3-13-2009

UAS Collision Avoidance Algorithm that Minimizes the Impact on Route Surveillance

Austin L. Smith

UAS COLLISION AVOIDANCE ALGORITHM THAT MINIMIZES THE
IMPACT ON ROUTE SURVEILLANCE

THESIS

Austin L. Smith

AFIT/GAE/ENY/09-M18

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

AFIT/GAE/ENY/09-M18

UAS COLLISION AVOIDANCE ALGORITHM THAT MINIMIZES THE IMPACT ON ROUTE SURVEILLANCE

THESIS

Presented to the Faculty

Department of Aeronautics and Astronautics

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Aeronautical Engineering

Austin L. Smith, BS

March 2009

AFIT/GAE/ENY/09-M18

UAS COLLISION AVOIDANCE ALGORITHM THAT MINIMIZES THE IMPACT ON ROUTE SURVEILLANCE

Austin L. Smith, BS

Approved:

Frederick G. Harmon	13 Mar 09
Frederick G. Harmon, Lt Col, USAF (Chairman)	Date
Richard G. Cobb	13 MAR 09
Dr. Richard G. Cobb, (Member)	Date
David R. Jacques	13 MAR 09
Dr. David R. Jacques, (Member)	Date
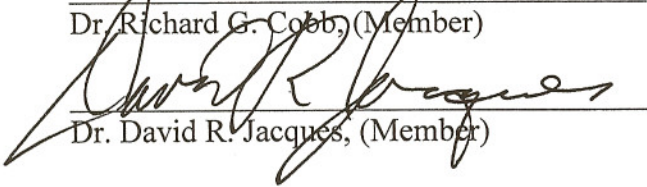
AFIT/GAE/ENY/09-M18

Abstract

A collision avoidance algorithm is developed and implemented that is applicable to different types of unmanned aerial systems ranging from a single platform with the ability to perform all collision avoidance functions independently to multiple vehicles performing functions as a cooperative group with collision avoidance commands computed at a ground station. The algorithm draws on the unique benefits of several theoretical approaches to conflict detection and resolution and combines them into one algorithm while addressing the limitations of those individual methods. Techniques and concepts from the three theoretical fields of robotics, homing guidance, and airspace management are used to complete the algorithm. The algorithm is developed with a focus on current Air Force systems used in route surveillance missions in hostile environments. The collision avoidance system is exercised and tested using hardware and platforms from the Advanced Navigation Technology Center at the Air Force Institute of Technology.

The results presented are the first known flight tests of a global, three-dimensional, geometric collision avoidance system on an unmanned aircraft system. Novel developments using an aggregated collision cone approach allows each unmanned aircraft to detect and avoid collisions with one or more other aircraft simultaneously. The collision avoidance system is implemented using a miniature unmanned aircraft with an onboard autopilot. Various simulation and flight test cases are used to demonstrate the algorithm's robustness to different collision encounters at various engagement angles. The flight test results are compared with ideal, software-in-the-loop, and hardware-in-the-loop tests.

Acknowledgments

I would like to express my sincere appreciation to my faculty advisor, Lt Col Fred Harmon, for his guidance and support throughout the course of this thesis effort. The insight, direction, and advice were certainly appreciated. I would also like to thank the Route Surveillance team, as well as Mr. Don Smith and Mr. John McNees, for their assistance and flight test support.

I would also like to thank my parents for instilling in me the character, dedication, and proper work ethic required to complete tasks such as these. Lastly, and foremost, I would like to thank my wife for providing the support and motivation I needed over the past few years; you are my source of inspiration.

Austin L. Smith

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AFIT	Air Force Institute of Technology
AFRL	Air Force Research Laboratory
AGL	Above Ground Level
ANT	Advanced Navigation Technology Center
ATMS	Air Traffic Management System
ATR	Autonomous Target Recognition
BAO	Battlefield Air Operations
CA	Collision Avoidance
CD&R	Conflict Detection and Resolution
CPA	Closest Point of Approach
DSA	Detect, Sense and Avoid
EO/IR	Electro-optical / Infrared
FAA	Federal Aviation Administration
GCS	Ground Control Station
GPS	Global Positioning System
HIL	Hardware-in-the-Loop
HUD	Heads-up Display
IED	Improvised Explosive Device
ISR	Intelligence, Surveillance and Reconnaissance
MAV	Micro Aerial Vehicle
MCR	MATLAB Compiler Runtime
NAS	National Airspace System
PID	Proportional-Integral-Derivative
R/C	Radio Control
SA	Separation Assurance
SAA	Sense and Avoid
SIL	Software-in-the-Loop
SOCOM	Special Operations Command
TCAS	Traffic Alert and Collision Avoidance System
UAS	Unmanned Aircraft System

UAS COLLISION AVOIDANCE ALGORITHM THAT MINIMIZES THE IMPACT ON ROUTE SURVEILLANCE

I. Introduction

1. Background

Current ground missions in Operation Iraqi Freedom and in the war on terror involve convoy transportation and the security of those convoys. Dangers to convoys include Improvised Explosive Devices (IED) that are placed on or near the road being used by Unites States military and coalition vehicles. One solution proposed to increase security around these mobile units is an unmanned aircraft system (UAS) to monitor the route before and during the convoy movement. The UAS could be used to detect insurgents placing the IEDs or to detect the IEDs themselves and alert convoy security before soldiers, civilians, or property are harmed.

Many options exist for a route surveillance system concept. A single aircraft with on-board sensing and processing could be used, but the effectiveness would be limited due to the required revisit rates. A multi-aircraft system could be used to monitor an entire stretch of road simultaneously but may require off-board processing and a more complicated communication and relay system. Both system types will be exposed to an environment where collision potential exists with non-cooperative air traffic or cooperative traffic within the UAS. In order to ensure completion of its mission and the safe return of the aircraft, separation must be maintained between vehicles in the UAS and between the UAS and non-cooperative traffic whether by procedures, human interference, or a last line of defense, a collision avoidance system.

2. Motivation

The intuitive need for collision avoidance systems in unmanned aircraft is apparent in the abundance of current algorithms, hardware, and complete-system developments for UAS of all sizes and complexity. In particular, military applications of UAS for defense and intelligence missions, and requirements for those missions, are laid out in the Unmanned System Roadmap 2007-2032 [1] and collision avoidance is specifically addressed in this roadmap. In fact, Chapter 6, Technologies for Unmanned Systems, Section 6.1, Technology Challenges, of the Roadmap states "the single most important near-term technical challenge facing unmanned systems is to develop an autonomous capability to assess and respond appropriately to near-field objects in their path of travel." This technical challenge is addressed by providing "direction for future investments" for collision avoidance systems.

6.6.8. Dynamic Obstacle/Interference/Collision Avoidance (Including Humans)

All unmanned systems except the smallest special purpose vehicles must have the ability to autonomously avoid obstacles. In addition to the simple avoidance of obstacles (which is not simple if both the "obstacle" and the vehicle are moving independently), we must consider perception elements impacting trafficability, tactical maneuver, and mission execution. While most control algorithms are sufficiently mature, sensor processing is lacking for autonomous operations. Some combination of radar, optical, and infrared (IR) sensors will likely be required; and image processing algorithms, especially for the latter two, are in their infancy. Most of the mission capabilities also require the autonomous avoidance of threat systems, including ships, boats, craft, active sensor systems, and, to whatever extent possible, passive detection systems. The community would benefit greatly from increased developments in this area. [1]

This Roadmap subsection affirms the need for collision avoidance developments in algorithms, sensors, and implementations. While control algorithms may be sufficiently mature, the integration of these algorithms with collision detection and

tracking algorithms is not mature. Additionally, analysis of commanded maneuvers for particular collision encounters is in its infancy. Therefore, additional research in these areas is warranted.

3. Problem Statement

Unmanned aerial systems are being widely used for intelligence, surveillance and reconnaissance (ISR) missions in both peaceful and wartime missions at home and abroad. As the number of separate systems grows and the number of unmanned vehicles in a single system increases, the ability to ensure the safety and integrity of the vehicles and ensure successful completion of missions is increasingly more difficult. Multiple vehicles are currently being used or are being tested for route surveillance, border and perimeter patrol, and support missions for all of the Unites States military services and other United States government agencies such as the Department of Homeland Security and Customs and Border Protection. Collision avoidance systems, whether implemented on-board the air platform or through a cooperative network, are a necessary component of the overall system.

Collision avoidance systems will come in many varieties and levels of complexity. The type, reliability, and autonomy will depend on operational requirements and the system it will protect. Neidhoefer, et al. state "It was concluded that functional determinism in autonomous systems is crucial…both to maximize the performance and potential benefits of such systems and to ensure that the operational environment…is not degraded for any stakeholders with respect to safety, organization, or ease of operation [2]." Not only will collision avoidance systems need to demonstrate effectiveness at their

defined tasks, they will be subject to intense evaluations, both for systems with and without humans in the loop.

4. Research Objectives/Hypothesis

The objective of this research was to develop a UAS collision avoidance system that deconflicts potential collisions and minimizes the mission impact as several aircraft perform a route surveillance mission. Multi-vehicle teams will be used for persistent surveillance of routes that will be traveled by convoys, borders between designated geographic areas, and perimeters of military and civilian bases and camps. These systems will maintain constant coverage of the route and identify possible threats along the route or within the area. The persistent surveillance constraint may result in operating the UAS in close proximity to each other throughout their coverage pattern. Additionally, exogenous inputs such as wind could cause unexpected encounters between UAS in the coverage pattern. Research in the Air Force Research Laboratory's Air Vehicles Directorate (AFRL/RB) has successfully shown efficient path planning of UAS that provide coverage of a route or perimeter while adding or removing UAS to the pattern and while changing the boundaries of the route or perimeter [3]. This research uses encounter geometries that may occur in these surveillance patterns as scenarios for potential UAS collisions. It is desired that the UAS successfully avoid the collisions and return to the prescribed search pattern while minimizing the impact on the sensor's route coverage. Altitude separation may not be a viable separation assurance method depending on sensor requirements, optimal operating conditions, and surveillance methods, so this research does not assume trivial collision avoidance measures (e.g. altitude separation).

As a proof of concept, the developed algorithm was tested on a small-scale micro air vehicle (MAV) testbed present within the Advanced Navigation Technology (ANT) laboratory. The testbed allows for a scaled version of UAS collision avoidance in a representative route surveillance mission.

The author asserts that a modified three-dimensional collision cone approach using aggregated cones and proportional navigation can successfully deconflict cooperative UAS in a range of encounter geometries. The algorithm will not rely on scripted maneuvers nor be limited to a particular spatial dimension and will provide commands to multiple aircraft in a cooperative network. The results will be the first known flight tests of a global, three-dimensional, geometric collision avoidance system on an unmanned aircraft system.

5. Research Focus

Significant amounts of research, development, and discussions in the literature involve the current issues of cooperative operations of UAS and airspace integration of those systems into airspace systems shared by manned aircraft. The term Sense and Avoid (SAA) is typically used to refer to the ability of an aircraft, autonomously for UAS and both autonomously and pilot controlled for manned aircraft, to detect a potential collision and command a resolution maneuver. This author defines two components to SAA: the longer time horizon aspect referred to as Separation Assurance (SA) that is dependent on procedures, mission plans, and possibly control station functions, and the shorter time horizon aspect Collision Avoidance (CA) that is dependent on aircraft

performance and response times as a last line of defense. The relationship between the three terms is shown in Figure 1-1.
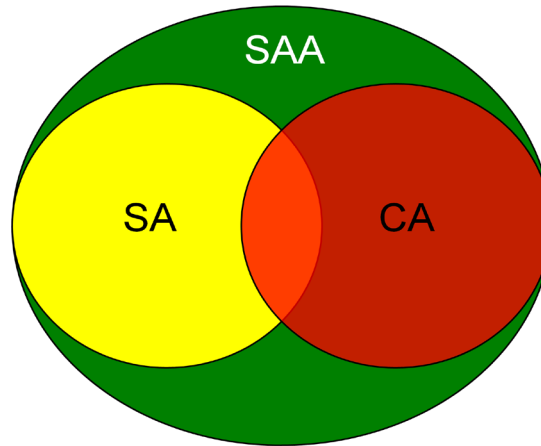
Figure 1-1: Sense and Avoid Components

SA and CA are not always distinguishable depending on the encounter, platform, and environmental circumstances; thus, there is overlap between them.

The algorithms developed in this thesis are intended to fulfill the CA function of a SAA system. It is assumed that mission procedures, human operators and the ground control station, if applicable, conduct SA actions but fail as a result of errors or exogenous inputs into the system and CA actions are required.

Planar collision encounter scenarios are of primary concern in this research because of the route surveillance mission operations. Planar, in this sense, describes multiple aircraft operating at constant above ground level (AGL) altitudes, thereby introducing collision possibilities while still allowing three-dimensional translational motion and collision avoidance reactions. Requiring constant AGL operation negates the trivial separation assurance procedure of altitude separation and is justified by any of the following reasons:

1. Surveillance pattern may require close proximity between platforms,

2. Sensors may be optimized for specific AGL altitudes so altitude separation would adversely affect the sensor measurements, and/or

3. Change detection requires operation at corresponding altitudes between passes because altitude separation would severely increase the false alarm rate.

6. Methodology

The steps necessary for successful development and testing of such an algorithm are now described. A significant amount of published basic research is used to develop the algorithms described in this thesis. Nonetheless, additional theory development is completed to extend the published theory for application to this problem. Following theory development, application to UAS CA is completed by focusing the research on a particular type of system and operation. With this information, tests can be identified to exercise the algorithms and performance measures can be enumerated.

Testing must be performed in a sequential manner with increasing uncertainty and complexity added in each step. First, ideal simulations are used to verify algorithm theoretical capabilities. For example, collision detection is tested in an ideal simulation by constructing an encounter guaranteed to result in a collision. Similarly, collision avoidance is tested using the same encounter and is successful if the collision is evaded. Assumptions are applied in these simulations such as simplified three degree-of-freedom dynamics and perfect command tracking, thereby, alleviating any uncertainty.

Next, complexity is added by testing the algorithm in a representative environment that it will ultimately operate in. Software-in-the-loop (SIL) simulation

7

capability is offered by the manufacturers of the UAS selected for integration. Interfaces between the CA algorithm and the system environment are developed in order to complete SIL and follow-on tests. The interface is defined in such a way so that it can be used on the actual operational system. SIL simulations add necessary uncertainty that exists in real-world applications and can be used as a gateway so that if they are not successful, progression to the next step is halted until major problems are resolved.

Further complexity and uncertainty is added with the next test type that is also offered by the manufacturer of the selected UAS. Hardware-in-the-loop (HIL) tests require use of actual external hardware and firmware that will either be onboard the aircraft platforms or used on the ground during actual operation. Most firmware and integrated software should be identical to that in the SIL tests, but additional communication and processing uncertainty now exists when operating on several different machines. When HIL and SIL results compare favorably, the algorithms and associated interfaces are ready for testing in a fully operating system and are ready for the next phase of testing.

The culmination of the CA system's development is flight test. A successful demonstration in flight test, with real-world uncertainty, complexity, and environmental effects, will solidify claims of the algorithm's effectiveness in its particular application. Flight test procedures and objectives must be carefully planned and executed in order to demonstrate the CA system's intended operation and to return results supporting the system's use in future UAS missions. Flight test cases must be constructed properly to represent scenarios that will exist in actual operation. Finally, data reduction following

the flight test is necessary to communicate the test's successes, failures, and potential follow-on improvements for the CA system.

7. Assumptions/Scope

This research develops and exercises a UAS Collision Avoidance Algorithm That Minimizes the Impact on Route Surveillance. It should be noted that no optimal control or optimal trajectory generation is used in this algorithm. That is not to say, however, that portions of the algorithm would not benefit from the use of such theory in the future. The CA algorithm is intended to monitor traffic internal and external to a cooperative network of UAS platforms. It will detect imminent collisions between any one of the platforms and another aircraft and command an appropriate guidance maneuver. The maneuver, based on the geometry of the collision and minimum separation definitions, tends to command small deviations to maintain minimum separation; thus, minimizing the maneuvering and its effect on the mission although not in an optimal sense. Additionally, the guidance laws applied here initially command small maneuvers that will grow in magnitude as the range decreases between the aircraft. Thus, collision encounters that are mitigated early in the encounter timeline will have been resolved with small commands minimizing impact on the mission. The algorithm does not provide a recovery course of action or commands to return the platforms back to their original trajectories. It does, nonetheless, return control of the UAS, after the collision encounter has been abated, back to the navigation algorithm embedded in the hardware. The navigation function is then used to determine the appropriate route back to the surveillance pattern. The algorithm is applicable to many different systems and operations but is limited in this

research to a single available system. This system does not allow onboard processing or sensing, and therefore, cannot detect external collisions outside of the cooperative network. Throughout the development of the CA algorithm, the potential for external threats is considered and the CA algorithm supports inputs from any sensing device provided the data is sufficient and in the proper format.

8. Preview

Chapter II of this thesis will review applicable theory and applications developed in seminal and contemporary literature. Chapter III discusses original theory, algorithm, and software developments by this author in addition to hardware and software provided by the ANT lab and used in this research. Chapter IV provides analysis and results of all testing performed throughout the research. Chapter V discusses the results and communicates the author's conclusions, conjectures, and recommendations for future researchers and/or users.

II. Literature Review

1. Chapter Overview

A significant amount of research and development has been ongoing for years involving the higher level topics of SAA and conflict detection and resolution (CD&R) and the lower level functions of SA and CA that are encompassed by SAA and CD&R. Collision avoidance, as discussed in this thesis, is defined as the detection of an imminent collision or violation of some minimum separation distance and a commanded avoidance maneuver after SA has failed. Before unmanned aircraft became prevalent, most of this research was directed towards commercial aircraft in the United States National Airspace System (NAS) and in ground-based robotics. Similar techniques and systems have been applied to the UAS collision avoidance problem as well as a variety of newly proposed methods. The UAS sub-systems involved in these processes are commonly referred to as SAA systems and detect, sense, and avoid systems (DSA). The following sections describe common methods applied to UAS CA and examples of implementation or the development of each.

Kuchar and Yang present a broad review and survey of CD&R methods in two papers [4] [5]. These papers discuss a wide array of methods applied to the CD&R problem to the date of their publications. Section 2 of this chapter reveals current developments in recent years to this field and key information for each.

When characterizing, comparing and contrasting collision avoidance approaches, one must describe certain properties of the proposed methods. Kuchar and Yang group all algorithms and approaches based on state dimensions, resolution maneuvers, and multiple

conflict properties of the algorithms. Dowek and Muñoz further define these categories

[6]. State dimension refers to two or three-dimensional conflict modeling for the

detection and avoidance of collisions. Two-dimensional modeling typically concentrates

on a horizontal or vertical plane only. Resolution maneuvers are characterized also by the

number of dimensions they inhabit. For instance, a maneuver only in the vertical plane

would be limited to altitude and airspeed changes and would not allow a turn maneuver.

Multiple conflict properties describe whether a system views CA in the global or pair-

wise sense. Global CA involves all aircraft in the airspace which the algorithm considers

when monitoring the dynamic airspace, detecting collisions, and commanding

maneuvers. This is global in the sense that the UAS considers all threats that it is aware

of simultaneously. This method applies to both cooperative and non-cooperative aircraft.

Pair-wise avoidance involves one UAS and one intruder regardless of the surrounding

environment. Encounters between aircraft are resolved one at a time but can include

considerations for future conflicts or maneuvers. Nonetheless, subsequent avoidance

maneuvers by one UAS avoiding multiple intruders in a pair-wise fashion are not

considered global. Pair-wise is also applicable to cooperative and non-cooperative

aircraft.

2. Methods

2.1. Geometric

Geometric collision detection and avoidance methods involve geometric

properties of aircraft trajectories and utilize positions and velocity vectors of all or

some aircraft involved in the encounter. Geometric methods can be used for collision detection by comparing velocity vectors of vehicles and obstacles, and can aide in collision resolution/avoidance by providing encounter geometry to the resolution guidance algorithm.

Chakravarthy and Ghose proposed a geometric collision detection and avoidance method in a dynamic environment with no constraints on vehicle shape or size. Their concept, the collision cone approach, originally developed for robotics, has proven to be a valuable foundation for other geometric approaches and methods and has been cited many times in the literature [7]. Using state information from the vehicle and obstacles, the collision cone approach analytically defines a collision region for which a collision is imminent if the vehicle velocity vector lies in this region. A thorough outline of the algorithm is given in addition to a number of examples by the authors in their publication.

A collision detection and avoidance approach referred to as the geometric optimization approach has been proposed [8]. This geometric collision detection method is a typical comparison of velocity vectors, but the resolution is optimized in the sense that it attempts to minimize the deviation from the nominal trajectory. The author also discusses the geometric optimization approach for multiple intruders but limits the effort to sequential avoidance of the most critical encounter. This is still considered pair-wise CD&R as opposed to global. Bilimoria noted "that resolutions for multiple-aircraft conflicts obtained by sequential pair-wise solutions do not necessarily minimize deviations from the nominal trajectories." Therefore, the

13

minimum deviation from the nominal trajectory may not be a combination of pair-wise encounter maneuvers, and the solution must be examined in the global sense.

Goss, Rajvanshi, and Subbarao consider the conflict detection and resolution problem using geometric and collision cone approaches for two aircraft in a three dimensional environment [9]. The pair-wise avoidance solution is found using the collision cone approach to detect a collision and generates a combination of velocity, heading, and elevation changes to avoid the collision. Analytical solutions are rigorously found for special encounter situations, but a numerical solver is used for more general cases. Numerical solutions are not ideal for real-time applications, and according to the authors, "the nonlinear equation solver has a tendency to get stuck at spurious updates in more complex scenarios." This pair-wise solution method would only increase in difficulty for simultaneous multiple intruders.

A pair-wise non-cooperative decision making algorithm for three-dimensional collision avoidance was presented by Carbone, et al. The authors contend their collision avoidance method is suitable for real-time applications because of analytical solutions that do not require numerical programming [10]. Similarly to Goss, Rajvanshi, and Subbarao, Carbone's three-dimensional geometric method is based on the collision cone. Numerical simulations are offered to demonstrate the algorithm's ability to maintain minimum separation while including sensor field-of-view limitations. The resolution algorithm, nevertheless, does not utilize all three of its control variables (i.e. longitudinal, lateral-directional, and speed) at the same time. However, comparisons of the three are presented.

14

2.2. Force Field

Force field methods are global approaches to CD&R. Vehicles can individually be represented as charged particles and repulse each other given position and velocity information of each or the entire airspace can be defined as a potential field or magnetic field and vehicles are maneuvered based on the global environment. This method is suited for distributed collision avoidance where state information is readily available from all vehicles, but can be applied to local avoidance when the number of vehicles is small.

Despite the distributed and global aspects of force field methods, several elements of these approaches are difficult to incorporate into practical systems. When generating a dynamic potential field, saddle points and local minima can disrupt the flow of vehicles and introduce additional problems such as aircraft stall or further collision threats. As eluded to previously, aircraft performance and dynamic characteristics must be taken into account when generating a field or evasive commands. If complete state information is not known for all vehicles or if a magnetic/potential field is not properly formed, aggressive control commands may be generated that are outside of the vehicles' abilities.

Sigurd and How investigate a "total-field sensing approach of magnetic nature" focused on systems with a large number of N vehicles [11]. The authors assert that local control approaches break down as N grows and the complexity and potential for collision grow exponentially. Also, many previous potential field approaches required perfect information or perfect sensing for safe maneuvering

through the obstacle field. Complexity and imperfect information, or lack of

information, say Sigurd and How, necessitate a distributed control approach. The

authors provided simulation results and a hardware experiment as advocates for their

algorithm. Although their discussions of increasing collision potential as N grows and

the benefits of distributed control are thorough, the authors' approach has many

drawbacks to aircraft collision avoidance. All vehicles would be required to carry a

magnetic field generating device and a magnetic field sensing system, limiting this

approach to cooperative collision avoidance.

