. Thrustmaster Rear View

Table 6. Operator Inputs

Operator Input	Function
PinchGlove	**Used to enter demonstration**
Haptic Joystick X Position	**Controls X position of camera wrt aircraft**
Haptic Joystick Y Position	**Controls Y position of camera wrt aircraft**
Haptic Joystick Top Hat Switch	**Toggles Operator modes**
Haptic Joystick Trigger	**Places Bombing run target**
Thrustmaster 4 way (Up-Down)	**Changes Desired Altitude**
Thrustmaster 4 way (Left-Right)	**Changes Desired Velocity**
Thrustmaster 3 way	**Toggles between UAV demo mode and normal control mode**
Thrustmaster 2 way	**Switches Speedbrake in and out**
Thrustmaster Radial #1	**Changes operator's overhead view (zooms in and out from cockpit)**
Thrustmaster Radial #2	**Changes operator's front view (rotates around the aircraft)**
Thrustmaster Switch #1	**Exits Demo (Eject)**
Thrustmaster Switch #2	**Changes starting locations**
Thrustmaster Sliding position	**Disabled in UAV mode**

To support a broad variety of synthetic environment research activities, with special focus on defense-related applications, commercial image generation software with the CAVE projection systems (Vega, Multigen-Paradigm) has been integrated. This software provides a number of high-level development tools but still allows access and the optimization of the underlying capability of the hardware platform (via SGI Performer and OpenGL). Furthermore, the chosen software platform permits options such as importing CAD and terrain data (MultiGen II, Multigen-Paradigm), simulation of flight dynamics (FLSIM, Virtual Prototypes, Montreal, Canada), simulation of ground vehicle dynamics (Clarus Drive), simulation of manufacturing processes (Clarus Manufacturing), and interfacing with display and control devices (Clarus InteractiveVR).

In the current application within the CAVE, a scene representing a 60km by 80km of geotypical terrain is used. The terrain features are a composite of several real locations and have been selected to provide specific operational challenges in close physical proximity (e.g., mountains and canyons next to open plains). The software tools available in VERITAS support the creation of large geospecific terrain scenes based on Defense Mapping Agency data (e.g., Digital Terrain Elevation Data, DTED, for terrain heights and Digital Feature Analysis Data, DFAD, for cultural features, with satellite or other photography providing texturing data). The limit on representing any specific physical location is the availability of digital data for that location.

4.2 Simulation vs. Real

As with any simulation, there are aspects that differ from the real world. The selection of the CAVE facility was based on the desire to minimize those differences. The unique characteristics of the CAVE facility, allows a user to be immersed in a virtual environment similar to the real world, but there are some differences. The cave has no software or hardware tools available to emulate actual camera angles. For this reason, the real camera angles are calculated (i.e. simulated) the same way the virual camera angles are determined.

Depending on future control sites, the simulation of flying and being able to see all the surrounding area may or may not be real. It is possible to achieve this but may not be practical. A real control site may limit the operator to only the camera view, while the CAVE views everything. The demonstration fixed the operator's chair to the aircraft, having the front wall always display where the aircraft was heading. This allowed the operator to sense what the aircraft doing. It is possible to fix the chair to the camera and have the front wall show what the camera is seeing. This configuration is not demonstrated and is left for future evaluation.

The actual experience of flying is only as real as the flight dynamics being modeled. The use of FLSIM with the F-16 flight characteristics provided a realism beneficial to the demonstration. Aircraft response to operator inputs simulated actual expected results. The software used to model the environment allows for disturbance simulation. If desired, wind and wind gusts can be simulated allowing the evaluation of aircraft response. All results presented in the demonstration do not included any wind or wind gusts.

Despite the shortcomings of the CAVE, the demonstration developed proved to be very realistic and successfully validated CSRC.

4.3 Simulation Hierarchy

In order to understand the source code developed for the demonstration, an understanding of the software structure used by the simulator is required.

Figure 18 depicts the structure of the simulation from the operator input to the aircraft response.

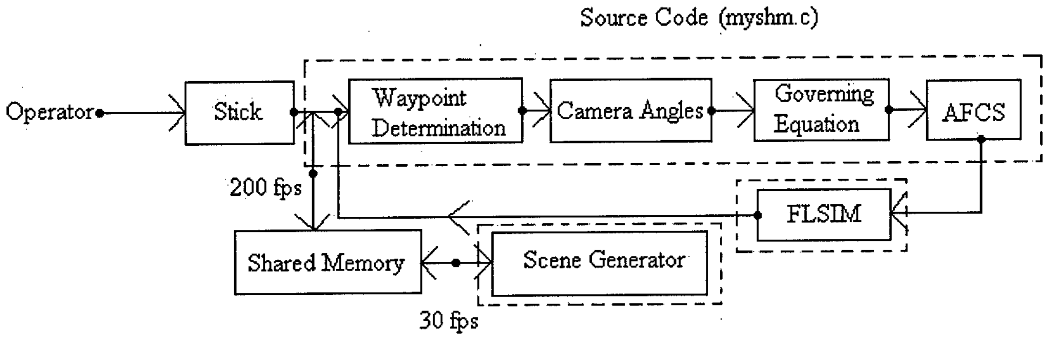

Source Code (myshm.c)
Operator
Stick
Waypoint Determination
Camera Angles
Governing Equation
AFCS
200 fps
FLSIM
Shared Memory
Scene Generator
30 fps

Figure 18. Simulation Structure

The source code developed is primarily a front-end processor. It is titled myshm.c, found in Appendix A, and is coded in UNIX C. This code communicates with FLSIM and with the shared memory. The shared memory is a segment of memory that is accessed by both the source code and by myvega.c, the scene generation code. Since the source code is the only segment that contains the control theory, it is the only one provided in this thesis. The scene generation code is modified to place the camera targeting bubbles in the scene and has no impact on the theory presented. The code for FLSIM is only available in compilation format, which is unreadable.

The source code operates at a frame rate of 200 frames per second, while the scene generator operates at a frame rate of 30 frames per second. This indicates, that the source code updates both FLSIM and the shared memory and is re-updated by them at a rate of 200 times per second. Though the scene generation code does not contain any control theory, it provides a piece of information used by the source code to determine camera angles. For this reason, the difference in frame rates has an impact on the demonstration. This impact is minimized with code in myshm.c. The source code determines the camera angles based on the aircraft position and the waypoint position. The waypoint position is calculated using the height above target. The height above target is measured using an I vector. I vectors can only be measured by the scene generator. Therefore, this information needs to be passed from the scene generator to the source code. The difference in frame rates causes a lag in the proper height above target to be sent to the source code. This lag is only noticeable in operator mode two. In operator mode two, the waypoint position is normally locked the instant the operator toggles into this mode. The z position of the target may not be correct with the x and y position at that instant. In order to work around this issue, a heartbeat of the scene generator was established. This heartbeat indicates to the source code everytime it is updated by the scene generator. Delaying the locking of the waypoint until the source code is updated insures that the z position matches the x and y position.

4.4 Source Code

The source code is the heart of the simulation. The source code shown in Appendix A is the beginning of one encompassing demonstration to be used by the CAVE. Parts of the code are not used by CSRC but are needed by another task. Table 7 shows the various subroutines and gives a description of what is performed.

Table 7. Source Code Subroutines

Subroutine	Description
Declarations	**Includes header files, defines constants, declares variables and subroutines**
*u_shm_init	**Creates shared memory segment**
*u_shm_exit	**Detaches the shared memory segment from the process address space**
HapticStick	**Reads stick inputs**
GetInputs	**Reads all other input devices, (glove, thrustmaster, etc)**
main	**Calls all subroutine in the order necessary**
InitFlsim	**Initializes FLSIM**
ResetFlsim	**Resets FLSIM**
RunFlsim	**Sets variables in FLSIM and reads variables from FLSIM, Waypoint Position is also determined here**
CameraAngles	**Calculates the camera angles (real and virtual) from the waypoint**
Pilot	**This is CSRC, determines the desired and delta camera angles, determines steering requirements, controls altitude and velocity, and commands FLSIM**
WriteLandingData	**Not used by CSRC**
WriteCrashData	**Not used by CSRC**
WriteContinuousData	**Not used by CSRC**
ReadContinuousData	**Not used by CSRC**

As annotated by Table 7, the subroutine Pilot is where CSRC really occurs. RunFlsim represents the interface between the operator and the UAV, while CameraAngles calculates the camera angles that would be read directly from the camera if this were implemented as real hardware. All other subroutines are either a function of the CAVE environment itself or not used by CSRC.

4.5 Additional Code

In addition to controlling the UAV, the source code has incorporated a bombing run in the Pilot subroutine. This code was included to demonstrate the ability of the control theory to control an UAV all the way to and through putting bombs on target. The bombing run is based on gravity bombs with no environmental disturbances. Using the height above target and aircraft velocity, the time to impact and the range at which to drop the bombs is determined. During a bombing run, entry into flight mode two is prohibited until after bomb release. After bomb release, the aircraft will either enter the circling mode if within the turn range or wait for this condition to be met.

4.6 Results

The results, from the demonstration, successfully validate the proposed control theory. The theory is demonstrated to be robust enough to handle varying conditions and all modes of operation, operator and flight. A mission profile, figure 19, was developed to demonstrate all the capabilities of CSRC as coded in the demonstration.

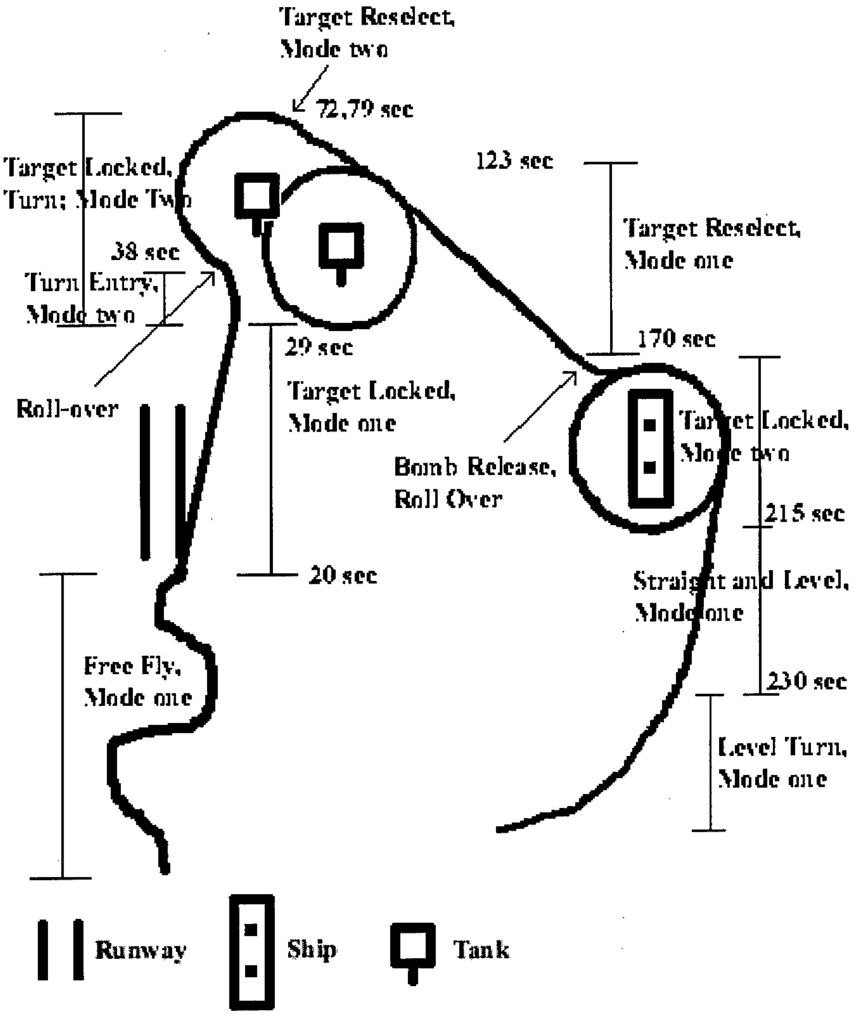

Target Reselect,
Mode two
72,79 sec
Target Locked,
Turn; Mode Two
38 sec
Turn Entry,
Mode two
Roll-over
29 sec
Target Locked,
Mode one
20 sec
123 sec
Target Reselect,
Mode one
170 sec
Bomb Release,
Roll Over
Target Locked,
Mode two
215 sec
Straight and Level,
Mode one
230 sec
Level Turn,
Mode one
Free Fly,
Mode one
Runway
Ship
Tank

Figure 19. Mission Profile

This mission profile was tested in the CAVE facility with the following results. The times indicated represent a change in flight mode or operator mode. Figure 20 shows the relationship between the roll command and the determined change in heading, $\Delta\psi$. Also shown is how the pitch command is influenced by the roll command. Of particular interest is how quickly the roll responds to $\Delta\psi$, especially in operator mode one.

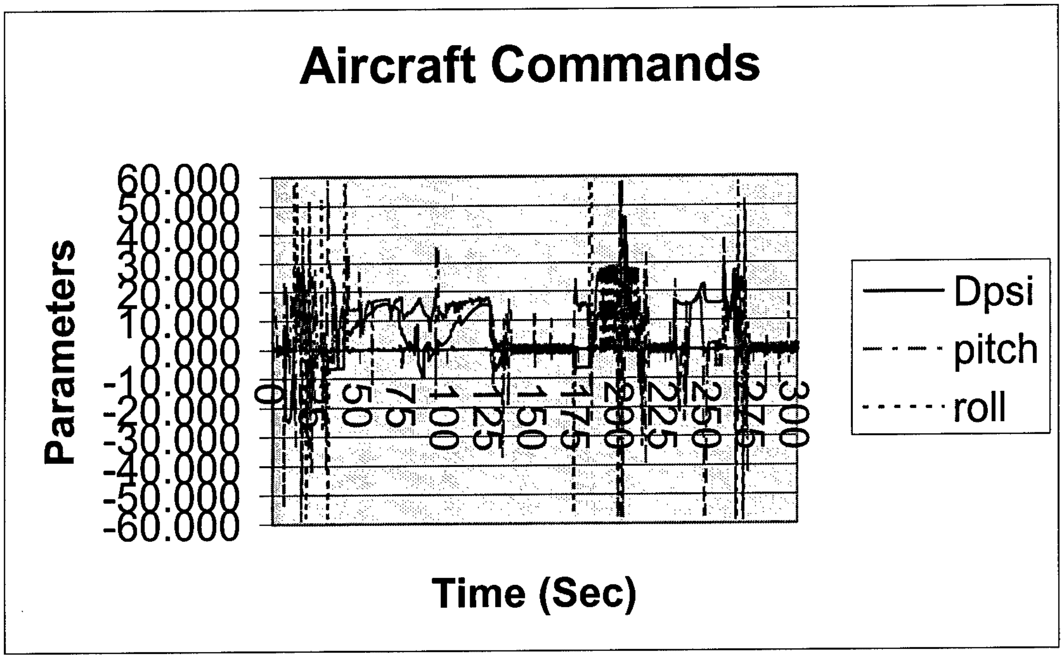


Figure 20. Aircraft Commands

Between 0 and 20 seconds, the operator is in free fly mode. In this mode, the $\Delta\psi$ is constantly changing with operator movement of the camera. This results in what appears to be an oscillation but is actually equation (23) reacting to operator inputs. During the time frame of 30 to 120 seconds the UAV is operating in the target locked mode and flying in flight mode two. In this region, the camera angles used are virtual camera angles and do not vary greatly. During this time, small corrections are made keeping the aircraft at the desired distance from the target. Figure 21 shows a closer view of what is occuring with $\Delta\psi$ and how it is related to the camera deflection angles during flight mode two in a right turn.

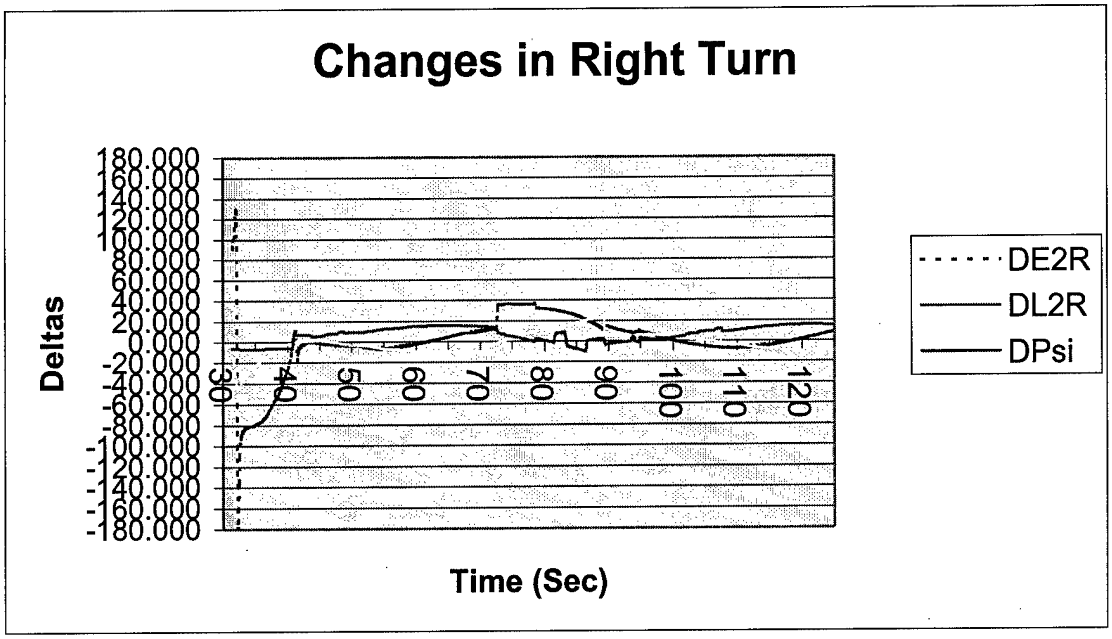


Figure 21. Right Turn

As seen, Δψ changes are small once a circling turn is entered and a direct reflection to the camera λ deflection can be seen. At 72 seconds, a change in the target is accomplished. This can be seen in Figure 21 by the sudden increase in Δε. The response to this sudden change as shown is relatively stable and the aircraft remains in the circling turn. The Δψ changes reorient the aircraft to turn around the new target without having to fly towards the target first. This result is of great significance. Current UAV operations do not respond in this manner and are unable to successfully track moving targets. Figure 22 shows the camera ε deflection angles as determined throughout the mission profile. This figure shows how Δψ responds to the different deflection angles depending upon which mode of flight the UAV is operating.