A multiple-vehicle UAS deconfliction algorithm based on potential functions,

referred to by the authors as navigation functions, was developed and presented by

Rahmani, et al. [12]. The approach addresses conflict prediction, resolution,

navigation and control of flying vehicles while obeying mission requirements.

Simulations are described that support their approach. This paper expands traditional

potential methods by using maneuvering obstacles, ensuring vehicles are in constant

motion and embedding mission requirements in the construction of the navigation

function. Aircraft operational limitations are included in the formulation of guidance

and control commands. Stagnation problems, a typical drawback of potential

functions for flying vehicle applications, are dealt with in this paper with the addition

of a swirling effect in the potential function. This, however, detracts from the

compliance with vehicle operational limitations formulated in the original function.

Consequently, vehicle motion constraints and maneuver constraints cannot be

guaranteed simultaneously. Saddle points are also a complication in the potential function and would cause a catastrophic effect in real applications.

2.3. Probabilistic

Probabilistic methods for aircraft CD&R may involve the calculation of the probability of collision based on current aircraft states and possible perturbations about that nominal state or the probability of collision based on all possible maneuvers and their likelihood of occurring. These methods avoid the conservativeness of worst-case prediction methods while maintaining robustness to uncertainty [13]. Statistical representations of the airspace environment, for global applications, and of its inhabitants, for pair-wise applications, must be characterized prior to algorithm design. Detailed knowledge of the airspace and its inhabitants is required and is used, typically, in Monte Carlo simulation for characterization.

Prandini, et al., 1999 and 2000, present approaches for probabilistic conflict detection for mid-range and short-range conflict scenarios [13] [14]. The authors define mid-range conflicts in the time horizon as tens of minutes and short-range conflicts as seconds to minutes. The probability of conflict is characterized by Monte-Carlo simulations and, in some special cases, closed form solutions are presented. These papers are focused primarily on the Air Traffic Management System (ATMS) and aircraft following flight plans and their respective waypoints. The CD&R functions are computationally intensive and require closed-form approximations and estimating algorithms for real-time applications.

Probability based methods have been applied to non-aircraft conflict detections [15]. Although the application is quite different, derivations of time to closest approach (TCA) and minimum miss distance (MMD) for spacecraft could be applied to aircraft as long as linear assumptions are valid. Simulation and experimental results were used to validate the probability of collision calculations.

Probabilistic methods specifically for UAS collision avoidance are discussed in three dimensions assuming UAS constant velocity [16]. The collision is decomposed into a horizontal plane and a vertical plane, and minimum separation criteria are defined for each plane. Probabilistic trajectory modeling is accomplished by modeling uncertainty in own-ship and intruder position and velocity obtained from a data-link system and in intruder maneuvering uncertainty. Threat levels are defined for probability of collision values determined from Monte-Carlo simulations. Scripted maneuvers are defined for each threat level in three dimensions although in simulation only vertical maneuvers are performed. The authors attribute this to the minimum separation definitions which make vertical maneuvers less aggressive.

2.4. Other Methods

Collision avoidance is a primary topic in swarming/flocking research of birds, insects, and other animals with applications to multiple aircraft operations in close proximity. Park, Tahk, and Bang discuss the historical evolution of swarming/ flocking research in computer graphics, gaming, and most importantly, aerospace applications [17]. The authors reference Reynolds' research of flocking behaviors and steering behaviors and his creation of "boids" (bird-oid) in flocking simulations [18]

[19]. Flocking can be modeled using three distinct and simultaneous behaviors, one of which is collision avoidance. Also, sub-behaviors were defined that make up the three main behaviors. Those sub-behaviors that relate to collision avoidance include fleeing, evading, and obstacle avoidance. Many of these basic functions are implemented using geometric techniques of summing or aligning velocity vectors.

Park, Tahk, and Bang implement CA in a pair-wise sense by defining a "safety-bubble" around the boid (aircraft) and commanding a scaled steering command opposite of the line of sight direction to the closest boid violating the bubble. This method is tested in three degree-of-freedom simulations.

A recent development in CA does not involve a detection or control method, but involves a spatial representation with which to define the avoidance approach. The Curvature-Velocity-Orientation (CVO) Method transfers aircraft motion from Cartesian space into CVO space and applies a potential field CA method for obstacle avoidance [20]. The potential field method transformed into CVO space is designed to take into account aircraft dynamic constraints making its application to UAS CA more achievable. Successful simulation results are shown but only for stationary obstacles. Also, the CVO results, as compared to the Cartesian space counterpart, contain undesirable oscillations in its final trajectory.

3. **Flight Tests and Notable Simulations**

Notable simulations of CA system responses to collision encounters include tests by Farley and Erzberger, and Paielli [21] [22]. Farley and Erzberger used recorded Federal Aviation Administration (FAA) air traffic data in the Cleveland Air Route Traffic

Control Center airspace to test their conflict resolution algorithm in nominal and heavy traffic conditions. Paielli used archived data of actual loss of separation due to controller error tracking data to test his method of solving imminent air traffic conflicts. The archived data consists of 100 operational error occurrences caused by the controller, which, according to Paielli, tend to be more difficult to detect and resolve than routine conflicts that get resolved successfully.

Another interesting simulation implementation of a CA system was focused on vision-based obstacle avoidance for UAS. The respective avoidance algorithm is based in Minimum Effort Guidance and was compared to proportional navigation guidance in a sequence of publications [23]. This guidance method was then tested using a six degree-of-freedom image-in-the-loop simulation set-up to exercise the vision-based detection algorithms and the subsequent avoidance maneuver [24]. Results of image processing, estimation, and guidance are analyzed. Image processing post-analysis shows the image processing algorithm did detect the obstacles but not as expected. The guidance system performed as expected and maintained required separation from the obstacles, but a significant limitation of these results is that the obstacles were stationary.

Significant amounts of hardware and software-in-the-loop simulations and calibrations, including communication system latencies, have been performed to prepare an obstacle detection, tracking, and CA system for flight test [25]. Ground tests characterizing system behavior and latencies are being performed for algorithms and sensors, both electro-optical (EO) and radar, and statistical performance properties of these have been defined.

A successful flight test of a maneuvering aircraft around another stationary, hovering aircraft was described by Neifhoefer, et al. [2]. The authors' intent was to demonstrate that practical implementations of highly autonomous functionally deterministic systems are possible. "Highly autonomous" in the previous sentence is a broad statement that could include traditional fly-by-wire autopilots or systems with high-level autonomy that involve complex decision-making or interaction with humans. The UAS in this experiment was commanded to fly a straight line path to a point along which it would collide with another aircraft. The collision avoidance system detected a collision and generated a safe, modified trajectory around the other aircraft to the goal point. Additionally, the modified path could be controlled and the resulting direction of the avoidance maneuvers was thereby changed. Results of the flight test are shown and conclusions are made about the feasibility of functionally deterministic systems being used on UAS.

Large scale flight tests have been completed by Northrop Grumman Corp. and AFRL using the variable-stability Calspan Learjet as a UAS surrogate aircraft [26]. These flight tests investigated the feasibility and effectiveness of the Traffic Alert and Collision Avoidance System (TCAS), a human-in-the-loop collision warning system used on today's commercial aircraft, in an autonomous collision avoidance role. The benefits of using TCAS in future UAS sense and avoid systems were shown through analyses of the flight test results. The tests consisted of a variety of encounter scenarios between the surrogate UAS and intruder aircraft: 1) level head-on, 2) abeam, 3) ascending head-on, and 4) descending head-on.

4. Summary

Chapter II provided a comprehensive, but not all-inclusive, overview of methods and techniques that have and are being applied to collision detection and avoidance. The most common methods were discussed by providing examples of their uses, from theoretical derivations to modern applications. The topic of UAS CA has been exhaustively researched, but no solution yet exists that can provide a generic capability for effective and safe CA for all aerospace applications. It is not this author's intent to undertake this daunting task; this thesis is meant to provide a capability for the UAS mission described in Chapter I and that is applicable to other UAS civil and military operations.

Geometric methods provide the most straightforward and extensible collision detection and avoidance techniques. A large amount of research has been completed on these approaches as they apply to UAS CA. However, no single existing solution addresses three-dimensional CA in the global sense for cooperative UAS operations. Cooperative, as defined in this research, is the exchange of information between UAS, either directly between platforms or through some single control station. Force field methods are well suited for global collision avoidance, but drawbacks such as including UAS performance limits and expansion to three-dimensional applications hinders their use. Probabilistic methods accommodate uncertain air traffic environments that can be described by statistical properties. This, however, requires modeling the probability of future trajectories for all entities in the environment, both cooperative and non-cooperative. Geometric methods address cooperative and non-cooperative traffic in the

22

same manner by comparing nominal trajectories using state information from either a cooperative network or sensors. This research focuses on the cooperative network of UAS, but the resulting algorithm is compatible with other sensed traffic. A geometric method, based on the collision cone approach [7], is used as the foundation for a three-dimensional global collision avoidance algorithm for UAS collision avoidance.

The collision cone approach has concurrently been expanded to three dimensions and applied in the global sense by this thesis' author but only for a single UAS [27]. Considerable amounts of additional research and development is needed for a novel and robust algorithm that manages both cooperative and non-cooperative traffic and detects imminent collisions and issues avoidance commands to a group of UAS. That type of algorithm does not yet exist in a single solution and certainly has not been flight tested according to an extensive literature review. This research engages both the algorithm and implementation deficiencies, and flight tests the resultant solutions.

III. Methodology

1. Chapter Overview

As discussed in Chapter II, the collision detection and avoidance method being employed in this research is based on the collision cone approach. According to the geometry of the encounter and the rates of change of translation and orientation, safe regions of flight are defined by the algorithm for each UAS in the cooperative network. In order to complete the algorithm, each UAS must be directed to safety in a manner consistent with the geometry of the encounter. Consequently, an algorithm providing guidance commands that are integrated with the detection algorithm is developed and refined for UAS CA.

A generic architecture, represented in pseudo-code, is shown in Figure 3-1 and describes in detail the process flow of a CA algorithm integrated with the navigation system of a UAS. The acronym GCS stands for ground control station which is the controlling unit that commands all UAS in the cooperative group. This research will not address all possible CA functions shown in the figure (see Chapter I, Section 7). Non-cooperative traffic will not be considered so Function 2 under "IF COMM LINK -> GUID_MODE" is not performed. Similarly, "ELSEIF NO COMM_LINK" is not considered, because information must be transmitted between aircraft for cooperative CA. Lastly, RECOVER_MODE is assumed to be the navigation mode of the UAS and is not considered in this CA system.

```
IF COMM_LINK

    GUID_MODE
            1)  Monitor Cooperative Airspace (GCS)
                -   Collision detection algorithm

        IF NO_CONFLICT
                1)  Fly Surveillance Pattern

        ELSEIF CONFLICT
            CA_MODE
                    1)  Do Not Maneuver
                        -   Not immediate threat
                        -   Continue monitoring cooperative airspace
                    2)  Maneuver
                        -   Function of range, threat maneuvering, priority
                            to maneuver, own status (are you the threat)
                        -   Rules of the road
            RECOVER_MODE
                IF NO_NEW_CONFLICT
                1)  Recover
                ELSEIF NEW_CONFLICT
                    CA_MODE

        2)  Monitor Non-cooperative Airspace (On-board sensing)
            -   Requires on-board detection algorithm for non-cooperative traffic
                in addition to GCS detection algorithm for out-of-view cooperative
                traffic

        IF NO_CONFLICT
            Same as above
        ELSEIF CONFLICT
            Same as above

ELSEIF NO COMM_LINK

    GUID_MODE
        1)  GCS Only

                i)      Prescribed altitude separation
                ii)     Loiter at current position
                iii)    Loiter at prescribed location
                iv)     RTB

        2)  On-board sensing
```

Modes:
GUID_MODE – route surveillance guidance mode, collision detection algorithm
    monitors UAS airspace
CA_MODE – collision avoidance system mode, assumes control from guidance mode,
    flagged by detection algorithm
RECOVER_MODE – recovery mode from collision avoidance to route surveillance, only
    initiated after CA_MODE

Status:
COMM_LINK – health of communication link (binary or threshold)
NO_CONFLICT – detection algorithm status
CONFLICT – detection algorithm status, initiates CA_MODE
NO_NEW_CONFLICT – recovery mode status, initiates recovery maneuver
NEW_CONFLICT – recovery mode status, initiates CA_MODE

Figure 3-1: CA Algorithm Pseudo-Code

2. Theory and Algorithms

The collision cone approach uses position, velocity, orientation, and orientation rate to determine if the nominal, or dead-reckoned, trajectories of two or more vehicles will result in a violation of some minimum miss distance [7]. This approach applies to any irregularly shaped object and depends only on current flight path information. Chakravarthy and Ghose derived the planar algorithm that defined the angular bounds of the collision cone. Smith, et al., derived the angular rates of change of the cone bounds and described their use in a three-dimensional CA control scheme based on Proportional Navigation (PN) guidance [27]. These two algorithms combined form the foundation of a robust and expandable CA algorithm that is compatible with real-time applications.

A collision cone is the region within which the velocity vector of a vehicle will violate an obstacle separation zone or collide with that object. The cone is a function of current states only but is numerically straightforward and not computationally intensive. A comprehensive description of the algorithm is given in Reference [7]. The nomenclature is repeated here for convenience. Figure 3-2 shows two-dimensional collision cones for three generic encounter cases. A single intruder can have as many as two collision cones if it is a collision threat, and for N intruders, there can be as many as $2N$ cones. In Figure 3-2, a blue arrow represents an aircraft velocity vector, a black circle defines the minimum separation area around a collision threat, a blue line represents the line of sight, red lines are the collision cone bounds, and green arrows point to the interior of the collision cone.

| (a) | (b) | (c) |
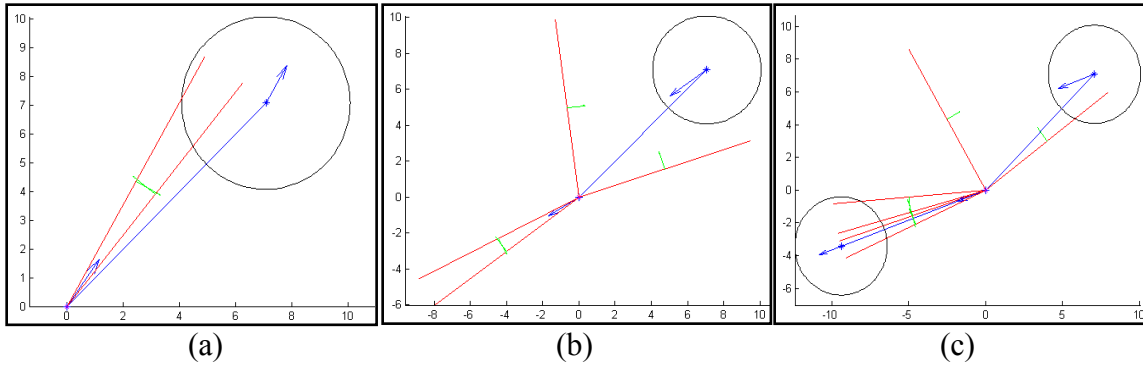
Figure 3-2: Two Dimensional Collision Cone Configurations (a) Single Cone (b) Split Cone (c) Multiple Intruders, Single and Split Cones [27]

Alpha, α, is the UAS velocity vector direction defined as positive counter-clockwise (CCW) from the x-axis. Beta, β, is a potential threat's velocity vector direction also defined positive CCW from the x-axis. Gamma, γ, is the UAS vertical flight path angle defined positive up. Chi, χ, is the potential threat's vertical flight path angle. Theta, θ, is the horizontal plane line-of-sight (LOS) angle from a UAS to a particular threat defined as positive CCW. Phi, φ, is the vertical plane LOS angle defined along the UAS velocity vector and positive up from the horizontal plane. Range, r, can be used to represent the horizontal distance between two aircraft or the slant-range distance. The horizontal minimum separation radius is given by R and defines the lateral separation. The vertical minimum separation is referred to as the vertical offset (*VO*). The collision cone boundaries are given as a lower and upper angular bound, α_1 and α_2, respectively. Similarly, their rates of change are $\dot{\alpha}_1$ and $\dot{\alpha}_2$, where (˙) represents time rate of change. Smith, et al., derived the rates of change of the cone bounds, and these equations are repeated in Appendix B with some additional simplifications. Each collision cone has its own set of boundaries and rates, and each plane has its own set of cones. The bounds and

27

their rates of change are defined entirely by the encounter geometry, including range, speed, heading, and the minimum separation distance. The bounds and bound rates returned by the extended collision cone algorithm are normalized by the LOS angles and their rates and are given by η_1, $\dot{\eta}_1$, η_2, and $\dot{\eta}_2$, respectively. The absolute bounds are given by Eq. (1), and their rates by Eq. (2). The vertical plane representation is given by simply replacing θ with φ and applying the appropriate upper and lower bounds.

$$\begin{aligned} \alpha_1 &= \eta_1 + \theta \\ \alpha_2 &= \eta_2 + \theta \end{aligned}$$
(1)

$$\begin{aligned} \dot{\alpha}_1 &= \dot{\eta}_1 + \dot{\theta} \\ \dot{\alpha}_2 &= \dot{\eta}_2 + \dot{\theta} \end{aligned}$$
(2)

The collision cone approach is valid for irregularly shaped objects. These oddly shaped objects are decomposed into circular regions that can be used to define a minimum separation zone around the object. This also allows different definitions of minimum separation in the lateral and vertical directions. Many of the geometric-based CA algorithms discussed in Chapter II assume a spherical safety zone around obstacles and other aircraft. This assumption simplifies the derivation of some detection and avoidance algorithms but does not allow the flexibility of independent separation distances in the horizontal and vertical planes. The minimum separation volume in this research uses definitions from the Federal Aviation Administration (FAA) of aircraft encounters. The FAA classifies the encounter's criticality by the horizontal separation and vertical separation of the aircraft involved. Likewise, a lateral and vertical separation forming a cylinder in three dimensions is used for this research. When viewed in the

horizontal plane, the cylinder appears as a circle and the collision cone algorithm can be applied directly. When viewed in the vertical plane, the cylinder appears as a rectangle that must be decomposed into a circle for application of the algorithm. Lines extending from a particular UAS location tangent to the rectangle's protruding corners in the vertical plane can be found, and a circle residing in these lines whose boundary is also tangent to them can be defined. The radius of this circle changes as the range from the UAS to the center of the circle changes. By selecting the range to the center of the circle to equal the range from the UAS to the threat, the radius of the circle can be found. Reference [7] gives this relationship and it is repeated in Eq. (3).

$$R_v = r sin(\psi_v/2) \tag{3}$$

With the definition of a radius for the minimum separation in the vertical plane, the collision cone approach can now be directly applied. The vertical plane geometry as viewed by the algorithm is shown in Figure 3-3.

(a) (b)
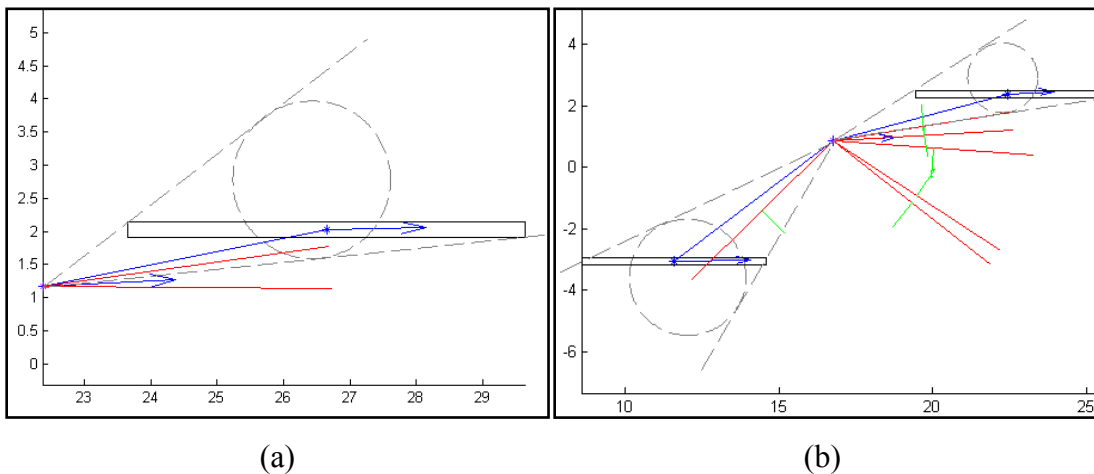
Figure 3-3: Collision Cone Approach in the Vertical Plane (a) Single Cone (b) Multiple Intruders, Single and Split Cone [27]

The line and arrow features in Figure 3-3 are the same as described for Figure 3-2, except the black circle is replaced by a black rectangle for the minimum separation area. Grey dashed lines project from the UAS velocity vector and are tangent to the rectangles protruding corners. The resultant separation circle calculated using Eq. (3) resides inside these lines and is also grey and dashed.

Given collision cone angular bounds and their rates of change defined in two planes, the horizontal and vertical, as a function of different separation criteria for minimum lateral separation and vertical offset, a sufficient description of the encounter geometry exists for CA. It is a simple extension from a single three-dimensional cone for a single obstacle to multiple cones for several obstacles. The modified collision cone algorithm is simply executed for each obstacle. However, this gives only a disassociated view of the threat environment and represents sequential pair-wise collision detection. A conservative, yet effective, approach to providing global collision detection is to aggregate overlapping individual collision cones in their respective planes into a single, all-encompassing cone that describes every potential obstacle threat in that region. Multiple collision cones are still possible if there is no overlap. The aggregate cone method guarantees the velocity vector will only exist in a single cone, and any necessary avoidance maneuvers are with respect to that single cone. Thus, the assignment of commands is not confounded by prioritization of individual threats or selection schemes of a set of maneuvers and no conflicting commands are issued. Figure 3-4 gives a visual representation of aggregating cone bounds.

Figure 3-4: Aggregate Cone Bounds

Figure 3-5 gives a complete description of key attributes of a two-ship encounter as viewed by the aggregate multiple vehicle UAS collision cone algorithm.

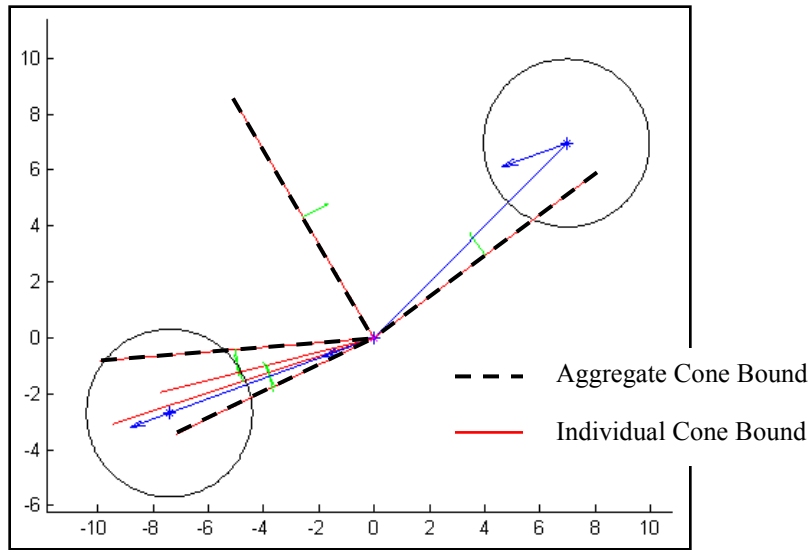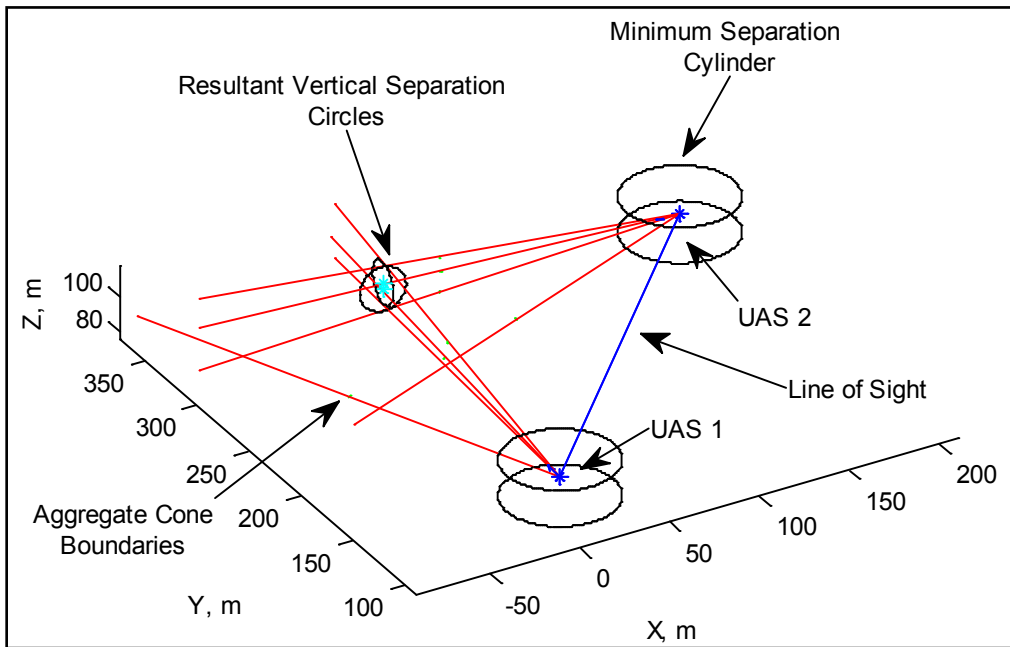**Figure 3-5: Full Encounter Description**

Using collision cone information about the angular bounds and their rates of change, guidance commands can be generated in concert with the geometry of the collision. Han and Bang, and subsequently Han, proposed a proportional navigation based CA scheme for UAS CA [28] [29]. They found the relative velocity vector of the UAS could be guided to a "collision avoidance vector" with proportional navigation and thereby alleviating the collision. Han also went as far as deriving the optimal proportional gain assuming constant velocity and investigated convergence of the guidance law as a function of the navigation constant. In this application to cooperative UAS CA, a similar approach is used for the CA guidance commands. The UAS velocity vector is chosen instead of the relative velocity vector because the collision cone approach is used instead of the geometric configuration approach discussed by Han. Han uses a geometric comparison of the relative velocity vector and the tangents to a minimum separation zone around an obstacle to define the guidance parameters. By using the collision cone approach, a collision detection method is not limited to a single vector comparison but is able to define an entire region of unsafe operation. The UAS velocity vector can then be guided outside of this unsafe region instead of just a single obstacle cone. Once the velocity vector is coincident to the edge of the aggregate collision cone, the UAS is guaranteed to maintain minimum separation by flying a trajectory that results in tangency to the minimum separation volume. At this point, guidance commands may cease and the cone and velocity vector are monitored for future violations. The proportional navigation guidance law used in this UAS CA algorithm is shown in Eq. (4).

$$a = NV_{UAS}\dot{\alpha}_{pn} \tag{4}$$

In Eq. (4), a is the commanded acceleration, N is the navigation constant, V_{UAS} is the UAS velocity, and $\dot{\alpha}_{pn}$ is the cone bound rate used for guidance (read on for a description of its selection). This guidance law is a variation of generalized true proportional navigation [30] where the commanded acceleration a, or, synonymously, angular rate, is perpendicular to the line of sight offset by a fixed angle. This offset angle is the normalized cone bound angle calculated using the collision cone approach that is then added to the line of sight to form the absolute cone bound. The proportional gain, which Han proved must be greater than one for convergence, is chosen depending on aircraft maneuverability and the geometry of typical encounters.

Which cone bound angular rate is used in the guidance law is chosen based on the magnitude and direction of the bound movement. For instance, a contracting cone could either have both cone bounds converging towards the center of the cone, or one cone bound converging to the center faster than the other bound that is diverging. Similarly, an expanding cone either has both cones diverging or one bound diverging faster than the other converging bound. The angular rate used in the guidance law depends on whether the cone is diverging or converging, whether the bounds are moving in the same or opposite directions, and the magnitude of each bound angular rate. The preceding discussion applies to both the horizontal plane and the vertical plane. The following tables, Table 3-1 and Table 3-2, define the logic for the guidance law angular rate choice and the corresponding angle choice. Blue shading denotes a diverging cone and green shading denotes a converging cone. Subscript 1 implies the upper cone bound and subscript 2 implies the lower cone bound.

33

Table 3-1: Guidance Law Angular Rate Matrix, Horizontal Plane

	$\dot{\alpha}_1 \geq 0$ & $\dot{\alpha}_2 \geq 0$	$\dot{\alpha}_1 \geq 0$ & $\dot{\alpha}_2 \leq 0$	$\dot{\alpha}_1 \leq 0$ & $\dot{\alpha}_2 \leq 0$	$\dot{\alpha}_1 \leq 0$ & $\dot{\alpha}_2 \geq 0$
$|\dot{\alpha}_1| \geq |\dot{\alpha}_2|$	$\dot{\alpha}_{pn} = \dot{\alpha}_2$ $\alpha_{pn} = \alpha_2$	$\dot{\alpha}_{pn} = \dot{\alpha}_1$ $\alpha_{pn} = \alpha_1$	$\dot{\alpha}_{pn} = \dot{\alpha}_2$ $\alpha_{pn} = \alpha_2$	$\dot{\alpha}_{pn} = \dot{\alpha}_1$ $\alpha_{pn} = \alpha_1$
$|\dot{\alpha}_1| \leq |\dot{\alpha}_2|$	$\dot{\alpha}_{pn} = \dot{\alpha}_1$ $\alpha_{pn} = \alpha_1$	$\dot{\alpha}_{pn} = \dot{\alpha}_2$ $\alpha_{pn} = \alpha_2$	$\dot{\alpha}_{pn} = \dot{\alpha}_1$ $\alpha_{pn} = \alpha_1$	$\dot{\alpha}_{pn} = \dot{\alpha}_2$ $\alpha_{pn} = \alpha_2$

Table 3-2: Guidance Law Angular Rate Matrix, Vertical Plane

	$\dot{\alpha}_1 \geq 0$ & $\dot{\alpha}_2 \geq 0$	$\dot{\alpha}_1 \geq 0$ & $\dot{\alpha}_2 \leq 0$	$\dot{\alpha}_1 \leq 0$ & $\dot{\alpha}_2 \leq 0$	$\dot{\alpha}_1 \leq 0$ & $\dot{\alpha}_2 \geq 0$
$|\dot{\alpha}_1| \geq |\dot{\alpha}_2|$	$\dot{\alpha}_{pn} = \dot{\alpha}_1$ $\alpha_{pn} = \alpha_1$	$\dot{\alpha}_{pn} = \dot{\alpha}_1$ $\alpha_{pn} = \alpha_1$	$\dot{\alpha}_{pn} = \dot{\alpha}_2$ $\alpha_{pn} = \alpha_2$	$\dot{\alpha}_{pn} = \dot{\alpha}_2$ $\alpha_{pn} = \alpha_2$
$|\dot{\alpha}_1| \leq |\dot{\alpha}_2|$	$\dot{\alpha}_{pn} = \dot{\alpha}_2$ $\alpha_{pn} = \alpha_2$	$\dot{\alpha}_{pn} = \dot{\alpha}_2$ $\alpha_{pn} = \alpha_2$	$\dot{\alpha}_{pn} = \dot{\alpha}_1$ $\alpha_{pn} = \alpha_1$	$\dot{\alpha}_{pn} = \dot{\alpha}_1$ $\alpha_{pn} = \alpha_1$

The logic is based on intuition and trial and error. For example, if both cone bound rates are positive, it would require minimal effort to deconflict by maneuvering in the opposite direction, allowing the cone to move away from the velocity vector in addition to the vehicle moving its velocity vector out of the cone. This is visually depicted in Figure 3-6.
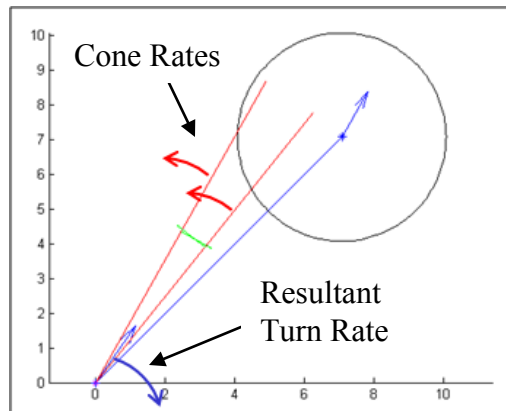

Figure 3-6: Guidance Logic Example

Assuming Figure 3-6 is in the horizontal plane, it corresponds to the first column in Table 3-1. More information (i.e. magnitude of the bound angular rates) is needed to determine the corresponding row.

When overlapping cones are combined into an aggregate cone, the new cone bounds are used in the guidance logic when choosing which angular rate to use in the guidance law. The bound corresponding to the angular rate choice used in the guidance law is maintained throughout the encounter as the guidance bound selection in order to prevent oscillating commands resulting from bound switching. Bound switching could occur if two separate cones overlap in the middle of the encounter or if an individual cone inside of the aggregate cone envelops another individual cone.

The guidance commands generated independently from the horizontal and vertical planes are decomposed into components along and perpendicular to the UAS velocity vector. Once decomposed, they can be combined into a single set of commands in three dimensions. These commands are turn rate ($\dot{\alpha}_c$), translational acceleration ($\dot{v}_c$), and rate of change of the vertical velocity vector ($\dot{\gamma}_c$). The formulation of the guidance commands is shown in Eqs. (5-7).

$$\dot{\alpha}_c = \frac{-a_h}{V_{UAS}} cos\left(\alpha_{h,12} - \alpha\right) \tag{5}$$

$$\dot{v}_c = -a_h sin\left(\alpha_{h,12} - \alpha\right)cos(\gamma) - a_v sin\left(\alpha_{v,12} - \gamma\right) \tag{6}$$

$$\dot{\gamma}_c = \frac{-a_v}{V_{UAS}} cos\left(\alpha_{v,12} - \gamma\right) - \frac{-a_h}{V_{UAS}} sin\left(\alpha_{h,12} - \alpha\right)sin(\gamma) \tag{7}$$

Subscript c denotes a command, h denotes horizontal plane, and v denotes vertical plane. Subscripts 1 and 2 represent the upper or lower cone bound, respectively. V_{UAS} is the

aircraft velocity. It should be noted that because of the definition of α, the turn rate command shown here is opposite in sign as typical turn rate definitions where heading is defined as positive clockwise (CW) from the y-axis (North).

Alternative commands can be derived from these basic commands depending on the application and autopilot. A set of such commands are shown below in Eqs. (8-10).

$$\dot{\psi}_c = -\dot{\alpha}_c \tag{8}$$

$$V_c = \int \dot{v}_c dt \tag{9}$$

$$\theta_c \approx \int \dot{\gamma}_c dt \tag{10}$$

Integration operators in the equations above are calculated using Euler integration. The symbol ψ represents UAS heading and is defined positive CW from the y-axis. Theta, θ, in this context, represents the UAS pitch angle. This command is generated under the assumption of small angle of attack and a small delay between changes in pitch and changes in flight path angle.

Smith, et al., described an own-ship UAS CA algorithm based on the collision cone approach [27]. This algorithm detected potential collisions with multiple intruders and commanded avoidance maneuvers to the UAS. It was assumed the intruder states were known, either from a communication network, or from on-board sensors. This algorithm is used as the foundation for a cooperative multiple vehicle UAS CA system. Because of the variety of UAS architectures, the cooperative UAS algorithm must be compatible with many systems. For instance, a UAS could consist of multiple decentralized platforms each with a CA system on-board, but with the systems acting in a

synergetic manner. Or, a system could consist of multiple platforms controlled by a centralized ground control station which commands the UAS and carries the burden of all CA processing. Therefore, the own-ship CA algorithm was modified to incorporate cooperative platform inputs internal to the UAS and non-cooperative threat inputs external to the UAS and provided by some sensor suite independently. It does not matter inside the modified collision cone algorithm whether the inputs are from cooperative or non-cooperative entities, but on real systems the means by which to acquire the inputs differs greatly (e.g. from the GCS or from the onboard sensors).

The modified and aggregate collision cone algorithm is extended to cooperative multi-vehicle UAS CA by executing the modified collision cone algorithm independently for each UAS. In each iteration, a different UAS is treated as the own-ship and the other UAS and any other sensed threat is treated as a moving obstacle. Consistent collision detections between conflicting UAS are inherent because the encounter geometry is a mirror image of the other. Commands generated by the proportional navigation guidance can be treated in a coordinated or uncoordinated manner. Coordinated is defined in this research as a synchronization of commands between multiple cooperative UAS reacting to the same collision encounter. Cooperative and coordinated CA is potentially extremely efficient because small maneuvers by two UAS with conflicting flight paths could be more energy efficient than one of the UAS performing an extreme maneuver. However, the coordination of commands becomes exponentially more difficult as the number of UAS increases. The following table, Table 3-3, shows what combinations of cooperative and coordinated CA are used in this application.

Table 3-3: Cooperation/Coordination Matrix

Information	Avoidance Commands	
	Uncoordinated	Coordinated
Non-Cooperative	Horizontal, Vertical	
Cooperative	Horizontal	Vertical

Uncoordinated maneuvers are the only option with non-cooperative traffic, and are required by UAS with onboard sensing and processing of external threats. A cooperative group of UAS can be guided by uncoordinated or coordinated commands. The proportional navigation guidance coupled with the collision cone approach addresses the direction and magnitude of commands; however, conflicting commands are possible for certain encounter geometries. For level, co-altitude flight encounters, flight path commands tend to be the same sign. However, logically, one would want the aircraft to move in opposite directions even if the horizontal commands maintain lateral separation. Thus, the vertical commands can be coordinated in an encounter to direct one aircraft to climb and the other to descend while allowing the horizontal avoidance to operate uncoordinated. Only when the difference in angular heading of approaching aircraft is approaching zero, nearly parallel converging trajectories, do the horizontal guidance commands have difficulty deconflicting the aircraft. In these extreme cases, the coordinated vertical guidance provides the primary separation commands. When three or four aircraft are in a collision encounter, a pair or pairs of aircraft will tend to maneuver in the same vertical direction. These pairs will rely on horizontal guidance commands to achieve separation. Any more than four aircraft will require more sophisticated command

coordination than what is discussed here, such as optimization techniques to reduce conflicting maneuvers.

3. Hardware

The UAS chosen for the CA algorithm integration is the AFIT ANT laboratory's Battlefield Air Targeting Camera Autonomous Micro-air vehicle (BATCAM). BATCAM is a miniature unmanned aircraft manufactured by Applied Research Associates, Inc., and is used by the United States Special Operations Command (SOCOM) for surveillance and reconnaissance missions.

Figure 3-7: BATCAM

BATCAM, shown in Figure 3-7, is categorized by the 2007-2032 UAS roadmap as a "Tactical 1 Special Operations Forces Team Small Unit Company and below" system that is a small, hand-launched, platform with electro-optical/infrared (EO/IR) sensors or communication equipment as the primary payload [1]. BATCAM is also classified by the roadmap as a Level 0 domestic-use UAS which is described as a system under two pounds within line of sight control that operates in unregulated airspace. The

utility of the BATCAM system is outlined in a presentation by then Deputy Assistant Secretary of the Air Force for Science, Technology, and Engineering James Engle to the Senate Armed Services Committee, Subcommittee on Emerging Threats and Capabilities [31]. Mr. Engle says of the BATCAM in its role in the Battlefield Air Operations (BAO) kit, "BATCAM replaces the current UAV system in the BAO kit with one that is five times smaller and ten times lighter, yet still provides covert reconnaissance, is simple to operate, inexpensive enough to be expendable, and can provide real-time battle damage assessment." Table 3-4 lists the key characteristics of the BATCAM platform.

Table 3-4: BATCAM Platform/System Characteristics [32]

	BATCAM
Manufacturer	ARA
User Service	SOCOM
Weight	0.84 lb
Length	24 in
Wingspan	21 in
Payload Capacity	0.09 lb
Engine Type	Battery
Ceiling	11,000 ft [33]
Radius	1.6 nm
Endurance	18 min
Number Planned	23 systems
Number UA/System	2

The EO/IR sensor system on the BATCAM consists of two cameras: one mounted forward-looking and the other mounted side-looking. The two camera angles provide a powerful surveillance/reconnaissance capability allowing forward views in straight-level flight and side views for single-point monitoring and loitering. For a route surveillance mission, long stretches of road can be viewed by a human operator or autonomous target

recognition (ATR) user, and single points of interest can be monitored for the endurance of the platform.

An important component of the BATCAM system is the autopilot onboard which is responsible for autonomously operating the aircraft, receiving and responding to commands from the human controller, and relaying information back to the controller. The autopilot used in the ANT laboratory's BATCAM systems is the Kestrel Autopilot System from Procerus Technologies [34]. The Kestrel autopilot is designed for miniature and micro aerial vehicles.
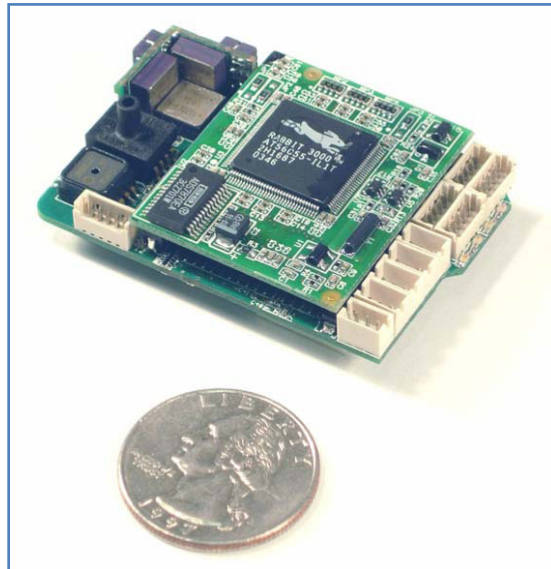

Figure 3-8: Kestrel Autopilot [34]

The Kestrel Autopilot (see Figure 3-8) provides autonomous flight, takeoff, and landing capability via tunable control laws and Global Positioning System (GPS) navigation. The system is complete with wireless communication equipment for uploading/downloading of information to/from an integrated GCS which will be described in detail shortly. The autopilot includes a full sensor suite containing three-axis

accelerometers and gyros for acceleration, attitude and rate information, and a pitot-static system for pressure measurements and calculations. A communication box ground component is the interface between the GCS (installed on a laptop) and the Kestrel autopilot. The Kestrel Autopilot also allows manual control via a radio control (R/C) device. Combined with the live and recordable telemetry capabilities of the autopilot, and an experienced R/C pilot, this system is well-suited for developmental and operational flight tests from safety and technical standpoints.
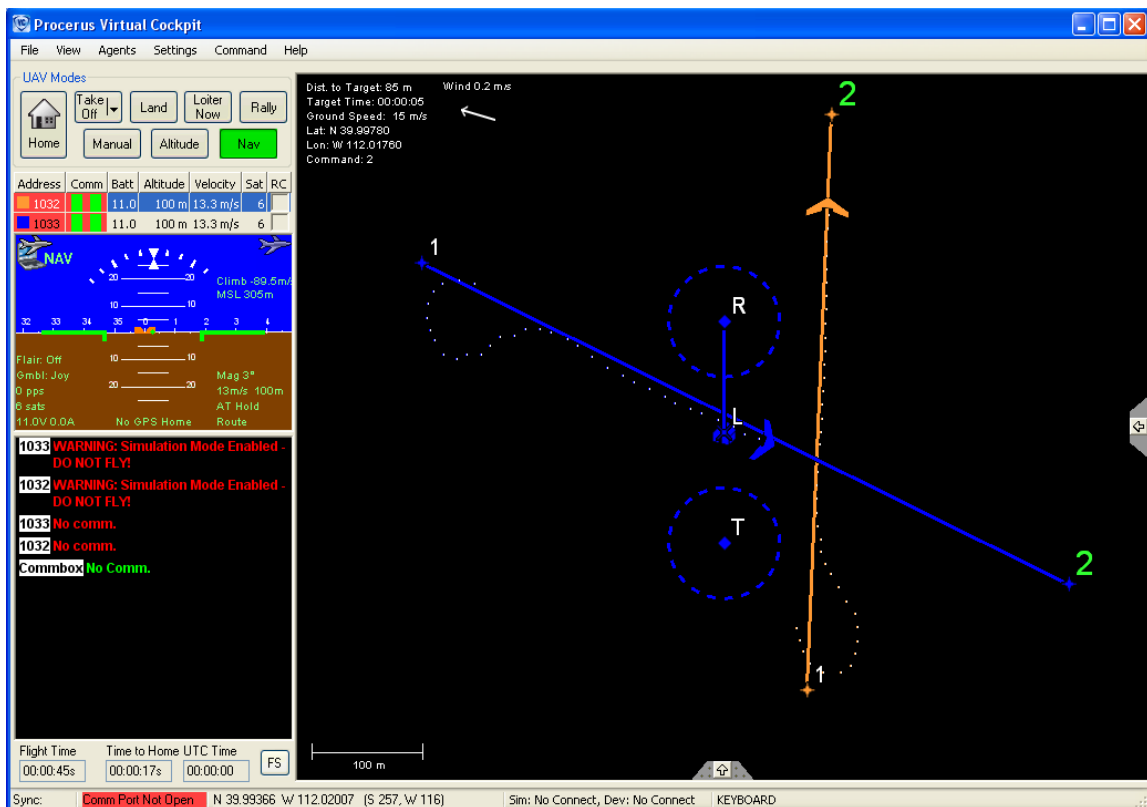

Figure 3-9: Virtual Cockpit

The GCS component of the Kestrel Autopilot system is the Virtual Cockpit Windows-based software system installed on a compatible laptop. Virtual Cockpit, shown in Figure 3-9, enables multiple vehicle UAS monitoring and control, and most

importantly, an external interface for user-developed algorithms. The GCS displays

spatial information of all UAS in a flight plan and map display window, as well as

attitude information in a heads-up-display (HUD) virtual window. Communication

information is displayed for each UAS, and real-time status alerts are provided to the user

with voice announcements. Virtual Cockpit provides in-flight autopilot tuning

capabilities via graphical variable interfaces, and in-flight mode switching between

navigation, manual, altitude, etc., modes.