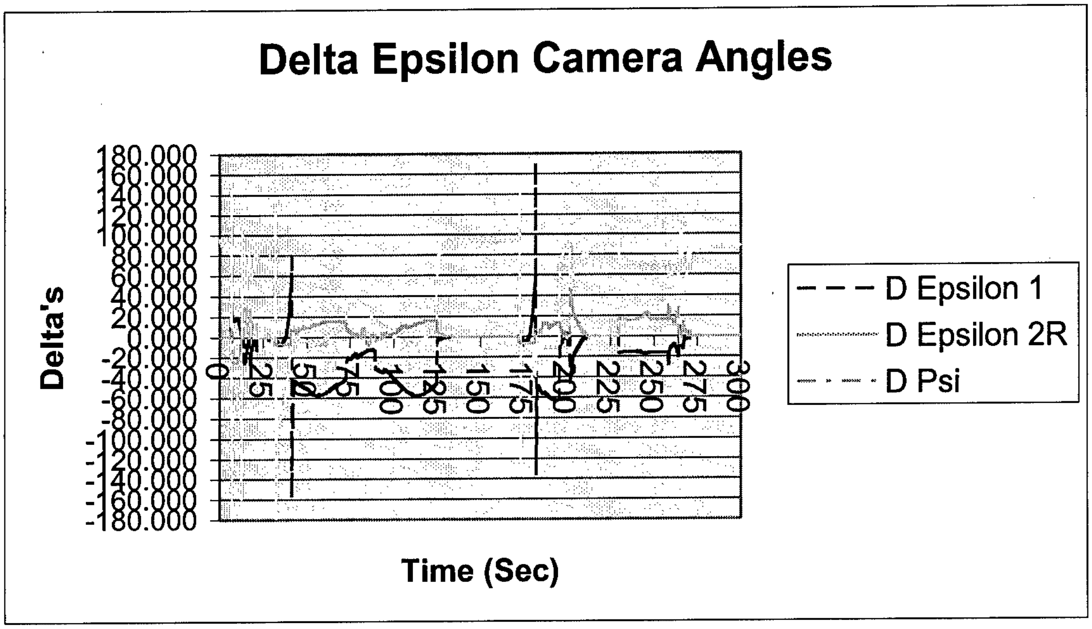


Figure 22. Δε Camera Angles

As seen between 0 and 30 seconds the Δψ is determined from Δε1 and between 30 and 125 seconds, where the UAV is in a right turn, Δψ is determined from Δε2R. Figure 23 shows how well altitude was held to the desired altitude during the mission flown. As seen, the altitude was relatively stable with only a deviation of approximately 50 meters until the end of the profile where a shallow level turn was occuring. Given enough time, the altitude hold algorithm would have corrected this deviation and stabilize on the desired altitude.

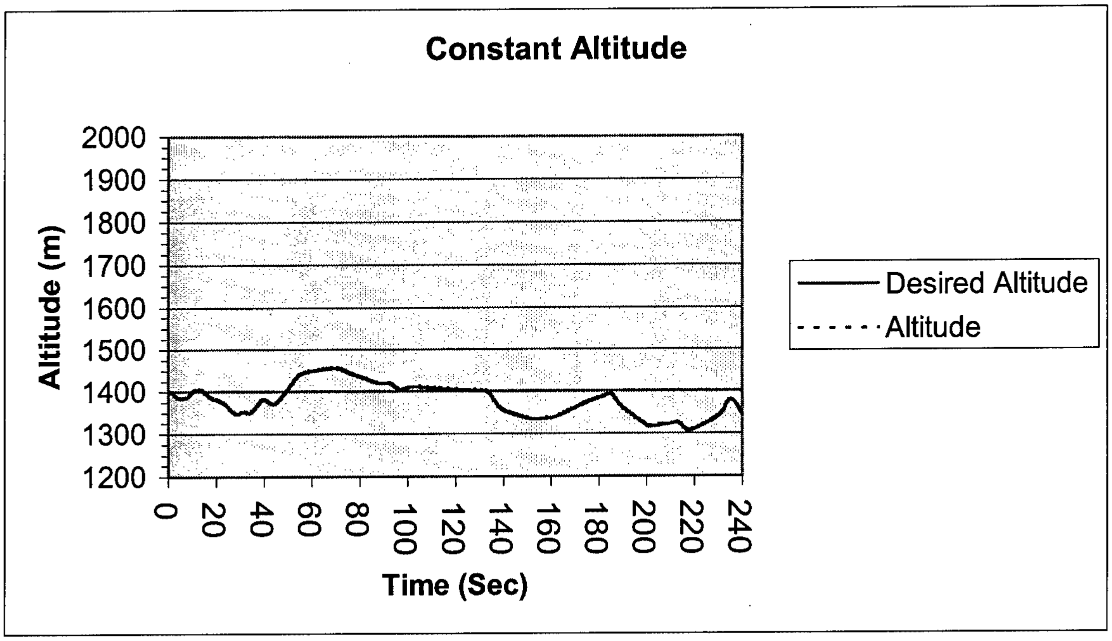


Figure 23. Altitude Hold

Finally, figure 24 demonstrates how well velocity was held to the desired velocity. Velocity, as shown, is somewhat oscillitory. However, the maximum deviation was only 4 m/s from the desired and this occurred early in the mission. As the mission progressed the deviation decreased. Changes in flight modes can be seen by the sudden increases in velocity as the aircraft's bank angles changes. This is a result of velocity being a function of pitch which is dependent upon the bank angle.

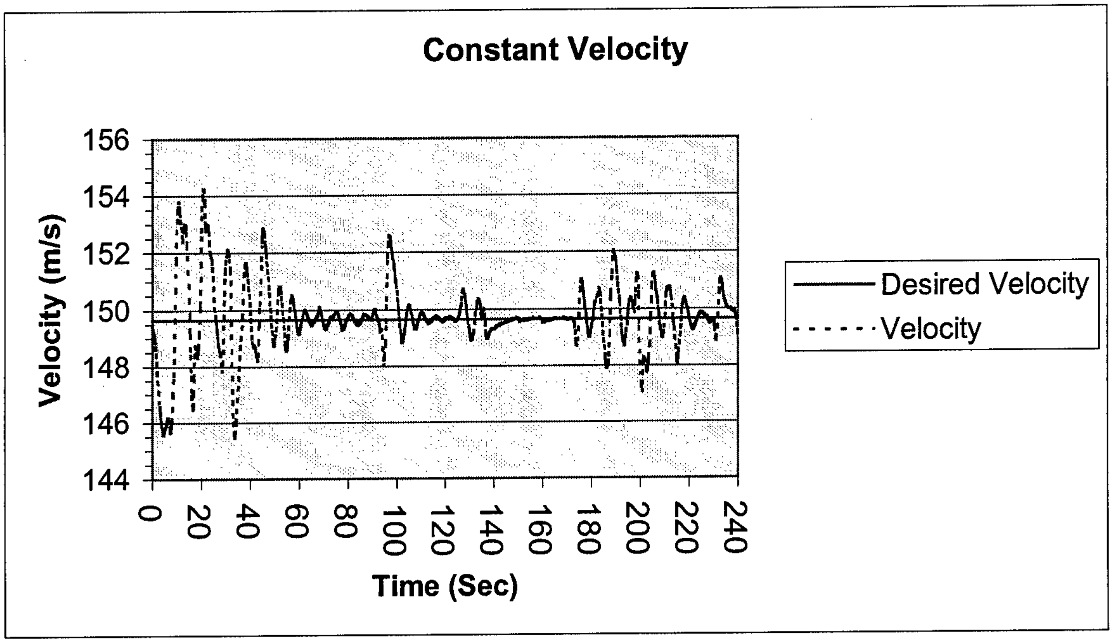


Figure 24. Velocity Hold

The results shown demonstrate the ability of CSRC to handle a diverse mission profile. CSRC is capable of handling an aggressive mission without any significant adverse reactions. Overall, the demonstration confirms the stability of CSRC.

Chapter 5. Conclusion/Recommendations

In conclusion, the theory developed and presented here is valid. As demonstrated, CSRC works. The goal of reducing the operator workload by allowing autonomous control of the UAV flight was achieved. Though the control theory was presented as a whole package, the individual parts are the strengths of CSRC. CSRC demonstrated the ability to track moving targets without the need for the aircraft to exit it's current turn. By remaining in a turn, aircraft response is stable and target tracking is improved. CSRC also provides a method to determine waypoints autonomously from aircraft position and a selected target. For UAVs that operate using waypoints, but require human operators to determine them manually, CSRC can provide a reduction in operator workload required during a mission. Using CSRC, new waypoints can be determined in a more timely fashion while under flight, increasing mission efficiency.

The development of the control algorithm for velocity and altitude hold, as required for the demonstration shows that an aircraft can be successfully controlled by a method other than classical or modern control theory. With the ability of modern control theory to optimize control surface deflections or the development of proper gains based upon classical control theory to increase aircraft performance stability, the best method of control remains an issue for future evaluation.

Of considerable benefit is CSRC's ability to triangulate using two data points. This method shows that any aircraft capable of obtaining its global position and measuring two sensor angles relative to a target can determine a target's position.

Additionally, the use of CSRC can be applied to all types of missions. In missions where close proximity to the Earth is necessary and terrain following is required, CSRC can be combined with an terrain mapping algorithm using task priority. As demonstrated in the CAVE, CSRC can be turned on and off without any impact upon aircraft performance.

It is recommended that continuation of research pertaining to CSRC should occur. The following is a recommended list of potential applications and investigations:

Man-machine interface studies:

1) Four wall projection versus one wall.
2) Various forms of human interaction with the UAV.
3) Capability to handle rapidly moving targets.
4) Level of parameter settings available to the operator.
5) Effects of time delay and/or communication loss.
6) Evaluation of the seat fixed to the aircraft versus fixed to the camera.
7) Comparison of CSRC versus the Predator 2-Operator workstation.

CSRC Extensions:

8) Incorporation of an autonomous threat reaction.
9) Incorporation of autonomous terrain mapping.
10) Alternative aircrafts (F-16 vs A-10)

Improvements/Evaluations:

11) Improvments to the velocity and altitude hold.
12) Test with aircraft perturbations.
13) Test with camera servo dynamics.
14) Test with wind and sensor noise.

This list is far from complete. It is only the beginning of where CSRC can lead. With the Navy and other Department of Defense offices investigating options for new Tactical Terminal to interface with various UAVs, suggestions 1 and 2 are recommended as the front runners for future investigation. The Human Factors issues need to be addressed and employed with CSRC.

As an added safety benefit, collision avoidance can be incorporated into navigation control via priority tasking [8]. Task priority allows an automated system to attempt to accomplish a set of goals. By prioritizing, the system must first be able to accomplish the first task before consideration to the second task can be given [8]. Collision avoidance can be given either first priority or second priority and is accomplished autonomously without operator involvement. By giving collision avoidance first priority, the UAV will avoid a collision, if possible, and then will navigate from the camera angles.

Finally, in support of public consideration is the fact that this is a shared control problem. At all times, there is a man-in-loop. During NASA's history, this has always been an issue of concern. The public fears the negative impacts a completely autonomous system is capable of achieving.