Table 3-5: Collision Avoidance Algorithm Function Descriptions

Function Name	Description
mult_uas_aa	Performs parsing of cooperative and non-cooperative UAS inputs. Performs calculations and organizational tasks for collision cone avoidance algorithm inputs. Calls collision avoidance algorithm. Processes avoidance algorithm outputs and retains *persistent* variables for next algorithm call. Completes coordination of commands, as necessary. Calculates Kestrel Autopilot commands from avoidance algorithm commands.
cc_pn_aa	Performs calculations for collision cone approach algorithm. Uncouples horizontal and vertical collision cone planes. Calls collision cone algorithm for each threat object/obstacle for horizontal and vertical planes. Processes normalized collision cone approach outputs to absolute representation. Aggregates collision cones based on existing overlap. Determines cone violation via velocity vector comparison. If conflict exists, determines cone angle and rate to use in guidance. Applies proportional navigation guidance to both planes to generate avoidance commands. Retains flags for use in next function call.
f_collisioncone4	Collision cone approach implementation function. Calculates collision cone bound angles and angular rates between own-ship and intruder.
wrap_mpi2pi	Wraps angles from minus pi to pi
wrap_pos	Wraps angles to positive values
wrap_neg	Wraps angles to negative values

4. Implementation

CA algorithm development is completed in the MATLAB environment because of the author's familiarity with the application, the ANT laboratory's organizational preference for its use, its flexibility for algorithm design and testing, and its built-in portability to programming languages such as C and C++. The CA algorithm developed in MATLAB code (m-code) is executed in the MATLAB environment and tested in ideal simulations containing three degree-of-freedom dynamics and perfect tracking of commands. The algorithm consists of six MATLAB functions in separate m-files. Table 3-5 gives descriptions of these functions and Appendix H contains full listings of the MATLAB code.

4.1. MATLAB Algorithm Deployment

Implementation of the collision avoidance algorithm into external applications requires auto-coding of the algorithm into a C shared library and supporting files with MATLAB's Compiler application. The compiler allows many user options; the configuration chosen for this algorithm generates the necessary file types in Table 3-6.

Table 3-6: Necessary MATLAB Compiler Generated Files

File Type	File Extension
Dynamic link library	*.dll
Static library	*.lib
Header file	*.h
CTF file	*.ctf

Additional C files are generated, but are not necessary. The following commands at the MATLAB prompt will auto-code the collision avoidance algorithm and return the files listed in Table 3-6.

    mcc –W lib:CCAA –T link:lib mult_uas_aa cc_pn_aa f_collisioncone4 wrap_mpi2pi wrap_pos wrap_neg

The names of the files are specified by the *lib:CCAA* option. This will name all associated files with different extensions *CCAA.xxx*, where xxx represents one of the extensions in Table 3-6.

4.2. C++ Application and GUI Development

Once the necessary compiler-generated files are available, the collision avoidance algorithm can be deployed to an external application. The Kestrel Developer's Kit, an add-on package available from Procerus Technologies, is the interface to Virtual Cockpit and the Kestrel Autopilot necessary for user-developed applications to communicate with the aircraft. The CA algorithm is integrated with the GCS using this Kit and MATLAB Compiler Runtime (MCR), an application that must be distributed with a MATLAB Compiler-created application. The CA algorithm is linked to Virtual Cockpit through library functions and interfaces in the Developer's Kit. A graphical user interface (GUI) and associated C++ functions were developed and written to process information from Virtual Cockpit for input to the CA algorithm and to process commands from the algorithm and send to the autopilots. Functions contained in the MCR are necessary to create the input and output structure required by the collision avoidance algorithm shared library. Figure

3-10 is the GUI for UAS collision avoidance monitoring and defining user parameters, in addition to other unrelated developmental testing windows.
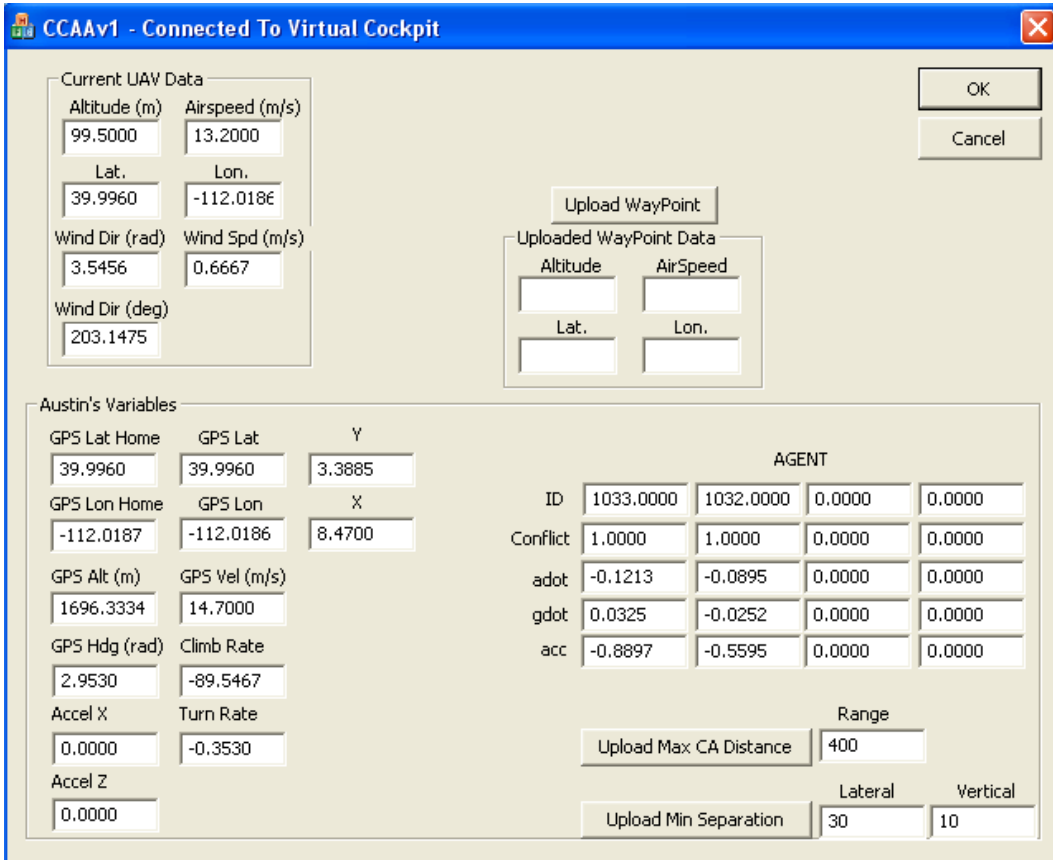

Figure 3-10: Collision Avoidance Application GUI

The collision avoidance GUI displays its Virtual Cockpit status information in the title bar. The lower right quadrant of the GUI is the collision avoidance display and configuration area (CADC). All other displays are for additional variable monitoring. The CADC display matrix shows the user which agents (i.e. UAS) are being processed by the algorithm and are identified by their Agent ID number. Currently, the GUI is compatible with up to four UAS platforms. This corresponds with current operational mission configurations that use four UAS. The next row in the matrix is a

binary flag informing the user whether the respective UAS has a conflict. If a conflict exists, the flag is a one, and the guidance commands from the algorithm appear in the remaining matrix locations. Under the agent matrix are user-defined parameter controls. Default settings appear when the GUI is initiated, but can be changed at runtime by the user and sent to the algorithm by pressing the appropriate button. The user-defined variables are described in Table 3-7.

Table 3-7: User Parameters, CA GUI

Parameter	Description	Default Value	Units
Range	Maximum range to begin executing collision avoidance algorithm	400	m
Lateral	Lateral minimum separation distance	30	m
Vertical	Vertical minimum separation distance (altitude offset)	10	m

The default values were defined iteratively by running several simulations. The default maximum range value allows ample time for the aircraft to align themselves for collision encounters while still challenging the algorithm to provide sufficient avoidance commands. This is a function of typical speeds for the aircraft, and would need to be changed for other platforms. The default separation values were selected to account for uncertainty in aircraft trajectories. They are large enough so that CA will still be activated even if the aircraft are not precisely flying the desired path. The separation values are a function of typical aircraft speeds and accuracy in maintaining desired trajectories and would need to be changed for other aircraft.

47

There exists a tradeoff when integrating this CA algorithm into different UAS platforms. At large ranges relative to the size and speed of a particular aircraft, more uncertainty exists when determining if the aircraft are on a collision course because of noisy measurements and the possibility of maneuvers. This uncertainty can be reduced by waiting until the aircraft are closer together before commanding CA maneuvers. The tradeoff exists between reducing the CA range and the maneuver capability of the aircraft. A sufficient amount of range and synonymously time must be available for the aircraft to maintain separation. Too little range will increase the possibility of violating minimum separation. Too much range increases the false alarm rate.

Novel methods of processing UAS information from multiple platforms and organizing the data for use in the CA algorithm are developed. First, it is necessary to understand how information is received and sent to Virtual Cockpit and the autopilots. Data packets are wirelessly transmitted between Virtual Cockpit and the Kestrel Autopilot. Many types of packets are defined by Procerus and each type contains different data. Therefore, for a particular algorithm, the proper packets that contain data required by the algorithm must be identified and received by the application. For the CA algorithm, two specific packets are required for proper operation, and one particular packet is optional. The packets are also specific to a single platform and contain only its information. Communication between the C++ CA application is configured so packet information is passed directly from Virtual Cockpit to the algorithm for processing. Consequently, packets received by the

application must be sorted for the proper packet type and sorted for their respective platforms. Platforms are identified in Virtual Cockpit by an Agent ID number. The processing scheme for packet data identifies any Agent ID number and determines whether or not a packet has been received from that platform. If not, it is a newly recognized platform and is added to a persistent list of IDs. A flow diagram of this portion of the algorithm is shown in Figure 3-11.
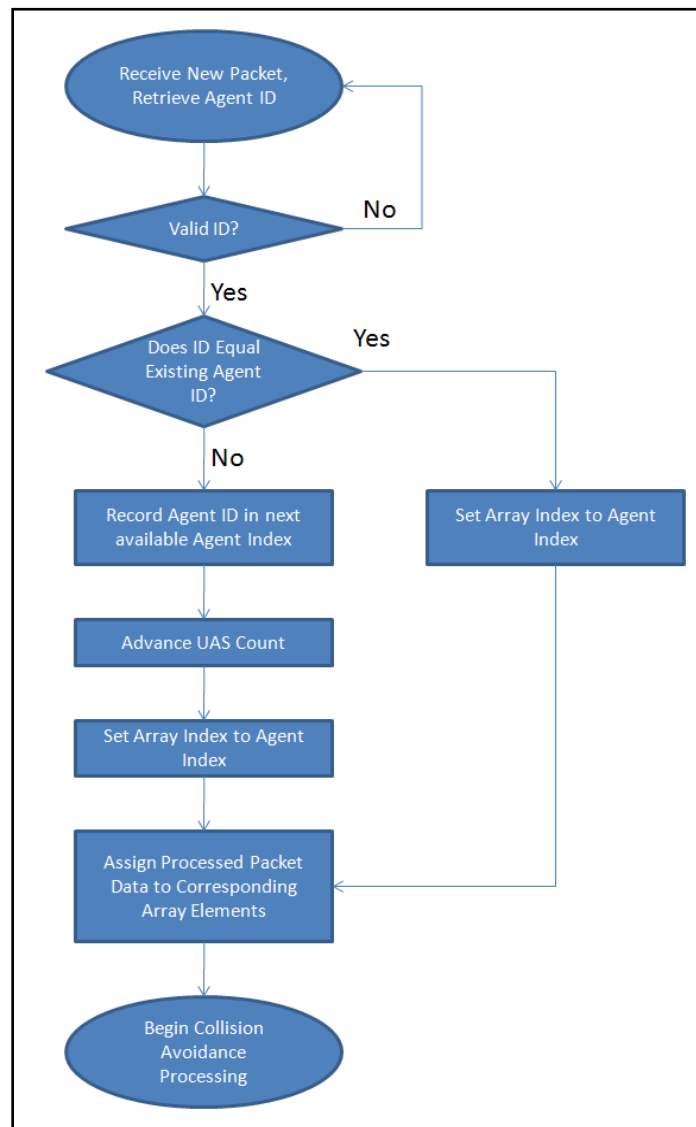

Figure 3-11: Agent ID Processing

Next, the particular packet type is recorded and appropriate data is accessed in that packet. Recorded algorithm input data is updated using the new packet data. Another persistent list is maintained by the processing scheme; it is a list of the packet types that have been received for a particular platform. This is done so the application tracks whether each platform has the minimum amount of information to execute the CA algorithm. If so, the platform is said to be "full-state".

The CA algorithm accepts state data for any number of UAS. However, due to communication limitations, only a single packet for a single UAS is collected at each measurement epoch. Thus, a particular set of UAS state data is added to the input arrays sent to the algorithm if and only if there is full state data for that UAS. That is, the two required packets for that UAS have been received and its respective input array elements have been populated. Regardless of whether the current packet information is used at that particular epoch, the data is used to populate the appropriate array element and saved for the next epoch. If full state data does not exist for any detected UAS or exists for only one UAS, then the CA algorithm is not executed.

As eluded to previously in this section, packet information is sorted according to new and existing Agent IDs, and UAS state information is added to the input arrays as they achieve full state information. Hence, the input and output arrays to and from the CA algorithm are dynamically sized at runtime. This is facilitated by MCR library functions that perform all memory allocation tasks automatically and appropriate array sizing in the MATLAB code before compiling.

4.3. Collision Avoidance Algorithm/Autopilot Interface

In Section 4.2, the direct pass-through of the packet information from the autopilot, through Virtual Cockpit, and finally to the collision avoidance C++ application is discussed. Also, extraction of UAS state information from the packet data is mentioned. The packets are structured so specific data always resides in a packet location. Specific state variables can therefore be withdrawn from the packet in short order. Procerus Technologies provides detailed information on the packet structure of the Kestrel Autopilot system in its Communications Documentation. A detailed interface description specific to this CA algorithm is provided in Appendix A. This description provides packet locations of required UAS states for the CA algorithm, as well as a thorough description of the variables themselves.
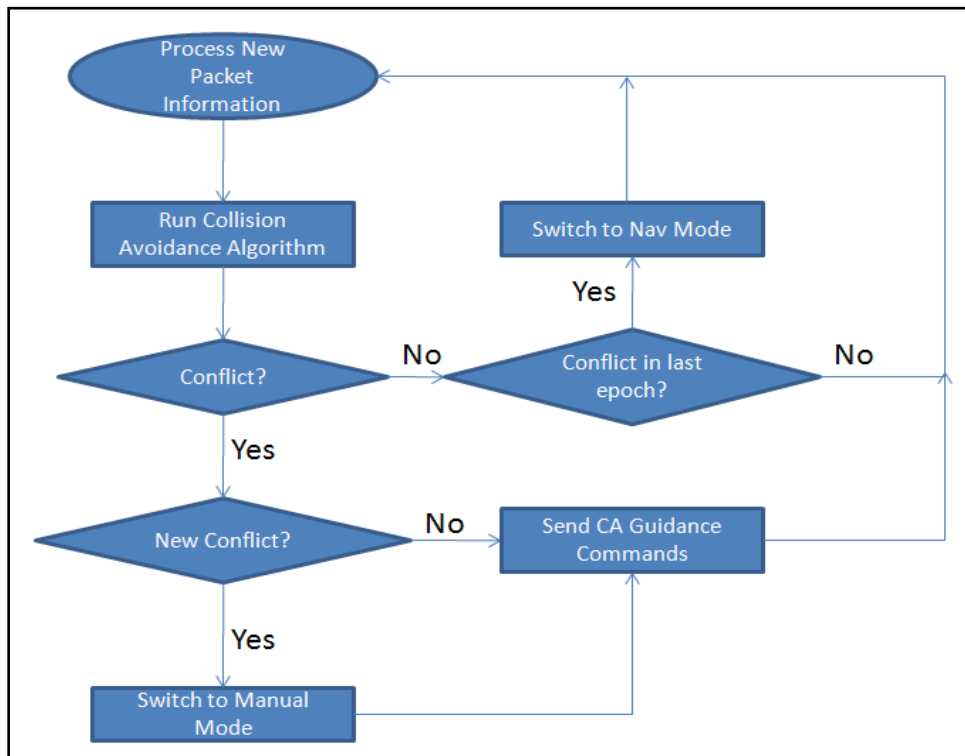
Figure 3-12: Collision Avoidance Command Processing

A similar process to extracting information from received Kestrel Autopilot packets is used to populate transmittable packets with guidance command information from the CA algorithm, if necessary. Figure 3-12 shows a flow diagram of the guidance command processing. This diagram is initiated with new packet processing, described above, that is an extremely non-trivial process.

The CA algorithm is executed and returns binary flags to denote conflicts: zero for no conflict and one for at least one conflict. Conflict flags are returned for each UAS. If a conflict exists, then it is determined whether this is a new conflict or an existing conflict. If new, the UAS is switched to MANUAL mode, and the guidance commands provided by the CA algorithm are sent to the UAS. The mode switch is controlled by a callback function that populates the proper Kestrel autopilot packet with the mode identifier and instructs Virtual Cockpit to transmit the packet to the UAS autopilot. The command transmission is also facilitated by callback functions that populate the proper transmission packet with the command in the appropriate packet location. As with the mode switch, Virtual Cockpit transmits this packet directly to the UAS autopilot. If the current conflict is an existing conflict, then the UAS is already in MANUAL mode, and only commands are transmitted. If no conflict exists, but one did exist for that UAS at the last measurement epoch, then the UAS is switched back to NAV mode. If no conflict existed in the last epoch or exists in the current epoch, then the collision detection algorithm continues to collect new packets and monitors the airspace without issuing commands.

IV. Analysis and Results

1. Chapter Overview

Chapter IV discusses the tests performed on the CA algorithm and the pertinent results obtained from each type of test. Simulation results are discussed in sequential order from the least complexity and uncertainty to the greatest. Simulations include ideal simulations performed in MATLAB, SIL simulations performed in Virtual Cockpit and Aviones, an aerial vehicle simulator, and finally, HIL simulations performed with Virtual Cockpit, Aviones, and the Kestrel Autopilot. Testing is concluded with flight tests performed with BATCAMs, Virtual Cockpit, and all other required hardware.

Each test is conducted with the same scenario sets. The sets are test cases that describe particular encounter geometries. A full spectrum of cases is used to evaluate the robustness of the algorithm. In flight test, the UAS was placed in collision encounters that included altitude separation for safety purposes. The minimum separation volume was defined large enough in flight test to activate collision avoidance even though a near-mid-air close-encounter relative to the size of the aircraft did not occur. Scenarios consist of a span of engagement angles (i.e. the angles at which the nominal trajectories of two aircraft meet). Opposing trajectories, 180° engagement angle, are called Head-on encounters. Trajectories with less than 180° and greater than 90° engagement angles are called Approaching encounters. A 90° engagement angle is an Abeam encounter. And finally, an engagement angle of less than 90° is a Converging encounter. Figure 4-1 depicts these encounters visually.
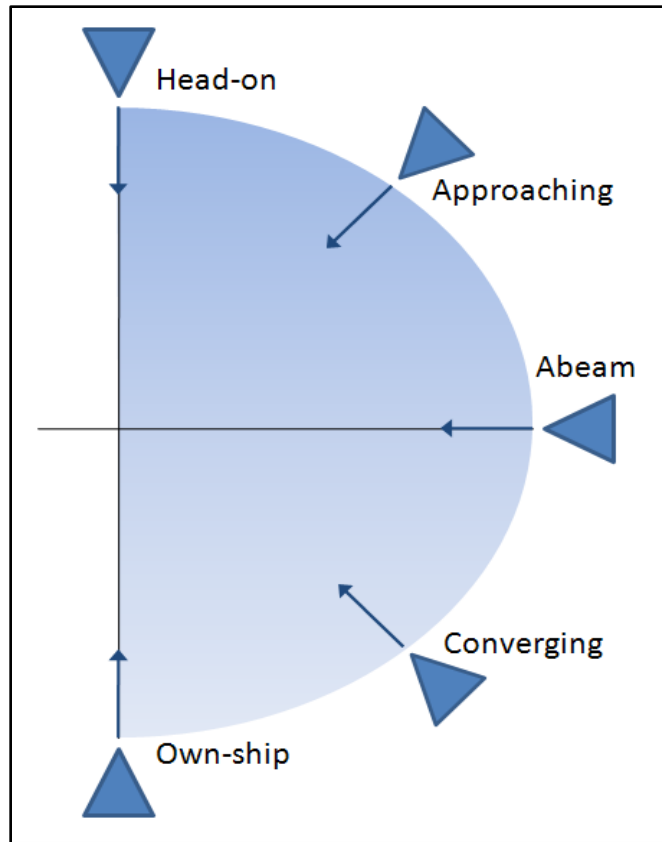
Figure 4-1: Test Case Geometries

Virtual Cockpit's standard telemetry recording feature was utilized for all testing. The recorded telemetry in flight test is data transmitted from the autopilot down to the GCS. This data gives a clear indication of commands and responses as reported by the autopilot, but is bounded by the rate of transmission. No communications between the CA algorithm and Virtual Cockpit are recorded because this data would be uncorrelated in time with actual packets received by the autopilot.

2. Simulation Results

2.1. Ideal

Ideal simulations were performed for a two-ship encounter at each of the engagement geometries discussed above. Three degree-of-freedom dynamics with perfect tracking of commands is assumed. The CA commands essentially control the magnitude and direction of the velocity vector of the aircraft. Reference [28] proves that the guidance law converges for a proportional gain greater than one. For simplicity, the gain was set to the next positive integer value satisfying the convergence property. Hence, the guidance law proportional gain was set to two for all subsequent tests.
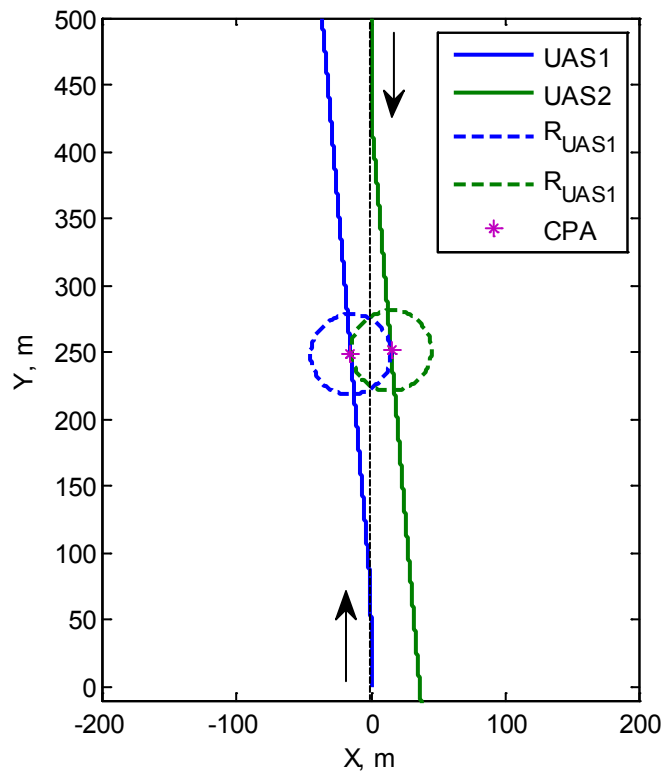
Figure 4-2: Ideal Head-on Simulation Trajectories

The results for the head-on encounter are shown in Figure 4-2. The closest point of approach is denoted as CPA and is shown in the figure as stars on each of the trajectories. Dashed circles represent the horizontal minimum separation at CPA. CPA representations are the same for all subsequent top-view figures for all encounters. The two UAS are initialized on coincident trajectories and flying in opposite directions at a nominal speed of 14 m/sec. This nominal trajectory is shown in the figure as a dashed black line. The UAS begin their maneuvers after the minimum CA range, 400 m, is reached. 400 m is the default value discussed in Chapter III. Avoidance commands last approximately 2.5 sec, 4 to 6.5 sec in the simulation, and CPA occurs at 19.5 sec into the simulation. The maneuvers follow the commands shown in Figure 4-3 generated by the proportional navigation guidance.
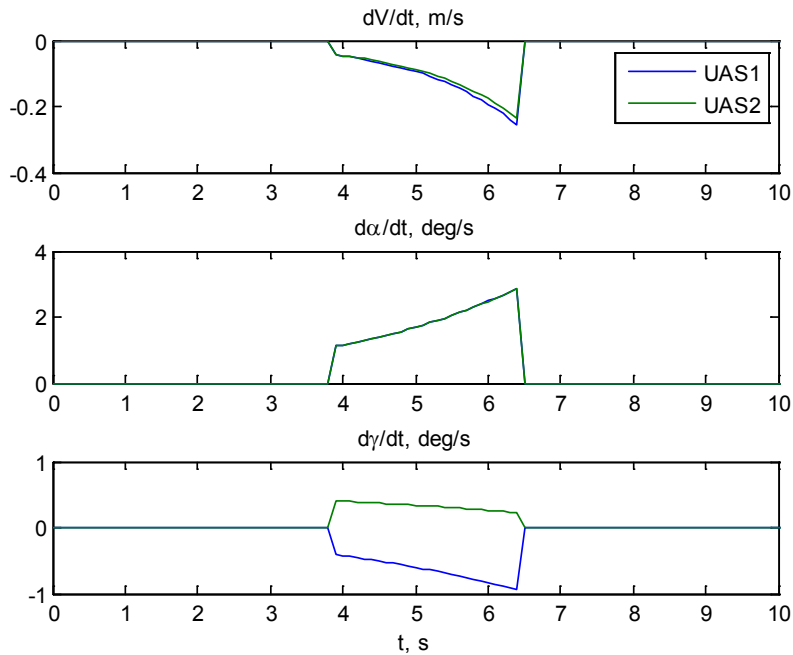
Figure 4-3: Ideal Head-on Simulation Avoidance Algorithm Commands

Attention should be paid to the intuitiveness of the commands generated by the CA guidance. The turn rate commands, as expected, are symmetric and have the same sign because each aircraft's approach to the collision point is a mirror image of the other. The coordinated flight path rate commands are opposite in sign, but not symmetric. This is due to the cone bounds used in the guidance for a particular UAS are rotating at different rates as the aircraft climb and dive in relation to each other. The translational acceleration commands are the remaining components of the proportional navigation acceleration command ensuring it is perpendicular to the appropriate cone bound. It is clear that the UAS are within CA range at approximately four seconds into the simulation.