Appendix A

Demonstration Source Code

```
/**********************************************************************
 **********************************************************************/

/**********************************************************************/
/*                         MODULE PURPOSE                             */
/*                                                                    */
/*                                                                    */
/**********************************************************************/

/**********************************************************************/
/*                         INCLUDED FILES                             */
/**********************************************************************/

int fn;
int CrashDataWritten=0;
int GoodLandings=0;
#define CRASH 0
#define LAND  1

float RMSx,RMSy;
double TotalSquaresX,TotalSquaresY;
long TotalN;

int Flag=0;

float AngleA,StickExponent;

#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <time.h>
#include <math.h>

#include "sig_earth_obj.h"
#include "flsim_manager.h"
#include "haptics.h"
```

```
static Flsim_Global_Data *gdp;

#define Xnav 0
#define Ynav -1000.0
#define GLOVE      0
#define THRUSTMASTER 1
#define IMMERSION   2
int InputMode = IMMERSION;

#define NO_DATA      0
#define LANDING_DATA 1
#define CONTINUOUS   2
#define BOTH         3
#define PLAYBACK     4
#define F16  0
#define C130 1
#define B747 2

/*control variables*/
int XmitCount = 0;
int AirCraft = F16;
int FlsimRunning=0;
float Thrust=0.0;
float StickJoyX,StickJoyY;
float JoyX=0.0,JoyY=0.0;
int StickLatched=0,StickLatchCount=0;
float JoyXlatched=0,JoyYlatched=0;
float WheelX=0.0;
float Flaps=0.0,Spoiler=0.0;
int SpeedBrake = 1,Gear = 1;
int HapTrig=0,HapTop=0;
float Ant=0.0,Rng=0.0;

/*Wind Variables*/
#define TURB_MAX 0.33333
#define TWO_PI 2 * 3.14159
#define MAX_SINES 7
float Kx;
float Ky;
float WindTime = 0.0;
float WindX,WindY;
float PerX[MAX_SINES] = {0.251,2.135,3.894,5.401,6.657,8.415,9.922};
float PerY[MAX_SINES] = {0.251,2.135,3.894,5.401,6.657,8.415,9.922 };
float AmpX[MAX_SINES] = { 0.99, 0.95,0.93, 0.85, 0.75, 0.68, 0.59 };
float AmpY[MAX_SINES] = { 0.99, 0.95,0.93, 0.85, 0.75, 0.68, 0.59 };
float OffsetX[MAX_SINES];
float OffsetY[MAX_SINES];

#define LANDING_TIMEOUT 10 /*in seconds*/
#define CRASH_TIMEOUT 1 /*in seconds*/
long FrameCounter;
long OOBcounter;
int FileContinuous = 0;
int Crashed=0, Wow=0, OutOfBounds=0, IdleFlsim=1, SnapShotTaken=0;
int TakeData=0, Practice=0, Subject, Block, N, Location, Haptics=0;
int Turbulance=0, Clouds=0, Trial;
int StartFlag = 1;
int count = 0;
float last_x,last_y,last_z;
float X_Start;
float Y_Start;
float Z_Start;
float H_Start;
float V_Start;

/*UV Variables */
int   uv_latch=0, uv_b = 0, uv_targ_lock, uv_switch, uv_mode = 1;
int   uv_hat_latched = 0, uv_lock_on =0, uv_bomb_hit = 0;
int   uv_bomb_run = 0, uv_bomb = 0, uv_bomb_away = 0, uv_bomb_drop = 0;
int   uv_on = 0, uv_turn_dir, uv_turn_dir_init, uv_theta_des_reset = 1;
float uv_hat,uv_hat_offset, uv_radar, uv_bomb_range, uv_drop_time;
float uv_bomb_x, uv_bomb_y, uv_bomb_z, uv_bomb_h, uv_bomb_v;
float uv_bomb_time = 0.0, uv_bomb_z_init, uv_z_GC;
float uv_bomb_dt, uv_drop_time_rel, uv_bomb_x_init, uv_bomb_y_init;
float uv_stick_angle, uv_stick_x, uv_stick_y, uv_targ_angle_AC;
float uv_targ_angle, uv_targ_angle_GC, uv_targ_angle_GC_Stick;
float uv_targ_dx_AC, uv_targ_dy_AC, uv_targ_dz_AC, uv_targ_hyp;
float uv_targ_max_stick, uv_targ_min_stick, uv_targ_angle_AC_Stick;
float uv_targ_dx_AC_stick, uv_targ_dy_AC_stick, uv_targ_dz_AC_stick;
float uv_targ_hyp_stick, uv_targ_angle_stick, uv_AC_dz_stick;
float uv_targ_dx_GC, uv_targ_dy_GC, uv_targ_dz_GC, uv_x_GC, uv_y_GC;
float uv_targ_dx_GC_stick, uv_targ_dy_GC_stick, uv_targ_dz_GC_stick;
float uv_cam_dx, uv_cam_dy, uv_cam_dz, uv_cam_epsilon, uv_cam_lambda;
float uv_cam_Depsilon_mode1, uv_cam_Dlambda_mode1;
float uv_cam_Depsilon_mode2, uv_cam_Dlambda_mode2L;
float uv_cam_Dlambda_mode2, uv_cam_Depsilon_mode2L;
float uv_cam_Depsilon_mode2R, uv_cam_Dlambda_mode2R;
float uv_cam_lambda_mode2L, uv_cam_epsilon_mode2R;
float uv_cam_lambda_mode2R, uv_cam_epsilon_mode2L;
float uv_cam_lambda_mode1, uv_cam_epsilon_mode1, uv_Rx_mode1;
float uv_Ry_mode1, uv_Rx_level, uv_Rz_mode2L;
float uv_Rz_mode1, uv_cam_lambda_level, uv_cam_epsilon_level;
float uv_Ry_level, uv_Rz_level, uv_Rx_mode2L, uv_Ry_mode2L;
float uv_Rx_mode2R, uv_Ry_mode2R, uv_Rz_mode2R, uv_targ_dx_mode2R;
float uv_targ_dy_mode2R, uv_targ_dy_mode2L, uv_targ_dx_mode2L;
float uv_targ_dx_mode1, uv_targ_dy_mode1, uv_targ_dx_level;
float uv_G_load, uv_VVI, uv_VVI_set, uv_VVI_des, uv_theta_des=0.0;
float uv_turn_rate, uv_pitch_rate_turn, uv_pitch_rate_alt;
float uv_desired_alt, uv_targ_dy_level, uv_pitch_rate_turn_init;
float uv_pitch_rate_turn_mod, uv_theta_zero = 0.0;
float uv_h, uv_p, uv_r, uv_Rx, uv_Ry, uv_Rz, uv_R;
float uv_dpsi, uv_psi, uv_dpsi_set, uv_theta, uv_phi, uv_cos_psi;
float uv_cos_theta, uv_cos_phi, uv_sin_psi, uv_sin_theta, uv_sin_phi;
float uv_roll_rate, uv_pitch_rate, uv_roll_desired;
float uv_pitch_rate_old, uv_pitch_rate_new, uv_pitch_Drate, uv_Dtheta;
float uv_n=2.0, uv_dg, uv_bank_desired, uv_Droll, uv_Alt_hold, uv_Dalt;
float uv_targ_locked_x, uv_targ_locked_y, uv_targ_locked_z;
float uv_stick_angle_angle, uv_targ_angle_angle, uv_targ_angle_zero;
float uv_targ_dx_zero, uv_targ_dy_zero, uv_targ_dz_zero;
float uv_targ_dx_angle, uv_targ_dy_angle, uv_targ_dz_angle;
float uv_cam_Depsilon_mode2_init, uv_turn_radius_stick;
float uv_targ_x_GC_stick, uv_targ_y_GC_stick, uv_targ_z_GC_stick;
float uv_Vel_des, uv_Vel=200.0, uv_Dvel, uv_Vel_set, uv_Vel_new;
float uv_Theta_lock_one = 0.0, uv_theta_set, range_check, uv_thr;
float uv_targ_range, uv_turn_range, uv_turn_radius;
float uv_cam_epsilon_level_init, uv_h_vel, uv_bank_max_var;
float uv_x_vel, uv_y_vel, uv_z_vel, uv_x_acc, uv_y_acc, uv_z_acc;
float uv_p_vel, uv_r_vel, uv_h_acc, uv_p_acc, uv_r_acc;
float uv_x_last,uv_y_last,uv_z_last, uv_x_vel_last, uv_y_vel_last;
flaot uv_z_vel_last;

/*********************************************************************/
/*                    L O C A L   C O N S T A N T S                  */
/*********************************************************************/

#define PI 3.14159
#define MAX_SECONDS 120
#define AIRPORT_ALT 0.0
#define METERS_PER_FEET 1.0/3.28
#define FEET_PER_METER  3.28
#define SECS_PER_HOUR    3600.0f
#define METERS_PER_KNOT  1852.0
#define DEFAULT_SHMKEY 0x30f7d101
#define FRAMES_PER_SECOND 200.0
```

```
#define NOONE_ACCESSING 0
#define MAIN_ACCESSING 1
#define FLSIM_ACCESSING 2
#define FLSIM_RESET 0
#define FLSIM_RUN   1
#define FLSIM_IDLE  2

/*********************************************************************/
/*                          LOCAL DATA                               */
/*********************************************************************/
/*-----------------------------*/
/* Shared-memory segment layout */
/*-----------------------------*/

typedef struct {

        char status;
        char mode;
        int uv_on;
        float ant;
        float rng;
        int start;
        int stop;
        int haptrig;
        int haptop;
        int aircraft;
        int TakeData;
        int practice;
        int Subject;
        int Block;
        int N;
        int Location;
        int Haptics;
        int Turbulance;
        int Clouds;
        int Trial;
        int uv_mode;
        float AC_dz;
        float targ_dz_GC;
        float targ_dz_zero;
        float targ_dz_angle;
        int targ_lock;
        int uv_switch;
        float stick_x;
        float stick_y;
float xstart;
float ystart;
float zstart;
float hstart;
float vstart;
int   onground;
float glove_joyy;
float glove_joyx;
float glove_thr;
float joyy;
float joyx;
float thr;
float flaps;
float spoiler;
int   speedbrake;
int   gear;
float hat;
float vvi;
float x;
float y;
float z;
float h;
float p;
float r;
float h_instr;
float p_instr;
float r_instr;
float knots;
float slip;
int   flightended;
int   landingstatus;
float angleA;
float stickexponent;
int   shmOK;
int   cockpitOK;
int   vegaheartbeat;
int   bomb;
int   bomb_drop;
int   bomb_hit;
float bomb_x;
float bomb_y;
float bomb_z;
int   demohaptics;
int   democlouds;
int   dog;
```

```
} U_Otw_Shm;

static U_Otw_Shm  *shmptr = NULL;
static int        shmid  = -1;

static volatile U_Otw_Shm *shm = 0;

extern void cpt_TransmitInfo(void);
extern void cpt_ReceiveInfo(void);
extern void cpt_Init(void);

void InitFlsim(void);
void ResetFlsim(void);
void RunFlsim(void);
void CameraAngles(void);
void Pilot(void);
void WriteLandingData(void);
void WriteCrashData(void);
void WriteContinuousData(void);
void ReadContinuousData(void);

/**********************************************************************/
/**********************************************************************/
/*              PUBLIC FUNCTIONS CODE SECTION                         */
/**********************************************************************/
/**********************************************************************/

/*----------------------------------------------------------------*/
/*u_shm_init()Creates,initializes and attaches the OTW shared-memory */
/*segment.                                    */
/*                                    */
/* Returned value   A valid pointer to the shared-memory segment if  */
/*              successful or NULL if an error occured.         */
/*----------------------------------------------------------------*/
U_Otw_Shm *u_shm_init( void )
{
        key_t shmkey;

        shmkey = DEFAULT_SHMKEY;

        printf("key = %x, size = %d \n", shmkey, sizeof(*shmptr));

        /*-----------------------------------------------*/
        /* Create (if necessary) the shared-memory segment */
```

```
    /*-----------------------------------------------*/
    shmid = shmget(shmkey, sizeof(*shmptr), IPC_CREAT | 0666);

    if (shmid == -1) {
        perror("\nError - Can't get (create) shared-memory segment");
        return(NULL);
    }

    /*----------------------------------*/
    /* Attach the segment to the process */
    /*----------------------------------*/

    shmptr = (U_Otw_Shm *)shmat(shmid,0,0);

    if (shmptr == (U_Otw_Shm *) -1) {
        perror("\nError - Can't attach to shared-memory segment");
        return(NULL);
    }

    /*---------------------------*/
    /* Lock the segment in memory */
    /*---------------------------*/

    if ( shmctl( shmid,SHM_LOCK,0 ) )
        perror("\nWarning - Can't lock shared-memory segment");

    /*-------------------------------------*/
    /* Initialization completed successfully */
    /*-------------------------------------*/

    return(shmptr);

}

/*---------------------------------------------------------------*/
/*u_shm_exit()Detaches the OTW shared-memory segment from the process*/
/*address space and removes the segment if no more              */
/*processes are attached.                                 */
/*                                                    */
/* Returned value   None                                   */
/*---------------------------------------------------------------*/
void u_shm_exit( void )
{
    struct shmid_ds shmds;
```

```
        /*-----------------------------------------*/
        /* Step 1: Detach the shared-memory segment */
        /*-----------------------------------------*/

        /*----------------------------------------------------*/
        /* The segment is attached if its pointer is non-NULL. */
        /*----------------------------------------------------*/

        if (shmptr)
                if (shmdt(shmptr))
         perror("\nWarning - Can't detach shared-memory segment");
                else
         shmptr = NULL;

/*------------------------------------------------------------*/
/* Step 2: Remove the shared-memory segment if no longer in use */
/*------------------------------------------------------------*/

/*---------------------------------------------------*/
/* Do we previously obtained a valid shared-memory ID? */
/*---------------------------------------------------*/

        if (shmid != -1)

                /*-----------------------------------*/
                /* Get the associated data structure */
                /*-----------------------------------*/

                if (shmctl(shmid,IPC_STAT,&shmds))
         perror("\nWarning - Can't obtain shared-memory data structure");
                else

         /*-----------------------------------*/
         /* How many processes are attached? */
         /*-----------------------------------*/

         if (shmds.shm_nattch == 0)

                 /*----------------------------------*/
                 /* Remove the shared-memory segment */
                 /*----------------------------------*/

                 if (shmctl(shmid,IPC_RMID,0))
                         perror("\nError - Can't remove shared-memory segment");
                 else
```

```
                    shmid = -1;

}

/*** Haptics **********************************************************/

void HapticStick( void )
{
 Haptics = shm->Haptics;

 AngleA = 180.0 - (atan2f(Xnav - uv_x_GC, Ynav - uv_y_GC) * FLSIM_RTD);
        if (AngleA > +180.0) AngleA -= 360.0;
        if (AngleA < -180.0) AngleA += 360.0;

/*   if ((AngleA < -90.0) || (AngleA > 90.0))
          OutOfBounds = 1;
RFG*/

/*printf("\n\rAngleA = %7.2f",AngleA);*/


        if (abs(AngleA) < 30.0) {

          if (StickExponent == 2.0) {
                if (((1.0/30.0) * AngleA) >= 0)
                        hap_Xin = -powf((1.0/30.0) * AngleA,StickExponent);
                else
                        hap_Xin = +powf((1.0/30.0) * AngleA,StickExponent);
          }
          else
                hap_Xin = -powf((1.0/30.0) * AngleA,StickExponent);

          hap_Yin = 0.0;
        }
        else
          hap_Xin = 0.0;

        if (!Haptics) {
          hap_Mode = CPT_HAP_POSITION;
          hap_Yin = 0;
          hap_Xin = 0;
        }
        else
          hap_Mode = CPT_HAP_FORCE;
        if (abs(AngleA) < 30.0) {

         if (StickExponent == 2.0) {
                if (((1.0/30.0) * AngleA) >= 0)
                        hap_Xin = -powf((1.0/30.0) * AngleA,StickExponent);
                else
                        hap_Xin = +powf((1.0/30.0) * AngleA,StickExponent);
         }
         else
                hap_Xin = -powf((1.0/30.0) * AngleA,StickExponent);

         hap_Yin = 0.0;
        }
        else
         hap_Xin = 0.0;

        if (!Haptics) {
         hap_Mode = CPT_HAP_POSITION;
         hap_Yin = 0;
         hap_Xin = 0;
        }
        else
         hap_Mode = CPT_HAP_FORCE;
}

/* GET INPUTS ***************************************************** */

void GetInputs(void)
{
/*      if (XmitCount < 50) {
                        cpt_TransmitInfo();
                        XmitCount++;
                }
*/
                cpt_TransmitInfo();
                cpt_ReceiveInfo();

                StickExponent = shm->stickexponent;

                if (InputMode == GLOVE)
                {
                        Thrust  = shm->glove_thr;
                        JoyX    = shm->glove_joyx;
                        JoyY    = shm->glove_joyy;
                        Flaps   = 0.0;
                        Spoiler = 0.0;
                        SpeedBrake = 1;
                }

                if (InputMode == THRUSTMASTER)
                {
                        JoyX = cpt_Analog[CPT_THM_JOYX];
                        JoyY = cpt_Analog[CPT_THM_JOYY];
                }

                if (InputMode == IMMERSION)
                {
                        JoyX = cpt_Analog[CPT_HAP_JOYX] - JoyXlatched;
                        JoyY = cpt_Analog[CPT_HAP_JOYY] - JoyYlatched;
                }

            if (!StickLatched && (StickLatchCount++ > 400)) {

              JoyXlatched = JoyX;
              JoyYlatched = JoyY;
              StickLatched = 1;

            }

/*  printf("JoyX = %5.2f   JoyY = %5.2f\n\r",JoyX,JoyY);  */

        if ((InputMode == THRUSTMASTER) || (InputMode == IMMERSION)) {

                        Thrust = cpt_Analog[CPT_THM_THR];
                        Flaps = cpt_Analog[CPT_THM_ANT];
                        Spoiler = cpt_Analog[CPT_THM_RNG];
                        Ant = cpt_Analog[CPT_THM_ANT];
                        Rng = cpt_Analog[CPT_THM_RNG];
/*       if (cpt_PosTrans(CPT_HAP_TOP)) SpeedBrake++;
                        if (SpeedBrake > 1) SpeedBrake = 0;
*/
SpeedBrake = 1;

/*                      HapTrig = cpt_PosTrans(CPT_HAP_TRIG);
        HapTop = cpt_PosTrans(CPT_HAP_TOP);

                        if (cpt_PosTrans(CPT_HAP_TRIG)) {
                                HapTrig = 1;
                                shm->haptrig = 1;
                        }
```

```
*/
/*
                                if (cpt_PosTrans(CPT_HAP_TOP)) {
                                        HapTop = 1;
                                        shm->haptop  = 1;
                                }
*/
                                if (cpt_PosTrans(CPT_T6)) {
                                        shm->start  = 1;

                                }
                                if (cpt_PosTrans(CPT_T1)) {
                                        shm->stop  = 1;  /*now turns on uav*/
                                }
                        }

/*   printf("Ant = %5.2f   Rng = %5.2f\n\r",Ant,Rng); */

/*   printf("\n\rTG1 = %5i   T6 = %5i   DOG = %5i",

cpt_Switch(CPT_TG1),cpt_Switch(CPT_T6),cpt_Switch(CPT_DOG));
*/
                        shm->joyx  = JoyX;
                        shm->joyy  = JoyY;
                        shm->ant   = Ant;
                        shm->rng   = Rng;
                        shm->thr   = Thrust;
                        shm->flaps = Flaps;
                        shm->spoiler    = Spoiler;
                        shm->gear       = Gear;
                        shm->speedbrake = SpeedBrake;