The ideal algorithm commands must be transformed for compatibility with the Kestrel autopilot. The transformed commands are also calculated in the ideal simulations. These commands for the head-on case are shown in Figure 4-4. The Kestrel autopilot commands are speed (V), integrated from the translational acceleration command, turn rate ($d\psi/dt$), which is the negative of the rate of change of alpha, and pitch angle (θ), which is assumed to equal flight path angle, and is integrated from flight path rate.

The aircraft slant range is plotted with the minimum CA range and the minimum separation distance in Figure 4-5. The avoidance maneuver commands start at approximately four seconds into the simulation. This corresponds to the maximum range allowed for CA (400 m). Also, CPA corresponds to near-tangency of the range curve to the minimum limit at 19.5 sec.
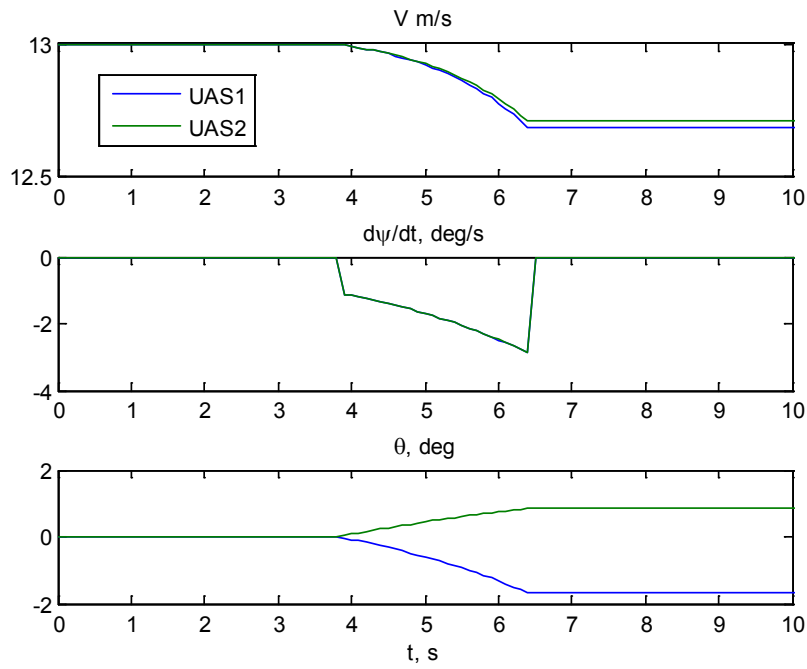
57

Figure 4-4: Ideal Head-on Simulation Kestrel Autopilot Commands
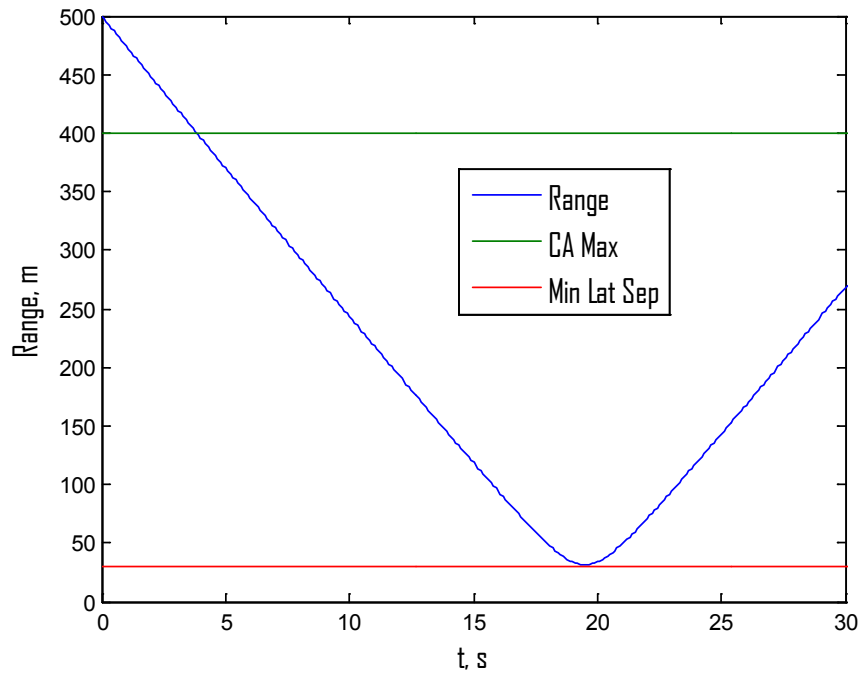

Figure 4-5: Ideal Head-on Simulation Range

The altitude of each aircraft is shown in Figure 4-6 along with the vertical separation, VO (Vertical Offset), dashed lines. It is clear in Figure 4-2 that the aircraft maintain horizontal separation but are also vertically separated now by almost eight meters. This is because both horizontal and vertical channels continue commanding until one achieves its perceived minimum separation. The separation is perceived because the aircraft may not follow their nominal trajectory after the avoidance maneuvers due to navigation commands or other obstacles to avoid. Figures for the other two-ship encounters, Approaching, Abeam, and Converging, are in Appendix C.
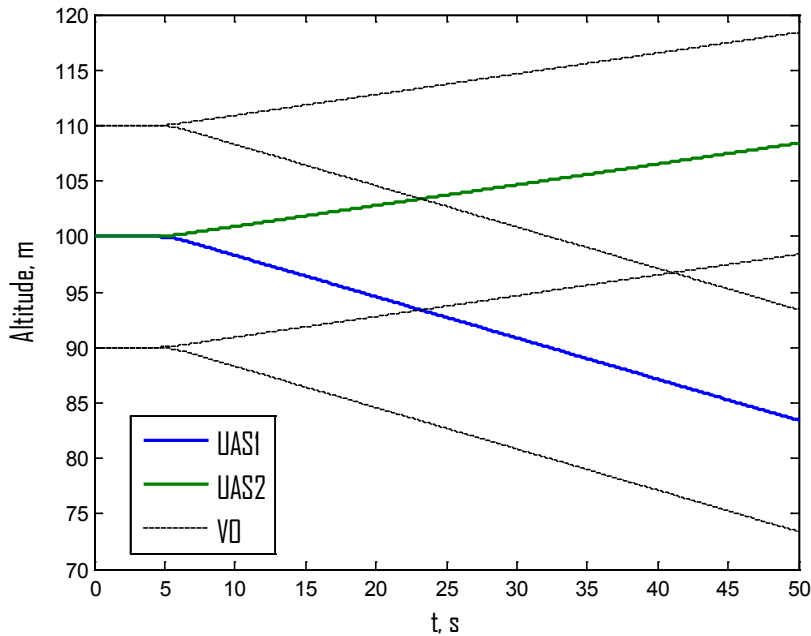
Figure 4-6: Ideal Head-on Simulation Altitude

Three-ship encounters were also completed with ideal simulations. One stressing case for collision avoidance is three UAS approaching at equal engagement angles, 120 deg, to the same point in space. Each UAS must react in a manner to

avoid both of the other UAS approaching the point. Without coordination of commands in the horizontal plane, the geometry of the encounter and the guidance equations command coherent maneuvers reflecting the symmetry of the encounter. Figure 4-7 shows the trajectories of this three-ship encounter.
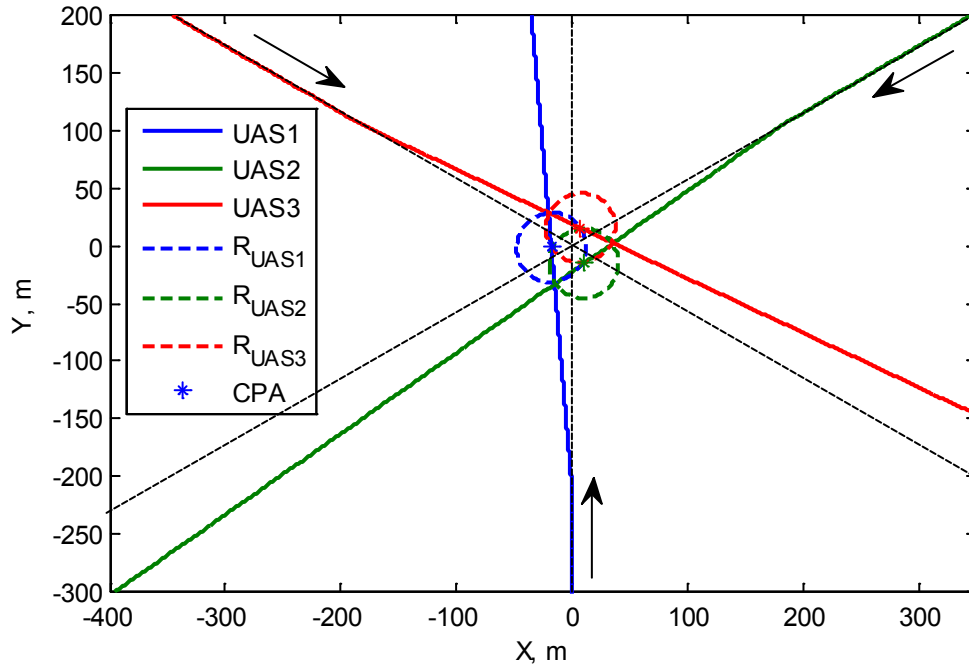

Figure 4-7: Ideal Three-Ship Simulation Trajectories

It is clear from the trajectories and the minimum separation areas shown with dashed circles that the UAS maintain their lateral separation because of symmetric maneuvers which were not coordinated by any hard-coded logic, but are inherent in the guidance law. CPA, for all UAS combinations, occurs at 31.1 sec into the simulation. Commands last approximately three seconds from 13 to 16 sec into the simulation. Figures 4-8 and 4-9 show the algorithm and Kestrel commands, respectively.
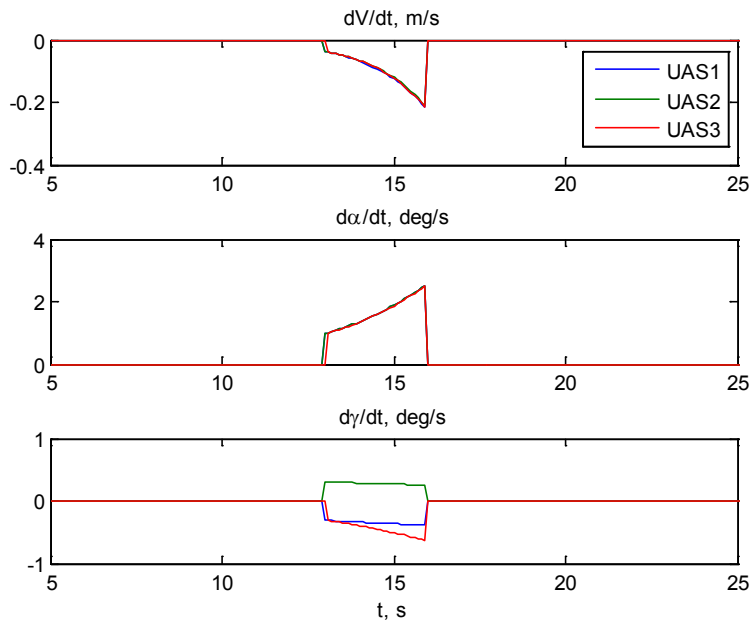
Figure 4-8: Ideal Three-Ship Simulation Avoidance Algorithm Commands
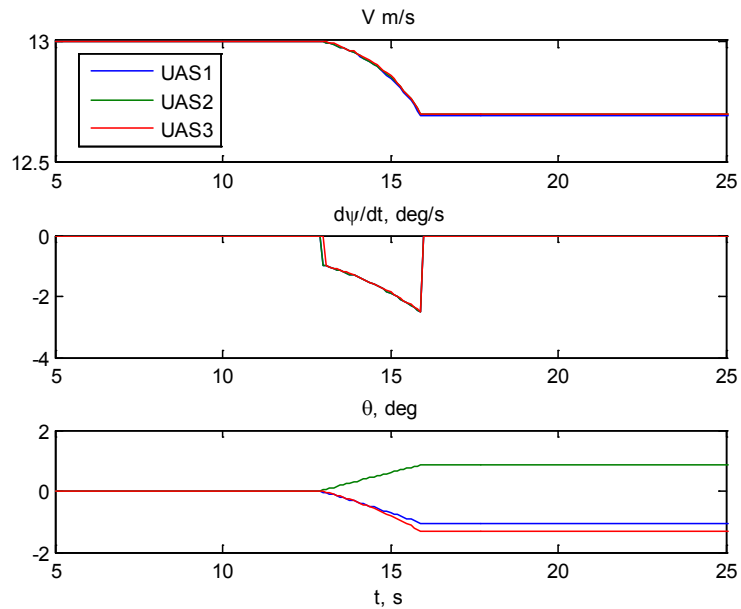

Figure 4-9: Ideal Three-Ship Simulation Kestrel Commands

The UAS, spaced 120 deg apart, are all equidistant from the collision point at initialization. They maintain this angular and range spacing throughout the encounter.

Distinct ranges (see Figure 4-10) between UAS are shown as 'R 1-2', range between

UAS 1 and UAS 2, 'R 1-3', range between UAS 1 and UAS 3, and 'R 2-3', range

between UAS 2 and UAS 3. All UAS maintain lateral separation with each other and
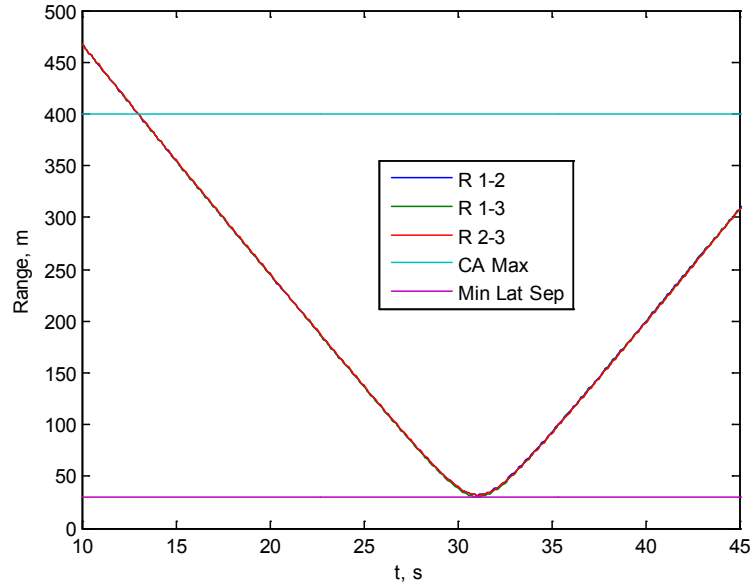
each encounter is temporally symmetric.

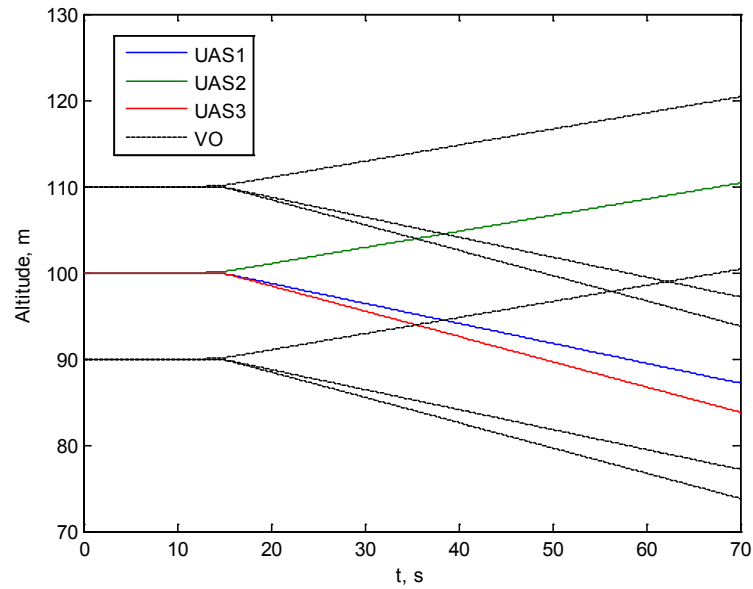**Figure 4-10: Ideal Three-Ship Simulation Range**

Figure 4-11: Ideal Three-Ship Simulation Altitude

Altitude (see Figure 4-11) displays the same pattern as the two-ship encounters except for one important feature. Now that there are three UAS, two will adjust their flight path in the same vertical direction. The burden of maintaining separation for these two aircraft is now solely placed on the horizontal maneuvers which were successful.

2.2. Software-in-the-Loop

SIL simulations were configured and executed on a single laptop in Virtual Cockpit's loop-back mode. Aviones was configured so that it used its default aircraft dynamics model and was started first. Once Aviones is running, Virtual Cockpit can be opened and Agents can be added with distinct Agent IDs and set to operate in SIL mode. Once added, these UAS and their respective Agent IDs appear in Aviones. The Aviones graphics screen appears as in Figure 4-12.
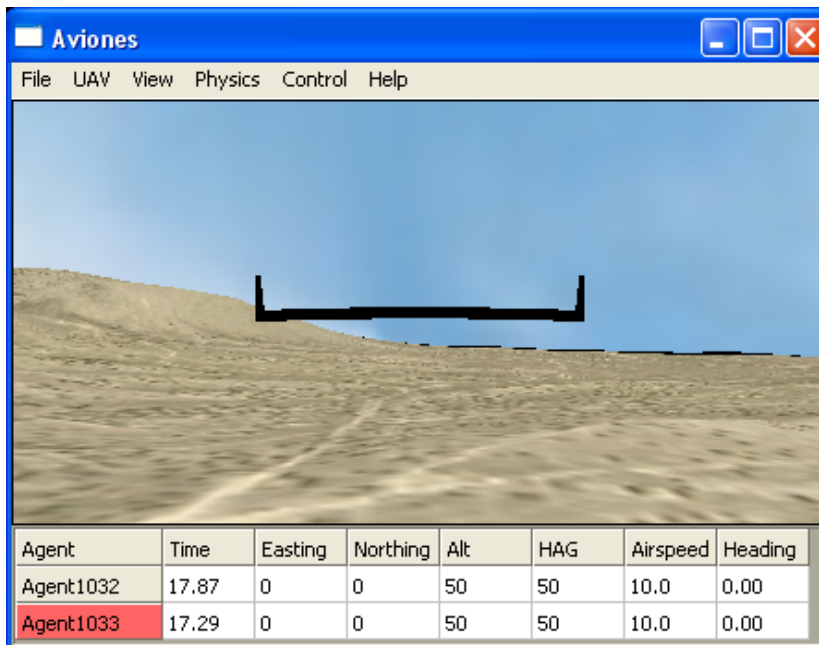

Figure 4-12: Aviones with Agents

Virtual Cockpit's GUI is designed for interactive mission set-up, both before

UAS launch and after. Waypoints are added to a particular scenario so that the

encounter will be at the appropriate engagement angle for each test case. This does

not, unfortunately, guarantee that the UAS will always encounter each other in the

desired geometry and location. Significant amounts of re-routing were necessary after

waypoint following in navigation mode was initiated to align the aircraft properly.

Aircraft dynamics and autopilot responses were the causes of configuring the

encounters manually.

Table 4-1: Manual Mode Collision Avoidance Autopilot Settings

Name	Setting
Level 1 Loops	
Roll	Checked
Roll Rate	Checked
Pitch	Checked
Pitch Rate	Checked
Yaw Rate	Unchecked
Throttle	"Airspeed"
Level 2 Loops	
Pitch Dynamic Input	"Fixed"

Waypoint following is achieved in navigation (NAV) mode for all UAS.

Collision avoidance operates in manual (MAN) mode when commanding maneuvers

and returns control of the UAS back to NAV mode once the collision threat has been

abated. Modes were configured in Virtual Cockpit for each control loop in the

autopilot. For instance, in MAN mode, a user can select the pitch loop to control

either airspeed or altitude. These settings must be configured appropriately for all

UAS autopilots to obey CA commands properly. The MAN mode configuration for

CA in SIL will also apply to HIL and flight test operations. Proper CA MAN settings are shown in Table 4-1.

To access these settings in Virtual Cockpit, follow the path below starting at the main window menu '*Settings->Autopilot Config*'. The 'Global Settings' window will appear and will reflect the currently selected Agent autopilot. The following steps must be completed for each autopilot.

1) Expand Mode Configuration

2) Expand Manual Mode

3) Expand both PID Loops lists (Level 1 and Level 2 Loops)

4) Set variables according to Table 4-1

5) Press Upload Config

6) Press Update Flash

7) Repeat Steps 1-6 for each UAS

The MAN mode autopilot settings for CA ensure that the autopilot control loops do not generate conflicting commands. The Level 2 loop setting ensures the aircraft will follow the CA Kestrel pitch command. Level 1 loop settings ensure that the throttle controls airspeed in accordance with the CA Kestrel airspeed command. A combination of the other Level 1 loops controls turn rate according to the CA Kestrel turn rate command.

Once Aviones and Virtual Cockpit are initialized and properly configured, the aircraft are launched. It takes combinations of active waypoint switching and timing to get them out of phase by 180 deg. Once they are in opposing route locations, the

CA GUI is executed and the CA algorithm immediately begins communicating with virtual cockpit, receiving packets, and monitoring the UAS. The GUI is monitored as the UAS cross the 400 m range limit, as the CA algorithm detects the collision, and as it switches the UAS to MAN mode from NAV mode. Immediately after the switch, commands populate the Agent matrix in the CA GUI. Mode switches are visually distinguishable in Virtual Cockpit because the current mode button is highlighted in green and all others are grey.
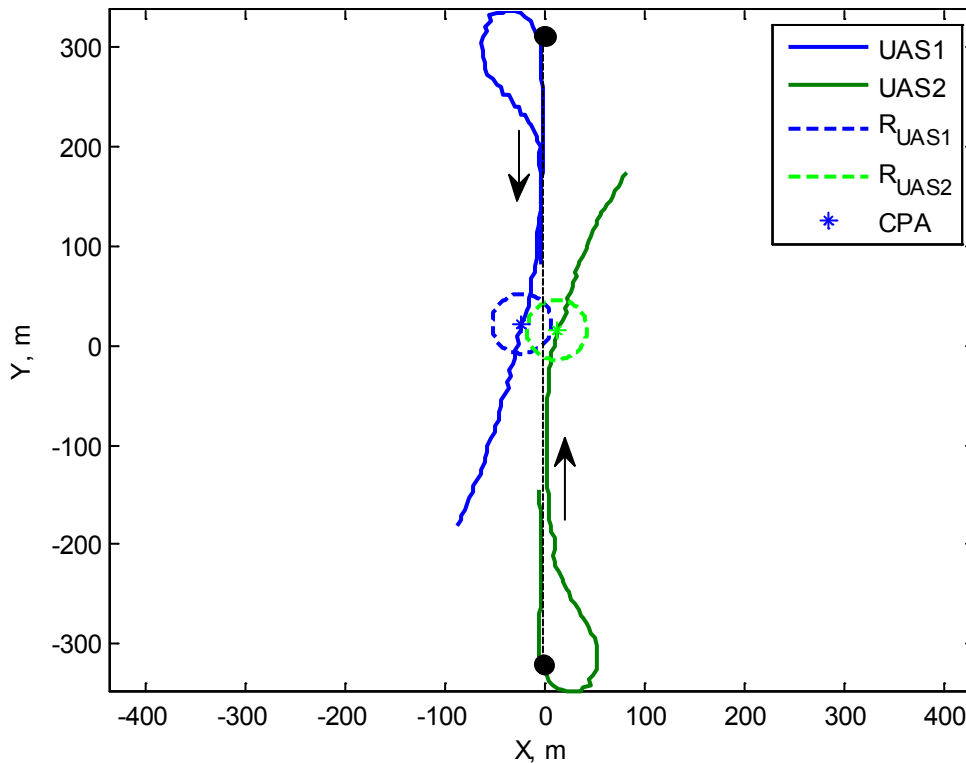
Figure 4-13: SIL Head-on Simulation Trajectories

SIL test cases are completed in the same manner as the ideal simulations. Actually flying out the encounters, mentioned earlier, requires user interaction to properly align the aircraft for the encounter. For the Head-on case, both UAS are

66

commanded to fly back and forth between the same waypoints and along the same path, but 180 deg out of phase. This pattern will result in a collision encounter at the center point along the route. The waypoints are spaced enough to allow the aircraft to align themselves along the straight route after their turn around the waypoint. The maximum CA range and separation volumes are identical to those in the ideal simulations. Figure 4-13 shows the resultant trajectories for the SIL Head-on test case. The UAS successfully maintained lateral separation and obeyed the CA commands.
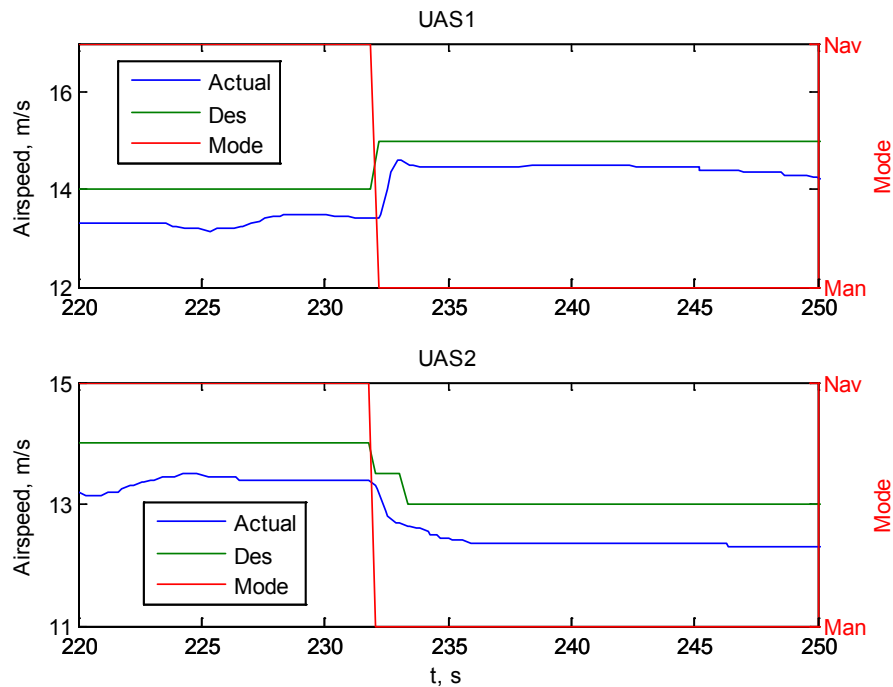
Figure 4-14: SIL Head-on Simulation Airspeed Response

Figures 4-14 through 4-16 show the CA commands for each UAS along with the autopilot mode. The mode crossing from NAV to MAN illustrates the collision detection and initiation of avoidance commands being sent to the autopilot.

Commands are initiated when the mode is switched at 232.17 sec into the simulation, and CPA occurs at 242.05 sec. Using collision timelines like this (10 sec elapsed from detection to CPA), CA algorithms can be tailored to particular aircraft and particular encounter situations in future research.
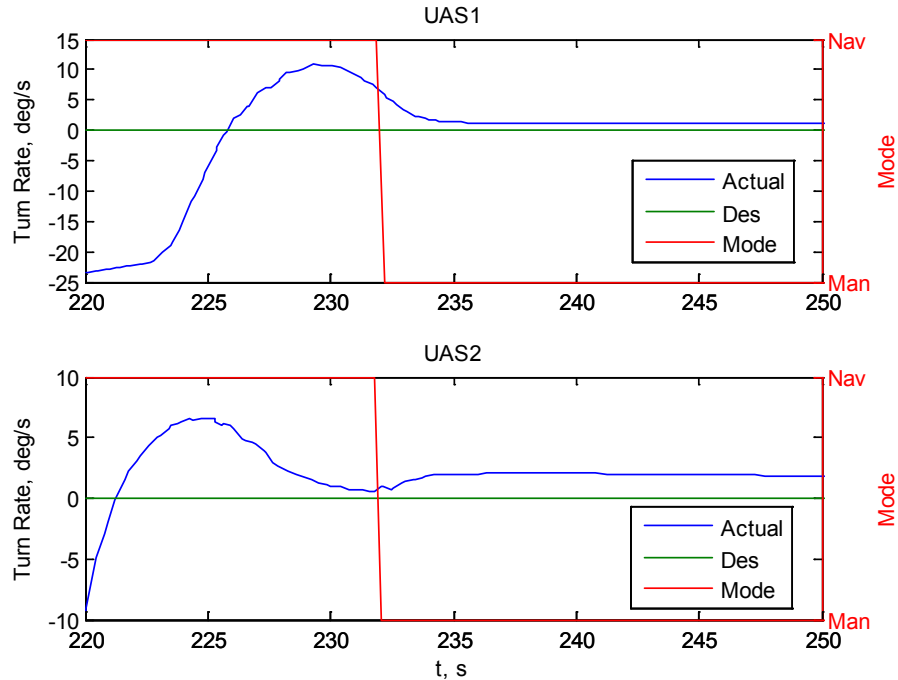

Figure 4-15: SIL Head-on Simulation Turn Rate Response[1]

The airspeed commands and response demonstrate acceptable command change tracking, but there exists a steady-state error throughout the simulation. This is not a result of the CA algorithm because it also exists in NAV mode. The error is caused by control loop settings in each autopilot's proportional-integral-derivative (PID) controller, and could be reduced by additional tuning of each PID controller for

---

[1] 'Des' signal not recorded and is all zeroes

this particular aircraft model. In flight test, PID settings must be tuned for each
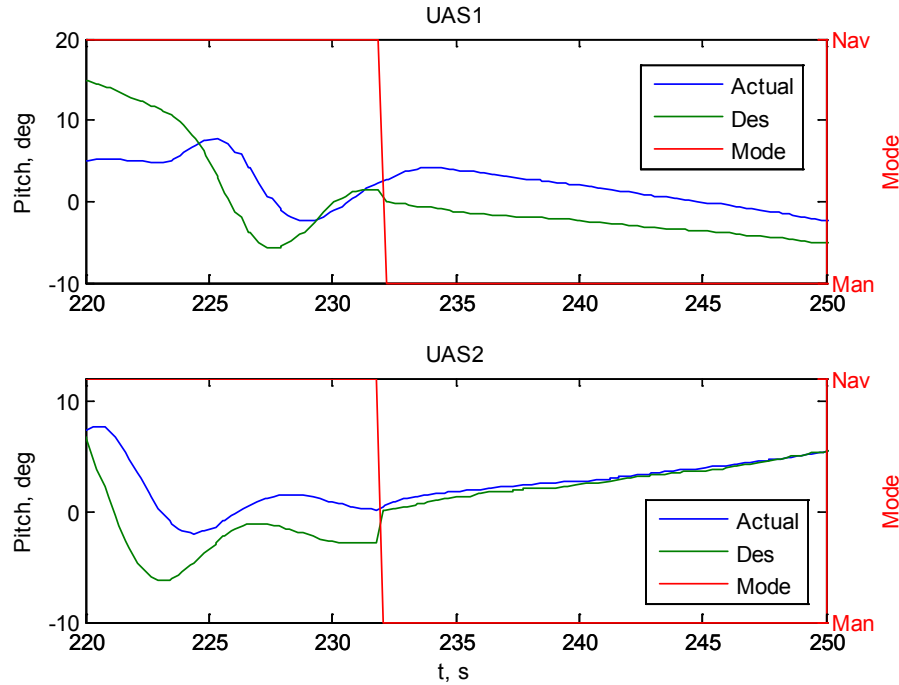
aircraft separately.

Figure 4-16: SIL Head-on Simulation Pitch Response

Unfortunately, the turn rate command is unable to be recorded through Virtual

Cockpit. This is a limitation of the software and firmware on the autopilot, and is

acknowledged by Procerus, the developer of Virtual Cockpit and the Kestrel

autopilot. Thus, command tracking comparisons for turn rate cannot be made, but it is

apparent that commands sufficient to provide lateral separation are being generated

and followed by the UAS.

UAS 2 tracks the pitch command very well, with little delay and effectively

zero steady-state error. UAS 1, however, is slow to respond to the command and

appears to be rate limited as it slowly converges to the command. It is more likely

that this is actually another steady-state error. The effect of pitch response can clearly

be seen in the altitude plot and altitude separation is minimal at the end of the
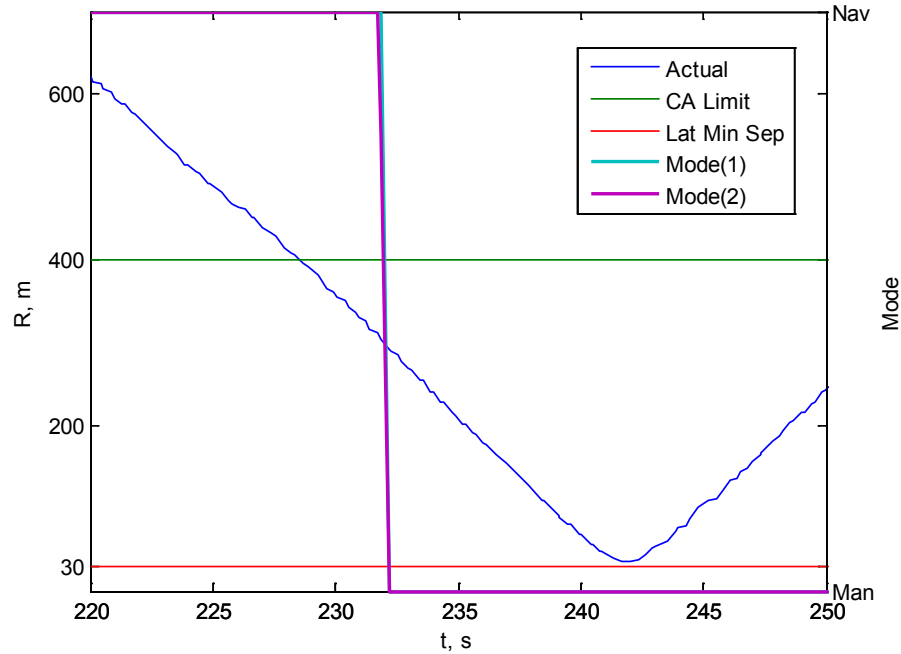
collision encounter.

Figure 4-17: SIL Head-on Simulation Range

Range is shown in Figure 4-17. It is interesting that a collision is not detected

and the mode is not changed until approximately four seconds after the 400 m range

is crossed. The aircraft are still finishing their convergence onto the route line and are

turning into the collision path after the 400 m mark was reached. Once their paths

violated the minimum separation, the mode switch and CA were initiated.

The UAS do not achieve altitude separation because of the slow response of

the UAS 1 pitch (see Figure 4-18). Nevertheless, they do achieve horizontal

separation, and the altitude, for this one test of one case, is irrelevant. The reader

should notice that the aircraft dynamics and communication effects, discussed below,

70

caused UAS 1 to be above UAS 2 at CPA (242.05 sec) even though UAS 1 was commanded to pitch down and UAS 2 was commanded to pitch up. These uncertainties are inevitable, and must be taken into account by the robustness of the algorithm (i.e. multiple dimensions for separation and sufficiently large separation volume definitions).
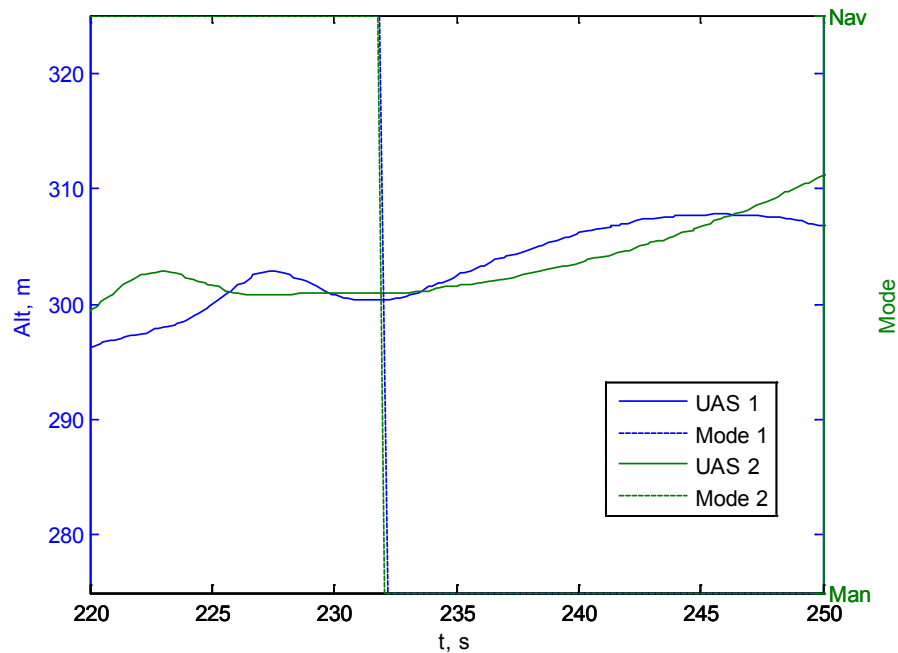

Figure 4-18: SIL Head-on Simulation Altitude

An issue attributed to communication between Virtual Cockpit and the autopilot has potential negative impacts on CA and mission execution for this specific implementation. After CA, the Mode is switched back to NAV mode from MAN mode. There exists a delay in simulation of this mode switch and could indicate a delay in the receipt of avoidance commands. This is probably due to a build-up of avoidance commands in a communication "buffer" that processes and sends commands to the autopilots at a slower rate than they are being generated. Potential

repercussions are increased collision likelihood because of delayed responses and poor mission performance because of large deviations from the nominal trajectory. This is a system-dependent issue that is not an indication of the CA algorithm's performance. Fortunately, the Mode switch from MAN to NAV at the beginning of the encounter happens immediately, and commands are received by the autopilots that successfully deconflict the aircraft.

Table 4-2: SIL Simulation Results, Key Statistics

Parameter	Value			
	Head-on	Approach	Abeam	Converge
Time of Minimum Lateral Separation	242.05 s	462.44 s	121.15 s	153.47 s
Minimum Lateral Separation	37.5 m	49.26 s	45.18 m	35.41 m
Time of Minimum Slant Range	242.05 s	462.44 s	121.15 s	152.67 s
Minimum Slant Range	35.83 m	49.69 m	45.73 m	37.05 m
UAS1 Alt at $T_{min,LS}$	307 m	304 m	297.33 m	306 m
UAS2 Alt at $T_{min,LS}$	304.5 m	297.17 m	304.17 m	293.83 m
UAS1 Alt at $T_{min,SR}$	307 m	304 m	297.33 m	305.33 m
UAS2 Alt at $T_{min,SR}$	304.5 m	297.17 m	304.17 m	295 m

Figures for the other two-ship encounters, Approaching, Abeam, and Converging, are in Appendix D. Key statistics of each two-ship encounter are shown in Table 4-2. Lateral separation was achieved in each test case. Although altitude separation was not necessary due to the lateral separation, the converging case did

achieve full vertical separation. For the other cases, pitch changes were commanded by the CA system but the dynamic lag between pitch changes and altitude changes was too slow. Airspeed commands frequently exhibited oscillating patterns, but it is assessed that these are caused by autopilot attempts to overwrite the CA command. If this was caused by the avoidance algorithm, either the turn rate or pitch command and response would exhibit the same type of pattern, because each of the three independent commands are components of a single change in the velocity vector command. The autopilot may overwrite commands depending on control loop settings beyond what a user can adjust (see Table 4-1).

A three-ship encounter was tested in SIL for an encounter identical to the ideal three-ship simulation. All three UAS have even angular spacing and are intended to collide near the origin of the local reference frame. Uncertainties, such as aircraft dynamics, autopilot command tracking, and communication delays, will inevitably cause differences in the responses, but it is important to demonstrate functional determinism (the ability to repeat results within bounds given uncertainty in the system). By quantitatively repeating ideal response patterns in the SIL simulations within some margin of error, one can transition to higher levels of complexity with confidence.

Figure 4-19 shows the results of the three-ship SIL simulation. Results are very similar to ideal simulations, except for UAS 3 turning in the opposite direction. This is caused by non-ideal geometry and dynamics-effects at the beginning of the encounter. The discrepant turn direction does not negatively affect the outcome of the

encounter, and all aircraft avoid each other in a similar manner to the ideal
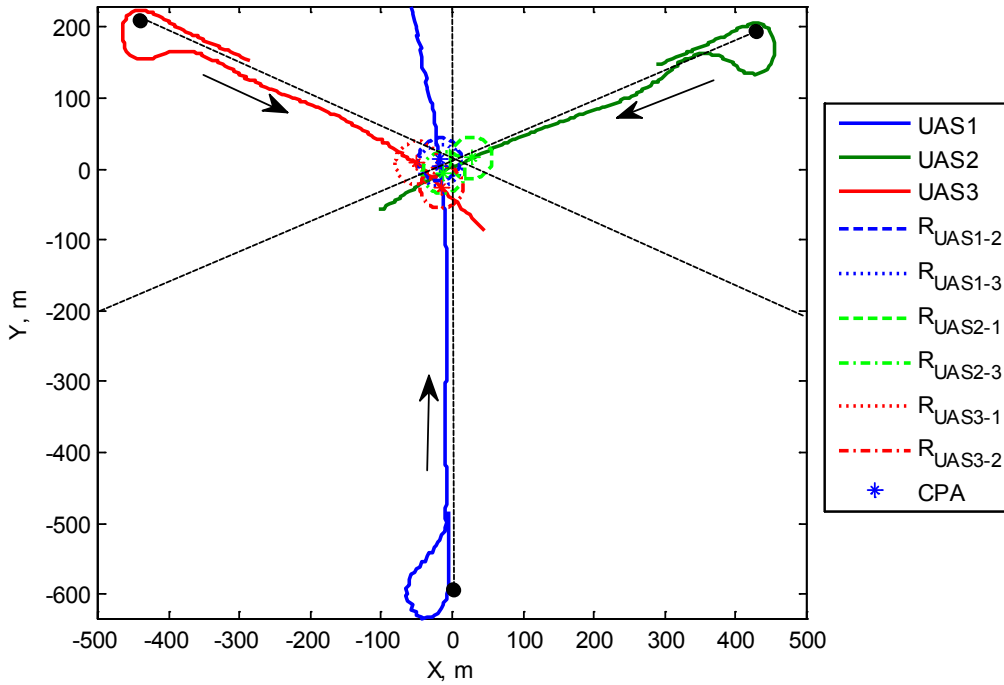
simulations.

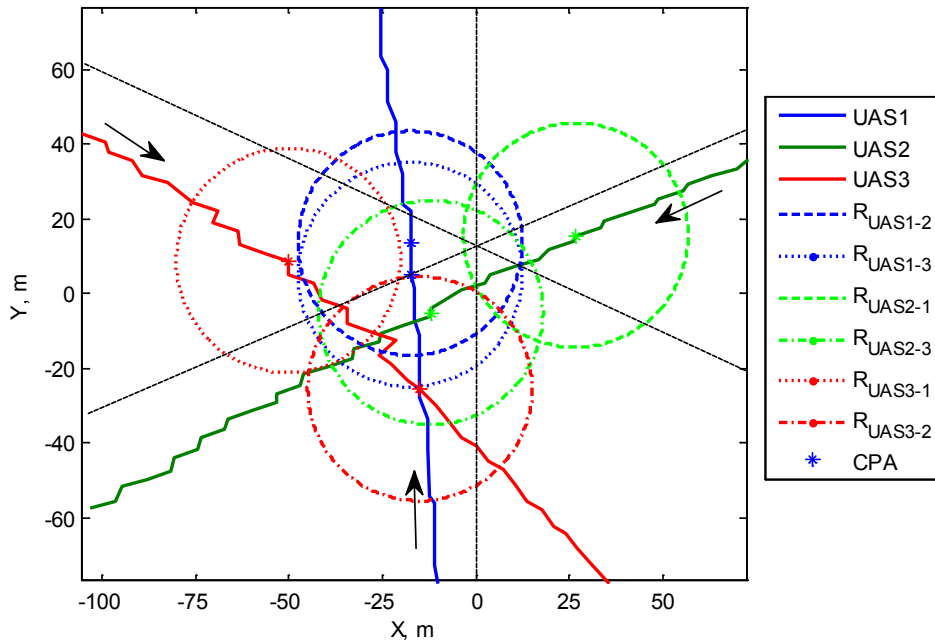**Figure 4-19: SIL Three-Ship Simulation Trajectories**

Figure 4-20: SIL Three-Ship Simulation Trajectories, Zoomed to Origin

74

Figure 4-20 is a close-up view of the locations of the CPA. The lateral

minimum separation areas are distinguishable by two factors. The line color of each

circle coincides with the color of each UAS's trajectory. The line type represents each

distinct CPA for each UAS combination. For instance, CPA between UAS 1 to 2 is

identical to CPA between UAS 2 to 1, and only one is shown. Key parameters of the

encounter are shown in Table 4-3.