                shm->dog = cpt_Switch(CPT_DOG);

}

/* ******************************************************************/

int main(void)
{
        timespec_t currentTime;
        timespec_t startTime;
        double seconds;
        int i;

         char str[200];
        char INPUT[50];
        int i1, i2;
        FILE *Fi;

/*
                sprintf( INPUT,"DATA.DEFAULT");

                Fi = fopen( INPUT, "r" );

fgets(str,81,Fi);
puts(str);
sscanf( str, "%i", &i1);
printf("%3i\n", i1);
fgets(str,81,Fi);
puts(str);
sscanf( str, "%i", &i2);
printf("%3i\n", i2);
*/
/*exit(0);*/

        /*----------------------------*/
        /* attach to shared memory     */
        /*----------------------------*/
        while(!(shm = u_shm_init())){
                sginap( 100 );
                printf( "Waiting for shared memory\n" );
        }

        Kx = 0.0;
        Ky = 0.0;

        /*Compute wind constants given TURB_MAX*/
        for(i=0;i<MAX_SINES;i++) {
                Kx += AmpX[i];
                Ky += AmpY[i];
        }
        Kx = TURB_MAX * (1.0/Kx);
        Ky = TURB_MAX * (1.0/Ky);

        cpt_Init();

        InitFlsim();

        shm->status = NOONE_ACCESSING;
        shm->mode = FLSIM_IDLE;
        shm->start = 0;
        shm->stop = 0;
        shm->cockpitOK = 0;
        shm->shmOK  = 0;
        shm->demohaptics = 0;
        shm->democlouds = 0;
        shm->dog = 0;

        cpt_PosTrans(CPT_T6);

        clock_gettime( CLOCK_REALTIME, &startTime );

        while (1==1) {

                shm->cockpitOK = cpt_OK;
                shm->shmOK  = 1;

                seconds = 0;

                while (seconds < (1.0/FRAMES_PER_SECOND)) {

                  clock_gettime( CLOCK_REALTIME, &currentTime );

                  seconds = ( currentTime.tv_sec - startTime.tv_sec) +
                (currentTime.tv_nsec - startTime.tv_nsec)/1000000000.0;
                }

                clock_gettime( CLOCK_REALTIME, &startTime );

/* printf("\n\rSECONDS = %12.5f %12.5f",(float)seconds,seconds-0.0025);
*/
                while (shm->status == MAIN_ACCESSING);

                shm->status = FLSIM_ACCESSING;

                cpt_OK = 0;

                GetInputs();

                if (shm->mode == FLSIM_RESET) {
                  ResetFlsim();
                  fm_manager_dispatch(1.0/FRAMES_PER_SECOND);
                  RunFlsim();
                  shm->mode = FLSIM_RUN;
                }
                FlsimRunning = 0;

                if (shm->mode == FLSIM_RUN)
                {
                 FlsimRunning = 1;

                 if (!IdleFlsim)
                 {
                        RunFlsim();

                        if (uv_on)
                        {
                               CameraAngles();
                               Pilot();
                        }
                 }
                }

                shm->status = NOONE_ACCESSING;

                if (FlsimRunning)
                 fm_manager_dispatch(1.0/FRAMES_PER_SECOND);

                if (!IdleFlsim)
                 HapticStick();
                else {
                 hap_Mode = CPT_HAP_POSITION;
                 hap_Yin = 0;
                 hap_Xin = 0;

                }
        }
}

float LatLon2Meters(float LatLon)
{
 return LatLon * FLSIM_RTD * 40030000.0 / 360.0;
}

float Meters2LatLon(float Meters)
{
 return Meters / ( FLSIM_RTD * 40030000.0 / 360.0 );
}
```

```
void InitFlsim()
{
          CrashDataWritten=0;

          shm->landingstatus = CRASH;

          TotalSquaresX = 0.0;
          TotalSquaresY = 0.0;
          TotalN = 0;

       uv_theta_des = 0.0;
       uv_lock_on = 0;
       uv_targ_lock = 0;

          if (StartFlag)
          {
                  gdp = fm_manager_init();
                  if (gdp == 0)
                  {
                          printf("fm_manager_init\n");
                          exit(0);
                  }
                  StartFlag = 0;
          }

          if (fm_profile_download_begin() < 0) {
                   printf("fm_profile_download_begin\n");
                   exit(0);  }

          if (fm_profile_atmos_by_name("Default") < 0) {
                   printf("fm_profile_atmos_by_name\n");
                   exit(0);  }

          if (fm_profile_user_by_name("Default") < 0) {
                   printf("fm_profile_user_by_name\n");
                   exit(0);  }

          fm_profile_earth(0.0, 0.0, SIG_EARTH_USER,
                                                   6378206.4, 6356583.8 );

          switch( F16/*AirCraft*/ ) {

                 case F16:
                                 if (fm_profile_aircraft_by_name("F16") < 0) {
                                                  printf("fm_profile_aircraft_by_name\n");
```

```
                                        exit(0);  }

                                break;
/*
                case C130:
                                if (fm_profile_aircraft_by_name("C130") < 0) {
                                        printf("fm_profile_aircraft_by_name\n");
                                        exit(0);  }

                                break;

                case B747:
                                if (fm_profile_aircraft_by_name("B747") < 0) {
                                        printf("fm_profile_aircraft_by_name\n");
                                        exit(0);  }

                                break;
*/
        }

        if (fm_profile_download_end() < 0) {
                printf("fm_profile_download_end\n");
                exit(0);  }

        fm_config_pilot_source( "all=EXTERNAL" );
}

void ResetFlsim()
{
        int i;

Flag = 0;

        last_x = 0.0;
        last_y = 0.0;
        last_z = 0.0;

        OutOfBounds = 0;
        Crashed = 0;
        Wow = 0;
        shm->flightended = 0;

        SpeedBrake = 1;

        /*randomize offsets for wind*/
        for (i=0;i<MAX_SINES;i++) {
                OffsetX[i] = (float)((int)rand()%360)*TWO_PI/360.0;
                OffsetY[i] = (float)((int)rand()%360)*TWO_PI/360.0;
        }

        FileContinuous = 0;
        FrameCounter = 0;
        OOBcounter = 0;
        IdleFlsim = 0;
        SnapShotTaken = 0;

        uv_hat_latched = 0;

        /*for experiment*/
        TakeData   = shm->TakeData;
        Practice   = shm->practice;
        Subject    = shm->Subject;
        Block      = shm->Block;
        N          = shm->N;
        Location   = shm->Location;
        Haptics    = shm->Haptics;
        Turbulance = shm->Turbulance;
        Clouds     = shm->Clouds;
        Trial      = shm->Trial;

        Y_Start = shm->ystart;
        X_Start = shm->xstart;
        Z_Start = shm->zstart;
        H_Start = shm->hstart;
        V_Start = shm->vstart;

        AirCraft = shm->aircraft;

/*V_Start = 0.0;*/

        if (fm_manager_stop() < 0) {
                printf("fm_manager_stop\n");
        }

        InitFlsim();

sginap(2);
/*      printf("STARTV = %5.2f\n",shm->vstart);   */

        fm_config_initial_conditions(
                Meters2LatLon(Y_Start*METERS_PER_FEET),  /*lat*/
                Meters2LatLon(X_Start*METERS_PER_FEET),  /*lon*/
                Z_Start*METERS_PER_FEET,               /*radar alt*/
                AIRPORT_ALT*METERS_PER_FEET,           /*ground elev*/
                0.0,                          /*ground pitch*/
                0.0,                          /*ground roll*/
                0.3,                          /*fuel load*/
                0.0,                          /*flap pos*/
                0.0,                          /*slat pos*/
                0.0,                          /*cla pos*/
                1,                            /*rwy condition*/
                shm->onground,                        /*on ground*/
                0,                            /*gear out*/
                V_Start*0.514,/*Knots2meterspere second*/ /*spd*/
                (360.0 - H_Start) / FLSIM_RTD          /*hdg*/
        );

        printf(
        "\n\rLocation :   X=%9.2f   Y=%9.2f  Z=%9.2f  H=%7.2f  V=%9.2f  OG=%1i
HAT=%9.2f",
        X_Start, Y_Start,
        Z_Start, H_Start,
        V_Start, shm->onground, shm->hat);

/*      55.667, 120mph*/

         if (fm_manager_start() < 0) {
                printf("fm_manager_start\n");
                exit(0);
         }

/*    fm_manager_dispatch(1.0/FRAMES_PER_SECOND);

         fm_runtime_set_flaps(Flaps);
         fm_runtime_set_throttle(Thrust, Thrust, Thrust, Thrust);
*/
}

void RunFlsim(void)
{
    int i;

      if (!uv_on) {

        if (cpt_Switch(CPT_RAD_U)) {
```

```
      shm->demohaptics = 1;
     }
       if (cpt_Switch(CPT_RAD_D)) {
      shm->demohaptics = 0;
     }
       if (cpt_Switch(CPT_RAD_R)) {
      shm->democlouds = 1;
     }
       if (cpt_Switch(CPT_RAD_L)) {
      shm->democlouds = 0;
     }

    }

printf("\n\rH=%5i  C=%5i",shm->demohaptics,shm->democlouds);


      WindX = 0.0;
      WindY = 0.0;

      for(i=0;i<MAX_SINES;i++)
   {
       WindX += Kx*(AmpX[i] * sinf(TWO_PI*WindTime/PerX[i]+OffsetX[i]));
       WindY += Ky*(AmpY[i] * sinf(TWO_PI*WindTime/PerY[i]+OffsetY[i]));
      }

      if (!Turbulance)
   {
       WindX = 0;
       WindY = 0;
      }

      WindTime += 1.0/((float)FRAMES_PER_SECOND);

      uv_on = shm->uv_on;

      fm_runtime_set_throttle(Thrust, Thrust, Thrust, Thrust);

   fm_runtime_ap_basic_off();

/*     printf("THRUST = %8.3f\n",shm->thr);   */

      StickJoyX = JoyX + WindX;
      StickJoyY = JoyY + WindY;
      if (StickJoyX > +1.0) StickJoyX = +1.0;
```

```
        if (StickJoyX < -1.0) StickJoyX = -1.0;
        if (StickJoyY > +1.0) StickJoyY = +1.0;
        if (StickJoyY < -1.0) StickJoyY = -1.0;

/*printf("\n\rX=%5.2f  Y=%5.2f",StickJoyX, StickJoyY);
*/

        if (!uv_on)
          fm_runtime_set_stick(StickJoyY, StickJoyX);

/*    fm_runtime_set_flaps(shm->flaps);
         fm_runtime_set_spoiler(shm->spoiler);
*/
        fm_runtime_set_flaps(1.0);
        fm_runtime_set_spoiler(1.0);

/*   if (!SpeedBrake)
          fm_runtime_set_spd_brk_out();
*/
/*   if (SpeedBrake)
*/
        fm_runtime_set_spd_brk_in();

        WheelX = JoyX;
        if ( (JoyX > -0.005) && (JoyX < 0.005) ) WheelX = 0.0;
          fm_runtime_set_steering_wheel(0.005*WheelX);

/*   if (!uv_hat_latched)
         {
          uv_hat_offset = shm->hat;
          uv_hat_latched = 1;
         }
*/
        uv_hat_offset = 0;
        uv_hat = shm->hat-uv_hat_offset;

    uv_radar =
      gdp->instruments.runtime_output.flsim.flight.alt_radar;
        Crashed =  gdp->position.runtime_output.flsim.crashed;
        Wow =  gdp->undercarriage.runtime_output.flsim.installation->wow;
/*
printf("\n\rCrashed = %i  Wow = %i  OutOfBounds = %i",
Crashed, Wow, OutOfBounds);
*/
                              OOBcounter++;
```

```
/*       if (OOBcounter > (MAX_SECONDS*FRAMES_PER_SECOND))
                                 OutOfBounds = 1;
RFG*/
        if (SnapShotTaken)
          FrameCounter++;
        else
{
          TotalSquaresX += (double)pow(JoyX,2);
          TotalSquaresY += (double)pow(JoyY,2);
          TotalN++;
        }

        if (Crashed || (Wow && (FrameCounter >= LANDING_TIMEOUT *
FRAMES_PER_SECOND)) || OutOfBounds)
    {
          if (!CrashDataWritten)
            WriteCrashData();
            shm->flightended = 1;
            shm->mode = FLSIM_IDLE;
            IdleFlsim = 1;
        }
/*
if (!Flag) {
printf("\n\rCrashed = %i  RadarAlt = %9.1f  Hat = %9.2f",
Crashed, uv_radar,uv_hat);
Flag=1;
}
*/
        if ((Crashed || Wow || OutOfBounds) && !SnapShotTaken)
        {
          RMSx = (float) sqrtf(TotalSquaresX/((double)TotalN));
          RMSy = (float) sqrtf(TotalSquaresY/((double)TotalN));

              if  ((TakeData  ==  LANDING_DATA)  ||  (TakeData  ==  BOTH))
WriteLandingData();
            SnapShotTaken = 1;
        }

        if (FrameCounter == (CRASH_TIMEOUT * FRAMES_PER_SECOND))
        {
          WriteCrashData();

          GoodLandings++;
```

```
  shm->landingstatus = LAND;

  printf("\n\r\n\rNO. OF GOOD LANDINGS = %i\n\r\n\r",GoodLandings);
 }

 if ((TakeData == CONTINUOUS) || (TakeData == BOTH))
WriteContinuousData();

 last_x = uv_x_GC;
 last_y = uv_y_GC;
 last_z = uv_z_GC;

 shm->vvi = uv_VVI = gdp->instruments.runtime_output.flsim.flight.vsi;
 shm->x = uv_x_GC =
   LatLon2Meters(gdp->position.runtime_output.flsim.ecg.lon);
 shm->y = uv_y_GC =
   LatLon2Meters(gdp->position.runtime_output.flsim.ecg.lat);
 shm->h_instr = uv_h =
  -gdp->instruments.runtime_output.flsim.flight.true_hdg * FLSIM_RTD;
 shm->p_instr = uv_p =
   gdp->instruments.runtime_output.flsim.flight.pitch * FLSIM_RTD;
 shm->r_instr = uv_r =
   gdp->instruments.runtime_output.flsim.flight.roll  * FLSIM_RTD;
 shm->z = uv_z_GC =
   gdp->instruments.runtime_output.flsim.flight.alt_baro;
 shm->knots = gdp->instruments.runtime_output.flsim.flight.ias
                    * (SECS_PER_HOUR / METERS_PER_KNOT);
 uv_Vel = gdp->instruments.runtime_output.flsim.flight.ias;

 if (!uv_on)
 {
  uv_Vel_new = uv_Vel;
  uv_Alt_hold = uv_z_GC;
 }

 uv_G_load = gdp->instruments.runtime_output.flsim.flight.g_load;

 shm->h = -gdp->attitude.runtime_output.flsim.psi*FLSIM_RTD;
 shm->p =  gdp->attitude.runtime_output.flsim.theta*FLSIM_RTD;
 shm->r =  gdp->attitude.runtime_output.flsim.phi*FLSIM_RTD;
 uv_h_vel = -gdp->attitude.runtime_output.flsim.psi_dot*FLSIM_RTD;
 uv_p_vel =
  gdp->attitude.runtime_output.flsim.theta_dot*FLSIM_RTD;
 uv_r_vel =  gdp->attitude.runtime_output.flsim.phi_dot*FLSIM_RTD;
```

```
        uv_h_acc =  -gdp->attitude.runtime_output.flsim.psi_dot_dot*FLSIM_RTD;
        uv_p_acc =  gdp->attitude.runtime_output.flsim.theta_dot_dot*FLSIM_RTD;
        uv_r_acc =  gdp->attitude.runtime_output.flsim.phi_dot_dot*FLSIM_RTD;

        shm->slip = gdp->forces.runtime_output.flsim.wcs.acc.y;

        uv_x_vel_last = uv_x_vel;
        uv_y_vel_last = uv_y_vel;
        uv_z_vel_last = uv_z_vel;

        uv_x_vel = (uv_x_GC - last_x) * FRAMES_PER_SECOND;
        uv_y_vel = (uv_y_GC - last_y) * FRAMES_PER_SECOND;
        uv_z_vel = (uv_z_GC - last_z) * FRAMES_PER_SECOND;

        uv_x_acc = (uv_x_vel - uv_x_vel_last) * FRAMES_PER_SECOND;
        uv_y_acc = (uv_y_vel - uv_y_vel_last) * FRAMES_PER_SECOND;
        uv_z_acc = (uv_z_vel - uv_z_vel_last) * FRAMES_PER_SECOND;

        if (TakeData == PLAYBACK) ReadContinuousData();

        fm_runtime_set_hat( shm->hat-uv_hat_offset, uv_z_GC, 0.0, 0.0 );

        shm->uv_mode = uv_mode;
        uv_stick_x = -JoyX;
        uv_stick_y = -JoyY;

        uv_AC_dz_stick = shm->AC_dz;
        uv_targ_dz_GC_stick = shm->targ_dz_GC;
        uv_targ_dz_zero = shm->targ_dz_zero;
        uv_targ_dz_angle = shm->targ_dz_angle;

/* Creates a dead spot in the stick if in flight mode 1 */
        if (uv_mode == 1)
        {
         if ((uv_stick_x < 0.05) && (uv_stick_x > -0.05))
           uv_stick_x = 0.0;
         if ((uv_stick_y < 0.05) && (uv_stick_y > -0.05))
           uv_stick_y = 0.0;
        }

/* Protects a divide by zero */
        if ((uv_stick_x == 0.0) && (uv_stick_y == 0.0))
         uv_stick_angle = 0.0;
        else
         uv_stick_angle = -atan2f(uv_stick_x,uv_stick_y) *FLSIM_RTD;
        uv_targ_angle_AC_Stick =  uv_stick_angle;

/* minus angle due to scene coordinate system being +90 deg equals west*/
        uv_targ_angle_stick = (-uv_h + uv_stick_angle);

/* Determines stick angle in global coordinate system */
        if (uv_targ_angle_stick > 180.0)
          uv_targ_angle_GC_Stick = uv_targ_angle_stick - 360.0;
        else if (uv_targ_angle < -180.0)
          uv_targ_angle_GC_Stick = uv_targ_angle_stick + 360.0;
        else
          uv_targ_angle_GC_Stick = uv_targ_angle_stick;

/* Limits the max range to a camera angle of 85 degrees */
        uv_targ_max_stick =
          sqrt(pow((uv_AC_dz_stick*tanf(85*FLSIM_DTR)),2));

/* Determines the turn radius based upon Velocity and max g's(n) */
        uv_turn_radius_stick = pow(uv_Vel,2)/
                    (9.8 * sqrtf(pow(uv_n,2)-1));

/* Forces the target to be outside 2*turn radius */
        uv_targ_min_stick = sqrt(4*pow(uv_turn_radius_stick,2));

/* Allows target to be anywhere when in turn mode */
        if (uv_mode == 23)
          uv_targ_min_stick = 0.0;

/* Determines the range to target based upon distance allowed and stick angle */
        uv_targ_hyp_stick =
          sqrtf(pow(uv_stick_x,2)+pow(uv_stick_y,2))*uv_targ_max_stick;

        if (uv_targ_hyp_stick < uv_targ_min_stick)
          uv_targ_hyp_stick = uv_targ_min_stick;

/* Calculates the distances based upon stick angle and range (both AC & GC) */
        uv_targ_dx_AC_stick = cosf(uv_targ_angle_AC_Stick *
                                  FLSIM_DTR)*uv_targ_hyp_stick;
        uv_targ_dy_AC_stick = sinf(uv_targ_angle_AC_Stick *
                                  FLSIM_DTR)*uv_targ_hyp_stick;
        uv_targ_dx_GC_stick = sinf(uv_targ_angle_GC_Stick *
                                  FLSIM_DTR)*uv_targ_hyp_stick;
        uv_targ_dy_GC_stick = cosf(uv_targ_angle_GC_Stick *
                                  FLSIM_DTR)*uv_targ_hyp_stick;
```

```
/*Determine the XYZ target position from the AC position and distances */
        uv_targ_x_GC_stick = uv_x_GC + uv_targ_dx_GC_stick;
        uv_targ_y_GC_stick = uv_y_GC + uv_targ_dy_GC_stick;
        uv_targ_z_GC_stick = uv_z_GC - uv_targ_dz_GC_stick;

        uv_switch = 1;

/* Toggles top hat through Operator modes */
        if (cpt_PosTrans(CPT_HAP_TOP))
        {
          uv_targ_lock++;
          uv_switch = 0;
      uv_lock_on = 0;

          if (uv_targ_lock > 3)
            uv_targ_lock = 0;

/* If in turn locks out operator mode 2 & 3 */
          if (uv_mode == 23 && uv_targ_lock > 1)
            uv_targ_lock = 1;

/* Sends synch signal to shared memory for coordination with scene */
          if (uv_targ_lock == 1)
          {
            uv_latch = 1;
            shm->vegaheartbeat = 0;
          }

/* Locks camera angle in operator mode 2 */