Table 4-3: SIL Three-ship Simulation Results, Key Statistics

Parameter	Value		
	1-2 (2-1)	1-3 (3-1)	2-3 (3-2)[2]
Time of Minimum Lateral Separation	176.18 s	176.16 s	187.46 s
Minimum Lateral Separation	44.37 m	32.84 m	20.59 m
Time of Minimum Slant Range	176.18 s	176.16 s	187.46 s
Minimum Slant Range	44.54 m	34.09 m	22.03 m
UAS1 Alt at $T_{min,LS}$	308.17 m	307.83 m	
UAS2 Alt at $T_{min,LS}$	304 m		304.83 m
UAS3 Alt at $T_{min,LS}$		299.33 m	298.33 m
UAS1 Alt at $T_{min,SR}$	308.17 m	307.83 m	
UAS2 Alt at $T_{min,SR}$	304 m		304.83 m
UAS3 Alt at $T_{min,SR}$		299.33 m	298.33 m

---

[2] Vertical separation was likely achieved, but not shown in data because of irregular measurements. Refer to discussion following chart for further information.

Unlike the ideal simulations where separation was maintained for every encounter, UAS 2 and 3 violate their lateral minimum separation criteria. This is due to communication delays causing avoidance commands to be executed seconds after they are generated by the avoidance algorithm. Although they do not maintain the 30 m separation, the UAS are still separated laterally by over 20 m, and are reported to have 6.5 m of altitude separation. Thus, uncertainties have caused suspected loss of separation even in SIL simulations. Furthermore, measurement uncertainty can skew the results because measurements are taken at discrete time intervals and must be sufficiently small to recreate the close-encounter spatially and temporally. Thirty meters was used as the minimum lateral separation for subsequent tests, but further studies in the future should concentrate on defining separation volumes with statistical encounter descriptions from Monte Carlo simulations.

The guidance commands are shown in Figures 4-21 through 4-23. Airspeed commands exhibit the same steady-state error as in the two-ship simulations. Turn rate commands are acceptable in pattern and magnitude, but because of the deficiency in Virtual Cockpit data recording, a definitive comparison of command tracking cannot be made. The pitch commands have poor tracking which causes difficulty in maintaining altitude separation at the CPA.
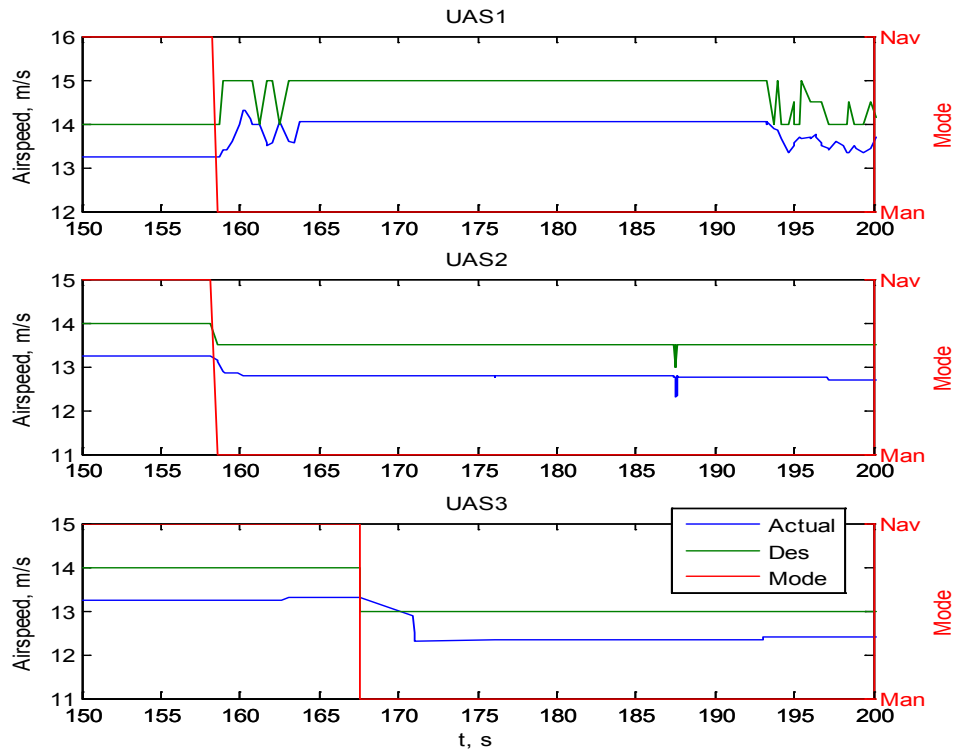
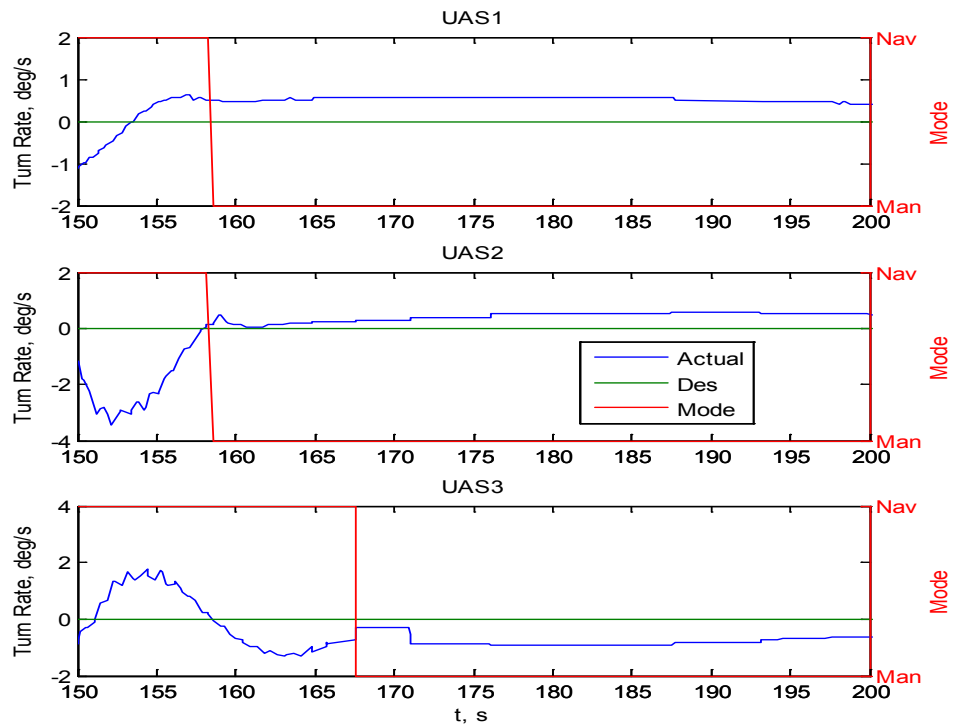**Figure 4-21: SIL Three-Ship Simulation Airspeed Avoidance Command**

Figure 4-22: SIL Three-Ship Simulation Turn Rate Avoidance Command
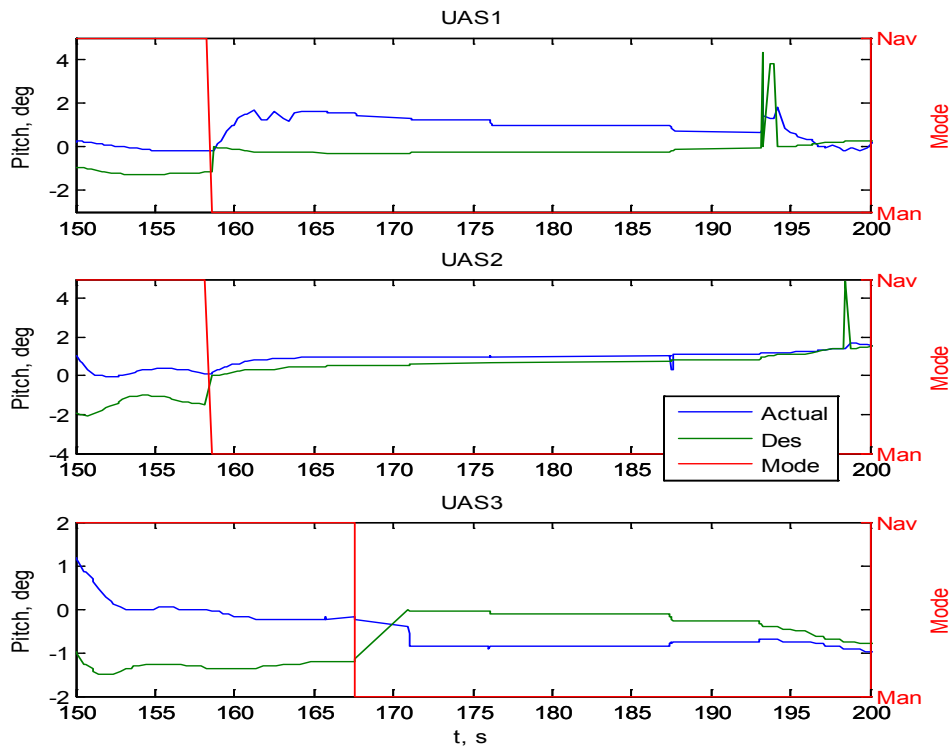
77

Figure 4-23: SIL Three-Ship Simulation Pitch Avoidance Command

Recorded range data, Figure 4-24, shows evidence of data transmission delays, another communication issue. Discrete measurements are recorded at large time steps during the period of CA. As a result, CPA estimates may not be accurate due to sporadic range measurements. Altitude measurements, Figure 4-25, show evidence of underestimation of the actual altitude separation due to irregular measurement intervals. By interpolating between altitude measurements at the time of minimum separation (equivalent for minimum lateral and slant range separation), 187.46 sec into the simulation, it is shown that the UAS were separated by as much as six meters more than the recorded value. This equates to an actual separation of nearly 12.5 m which meets the vertical separation criteria.
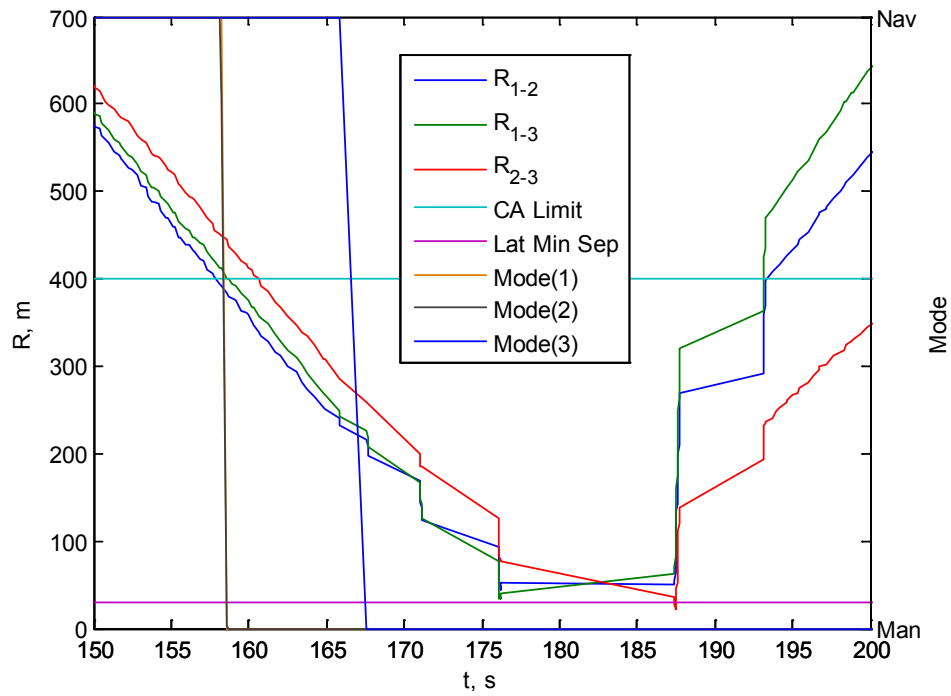
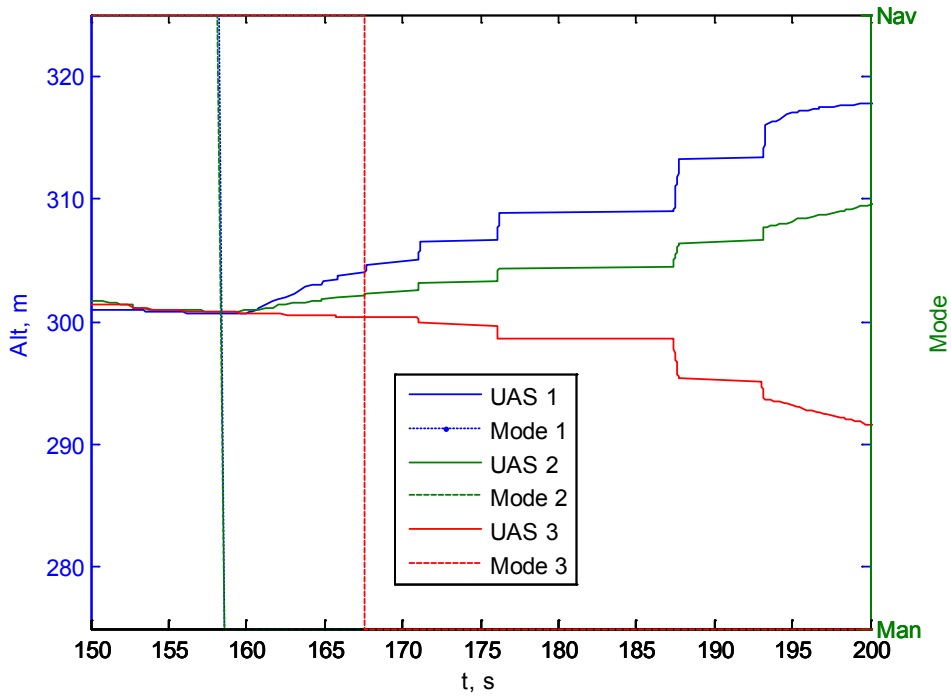**Figure 4-24: SIL Three-Ship Simulation Range**

Figure 4-25: SIL Three-Ship Simulation Altitude

79

2.3. Hardware-in-the-Loop

Hardware-in-the-loop (HIL) tests required significantly more test preparation than SIL tests. The autopilots were configured as though they were on an actual aircraft. They are battery powered with an antenna attached for wireless communication with the communication box (COMM BOX). The difference between actual operation and HIL is that the autopilots communicate with Aviones for simulated aircraft dynamics by configuring Aviones to communicate through USB communication ports and connecting the autopilots to these ports with adaptors. The COMM BOX is attached to the computer running Virtual Cockpit which is configured for autopilot operation in HIL mode and communicates with the COMM BOX.
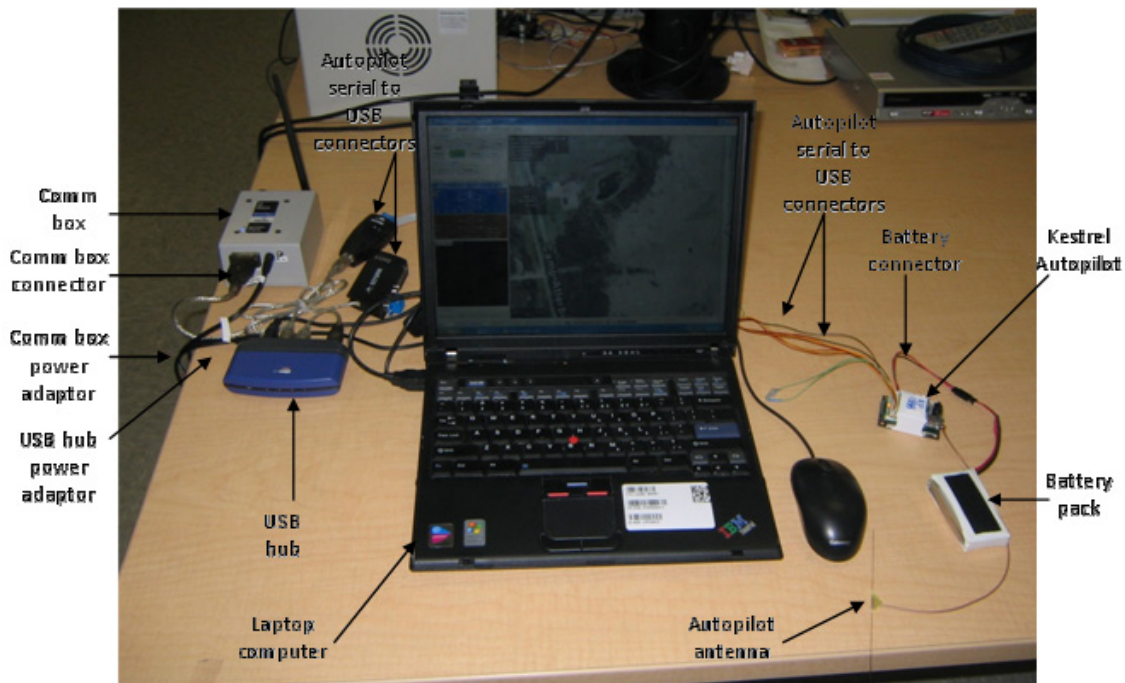

Figure 4-26: Single UAS HIL Set-up

Figure 4-26 shows the HIL set-up for a single vehicle UAS. For a multiple vehicle UAS, which is the case for the CA testing, each autopilot must have its own computer, battery, antenna, and Aviones instantiation. The COMM BOX will communicate with each autopilot and transfer communications to and from Virtual Cockpit.

Each autopilot is configured as in Table 4-1 and test cases are identical to those in SIL testing. Added complexity and uncertainty in HIL includes firmware and autopilot processing, battery power, and wireless communication. Communication between the CA application and the autopilot is now subject to wireless communication delays and dropouts in addition to independent firmware and autopilot processing instead of software-only processing on a single computer.
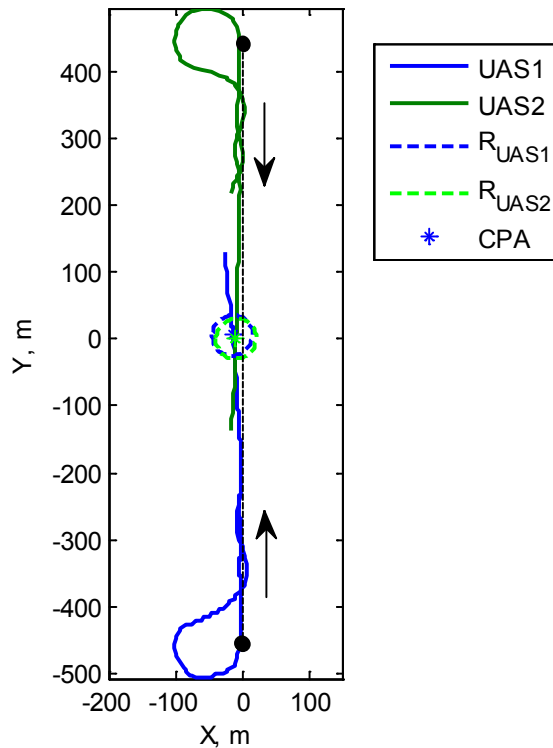

Figure 4-27: HIL Head-on Simulation Trajectories

Trajectories for the HIL head-on encounter are shown in Figure 4-27. It falsely appears in this horizontal plane view that the aircraft collide. However, in Figure 4-28, the altitude time-history shows that the aircraft achieved altitude separation at CPA and successfully avoided a collision.
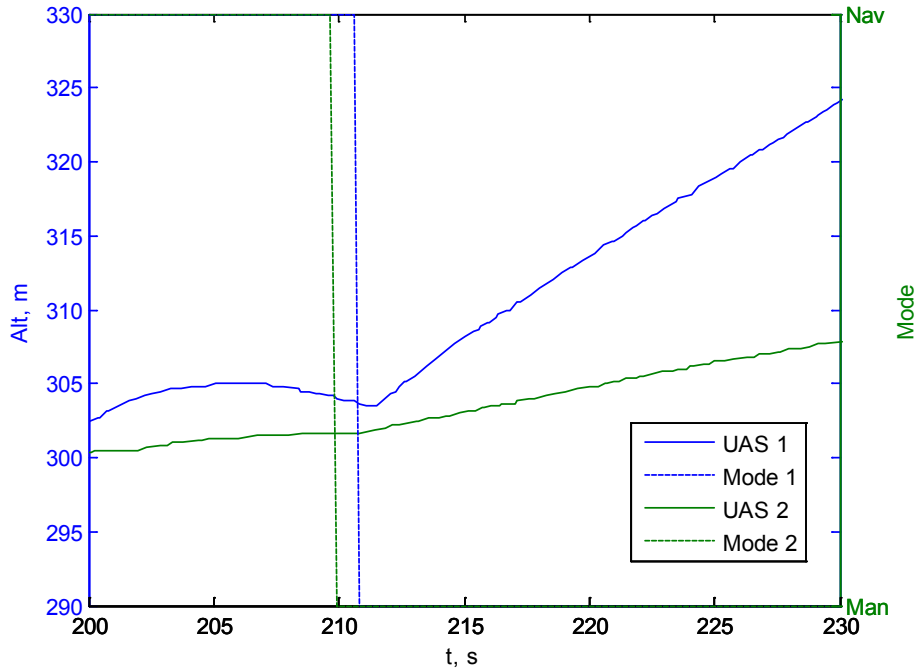

Figure 4-28: HIL Head-on Simulation Altitude

It is interesting that this encounter differs from the SIL test, which maintained lateral separation. Differences in the trajectories, though slight relative to the trajectory's scale, can cause significant differences in the resultant avoidance commands. Also, uncertainty in both tests will inevitably cause differences. Although differences are expected, the lack of any considerable lateral separation and opposing turn rate commands resulting in the aircraft turning towards each other are evidence of 1) poor command tracking and 2) conflicting interpretations of geometry between

the collision cone algorithm and guidance logic. This single encounter shows the
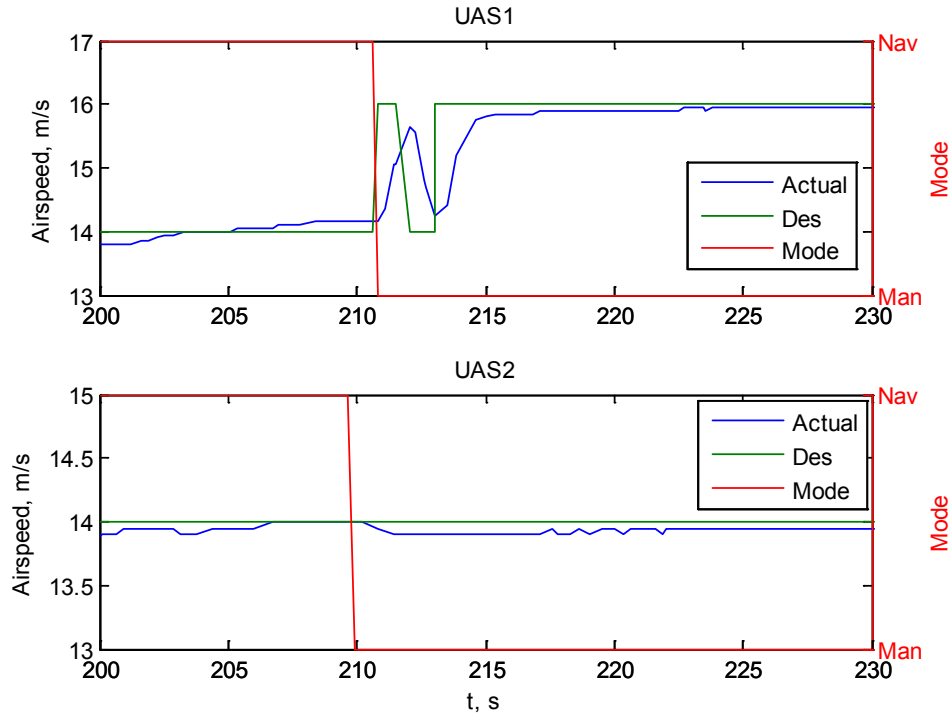importance of multi-dimensional commands in a geometric algorithm.