          if (uv_targ_lock == 2)
          {
            if (-uv_h > 180.0)
              uv_targ_angle_zero = -uv_h - 360.0;
      else if (-uv_h < -180.0)
                  uv_targ_angle_zero = -uv_h + 360.0;
            else
                  uv_targ_angle_zero = -uv_h;
          }

/* Locks camera angle in operator mode 3 */
          if (uv_targ_lock == 3)
            uv_stick_angle_angle = uv_stick_angle;
        }
```

```
/* Set target locked position for different operator modes */
        if (uv_targ_lock == 0)
        {
         uv_targ_locked_x = uv_targ_x_GC_stick;
         uv_targ_locked_y = uv_targ_y_GC_stick;
         uv_targ_locked_z = uv_targ_z_GC_stick;
        }

        if (uv_targ_lock == 1 && uv_latch && shm->vegaheartbeat)
        {
         uv_latch = 0;
         uv_targ_locked_x = uv_targ_x_GC_stick;
         uv_targ_locked_y = uv_targ_y_GC_stick;
         uv_targ_locked_z = uv_targ_z_GC_stick;
        }

        if (uv_targ_lock == 2)
        {
         uv_targ_dx_zero = sinf(uv_targ_angle_zero *
       FLSIM_DTR)*uv_targ_min_stick;

         uv_targ_dy_zero = cosf(uv_targ_angle_zero *
       FLSIM_DTR)*uv_targ_min_stick;

         uv_targ_locked_x = uv_x_GC + uv_targ_dx_zero;
         uv_targ_locked_y = uv_y_GC + uv_targ_dy_zero;
         uv_targ_locked_z = uv_z_GC - uv_targ_dz_zero;
        }

        if (uv_targ_lock == 3)
        {
         uv_targ_angle_angle = (-uv_h + uv_stick_angle_angle);

         if (uv_targ_angle_angle > 180.0)
          uv_targ_angle_angle = uv_targ_angle_angle - 360.0;
         else if (uv_targ_angle_angle < -180.0)
          uv_targ_angle_angle = uv_targ_angle_angle + 360.0;
         else
          uv_targ_angle_angle = uv_targ_angle_angle;

         uv_targ_dx_angle = sinf(uv_targ_angle_angle *
                                FLSIM_DTR)*uv_targ_min_stick;
        uv_targ_dy_angle = cosf(uv_targ_angle_angle *
                           FLSIM_DTR)*uv_targ_min_stick;

        uv_targ_locked_x = uv_x_GC + uv_targ_dx_angle;
        uv_targ_locked_y = uv_y_GC + uv_targ_dy_angle;
        uv_targ_locked_z = uv_z_GC - uv_targ_dz_angle;
       }


/* Calculates distance to target from locked position and AC position */
      uv_targ_dx_GC_stick = (uv_targ_locked_y - uv_y_GC);
      uv_targ_dy_GC_stick = (uv_targ_locked_x - uv_x_GC);
      uv_targ_dz_AC_stick = uv_targ_dz_GC_stick = (uv_z_GC –
               uv_targ_locked_z);

 /* Back calculates angle to target */
      if ((uv_targ_dy_GC == 0.0) && (uv_targ_dx_GC == 0.0))
        uv_targ_angle_GC = 0.0;
      else
        uv_targ_angle_GC =
     atan2f(uv_targ_dy_GC,uv_targ_dx_GC)*FLSIM_RTD;
      uv_targ_angle = uv_targ_angle_GC + uv_h;

      if (uv_targ_angle > 180.0)
        uv_targ_angle_AC = uv_targ_angle - 360.0;
      else if (uv_targ_angle < -180.0)
        uv_targ_angle_AC = uv_targ_angle + 360.0;
      else
        uv_targ_angle_AC = uv_targ_angle;

      uv_targ_hyp = sqrtf(pow(uv_targ_dx_GC,2)+pow(uv_targ_dy_GC,2));

      uv_targ_dx_AC = cosf(uv_targ_angle_AC * FLSIM_DTR)*uv_targ_hyp;
      uv_targ_dy_AC = sinf(uv_targ_angle_AC * FLSIM_DTR)*uv_targ_hyp;

/* Obtains Euler angles and trig values from FLSIM */
      uv_phi  = gdp->attitude.runtime_output.flsim.phi;   /*roll*/
      uv_psi      = gdp->attitude.runtime_output.flsim.psi;/*heading*/
      uv_theta     = gdp->attitude.runtime_output.flsim.theta;/*pitch*/
      uv_cos_phi   = gdp->attitude.runtime_output.flsim.cos_phi;
      uv_cos_theta = gdp->attitude.runtime_output.flsim.cos_theta;
      uv_cos_psi   = gdp->attitude.runtime_output.flsim.cos_psi;
      uv_sin_phi   = gdp->attitude.runtime_output.flsim.sin_phi;
      uv_sin_theta = gdp->attitude.runtime_output.flsim.sin_theta;
      uv_sin_psi   = gdp->attitude.runtime_output.flsim.sin_psi;
```

```
        if (!uv_switch)
          shm->uv_switch = uv_switch;

        shm->targ_lock = uv_targ_lock;

}

/* Determines the camera angles for simulation purposes */
void CameraAngles( void )
{
/* Calculates the distance in the Aircraft frame from GC position */
        uv_Rx = uv_cos_theta * uv_cos_psi * uv_targ_dx_GC_stick
               + uv_cos_theta * uv_sin_psi * uv_targ_dy_GC_stick
               - uv_sin_theta * uv_targ_dz_GC_stick;
        uv_Ry = (uv_sin_phi * uv_sin_theta * uv_cos_psi - uv_cos_phi *
                 uv_sin_psi)* uv_targ_dx_GC_stick
               + (uv_sin_phi * uv_sin_theta * uv_sin_psi + uv_cos_phi *
                 uv_cos_psi)* uv_targ_dy_GC_stick
               +  uv_sin_phi * uv_cos_theta * uv_targ_dz_GC_stick;
        uv_Rz = (uv_cos_phi * uv_sin_theta * uv_cos_psi + uv_sin_phi *
                 uv_sin_psi)* uv_targ_dx_GC_stick
               + (uv_cos_phi * uv_sin_theta * uv_sin_psi - uv_sin_phi *
                 uv_cos_psi)* uv_targ_dy_GC_stick
               +  uv_cos_phi * uv_cos_theta * uv_targ_dz_GC_stick;

/* Determines the range */
        uv_R = sqrtf(powf(uv_Rx,2) + powf(uv_Ry,2) + powf(uv_Rz,2));

/* Calculates epsilon, azimuth angle */

        if ((uv_Rx == 0.0) && (uv_Ry == 0.0))
          uv_cam_epsilon = 0.0;
        else
          uv_cam_epsilon = atan2f(-uv_Rx,uv_Ry);

/* Calculates lambda, elevation angle */

        if ((uv_Rz == 0.0) && ((uv_Rx * sinf(uv_cam_epsilon)
                    - uv_Ry * cosf(uv_cam_epsilon)) == 0.0))
          uv_cam_lambda = 0.0;
        else
          uv_cam_lambda =  atan2f(-uv_Rz,
                    (uv_Rx * sinf(uv_cam_epsilon)
                    - uv_Ry * cosf(uv_cam_epsilon)));
```

```
/* Checks */
        uv_cam_dx =  cosf(uv_cam_epsilon)* uv_Rx + sinf(uv_cam_epsilon)
                      * uv_Ry;
        uv_cam_dy = - cosf(uv_cam_lambda) * sinf(uv_cam_epsilon)* uv_Rx
                     + cosf(uv_cam_lambda) * cosf(uv_cam_epsilon)* uv_Ry
                     + sinf(uv_cam_lambda) * uv_Rz;
        uv_cam_dz =  sinf(uv_cam_lambda) * sinf(uv_cam_epsilon)* uv_Rx
                     - sinf(uv_cam_lambda) * cosf(uv_cam_epsilon)* uv_Ry
                    + cosf(uv_cam_lambda) * uv_Rz;
}


/* Actual flight command determination and commands */
void Pilot( void )
{
/* Toggles for changing desired altitude and velocity */
        if (cpt_Switch(CPT_RAD_U)) {
         uv_Alt_hold++;
        }
        if (cpt_Switch(CPT_RAD_D)) {
         uv_Alt_hold--;
        }
        if (cpt_Switch(CPT_RAD_R)) {
         uv_Vel_new++;
        }
        if (cpt_Switch(CPT_RAD_L)) {
         uv_Vel_new--;
        }

/* Toggles for changing desired altitude and velocity */


/* Set velocity desired to new velocity and resets counter for zero lift AOA */
        if ((uv_Vel_new - uv_Vel_des) != 0.0)
        {
         uv_Vel_des = uv_Vel_new;
         uv_b = 0;
        }

/* Determines velocity error */
        uv_Dvel = uv_Vel_des - uv_Vel;
```

```
/* Throttle algorithm, Alters throttle setting until velocity changes in direction of desired
velocity, when velocity is within 1m/s of desired velocity, gains are reduced from a
constant to a variable to help zero in on desired velocity */
        if (uv_Dvel > 1.0 && uv_Vel < uv_Vel_set)
          uv_thr = uv_thr + 0.001;
        else if (uv_Dvel < -1.0 && uv_Vel > uv_Vel_set)
          uv_thr = uv_thr - 0.001;
        else if (uv_Dvel > 0.0 && uv_Dvel < 1.0 && uv_Vel < uv_Vel_set)
          uv_thr = uv_thr + 0.001*uv_Dvel;
        else if (uv_Dvel < 0.0 && uv_Dvel > -1.0 && uv_Vel > uv_Vel_set)
          uv_thr = uv_thr + 0.001*uv_Dvel;

/* Sets check parameter to determine direction of velocity change */
        uv_Vel_set = uv_Vel;

/* Set throttle */
        fm_runtime_set_throttle(uv_thr, uv_thr, uv_thr, uv_thr);

/* Calculates aircraft position based upon Angles and range */

        uv_Rx = -cosf(uv_cam_lambda) * sinf(uv_cam_epsilon)* uv_R;
        uv_Ry =  cosf(uv_cam_lambda) *  cosf(uv_cam_epsilon)* uv_R;
        uv_Rz =  sinf(uv_cam_lambda) * uv_R;

        uv_targ_dx_GC = -((uv_cos_theta * uv_cos_psi) * uv_Rx
                       + (uv_sin_phi * uv_sin_theta * uv_cos_psi -
                          uv_cos_phi * uv_sin_psi) * uv_Ry
                       + (uv_cos_phi * uv_sin_theta * uv_cos_psi +
                           uv_sin_phi * uv_sin_psi) * uv_Rz);

        uv_targ_dy_GC = -((uv_cos_theta * uv_sin_psi) * uv_Rx
                       + (uv_sin_phi * uv_sin_theta * uv_sin_psi +
                           uv_cos_phi * uv_cos_psi) * uv_Ry
                       + (uv_cos_phi * uv_sin_theta * uv_sin_psi –
                          uv_sin_phi * uv_cos_psi) * uv_Rz);

        uv_targ_dz_GC = -((- uv_sin_theta) * uv_Rx
            + (uv_sin_phi * uv_cos_theta) * uv_Ry
            + (uv_cos_phi * uv_cos_theta) * uv_Rz);

        uv_targ_range = sqrt(pow(uv_targ_dx_GC,2)+pow(uv_targ_dy_GC,2));
        uv_turn_radius = pow(uv_Vel,2)/(9.8 * sqrtf(pow(uv_n,2)-1));

/* Determines epsilon as if Aircraft is in level flight */
```

```
        uv_Rx_level = cosf(-uv_h * FLSIM_DTR) * uv_targ_dx_GC
                      + sinf(-uv_h * FLSIM_DTR) * uv_targ_dy_GC;
        uv_Ry_level = - (sinf(-uv_h * FLSIM_DTR)) * uv_targ_dx_GC
                      + ( cosf(-uv_h * FLSIM_DTR)) * uv_targ_dy_GC;
        uv_Rz_level = uv_targ_dz_GC;

        if ((uv_Rx_level == 0.0) && (uv_Ry_level == 0.0))
         uv_cam_epsilon_level = 0.0;
        else
         uv_cam_epsilon_level = atan2f(-uv_Rx_level,uv_Ry_level);

        if ((uv_Rz_level== 0.0) && ((uv_Rx_level *
           sinf(uv_cam_epsilon_level) - uv_Ry_level *
           cosf(uv_cam_epsilon_level)) == 0.0))
         uv_cam_lambda_level = 0.0;
        else
         uv_cam_lambda_level =  atan2f(-uv_Rz_level,
                                  (uv_Rx_level * sinf(uv_cam_epsilon_level)-
                                   uv_Ry_level *cosf(uv_cam_epsilon_level)));

/* Determines the desired camera angles for Model flight */
/* Desires targ off nose ie. dx_AC = range, dy_AC = 0 */

        uv_targ_dx_model =  cosf(uv_h * FLSIM_DTR)*uv_targ_range;
        uv_targ_dy_model = -sinf(uv_h * FLSIM_DTR)*uv_targ_range;

        uv_Rx_model = uv_cos_theta * uv_cos_psi * uv_targ_dx_model
                      + uv_cos_theta * uv_sin_psi * uv_targ_dy_model
                       - uv_sin_theta * uv_targ_dz_GC;
        uv_Ry_model = (uv_sin_phi * uv_sin_theta * uv_cos_psi –
                        uv_cos_phi * uv_sin_psi) * uv_targ_dx_model
                     + (uv_sin_phi * uv_sin_theta * uv_sin_psi +
                        uv_cos_phi * uv_cos_psi) * uv_targ_dy_model
                      +  uv_sin_phi * uv_cos_theta * uv_targ_dz_GC;
        uv_Rz_model = (uv_cos_phi * uv_sin_theta * uv_cos_psi +
                         uv_sin_phi * uv_sin_psi) * uv_targ_dx_model
                      + (uv_cos_phi * uv_sin_theta * uv_sin_psi –
                        uv_sin_phi * uv_cos_psi) * uv_targ_dy_model
                      +  uv_cos_phi * uv_cos_theta * uv_targ_dz_GC;

        if ((uv_Rx_model == 0.0) && (uv_Ry_model == 0.0))
         uv_cam_epsilon_model = 0.0;
        else
         uv_cam_epsilon_model = atan2f(-uv_Rx_model,uv_Ry_model);
        if ((uv_Rz_mode1== 0.0) && ((uv_Rx_mode1 *
                sinf(uv_cam_epsilon_mode1)-
                uv_Ry_mode1 * cosf(uv_cam_epsilon_mode1)) == 0.0))
          uv_cam_lambda_mode1 = 0.0;
        else
          uv_cam_lambda_mode1 =  atan2f(-uv_Rz_mode1,
                                (uv_Rx_mode1 * sinf(uv_cam_epsilon_mode1)-
                                 uv_Ry_mode1 * cosf(uv_cam_epsilon_mode1)));

/* Determines the desired camera angles for Mode2 flight */
/* Desires targ off rt wing ie. dx = 0, dy =  turn_radius, mode 2R */

        uv_targ_dx_mode2R =  cosf((uv_h-90.0) *
                                  FLSIM_DTR)*uv_turn_radius;
        uv_targ_dy_mode2R = -sinf((uv_h-90.0) *
                                  FLSIM_DTR)*uv_turn_radius;

        uv_Rx_mode2R = uv_cos_theta * uv_cos_psi * uv_targ_dx_mode2R
                     + uv_cos_theta * uv_sin_psi * uv_targ_dy_mode2R
                     - uv_sin_theta * uv_targ_dz_GC;
        uv_Ry_mode2R = (uv_sin_phi * uv_sin_theta * uv_cos_psi –
                        uv_cos_phi * uv_sin_psi) * uv_targ_dx_mode2R
                     + (uv_sin_phi * uv_sin_theta * uv_sin_psi +
                       uv_cos_phi * uv_cos_psi)* uv_targ_dy_mode2R
                     +  uv_sin_phi * uv_cos_theta * uv_targ_dz_GC;
        uv_Rz_mode2R = (uv_cos_phi * uv_sin_theta * uv_cos_psi +
                        uv_sin_phi * uv_sin_psi)* uv_targ_dx_mode2R
                     + (uv_cos_phi * uv_sin_theta * uv_sin_psi –
                       uv_sin_phi * uv_cos_psi)* uv_targ_dy_mode2R
                    +  uv_cos_phi * uv_cos_theta * uv_targ_dz_GC;

        if ((uv_Rx_mode2R == 0.0) && (uv_Ry_mode2R == 0.0))
          uv_cam_epsilon_mode2R = 0.0;
        else
          uv_cam_epsilon_mode2R = atan2f(-uv_Rx_mode2R,uv_Ry_mode2R);

        if    ((uv_Rz_mode2R== 0.0) && ((uv_Rx_mode2R *
          sinf(uv_cam_epsilon_mode2R) - uv_Ry_mode2R *
          cosf(uv_cam_epsilon_mode2R)) == 0.0))
          uv_cam_lambda_mode2R = 0.0;
        else
          uv_cam_lambda_mode2R =  atan2f(-uv_Rz_mode2R,
                                     (uv_Rx_mode2R *
                                         sinf(uv_cam_epsilon_mode2R) -
                                      uv_Ry_mode2R *
```

```
                                    cosf(uv_cam_epsilon_mode2R)));

/*Desires targ off lt wing ie. dx = 0, dy = -turn_radius, mode 2L */
        uv_targ_dx_mode2L =  cosf((uv_h+90.0) *
                                  FLSIM_DTR)*uv_turn_radius;
        uv_targ_dy_mode2L = -sinf((uv_h+90.0) *
                                  FLSIM_DTR)*uv_turn_radius;

        uv_Rx_mode2L = uv_cos_theta * uv_cos_psi * uv_targ_dx_mode2L
                     + uv_cos_theta * uv_sin_psi * uv_targ_dy_mode2L
                     - uv_sin_theta * uv_targ_dz_GC;
        uv_Ry_mode2L = (uv_sin_phi * uv_sin_theta * uv_cos_psi –
                         uv_cos_phi * uv_sin_psi) * uv_targ_dx_mode2L
                     + (uv_sin_phi * uv_sin_theta * uv_sin_psi +
                        uv_cos_phi * uv_cos_psi) * uv_targ_dy_mode2L
                     +  uv_sin_phi * uv_cos_theta * uv_targ_dz_GC;
        uv_Rz_mode2L = (uv_cos_phi * uv_sin_theta * uv_cos_psi +
                         uv_sin_phi * uv_sin_psi) * uv_targ_dx_mode2L
                      + (uv_cos_phi * uv_sin_theta * uv_sin_psi –
                          uv_sin_phi * uv_cos_psi) * uv_targ_dy_mode2L
                      +  uv_cos_phi * uv_cos_theta * uv_targ_dz_GC;

        if ((uv_Rx_mode2L == 0.0) && (uv_Ry_mode2L == 0.0))
          uv_cam_epsilon_mode2L = 0.0;
        else
          uv_cam_epsilon_mode2L = atan2f(-uv_Rx_mode2L,uv_Ry_mode2L);

        if ((uv_Rz_mode2L== 0.0) && ((uv_Rx_mode2L *
            sinf(uv_cam_epsilon_mode2L) - uv_Ry_mode2L *
            cosf(uv_cam_epsilon_mode2L)) == 0.0))
          uv_cam_lambda_mode2L = 0.0;
        else
          uv_cam_lambda_mode2L =  atan2f(-uv_Rz_mode2L,
                                        (uv_Rx_mode2L *
                                    sinf(uv_cam_epsilon_mode2L)-
                                    uv_Ry_mode2L *
                                   cosf(uv_cam_epsilon_mode2L)));

/* Insures camera angles are in +- 180 degrees, ie. no 270 degrees */
        if (uv_cam_epsilon_mode1 < -PI)  uv_cam_epsilon_mode1  += 2*PI;
        if (uv_cam_epsilon_mode1 > PI)   uv_cam_epsilon_mode1  -= 2*PI;
        if (uv_cam_epsilon_mode2R < -PI) uv_cam_epsilon_mode2R += 2*PI;
        if (uv_cam_epsilon_mode2R > PI)  uv_cam_epsilon_mode2R -= 2*PI;
        if (uv_cam_epsilon_mode2L < -PI) uv_cam_epsilon_mode2L += 2*PI;
        if (uv_cam_epsilon_mode2L > PI)  uv_cam_epsilon_mode2L -= 2*PI;
        if (uv_cam_lambda_mode1 < -PI)  uv_cam_lambda_mode1   += 2*PI;
        if (uv_cam_lambda_mode1 > PI)   uv_cam_lambda_mode1   -= 2*PI;
        if (uv_cam_lambda_mode2R < -PI) uv_cam_lambda_mode2R  += 2*PI;
        if (uv_cam_lambda_mode2R > PI)  uv_cam_lambda_mode2R  -= 2*PI;
        if (uv_cam_lambda_mode2L < -PI) uv_cam_lambda_mode2L  += 2*PI;
        if (uv_cam_lambda_mode2L > PI)  uv_cam_lambda_mode2L  -= 2*PI;

/* Determines the delta camera angles */
        uv_cam_Depsilon_mode1 = uv_cam_epsilon_mode1- uv_cam_epsilon ;
        uv_cam_Dlambda_mode1 =  uv_cam_lambda  - uv_cam_lambda_mode1;
        uv_cam_Depsilon_mode2R = uv_cam_epsilon_mode2R - uv_cam_epsilon;
        uv_cam_Dlambda_mode2R =  uv_cam_lambda - uv_cam_lambda_mode2R;
        uv_cam_Depsilon_mode2L = uv_cam_epsilon_mode2L - uv_cam_epsilon;
        uv_cam_Dlambda_mode2L =  uv_cam_lambda - uv_cam_lambda_mode2L;

/* Insures camera angles are in +- 180 degrees, ie. no 270 degrees */
        if (uv_cam_Depsilon_mode1 < -PI)  uv_cam_Depsilon_mode1  += 2*PI;
        if (uv_cam_Depsilon_mode1 > PI)   uv_cam_Depsilon_mode1  -= 2*PI;
        if (uv_cam_Depsilon_mode2R < -PI) uv_cam_Depsilon_mode2R += 2*PI;
        if (uv_cam_Depsilon_mode2R > PI)  uv_cam_Depsilon_mode2R -= 2*PI;
        if (uv_cam_Depsilon_mode2L < -PI) uv_cam_Depsilon_mode2L += 2*PI;
        if (uv_cam_Depsilon_mode2L > PI)  uv_cam_Depsilon_mode2L -= 2*PI;
        if (uv_cam_Dlambda_mode1 < -PI)   uv_cam_Dlambda_mode1   += 2*PI;
        if (uv_cam_Dlambda_mode1 > PI)    uv_cam_Dlambda_mode1   -= 2*PI;
        if (uv_cam_Dlambda_mode2R < -PI)  uv_cam_Dlambda_mode2R  += 2*PI;
        if (uv_cam_Dlambda_mode2R > PI)   uv_cam_Dlambda_mode2R  -= 2*PI;
        if (uv_cam_Dlambda_mode2L < -PI)  uv_cam_Dlambda_mode2L  += 2*PI;
        if (uv_cam_Dlambda_mode2L > PI)   uv_cam_Dlambda_mode2L  -= 2*PI;

        uv_turn_range = sqrt(3*pow(uv_turn_radius,2));
        range_check = uv_targ_range - uv_turn_radius;

/* Set turn rate based upon velocity and turn radius, also protects a divide by zero by
always setting turn rate to at least 1.0 */
        if (uv_Vel == 0.0 || uv_turn_radius == 0.0)
          uv_turn_rate = 1.0;
        else
          uv_turn_rate = uv_Vel / uv_turn_radius;

/* If in mode 1 and target enters turn range, initializes turn, mode 21, else remains in
mode 1 or reenter mode 1 when target exits 2*turn radius */
        if (uv_targ_range <= uv_turn_range && uv_mode == 1)
          uv_mode = 21;
        else if (uv_targ_range >= (2 * uv_turn_radius))
          uv_mode = 1;
```

```
if (cpt_PosTrans(CPT_HAP_TRIG)  && uv_targ_lock == 1)
{
 uv_bomb++;
 if (uv_bomb > 1) uv_bomb = 0;
}

if (uv_targ_lock != 1)
 uv_bomb = 0;

if (uv_targ_dz_GC_stick >= 0.0)
 uv_drop_time = sqrtf(2.0*uv_targ_dz_GC_stick/9.8);
else
 uv_drop_time = 0.0;

uv_bomb_range = uv_turn_radius + uv_Vel * uv_drop_time;

if ((uv_drop_time > 0.0) && (uv_targ_range > uv_bomb_range) &&
          uv_bomb)
    uv_bomb_run = 1;

   if (!uv_bomb)
    uv_bomb_run = 0;

   shm->bomb = uv_bomb_run;

   if (uv_bomb_run)
    uv_mode = 1;

if (uv_targ_range <= uv_Vel * uv_drop_time && uv_bomb_run)
{
 uv_bomb_away = 1;
 uv_bomb_hit = 0;
 uv_bomb_x_init = uv_x_GC;
 uv_bomb_y_init = uv_y_GC;
 uv_bomb_z_init = uv_z_GC;
 uv_bomb_h = uv_h;
 uv_bomb_v = uv_Vel;
 uv_bomb_drop = 1;
 uv_bomb_time = 0.0;
 uv_bomb_dt = 1.0/FRAMES_PER_SECOND;
 uv_drop_time_rel = sqrtf(2.0*uv_targ_dz_GC_stick/9.8);
}
```

```
if (uv_bomb_drop)
{
 uv_bomb_x = uv_bomb_x_init - sinf(uv_bomb_h * FLSIM_DTR) *
     uv_bomb_v * uv_bomb_time;
 uv_bomb_y = uv_bomb_y_init + cosf(uv_bomb_h * FLSIM_DTR) *
     uv_bomb_v * uv_bomb_time;
 uv_bomb_z = uv_bomb_z_init - 0.5*9.8*pow(uv_bomb_time,2);
 uv_bomb_time += uv_bomb_dt;
 if (uv_bomb_time >= uv_drop_time_rel)
 {
  uv_bomb_hit = 1;
  uv_bomb_drop = 0;
  uv_bomb = 0;
 }
}

shm->bomb_drop = uv_bomb_drop;
shm->bomb_hit = uv_bomb_hit;
shm->bomb_x = uv_bomb_x;
shm->bomb_y = uv_bomb_y;
shm->bomb_z = uv_bomb_z;

if (uv_bomb_run && uv_bomb_away)
{
  uv_bomb_run = 0;
  uv_bomb_away = 0;
}
```

```
/* Determines turn direction and sets parameter for initialization termination, direction is
determined based upon initial target location */
    if (uv_cam_epsilon_level >= -PI/2 && uv_cam_epsilon_level <= PI/2
     && uv_mode == 1)
    {
     uv_turn_dir = +1;
     uv_cam_epsilon_level_init = uv_cam_epsilon_level;
    }
    else if ((uv_cam_epsilon_level < -PI/2 ||
       uv_cam_epsilon_level > PI/2) && uv_mode == 1)
    {
     uv_turn_dir = -1;
     uv_cam_epsilon_level_init = uv_cam_epsilon_level;
    }

/* Set direction for turn entry */
    if (uv_cam_epsilon_level < 0.0 && uv_mode == 1)
```

```
      uv_turn_dir_init = -1;
     else if (uv_cam_epsilon_level > 0.0 && uv_mode == 1)
      uv_turn_dir_init = +1;

/* Set turn rate and mode 2 values based upon turn direction */
     if (uv_turn_dir == 1)
     {
      uv_turn_rate = uv_turn_rate;
      uv_cam_Depsilon_mode2 = uv_cam_Depsilon_mode2R;
      uv_cam_Dlambda_mode2 = uv_cam_Dlambda_mode2R;
     }
     else if (uv_turn_dir == -1)
     {
      uv_turn_rate = -uv_turn_rate;
      uv_cam_Depsilon_mode2 = uv_cam_Depsilon_mode2L;
      uv_cam_Dlambda_mode2 = uv_cam_Dlambda_mode2L;
     }

/* Dampens mode 1 oscillation by small fluctations in deltas */
    if ((uv_cam_Dlambda_mode1*FLSIM_RTD < .01
     && uv_cam_Dlambda_mode1*FLSIM_RTD > -.01)
       && (uv_cam_Depsilon_mode1*FLSIM_RTD < .03
     && uv_cam_Depsilon_mode1*FLSIM_RTD > -.03))
      uv_lock_on = 1;

     if ((uv_cam_Dlambda_mode1*FLSIM_RTD > 1.0
    || uv_cam_Dlambda_mode1*FLSIM_RTD < -1.0
      || uv_cam_Depsilon_mode1*FLSIM_RTD > 3
    || uv_cam_Depsilon_mode1*FLSIM_RTD < -3)
    && uv_lock_on)
       uv_lock_on = 0;

     if (uv_cam_Dlambda_mode1*FLSIM_RTD < 1.0
    && uv_cam_Dlambda_mode1*FLSIM_RTD > -1.0 && uv_lock_on)
      uv_cam_Dlambda_mode1 = 0.0;

     if (uv_cam_Depsilon_mode1*FLSIM_RTD < 3
   && uv_cam_Depsilon_mode1*FLSIM_RTD > -3 && uv_lock_on)
      uv_cam_Depsilon_mode1 = 0.0;

/* Dampens mode 2 oscillations by small fluctations in deltas always gives lambda sole
control when target is at a greater distance than 100 meters.  This prevents epsilon from
slowing down range correction */
     if (uv_cam_Dlambda_mode2*FLSIM_RTD < 1.0
   && uv_cam_Dlambda_mode2*FLSIM_RTD > -1.0)
```

```
            uv_cam_Dlambda_mode2 = 0.0;
           if ((uv_cam_Depsilon_mode2*FLSIM_RTD < 1.0
        && uv_cam_Depsilon_mode2*FLSIM_RTD > -1.0)
            || ( range_check > 100.0 || range_check < -100.0))
            uv_cam_Depsilon_mode2 = 0.0;

/* Determines termination of turn entry and entry into roll over, mode 22 */
           if ((((uv_turn_dir == 1) && (uv_cam_epsilon_level_init < 0.0)
         && (uv_cam_epsilon_level > 0.0)) ||
              ((uv_turn_dir == 1) && (uv_cam_epsilon_level_init > 0.0)
         && (uv_cam_epsilon_level < 0.0)) ||
              ((uv_turn_dir == -1) && (uv_cam_epsilon_level_init < 0.0)
         && (uv_cam_epsilon_level > 0.0)) ||
              ((uv_turn_dir == -1) && (uv_cam_epsilon_level_init > 0.0)
         && (uv_cam_epsilon_level < 0.0))) && (uv_mode == 21))
                  uv_mode = 22;

/* Determines change in psi, for different flight modes */
           if (uv_mode == 1)
           {
            uv_dpsi = (((uv_sin_phi / uv_cos_theta) * (-
                                  uv_cam_Dlambda_mode1) *
                                  sqrt(pow(cosf(uv_cam_epsilon),2)))
                                -((uv_cos_phi / uv_cos_theta) *
                 uv_cam_Depsilon_mode1)) * FLSIM_RTD;
           }

           if (uv_mode == 21)
            uv_dpsi = uv_turn_dir_init * uv_turn_rate * FLSIM_RTD;

           if (uv_mode == 22)
            uv_dpsi = uv_turn_rate  * FLSIM_RTD;

           if (uv_mode == 23)
           {
        uv_dpsi = (((uv_sin_phi / uv_cos_theta) * (uv_sin_phi * uv_cos_theta *
                     uv_turn_rate - uv_cam_Dlambda_mode2
                   * sqrt(pow(cosf(uv_cam_epsilon),2))))
                   + ((uv_cos_phi / uv_cos_theta) * (uv_cos_phi *
                        uv_cos_theta * uv_turn_rate – uv_cam_Depsilon_mode2))) *
                        FLSIM_RTD;
           }

/* Insures change angle is +- 180 */
           if (uv_dpsi > 180.0) uv_dpsi -= 360.0;
        if (uv_dpsi < -180.0) uv_dpsi += 360.0;

/* Helps dampen oscillations in turn, by keeping delta psi constant within small camera
angle fluctations */
        if (uv_mode == 23 && uv_cam_Dlambda_mode2 == 0.0 &&
           uv_cam_Depsilon_mode2 == 0.0)
         uv_dpsi = uv_dpsi_set;

/* Determines desired bank angle based upon max g load desired */
        uv_bank_desired = acosf(1/uv_n)*FLSIM_RTD;

/* Determines desired roll angle based upon delta psi and mode */
/* In mode 1 all turns greater than 1 degree set to desired bank,
  else dampens out roll desired as delta psi approaches zero */
        if (uv_mode == 1)
        {
         if (uv_dpsi > 3.0)
          uv_roll_desired = uv_bank_desired;
         else if (uv_dpsi < -3.0)
          uv_roll_desired = -uv_bank_desired;
         else
          uv_roll_desired = uv_dpsi * uv_bank_desired / 3.0;
        }

/* If in operator mode 3, constant angle level turn sets roll angle to delta psi */
        if (uv_targ_lock == 3)
         uv_roll_desired = uv_dpsi;

/* If mode 2 roll desired is set to a ratio of delta psi to the desired turn rate, allows
correction for range, by increasing bank in range outside turn radius and decreases bank
for range inside turn radius.  Also for purposes of insuring no great changes in roll, limits
roll to a maximum variation. */
       if (uv_turn_radius == 0.0)
        uv_bank_max_var = 0.0;
       else
        uv_bank_max_var =
                 sqrt(pow((range_check/uv_turn_radius),2))*5.0;

        if (uv_mode == 21 || uv_mode == 22 || uv_mode == 23)
        {
         uv_roll_desired =
        (uv_dpsi/(sqrt(pow(uv_turn_rate,2))*FLSIM_RTD))*uv_bank_desired;
         if (uv_roll_desired > uv_bank_desired + uv_bank_max_var)
          uv_roll_desired = uv_bank_desired + uv_bank_max_var;
         else if (uv_roll_desired < -(uv_bank_desired +
             uv_bank_max_var))
            uv_roll_desired = -(uv_bank_desired + uv_bank_max_var);
          }

          if (uv_mode == 23 && uv_turn_dir == 1 && uv_roll_desired <
            (uv_bank_desired - uv_bank_max_var))
           uv_roll_desired = (uv_bank_desired - uv_bank_max_var);
          if (uv_mode == 23 && uv_turn_dir == -1 && uv_roll_desired > -
            (uv_bank_desired - uv_bank_max_var))
           uv_roll_desired = -(uv_bank_desired - uv_bank_max_var);

/* Prohibits rolling opposite direction once in a turn */
          if (uv_mode == 23 && uv_turn_dir == 1 && uv_roll_desired < 0.0)
           uv_roll_desired = 0.0;
          if (uv_mode == 23 && uv_turn_dir == -1 && uv_roll_desired > 0.0)
           uv_roll_desired = 0.0;

/* Prohibits rolls of +- 90 degrees */
          if (uv_roll_desired >= 89.0)
           uv_roll_desired = 89.0;
          else if (uv_roll_desired <= -89.0)
           uv_roll_desired = -89.0;

/* Calculates the roll error */
          uv_Droll=uv_roll_desired-uv_phi*FLSIM_RTD;

/* Determines if roll over is complete, then enters final turn */
          if (uv_mode == 22 && uv_Droll < 0.02 && uv_Droll > -0.02)
           uv_mode = 23;

/* Uses roll error to control roll rate, set for 2g turn, as roll error approaches zero, roll rate
is damped out */
          if ((uv_Droll) > 60.0)
           uv_roll_rate = 1.0;
          else if ((uv_Droll) < -60.0)
           uv_roll_rate = -1.0;
          else
           uv_roll_rate = (1.0/60.0)*(uv_Droll);

/* Set initial pitch rate for turn equal to roll angle desired */
          uv_pitch_rate_turn_init = sqrt(pow(uv_roll_desired,2)) / 100.0;

/* Set altitude error */
          uv_Dalt = uv_z_GC - uv_Alt_hold;
```

```
/* Set the climb/descent rate based upon altitude error, allows greater changes to correct
faster */
        uv_VVI_des = -(uv_Dalt/15.0);

      if (uv_phi*FLSIM_RTD > .5 || uv_phi*FLSIM_RTD < -.5)
        uv_theta_des_reset = 1;

      if (uv_phi*FLSIM_RTD < .5 && uv_phi*FLSIM_RTD > -.5
          && uv_theta_des_reset)
      {
       uv_theta_des_reset = 0;
       uv_theta_des = uv_theta_zero;
      }

/* Algorithm for control altitude, determines theta desired, control variable in
determining pitch error */
/* Modifies theta desired, if climing or descending to fast or in wrong direction, has two
value of modification, one when too fast or wrong direction and a second when the theta
error is within +-.25. */
        if (((uv_VVI < uv_VVI_des) && (uv_VVI < uv_VVI_set)
      && uv_Dalt > 0.0) || ((uv_VVI > uv_VVI_des)
          && (uv_VVI > uv_VVI_set) && uv_Dalt < 0.0))
         uv_theta_des = uv_theta_des - .005 * (uv_VVI - uv_VVI_des);
        else if ((uv_Dalt < 0.0 && uv_VVI < uv_VVI_des
      && uv_VVI < uv_VVI_set && uv_Dtheta < 0.25
          && uv_Dtheta > -0.25)|| (uv_Dalt > 0.0 && uv_VVI > uv_VVI_des
          && uv_VVI > uv_VVI_set && uv_Dtheta < 0.25
          && uv_Dtheta > -0.25))
         uv_theta_des = uv_theta_des - .01 * (uv_VVI - uv_VVI_des);

/* Initializes the theta for zero lift */
        if ((uv_Dalt < 0.50) && (uv_Dalt > -0.5) && uv_b == 0)
        {
         uv_theta_zero = uv_theta_des;
         uv_theta_des = uv_theta_zero;
         uv_b = uv_b + 1;
        }

/* Set desired theta to theta zero if altitude error +- .5 */
        if ((uv_Dalt < 0.50) && (uv_Dalt > -0.5))
         uv_theta_des = uv_theta_zero;

/* Modifies theta zero */
        if ((uv_Dalt < 0.00) && (uv_Dalt > -0.50) && uv_b == 1
      && uv_Dtheta < .05 && uv_Dtheta > -.05 && uv_VVI < 0.0
```

```
          && uv_VVI < uv_VVI_set)
        {
          uv_theta_zero = uv_theta_zero + .0001;
          uv_theta_des = uv_theta_zero;
        }
        else if ((uv_Dalt < 0.50) && (uv_Dalt > 0.0) && uv_b == 1
             && uv_Dtheta < .05 && uv_Dtheta > -.05 && uv_VVI > 0.0
             && uv_VVI > uv_VVI_set)
        {
          uv_theta_zero = uv_theta_zero - .0001;
          uv_theta_des = uv_theta_zero;
        }

/* Determines theta error for pitch rate determination */
        uv_Dtheta = (uv_theta_des-(uv_theta*FLSIM_RTD));

/* Determines pitch rate for altitude hold, maximizes rate if error greater than 5.0
degrees, else dampens out rate as error approaches zero, has three ranges < 1, 1 to 5, and
> 5.  */
        if (uv_Dtheta > 5.0 && uv_theta < uv_theta_set)
          uv_pitch_rate_alt = 1.0;
        else if (uv_Dtheta < -5.0 && uv_theta > uv_theta_set)
          uv_pitch_rate_alt = -1.0;
        else if (uv_Dtheta < 1.0 && uv_Dtheta > -1.0)
          uv_pitch_rate_alt = (5.0/6.75)*uv_Dtheta;
        else
          uv_pitch_rate_alt = (1/10.0)*uv_Dtheta;

/* Determines if pitch rate is stabilized for modification of turn pitch rate */
        uv_pitch_Drate = uv_pitch_rate_old - uv_pitch_rate_new;

/* Modifies pitch rate for turn if pitch rate is stable and aircraft is still climbing or
descending */
        if (uv_pitch_rate_turn_init != 0.0 && uv_pitch_Drate <= 0.0009
          && uv_pitch_Drate >= -0.0009 && uv_Dalt < 0.0
          && uv_VVI < uv_VVI_set && uv_VVI < 0.0)
            uv_pitch_rate_turn_mod = uv_pitch_rate_turn_mod + 0.001;
        else if (uv_pitch_rate_turn_init != 0.0
          && uv_pitch_Drate <= 0.0009 && uv_pitch_Drate >= -0.0009
          && uv_Dalt > 0.0 && uv_VVI > uv_VVI_set && uv_VVI > 0.0)
            uv_pitch_rate_turn_mod = uv_pitch_rate_turn_mod - 0.001;

/* Sets pitch rate for turn equal to the initial value plus the modification */
        uv_pitch_rate_turn = uv_pitch_rate_turn_init +
                                      uv_pitch_rate_turn_mod;
        if (uv_pitch_rate_turn_init == 0.0)
          uv_pitch_rate_turn = 0.0;

/* Sum the two variables of pitch rate */
        uv_pitch_rate = uv_pitch_rate_turn + uv_pitch_rate_alt;

/* Limits pitch rate to +- 1.0 */
        if (uv_pitch_rate > 1.0)
          uv_pitch_rate = 1.0;
        if (uv_pitch_rate < -1.0)
          uv_pitch_rate = -1.0;

/* Sets values for direction determination */
        uv_VVI_set = uv_VVI;
        uv_theta_set = uv_theta;
        uv_pitch_rate_old = uv_pitch_rate_new;
        uv_pitch_rate_new = uv_pitch_rate_alt;
        uv_dpsi_set = uv_dpsi;
/* Sets stick at desired rates */
        fm_runtime_set_stick(uv_pitch_rate, uv_roll_rate);

}

/*======================WRITE DATA========================*/

void WriteLandingData(void)
{
        int P, n, i;
        char str[250];
        char OUTPUT[50];

        if (Practice)
    {
          if (Clouds)
          sprintf( OUTPUT,"DATA.SJ%i.BK%i.N%i.PC", Subject, Block, N+1 );
          else
          sprintf( OUTPUT,"DATA.SJ%i.BK%i.N%i.P", Subject, Block, N+1 );
         }
         else
          sprintf( OUTPUT,"DATA.SJ%i.BK%i.N%i", Subject, Block, N+1 );
         fn = open( OUTPUT, O_WRONLY | O_CREAT | O_TRUNC );

         sprintf( str, " SUB BLK   N LOC HAP TUR CLD TRI X_RMS    Y_RMS
                X_POS    Y_POS    Z_POS    X_VEL    Y_VEL    Z_VEL    X_ACC
                Y_ACC    Z_ACC");
        strcat( str,"   H_POS    P_POS    R_POS    H_VEL    P_VEL
                R_VEL    H_ACC    P_ACC    R_ACC CRA WOW OOB\n");
        write( fn, str, 225 );

        sprintf( str,
"%4i%4i%4i%4i%4i%4i%4i%4i%9.5f%9.5f%9.2f%9.2f%9.2f%9.2f%9.2f%9.2f%9.2f
 %9.2f%9.2f%9.2f%9.2f%9.2f%9.2f%9.2f%9.2f%9.2f%9.2f%9.2f",
        Subject, Block, N+1, Location, Haptics, Turbulance, Clouds,
         Trial, RMSx, RMSy, uv_x_GC, uv_y_GC, uv_z_GC, uv_x_vel,
         uv_y_vel, uv_z_vel, uv_x_acc, uv_y_acc, uv_z_acc, uv_h, uv_p,
        uv_r, uv_h_vel, uv_p_vel, uv_r_vel, uv_h_acc, uv_p_acc, uv_r_acc
    );
        write( fn, str, 225-13 );

}

void WriteCrashData(void)
{
        char str[250];

        CrashDataWritten = 1;

        sprintf( str,
         "%4i%4i%4i\n",
         Crashed, Wow, OutOfBounds);

        write( fn, str, 13 );

        close( fn );
}

void WriteContinuousData(void)
{
        int P, n, i;
        char str[200];
        char OUTPUT[50];

        if (FrameCounter < 4) return;

        if (!FileContinuous) {

                sprintf(
OUTPUT,"DATA.%SBJ1i.BLK%1i.N%1i.LOC%1i.HAP%1i.TUR%1i.CLD%1i.TRL%
1i",
            Subject, Block, N+1, Location, Haptics, Turbulance, Clouds, Trial );

            FileContinuous = open( OUTPUT, O_WRONLY | O_CREAT |
                                    O_TRUNC );

            sprintf( str," WOW  X_JOY  Y_JOY  X_WND  Y_WND SBK  X_POS
            Y_POS  Z_POS  X_VEL  Y_VEL  Z_VEL  X_ACC  Y_ACC  Z_ACC
            H_POS  P_POS  R_POS  H_VEL  P_VEL  R_VEL  H_ACC  P_ACC
            R_ACC\n");
            write( FileContinuous, str, 163 );

        }

        sprintf( str,"%4i%4i%7.3f%7.3f%7.3f%7.3f%4i%7.2f%7.2f%7.2f%7.2f
           %7.2f%7.2f%7.2f%7.2f%7.2f%7.2f%7.2f%7.2f%7.2f%7.2f%7.2f
           %7.2f%7.2f%7.2f\n",
        Wow, Crashed, JoyX, JoyY, WindX, WindY, SpeedBrake, uv_x_GC,
        uv_y_GC, uv_z_GC, uv_x_vel, uv_y_vel, uv_z_vel, uv_x_acc,
        uv_y_acc, uv_z_acc, uv_h, uv_p, uv_r, uv_h_vel, uv_p_vel,
        uv_r_vel, uv_h_acc, uv_p_acc, uv_r_acc);

        write( FileContinuous, str, 163 );

        if (Crashed || OutOfBounds || (Wow && (FrameCounter >=
            LANDING_TIMEOUT * FRAMES_PER_SECOND))) {
            close( FileContinuous );
            IdleFlsim = 1;
        }
}

void ReadContinuousData(void)
{
        int P, n, i;
        char str[200];
        char INPUT[50];

        if (!FileContinuous) {

sprintf( INPUT,"DATA.DEFAULT",
Subject, Block, N, Location, Haptics, Turbulance, Clouds, Trial );

            FileContinuous = open( INPUT, O_RDONLY );

/*      fscanf( FileContinuous, str); */
```

```
        }
/*
        fscanf( FileContinuous,

"%4i%4i%7.3f%7.3f%7.3f%7.3f%4i%7.2f%7.2f%7.2f%7.2f%7.2f%7.2f%7.2f%7.2f
 %7.2f%7.2f%7.2f%7.2f%7.2f%7.2f%7.2f%7.2f%7.2f%7.2f\n",
        Wow, Crashed, JoyX, JoyY, WindX, WindY, SpeedBrake, uv_x_GC,
        uv_y_GC, uv_z_GC, uv_x_vel, uv_y_vel, uv_z_vel, uv_x_acc,
        uv_y_acc, uv_z_acc, uv_h, uv_p, uv_r, uv_h_vel, uv_p_vel,
        uv_r_vel, uv_h_acc, uv_p_acc, uv_r_acc);
*/
         if (Crashed || (Wow && (FrameCounter >= LANDING_TIMEOUT *
FRAMES_PER_SECOND))) {
                close( FileContinuous );
                IdleFlsim = 1;
        }
}
```

Appendix B

Triangulation Method

Triangulation is a method in which a position can be determined from outside sources. In normal practice, triangulation is usually accomplished with three data points. Using the algorithm in CSRC for range information, triangulation can be accomplished using only two points. Figure 25 shows an overview of how two data points represented by two different aircrafts can triangulate onto a target. Using the range information algorithm, points a and b can be obtained from the aircraft's position, orientation and sensor angles. These two points, along with two additional points directly below the aircraft, are positioned at sea level and are obtained from the assumption that DZ is the aircraft's position above sea level.

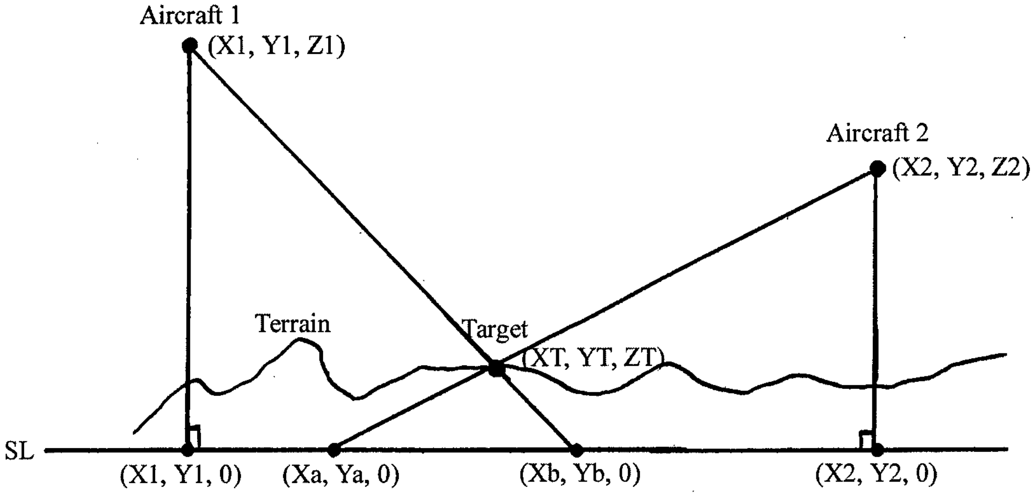


Figure 25. Triangulation Overview

From figure 25, there are six known data points. These six points located in a 3-dimensional plane are reduced to a 2-dimensional plane. By eliminating the y components, these data points are brought to an XZ plane, as shown in figure 26. Conversely, the x components could be eliminated to obtain a YZ plane.

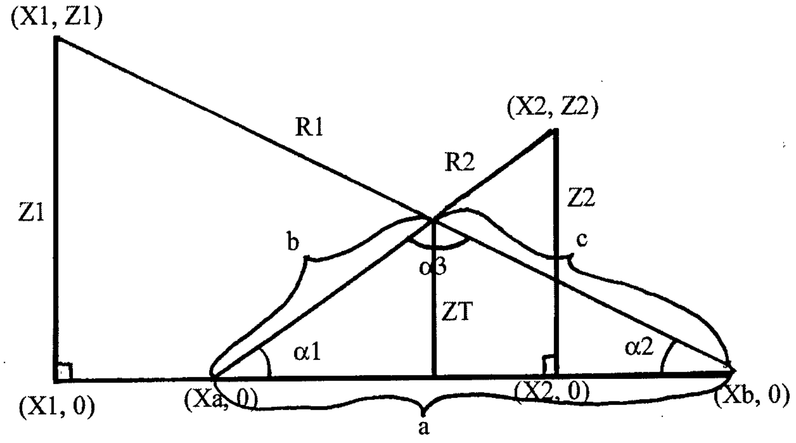


Figure 26. XZ Plane

In figure 26, the data points, x and z components, are used to determine the distances between each point. Using trigonometry, the three angles of a triangle formed with the target position can be calculated from these distances. With the calculated angles and distances, the height of the target above sea level can be determined. Reapplying the range information algorithm with the known DZ from either data point to the target, the target's XYZ position is determined. Two mathcad solutions are provided to demonstrate this method. Each solution has a different set of aircraft position and orientation along with a different target location. Both solutions were able to determine the target's height above sea level with an accuracy of 100%. This accuracy is subject to the two data points used. As the distance between the two data points is reduced a singularity is approached. The minimum distance required was not determined and is left for future evaluation.

Case One:

$$\theta 1 := 10.0\cdot\frac{\pi}{180}\quad \psi 1 := 0.0\cdot\frac{\pi}{180}\quad \phi 1 := 60.0\cdot\frac{\pi}{180}\quad \theta 2 := -0.0\cdot\frac{\pi}{180}\quad \psi 2 := -135.0\cdot\frac{\pi}{180}\quad \phi 2 := 0.0\cdot\frac{\pi}{180}$$

$$R1 := \begin{bmatrix} \cos(\theta 1)\cdot\cos(\psi 1) & \cos(\theta 1)\cdot\sin(\psi 1) & -\sin(\theta 1) \\ \sin(\phi 1)\cdot\sin(\theta 1)\cdot\cos(\psi 1) - \cos(\phi 1)\cdot\sin(\psi 1) & \sin(\phi 1)\cdot\sin(\theta 1)\cdot\sin(\psi 1) + \cos(\phi 1)\cdot\cos(\psi 1) & \sin(\phi 1)\cdot\cos(\theta 1) \\ \cos(\phi 1)\cdot\sin(\theta 1)\cdot\cos(\psi 1) + \sin(\phi 1)\cdot\sin(\psi 1) & \cos(\phi 1)\cdot\sin(\theta 1)\cdot\sin(\psi 1) - \sin(\phi 1)\cdot\cos(\psi 1) & \cos(\phi 1)\cdot\cos(\theta 1) \end{bmatrix}$$

$$R2 := \begin{bmatrix} \cos(\theta 2)\cdot\cos(\psi 2) & \cos(\theta 2)\cdot\sin(\psi 2) & -\sin(\theta 2) \\ \sin(\phi 2)\cdot\sin(\theta 2)\cdot\cos(\psi 2) - \cos(\phi 2)\cdot\sin(\psi 2) & \sin(\phi 2)\cdot\sin(\theta 2)\cdot\sin(\psi 2) + \cos(\phi 2)\cdot\cos(\psi 2) & \sin(\phi 2)\cdot\cos(\theta 2) \\ \cos(\phi 2)\cdot\sin(\theta 2)\cdot\cos(\psi 2) + \sin(\phi 2)\cdot\sin(\psi 2) & \cos(\phi 2)\cdot\sin(\theta 2)\cdot\sin(\psi 2) - \sin(\phi 2)\cdot\cos(\psi 2) & \cos(\phi 2)\cdot\cos(\theta 2) \end{bmatrix}$$

$$R1 = \begin{bmatrix} 0.985 & 0 & -0.174 \\ 0.15 & 0.5 & 0.853 \\ 0.087 & -0.866 & 0.492 \end{bmatrix}\quad R2 = \begin{bmatrix} -0.707 & -0.707 & 0 \\ 0.707 & -0.707 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$X1 := 200 \quad Y1 := 3000 \quad Z1 := 10000$

$X2 := -6000 \quad Y2 := -200 \quad Z2 := 500$

$XT := 100 \quad YT := 1000 \quad ZT := 50$

$DX1 := XT - X1 \quad DY1 := YT - Y1 \quad DZ1 := ZT - Z1$

$DX2 := XT - X2 \quad DY2 := YT - Y2 \quad DZ2 := ZT - Z2$

$DX1 = -100 \quad DY1 = -2\cdot 10^{3} \quad DZ1 = -9.95\cdot 10^{3}$

$DX2 = 6.1\cdot 10^{3} \quad DY2 = 1.2\cdot 10^{3} \quad DZ2 = -450$

Camera Angle Determination:

$$\begin{bmatrix} Rx1 \\ Ry1 \\ Rz1 \end{bmatrix} := R1\cdot\begin{bmatrix} DX1 \\ DY1 \\ DZ1 \end{bmatrix}\quad \begin{bmatrix} Rx1 \\ Ry1 \\ Rz1 \end{bmatrix} = \begin{bmatrix} 1.629\cdot 10^{3} \\ -9.501\cdot 10^{3} \\ -3.176\cdot 10^{3} \end{bmatrix}$$

$$\varepsilon 1 := \operatorname{atan}\left(\frac{-Rx1}{Ry1}\right)\quad \varepsilon deg := \varepsilon 1\cdot\frac{180}{\pi}\quad \varepsilon deg = 9.731$$

$$\lambda 1 := \operatorname{atan}\left[\frac{-Rz1}{(Rx1\cdot\sin(\varepsilon 1) - Ry1\cdot\cos(\varepsilon 1))}\right]\quad \lambda deg := \lambda 1\cdot\frac{180}{\pi}\quad \lambda deg = 18.236$$

$$\begin{bmatrix} Rx2 \\ Ry2 \\ Rz2 \end{bmatrix} := R2\cdot\begin{bmatrix} DX2 \\ DY2 \\ DZ2 \end{bmatrix}\quad \begin{bmatrix} Rx2 \\ Ry2 \\ Rz2 \end{bmatrix} = \begin{bmatrix} -5.162\cdot 10^{3} \\ 3.465\cdot 10^{3} \\ -450 \end{bmatrix}$$

$$\varepsilon 2 := \operatorname{atan}\left(\frac{-Rx2}{Ry2}\right)\quad \varepsilon deg := \varepsilon 2\cdot\frac{180}{\pi}\quad \varepsilon deg = 56.129$$

$$\lambda 2 := \operatorname{atan}\left[\frac{-Rz2}{(Rx2\sin(\varepsilon 2) - Ry2\cdot\cos(\varepsilon 2))}\right]\quad \lambda deg := \lambda 2\cdot\frac{180}{\pi}\quad \lambda deg = -4.14$$

Data Points a and b:

$DZ1 := 0 - Z1 \quad \varepsilon := \varepsilon 1 \quad \lambda := \lambda 1 \quad \theta := \theta 1 \quad \psi := \psi 1 \quad \phi := \phi 1$

$$L_{11} = \cos\varepsilon\cos\Theta\cos\Psi + \sin\varepsilon(\sin\Phi\sin\Theta\cos\Psi - \cos\Theta\sin\Psi)$$

$$L_{12} = \cos\varepsilon\cos\Theta\sin\Psi + \sin\varepsilon(\sin\Phi\sin\Theta\sin\Psi + \cos\Theta\cos\Psi)$$

$$L_{13} = -\cos\varepsilon\sin\Theta + \sin\varepsilon\sin\Phi\cos\Theta$$

$$L_{21} = -\cos\lambda\sin\varepsilon\cos\Theta\cos\Psi + \cos\lambda\cos\varepsilon(\sin\Phi\sin\Theta\cos\Psi - \cos\Phi\sin\Psi)$$

$$+\sin\lambda(\cos\Phi\sin\Theta\cos\Psi + \sin\Phi\sin\Psi)$$

$$L_{22} = -\cos\lambda\sin\varepsilon\cos\Theta\sin\Psi + \cos\lambda\cos\varepsilon(\sin\Phi\sin\Theta\sin\Psi + \cos\Phi\cos\Psi)$$

$$+\sin\lambda(\cos\Phi\sin\Theta\sin\Psi - \sin\Phi\cos\Psi)$$

$$L_{23} = \cos\lambda\sin\varepsilon c\sin\Theta + \cos\lambda\cos\varepsilon\sin\Phi\cos\Theta + \sin\lambda\cos\Phi\cos\Theta$$

$$L_{31} = \sin\lambda\sin\varepsilon\cos\Theta\cos\Psi - \sin\lambda\cos\varepsilon(\sin\Phi\sin\Theta\cos\Psi - \cos\Phi\sin\Psi)$$

$$+\cos\lambda(\cos\Phi\sin\Theta\cos\Psi + \sin\Phi\sin\Psi)$$

$$L_{32} = \sin\lambda\sin\varepsilon\cos\Theta\sin\Psi - \sin\lambda\cos\varepsilon(\sin\Phi\sin\Theta\sin\Psi + \cos\Phi\cos\Psi)$$

$$+\cos\lambda(\cos\Phi\sin\Theta\sin\Psi - \sin\Phi\cos\Psi)$$

$$L_{33} = -\sin\lambda\sin\varepsilon\sin\Theta - \sin\lambda\cos\varepsilon\sin\Phi\cos\Theta + \cos\lambda\cos\Phi\cos\Theta$$

$DX1 := \dfrac{L32 \cdot L13 - L33 \cdot L12}{L31 \cdot L12 - L11 \cdot L32} \cdot DZ1 \quad DY1 := \dfrac{L11 \cdot L33 - L31 \cdot L13}{L31 \cdot L12 - L11 \cdot L32} \cdot DZ1$

$DX1 = -100.503$ $\qquad$ $DY1 = -2.01 \cdot 10^{3}$

$Xb := X1 + DX1$ $\qquad$ $Yb := Y1 + DY1$

$Xb = 99.497$ $\qquad$ $Yb = 989.95$

$DZ2 := 0 - Z2 \quad \varepsilon := \varepsilon 2 \quad \lambda := \lambda 2 \quad \theta := \theta 2 \quad \psi := \psi 2 \quad \phi := \phi 2$

$$L_{11} = \cos\varepsilon\cos\Theta\cos\Psi + \sin\varepsilon(\sin\Phi\sin\Theta\cos\Psi - \cos\Theta\sin\Psi)$$

$$L_{12} = \cos\varepsilon\cos\Theta\sin\Psi + \sin\varepsilon(\sin\Phi\sin\Theta\sin\Psi + \cos\Theta\cos\Psi)$$

$$L_{13} = -\cos\varepsilon\sin\Theta + \sin\varepsilon\sin\Phi\cos\Theta$$

$$L_{21} = -\cos\lambda\sin\varepsilon\cos\Theta\cos\Psi + \cos\lambda\cos\varepsilon(\sin\Phi\sin\Theta\cos\Psi - \cos\Phi\sin\Psi)$$

$$+\sin\lambda(\cos\Phi\sin\Theta\cos\Psi + \sin\Phi\sin\Psi)$$

$$L_{22} = -\cos\lambda\sin\varepsilon\cos\Theta\sin\Psi + \cos\lambda\cos\varepsilon(\sin\Phi\sin\Theta\sin\Psi + \cos\Phi\cos\Psi)$$

$$+\sin\lambda(\cos\Phi\sin\Theta\sin\Psi - \sin\Phi\cos\Psi)$$

$$L_{23} = \cos\lambda\sin\varepsilon c\sin\Theta + \cos\lambda\cos\varepsilon\sin\Phi\cos\Theta + \sin\lambda\cos\Phi\cos\Theta$$

$$L_{31} = \sin\lambda\sin\varepsilon\cos\Theta\cos\Psi - \sin\lambda\cos\varepsilon(\sin\Phi\sin\Theta\cos\Psi - \cos\Phi\sin\Psi)$$

$$+\cos\lambda(\cos\Phi\sin\Theta\cos\Psi + \sin\Phi\sin\Psi)$$

$$L_{32} = \sin\lambda\sin\varepsilon\cos\Theta\sin\Psi - \sin\lambda\cos\varepsilon(\sin\Phi\sin\Theta\sin\Psi + \cos\Phi\cos\Psi)$$

$$+\cos\lambda(\cos\Phi\sin\Theta\sin\Psi - \sin\Phi\cos\Psi)$$

$$L_{33} = -\sin\lambda\sin\varepsilon\sin\Theta - \sin\lambda\cos\varepsilon\sin\Phi\cos\Theta + \cos\lambda\cos\Phi\cos\Theta$$

$DX2 := \frac{L32 \cdot L13 - L33 \cdot L12}{L31 \cdot L12 - L11 \cdot L32} \cdot DZ2 \qquad DY2 := \frac{L11 \cdot L33 - L31 \cdot L13}{L31 \cdot L12 - L11 \cdot L32} \cdot DZ2$

$DX2 = 6.778 \cdot 10^{3} \qquad DY2 = 1.333 \cdot 10^{3}$

$Xa := X2 + DX2 \qquad Ya := Y2 + DY2$

$Xa = 777.778 \qquad Ya = 1.133 \cdot 10^{3}$

Triangulation Method:

$D1 := X1 - Xb \qquad D1 = 100.503 \qquad a1 := \sqrt{D1^{2} + DZ1^{2}} \qquad a1 = 1 \cdot 10^{4}$

$D2 := X2 - Xa \qquad D2 = -6.778 \cdot 10^{3} \qquad a2 := \sqrt{D2^{2} + DZ2^{2}} \qquad a2 = 6.796 \cdot 10^{3}$

$A1 := \operatorname{asin}\left(\frac{|DZ2|}{a2}\right) \qquad A1deg := A1 \cdot \frac{180}{\pi} \qquad A1 = 0.074 \qquad A1deg = 4.219$

$A2 := \operatorname{asin}\left(\frac{|DZ1|}{a1}\right) \qquad A2deg := A2 \cdot \frac{180}{\pi} \qquad A2 = 1.561 \qquad A2deg = 89.424$

$a := Xa - Xb \qquad a = 678.28$

$A3 := \pi - A1 - A2 \qquad A3 = 1.507$

$A3deg := A3 \cdot \frac{180}{\pi}$

$A3deg = 86.357$

$c := \sin(A1) \cdot \frac{a}{\sin(A3)}$

$c = 50.003$

$z := c \cdot \sin(A2) \qquad \mathbf{z = 50}$

Case Two:

$\theta1 := -0.0\cdot\frac{\pi}{180}$ $\psi1 := 45.0\cdot\frac{\pi}{180}$ $\phi1 := 0.0\cdot\frac{\pi}{180}$ $\theta2 := -10.0\cdot\frac{\pi}{180}$ $\psi2 := -0.0\cdot\frac{\pi}{180}$ $\phi2 := 30.0\cdot\frac{\pi}{180}$

$$R1 := \begin{bmatrix} \cos(\theta1)\cdot\cos(\psi1) & \cos(\theta1)\cdot\sin(\psi1) & -\sin(\theta1) \\ \sin(\phi1)\cdot\sin(\theta1)\cdot\cos(\psi1) - \cos(\phi1)\cdot\sin(\psi1) & \sin(\phi1)\cdot\sin(\theta1)\cdot\sin(\psi1) + \cos(\phi1)\cdot\cos(\psi1) & \sin(\phi1)\cdot\cos(\theta1) \\ \cos(\phi1)\cdot\sin(\theta1)\cdot\cos(\psi1) + \sin(\phi1)\cdot\sin(\psi1) & \cos(\phi1)\cdot\sin(\theta1)\cdot\sin(\psi1) - \sin(\phi1)\cdot\cos(\psi1) & \cos(\phi1)\cdot\cos(\theta1) \end{bmatrix}$$

$$R2 := \begin{bmatrix} \cos(\theta2)\cdot\cos(\psi2) & \cos(\theta2)\cdot\sin(\psi2) & -\sin(\theta2) \\ \sin(\phi2)\cdot\sin(\theta2)\cdot\cos(\psi2) - \cos(\phi2)\cdot\sin(\psi2) & \sin(\phi2)\cdot\sin(\theta2)\cdot\sin(\psi2) + \cos(\phi2)\cdot\cos(\psi2) & \sin(\phi2)\cdot\cos(\theta2) \\ \cos(\phi2)\cdot\sin(\theta2)\cdot\cos(\psi2) + \sin(\phi2)\cdot\sin(\psi2) & \cos(\phi2)\cdot\sin(\theta2)\cdot\sin(\psi2) - \sin(\phi2)\cdot\cos(\psi2) & \cos(\phi2)\cdot\cos(\theta2) \end{bmatrix}$$

$$R1 = \begin{bmatrix} 0.707 & 0.707 & 0 \\ -0.707 & 0.707 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad R2 = \begin{bmatrix} 0.985 & 0 & 0.174 \\ -0.087 & 0.866 & 0.492 \\ -0.15 & -0.5 & 0.853 \end{bmatrix}$$

$X1 := 200$ $Y1 := 3000$ $Z1 := 10000$

$X2 := -6000$ $Y2 := 2000$ $Z2 := 50000$

$XT := 100$ $YT := 1000$ $\mathbf{ZT := 500}$

$DX1 := XT - X1$ $DY1 := YT - Y1$ $DZ1 := ZT - Z1$

$DX2 := XT - X2$ $DY2 := YT - Y2$ $DZ2 := ZT - Z2$

$DX1 = -100$ $DY1 = -2\cdot10^{3}$ $DZ1 = -9.5\cdot10^{3}$

$DX2 = 6.1\cdot10^{3}$ $DY2 = -1\cdot10^{3}$ $DZ2 = -4.95\cdot10^{4}$

Camera Angle Determination:

$$\begin{bmatrix} Rx1 \\ Ry1 \\ Rz1 \end{bmatrix} := R1\cdot\begin{bmatrix} DX1 \\ DY1 \\ DZ1 \end{bmatrix} \quad \begin{bmatrix} Rx1 \\ Ry1 \\ Rz1 \end{bmatrix} = \begin{bmatrix} -1.485\cdot10^{3} \\ -1.344\cdot10^{3} \\ -9.5\cdot10^{3} \end{bmatrix}$$

$\varepsilon1 := \operatorname{atan}\left(\frac{-Rx1}{Ry1}\right)$ $\varepsilon deg := \varepsilon1\cdot\frac{180}{\pi}$ $\varepsilon deg = -47.862$

$\lambda1 := \operatorname{atan}\left[\frac{-Rz1}{(Rx1\cdot\sin(\varepsilon1) - Ry1\cdot\cos(\varepsilon1))}\right]$ $\lambda deg := \lambda1\cdot\frac{180}{\pi}$ $\lambda deg = 78.097$

$$\begin{bmatrix} Rx2 \\ Ry2 \\ Rz2 \end{bmatrix} := R2\cdot\begin{bmatrix} DX2 \\ DY2 \\ DZ2 \end{bmatrix} \quad \begin{bmatrix} Rx2 \\ Ry2 \\ Rz2 \end{bmatrix} = \begin{bmatrix} -2.588\cdot10^{3} \\ -2.577\cdot10^{4} \\ -4.263\cdot10^{4} \end{bmatrix}$$

$\varepsilon2 := \operatorname{atan}\left(\frac{-Rx2}{Ry2}\right)$ $\varepsilon deg := \varepsilon2\cdot\frac{180}{\pi}$ $\varepsilon deg = -5.735$

$$\lambda 2 := \operatorname{atan}\left[\frac{-Rz2}{(Rx2\cdot\sin(\varepsilon 2) - Ry2\cdot\cos(\varepsilon 2))}\right] \quad \lambda deg := \lambda 2\cdot\frac{180}{\pi} \quad \lambda deg = 58.722$$

Data Points a and b:

$DZ1 := 0 - Z1 \quad \varepsilon := \varepsilon 1 \quad \lambda := \lambda 1 \quad \theta := \theta 1 \quad \psi := \psi 1 \quad \phi := \phi 1$

$$L_{11} = \cos\varepsilon\cos\Theta\cos\Psi + \sin\varepsilon(\sin\Phi\sin\Theta\cos\Psi - \cos\Theta\sin\Psi)$$

$$L_{12} = \cos\varepsilon\cos\Theta\sin\Psi + \sin\varepsilon(\sin\Phi\sin\Theta\sin\Psi + \cos\Theta\cos\Psi)$$

$$L_{13} = -\cos\varepsilon\sin\Theta + \sin\varepsilon\sin\Phi\cos\Theta$$

$$L_{21} = -\cos\lambda\sin\varepsilon\cos\Theta\cos\Psi + \cos\lambda\cos\varepsilon(\sin\Phi\sin\Theta\cos\Psi - \cos\Phi\sin\Psi)$$
$$+ \sin\lambda(\cos\Phi\sin\Theta\cos\Psi + \sin\Phi\sin\Psi)$$

$$L_{22} = -\cos\lambda\sin\varepsilon\cos\Theta\sin\Psi + \cos\lambda\cos\varepsilon(\sin\Phi\sin\Theta\sin\Psi + \cos\Phi\cos\Psi)$$
$$+ \sin\lambda(\cos\Phi\sin\Theta\sin\Psi - \sin\Phi\cos\Psi)$$

$$L_{23} = \cos\lambda\sin\varepsilon c\sin\Theta + \cos\lambda\cos\varepsilon\sin\Phi\cos\Theta + \sin\lambda\cos\Phi\cos\Theta$$

$$L_{31} = \sin\lambda\sin\varepsilon\cos\Theta\cos\Psi - \sin\lambda\cos\varepsilon(\sin\Phi\sin\Theta\cos\Psi - \cos\Phi\sin\Psi)$$
$$+ \cos\lambda(\cos\Phi\sin\Theta\cos\Psi + \sin\Phi\sin\Psi)$$

$$L_{32} = \sin\lambda\sin\varepsilon\cos\Theta\sin\Psi - \sin\lambda\cos\varepsilon(\sin\Phi\sin\Theta\sin\Psi + \cos\Phi\cos\Psi)$$
$$+ \cos\lambda(\cos\Phi\sin\Theta\sin\Psi - \sin\Phi\cos\Psi)$$

$$L_{33} = -\sin\lambda\sin\varepsilon\sin\Theta - \sin\lambda\cos\varepsilon\sin\Phi\cos\Theta + \cos\lambda\cos\Phi\cos\Theta$$

$$DX1 := \frac{L32\cdot L13 - L33\cdot L12}{L31\cdot L12 - L11\cdot L32}\cdot DZ1 \quad DY1 := \frac{L11\cdot L33 - L31\cdot L13}{L31\cdot L12 - L11\cdot L32}\cdot DZ1$$

$DX1 = -105.263 \quad DY1 = -2.105\cdot 10^{3}$

$Xb := X1 + DX1 \quad Yb := Y1 + DY1$

$Xb = 94.737 \quad Yb = 894.737$

$DZ2 := 0 - Z2 \quad \varepsilon := \varepsilon 2 \quad \lambda := \lambda 2 \quad \theta := \theta 2 \quad \psi := \psi 2 \quad \phi := \phi 2$

$$L_{11} = \cos\varepsilon\cos\Theta\cos\Psi + \sin\varepsilon(\sin\Phi\sin\Theta\cos\Psi - \cos\Theta\sin\Psi)$$

$$L_{12} = \cos\varepsilon\cos\Theta\sin\Psi + \sin\varepsilon(\sin\Phi\sin\Theta\sin\Psi + \cos\Theta\cos\Psi)$$

$$L_{13} = -\cos\varepsilon\sin\Theta + \sin\varepsilon\sin\Phi\cos\Theta$$

$$L_{21} = -\cos\lambda\sin\varepsilon\cos\Theta\cos\Psi + \cos\lambda\cos\varepsilon(\sin\Phi\sin\Theta\cos\Psi - \cos\Phi\sin\Psi)$$
$$+ \sin\lambda(\cos\Phi\sin\Theta\cos\Psi + \sin\Phi\sin\Psi)$$

$$L_{22} = -\cos\lambda\sin\varepsilon\cos\Theta\sin\Psi + \cos\lambda\cos\varepsilon(\sin\Phi\sin\Theta\sin\Psi + \cos\Phi\cos\Psi)$$
$$+ \sin\lambda(\cos\Phi\sin\Theta\sin\Psi - \sin\Phi\cos\Psi)$$

$$L_{23} = \cos\lambda\sin\varepsilon c\sin\Theta + \cos\lambda\cos\varepsilon\sin\Phi\cos\Theta + \sin\lambda\cos\Phi\cos\Theta$$

$$L_{31} = \sin\lambda\sin\varepsilon\cos\Theta\cos\Psi - \sin\lambda\cos\varepsilon(\sin\Phi\sin\Theta\cos\Psi - \cos\Phi\sin\Psi)$$
$$+ \cos\lambda(\cos\Phi\sin\Theta\cos\Psi + \sin\Phi\sin\Psi)$$

$$L_{32} = \sin\lambda\sin\varepsilon\cos\Theta\sin\Psi - \sin\lambda\cos\varepsilon(\sin\Phi\sin\Theta\sin\Psi + \cos\Phi\cos\Psi)$$
$$+ \cos\lambda(\cos\Phi\sin\Theta\sin\Psi - \sin\Phi\cos\Psi)$$

$$L_{33} = -\sin\lambda\sin\varepsilon\sin\Theta - \sin\lambda\cos\varepsilon\sin\Phi\cos\Theta + \cos\lambda\cos\Phi\cos\Theta$$

$$DX2 := \frac{L32 \cdot L13 - L33 \cdot L12}{L31 \cdot L12 - L11 \cdot L32} \cdot DZ2 \qquad DY2 := \frac{L11 \cdot L33 - L31 \cdot L13}{L31 \cdot L12 - L11 \cdot L32} \cdot DZ2$$

$$DX2 = 6.162 \cdot 10^{3} \qquad DY2 = -1.01 \cdot 10^{3}$$

$$Xa := X2 + DX2 \qquad Ya := Y2 + DY2$$

$$Xa = 161.616 \qquad Ya = 989.899$$

Triangulation Method:

$$D1 := X1 - Xb \qquad D1 = 105.263 \qquad a1 := \sqrt{D1^{2} + DZ1^{2}} \qquad a1 = 1 \cdot 10^{4}$$

$$D2 := X2 - Xa \qquad D2 = -6.162 \cdot 10^{3} \qquad a2 := \sqrt{D2^{2} + DZ2^{2}} \qquad a2 = 5.038 \cdot 10^{4}$$

$$A1 := \operatorname{asin}\left(\frac{|DZ2|}{a2}\right) \qquad A1 = 1.448 \qquad A1deg := A1 \cdot \frac{180}{\pi} \qquad A1deg = 82.975$$

$$A2 := \operatorname{asin}\left(\frac{|DZ1|}{a1}\right) \qquad A2 = 1.56 \qquad A2deg := A2 \cdot \frac{180}{\pi} \qquad A2deg = 89.397$$

$$a := Xa - Xb \qquad a = 66.879$$

$$A3 := \pi - A1 - A2 \qquad A3 = 0.133$$

$$A3deg := A3 \cdot \frac{180}{\pi} \qquad A3deg = 7.628$$

$$c := \sin(A1) \cdot \frac{a}{\sin(A3)} \qquad c = 500.028$$

$$z := c \cdot \sin(A2) \qquad \mathbf{z = 500}$$

Bibliography

[1] Anderson, John D. Jr. Introduction to Flight (Third Edition). New York: McGraw-Hill Book Company, 1989.

[2] ASI. `http://www.ga.com/asi/aero.html`.

[3] Craig, John J. Introduction to Robotics Mechanics and Control (Second Edition). Reading, Massachusetts: Addison-Wesley Publishin Company, 1989.

[4] Cypher. `http://www.sikorsky.com/programs/cypher/index,html`. 12 March 1999.

[5] ERAST "Environmental Research Aircraft and Sensor Technology". `http://www.dfrc.nasa.gov/projects/erast/erasth.html#partners`. 12 March 1999.

[6] Etkin, Bernard. Dynamics of Atmospheric Flight. New York: John Wiley & Sons, Inc., 1972.

[7] Etkin, Bernard. Dynamics of Flight – Stability and Control (Secopnd Edition). New York: John Wiley & Sons, Inc., 1982.

[8] Nakamura, Yoshihiko. Advance Robotics Redundancy and Optimization. Reading, Massachusetts: Addison-Wesley Publishin Company,1991.

[9] Nelson, Robert C. Flight Stability and Automatic Control (Second Edition). Boston: WCB/McGraw-Hill, 1998.

[10] Rechtin, Eberhardt and Maier, Mark W. The Art of Systems Architecting. Boca Raton, Florida: CRC Press, 1997.

[11] Sheridan, Thomas B. Telerobotics, Automation, and Human Supervisory Control. Cambride, Massachusetts: The MIT Press, 1992.

[12] Siouris, George M. Aerospace Avionics Systems, A Modern Synthesis. San Diego: Academic Press, Inc., 1993.

[13] US Atlantic Command (USACOM). HAE UAV Joint Employment, Concept of Operations. 15 July 1998.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE
June 1999

3. REPORT TYPE AND DATES COVERED
Master's Thesis

4. TITLE AND SUBTITLE
Closely Supervised Reactive Control of an Uninhabited Aerial Vehicle

5. FUNDING NUMBERS

6. AUTHOR(S)
Roy G. Glassco, Capt, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
AFIT/ENY BLDG 640
2950 P STREET
WRIGHT-PATTERSON AFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GAE/ENY/99J-01

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Dr. John M. Reising
AFRL/HEC
2255 H Street
Wright-Patterson AFB OH 45433-7022
DSN 785-8769 (937) 255-8769

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
Thesis Advisor: Dr. Curtis Spenny
(937)255-6565, ext 4320
e-mail: Curtis.Spenny@afit.af.mil

12a. DISTRIBUTION AVAILABILITY STATEMENT
Approved for public release; unlimited distribution

12b. DISTRIBUTION CODE
A

13. ABSTRACT *(Maximum 200 words)*

Closely Supervised Reactive Control is an alternative control method for uninhabited aerial vehicles that incorporates some of the benefits of both manual and autonomous operations. Utilizing an on-board camera, an operator can control a UAV by manually choosing desired targets. The flight path of the uninhabited vehicle is determined autonomously from the camera gimbal angles. The operator of the payload (i.e. camera), has close supervision of the aircraft. The aircraft, using an on-board computer, is given autonomous control to alter flight path to fly towards a target and at a specified range, loiter over the target. In the basic mode of operation, the camera operator must manually track the target providing continuous updates to the camera angles. In an advanced mode of operation with the use of an INS or GPS, the aircraft autonomously determines the camera angles from a single locked position that the operator specifies. In this mode, the operator is free to look for other targets. This control method is being validated in the CAVE Automated Virtual Environment (CAVE) facility located at Wright Patterson AFB and owned and operated by Wright State University. The aircraft flight simulator used is FLSIM with the F-16 flight characteristics.

14. SUBJECT TERMS
URVs, Reactive Control, CAVE, Triangulation, Waypoint Navigation, Autonomous Flight

15. NUMBER OF PAGES
138

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Standard Form 298 (Rev. 2-89) (EG)
Prescribed by ANSI Std. 239.18
Designed using Perform Pro, WHS/DIOR, Oct 94