Figure 4-29: HIL Head-on Simulation Airspeed Avoidance Command

The CA commands are shown in Figures 4-29 through 4-31. As alluded to
previously, the turn rate commands are opposite in sign, which for a head-on
trajectory turns the aircraft in the same direction. It can be seen in the trajectory figure
the aircraft are still oscillating around the nominal trajectory at the point of CA
activation. The aircraft were most likely at an orientation that warranted the
commanded turn rates, but based on future knowledge of the desired trajectory, they
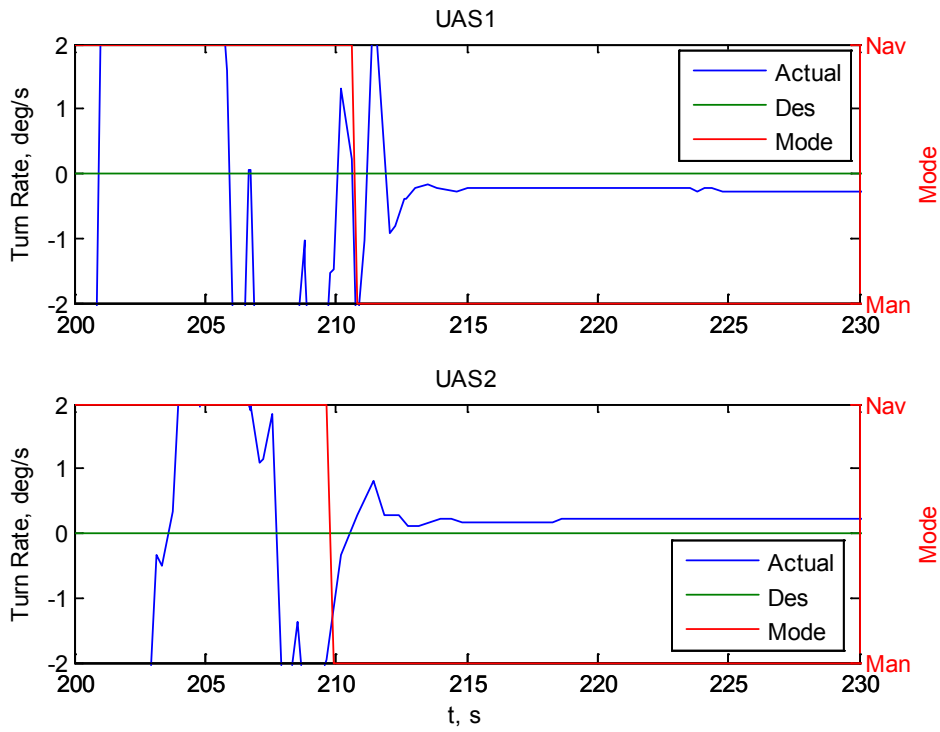are undesirable.

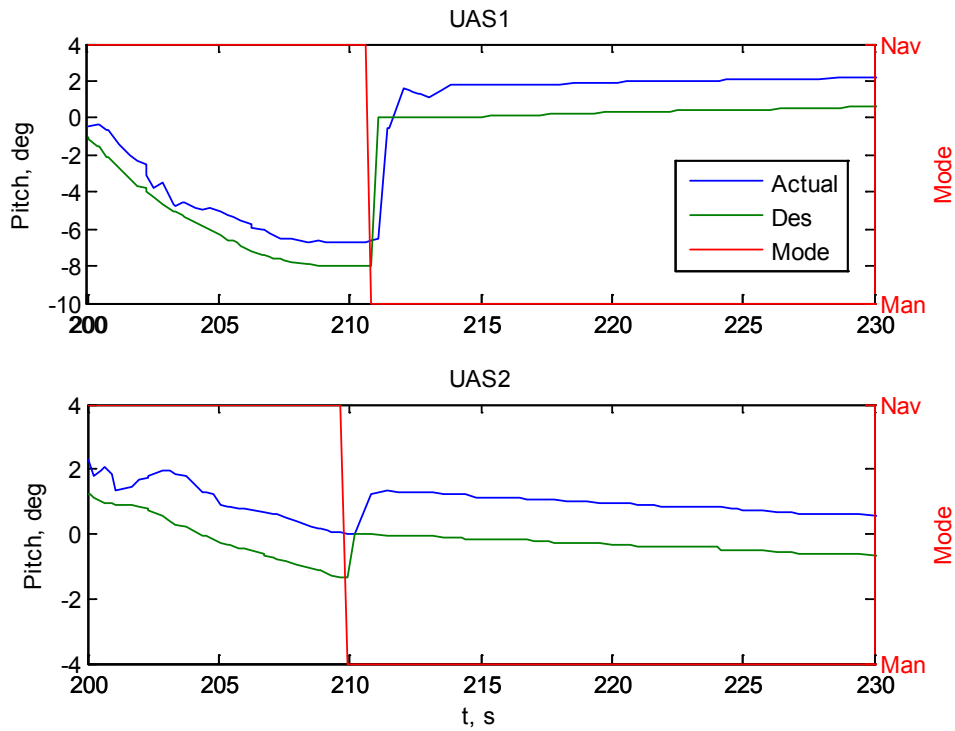**Figure 4-30: HIL Head-on Simulation Turn Rate Avoidance Command**

Figure 4-31: HIL Head-on Simulation Pitch Avoidance Command

In Figure 4-31, the aircraft pitch response to the avoidance commands contains a one to two degree steady-state error. The pitch commands from the CA algorithm for UAS 1 are positive and for UAS 2 are negative and should separate the aircraft. However, both responses are positive because the error is larger than the negative commands for UAS 2. This results in both aircraft climbing as seen in Figure 4-28 but at much different rates. This is an autopilot issue requiring PID gain tuning, but it did not cause loss of altitude separation. The airspeed command for UAS 2 remains at the nominal speed, but this is also an effect of autopilot configuration. The speed changes commanded by the CA algorithm are not large enough for interpretation by the autopilot.
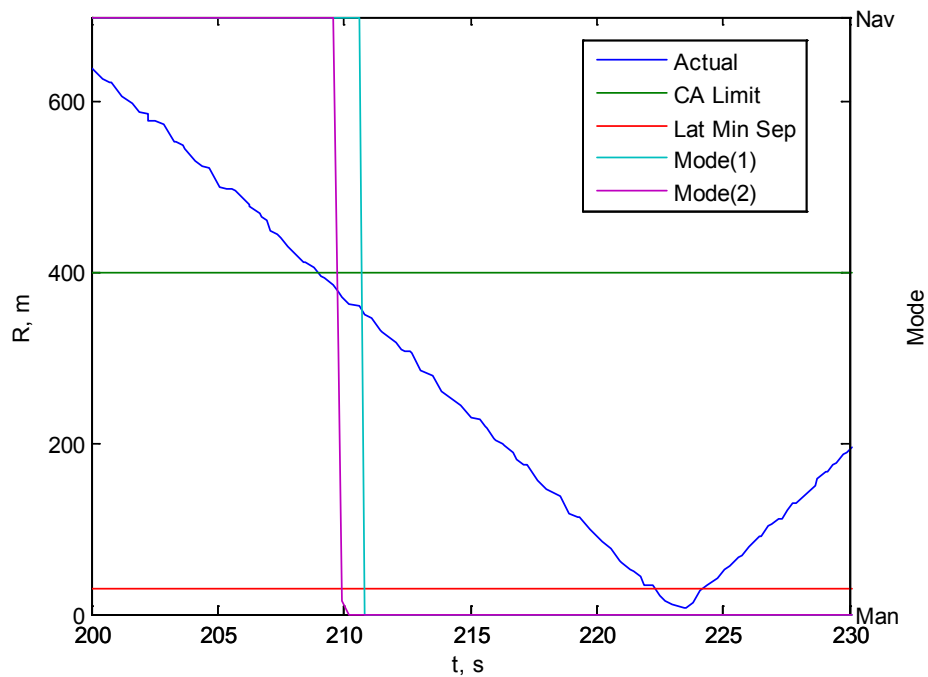

Figure 4-32: HIL Head-on Simulation Range

The range plot, Figure 4-32, indicates loss of lateral separation, but roughly shows the smaller altitude separation. Figures for the other two-ship encounters, Approaching, Abeam, and Converging, are in Appendix E. Table 4-4 shows key statistics of each of the two-ship encounters.

Table 4-4: HIL Simulation Results, Key Statistics

Parameter	Value			
	Head-on	**Approach**	**Abeam**	**Converge**
Time of Minimum Lateral Separation	223.47 s	584.83 s	608.36 s	189.2 s
Minimum Lateral Separation	7.3 m	24.77 s	27.83 m	65.2 m
Time of Minimum Slant Range	223.47 s	584.83 s	608.36 s	189.2 s
Minimum Slant Range	13.54 m	27.01 m	31.85 m	66.26 m
UAS1 Alt at $T_{min,LS}$	317.33 m	316.83 m	315.67 m	300.67 m
UAS2 Alt at $T_{min,LS}$	306.5 m	306.33 m	299.67 m	308.00 m
UAS1 Alt at $T_{min,SR}$	317.33 m	316.83 m	315.67 m	300.67 m
UAS2 Alt at $T_{min,SR}$	306.5 m	306.33 m	299.67 m	308.00 m

HIL two-ship simulations stressed the importance of minimum altitude separation by means of a pitch command. Only the Converging test case maintained the full lateral separation, though the Abeam and Approaching cases were over 20 m and also achieved altitude separation. Three of the cases also achieved their full altitude separation. The HIL results departed from the SIL results in separation trends, but are favorable in their own right and display successful CA. Communication

"buffer" delay issues, similar to those in SIL testing, caused delays in Mode switching back to NAV mode after CA was successful.
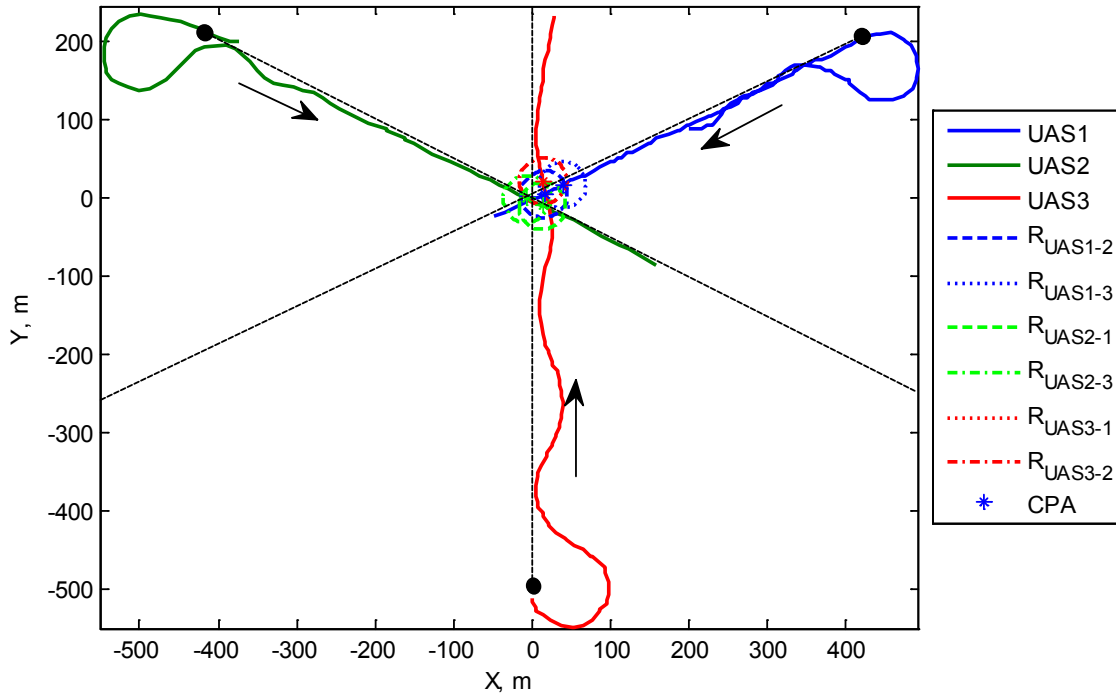

Figure 4-33: HIL Three-ship Simulation Trajectories

A three-ship HIL test is completed and compared to the SIL test. Figure 4-33 shows the resultant trajectories, and Figure 4-34 is a close-up view of the origin and the CPA locations. The CPA between distinct pairs of UAS is distinguished in the same manner used in SIL; color represents a particular UAS and line type denotes a distinct UAS pair. UAS 3 noticeably deviated from its nominal trajectory before the collision encounter and CA initiation. Its waypoint following, albeit poor, further challenged the CA algorithm because of the maneuvering.
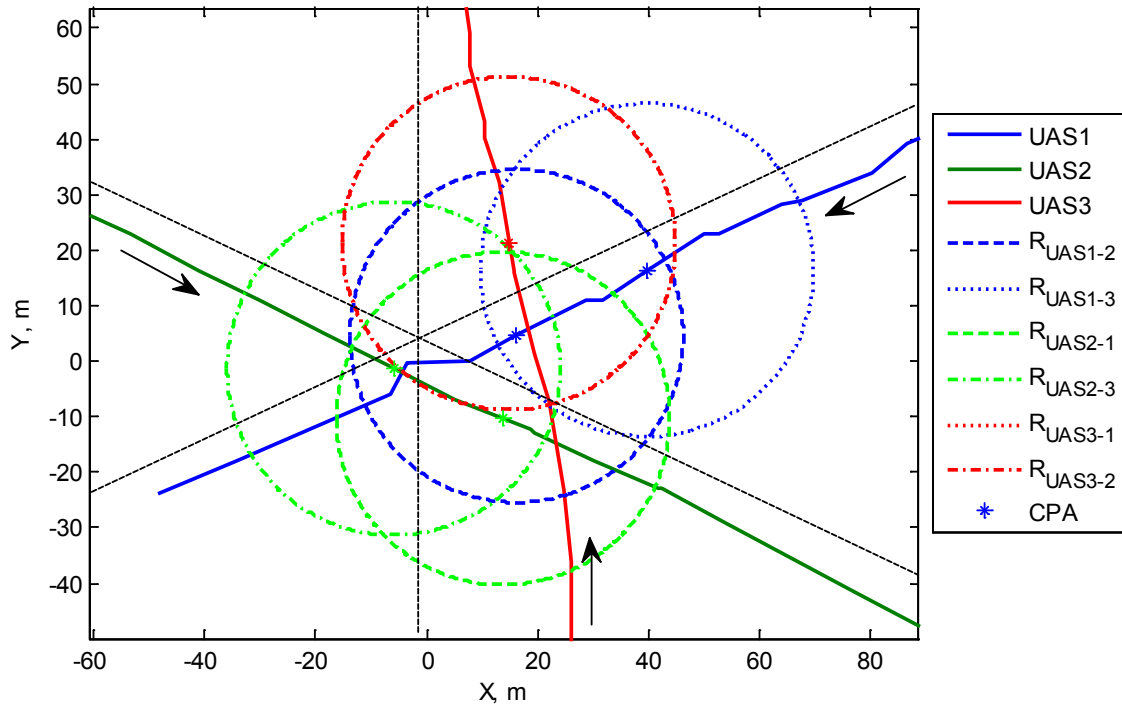
87

Figure 4-34: HIL Three-ship Simulation Trajectories, Zoomed to Origin

Table 4-5 shows key statistics for the HIL three-ship simulation. UAS 1 and 2 achieved more than the required altitude separation at CPA, UAS 1 and 3 also achieved full altitude separation as well as nearly achieving lateral separation, and UAS 2 and 3 achieved full lateral separation. Unfortunately, the separations cannot be fully attributed to CA commands. It will be seen in later figures that the Mode switch for CA for UAS 2 and 3 occurs after CPA. This is undoubtedly due to communication delays because commands are still issued after CPA, gleaning more evidence of a communication buffer that stores packets for transmission in a queue.

Table 4-5: HIL Three-ship Simulation Results, Key Statistics

Parameter	Value		
	1-2 (2-1)	1-3 (3-1)	2-3 (3-2)
Time of Minimum Lateral Separation	874.09 s	872.01 s	872.01 s
Minimum Lateral Separation	15.05 m	25.36 m	30.64 m
Time of Minimum Slant Range	874.09 s	872.01 s	872.01 s
Minimum Slant Range	22.46 m	30.35 m	30.66 m
UAS1 Alt at $T_{min,LS}$	317 m	316 m	
UAS2 Alt at $T_{min,LS}$	300 m		300 m
UAS3 Alt at $T_{min,LS}$		299 m	299 m
UAS1 Alt at $T_{min,SR}$	317 m	316 m	
UAS2 Alt at $T_{min,SR}$	300 m		300 m
UAS3 Alt at $T_{min,SR}$		299 m	299 m

Although individual separation criteria were met, it is important to understand why some separation criteria in particular dimensions were not achieved. One glaring piece of evidence exists in the airspeed command, shown in Figure 4-35. It is unclear whether no airspeed commands actually were received by the autopilots, or whether they were too small to be interpreted by the autopilots, but no speed changes were commanded. They were unquestionably commanded by the CA algorithm because the other commands were processed by the autopilots.
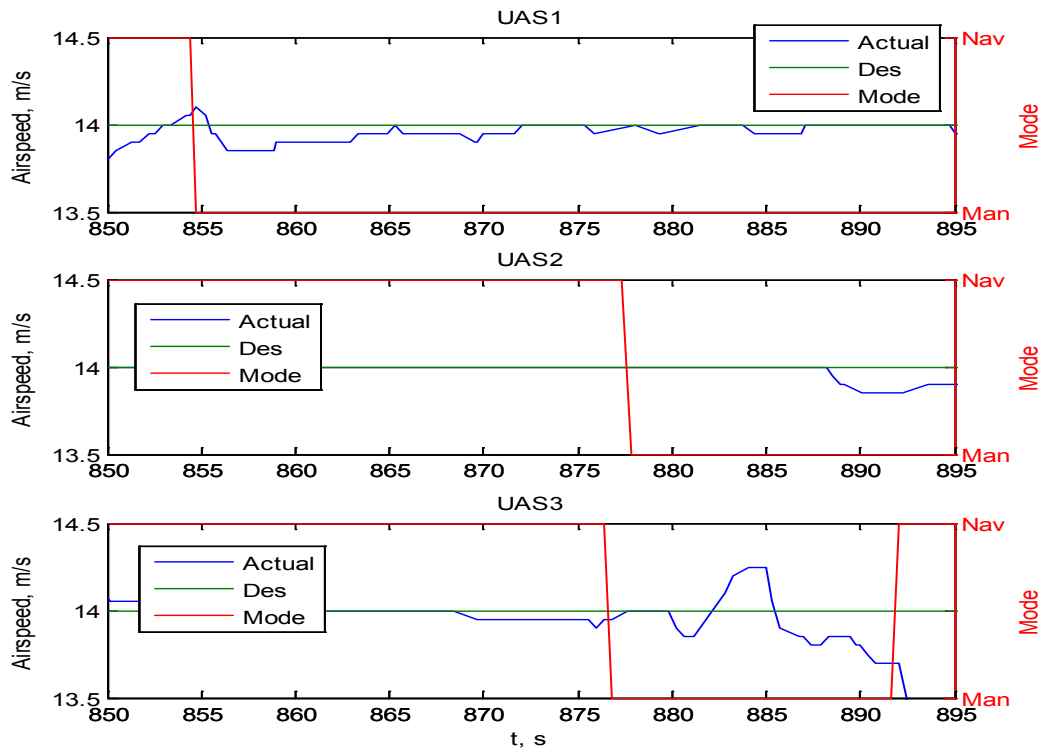
Figure 4-35: HIL Three-ship Simulation Airspeed Command

Both combinations of failed lateral separation involved UAS 1. It can be seen in Figure 4-36 why this possibly happened. UAS 1 did not respond to or did not receive a turn rate command. Unfortunately, because turn rate commands cannot be recorded, it is impossible to tell which is the case. It is speculated that if UAS 1 had responded to a turn rate command, the UAS 1-2 and UAS 1-3 combinations would have achieved lateral separation because 1) UAS 2-3 successfully achieved lateral separation with their commands which also included separation considerations for UAS 1 (global approach), and 2) UAS 1 responded successfully to its vertical commands to achieve altitude separation.
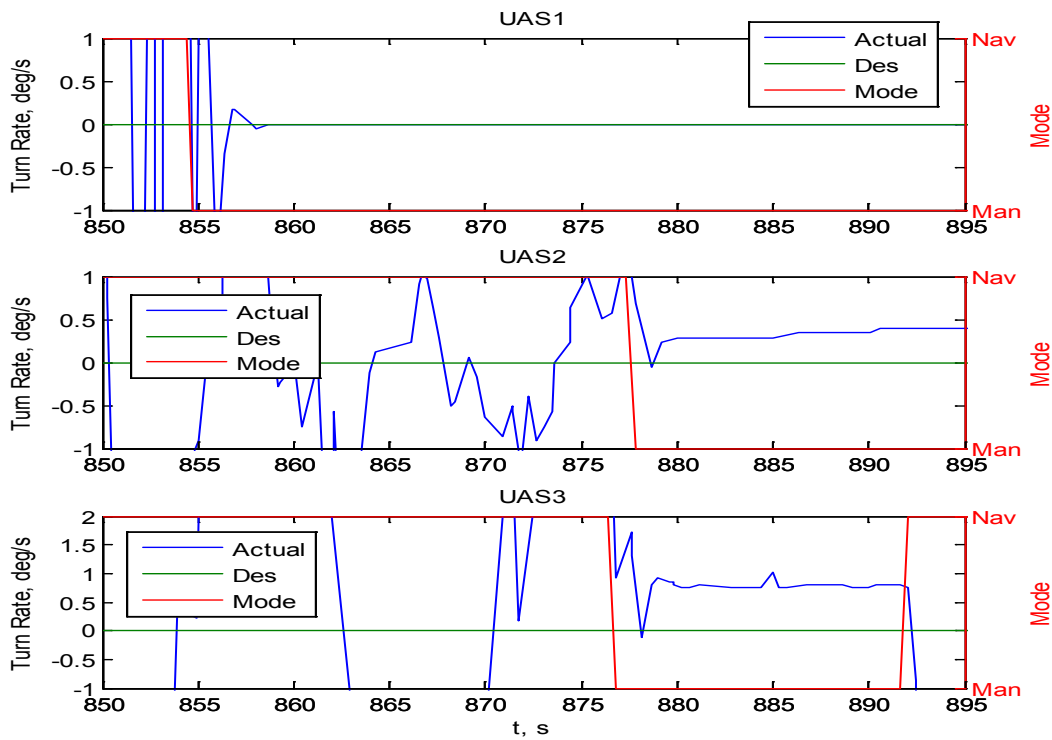
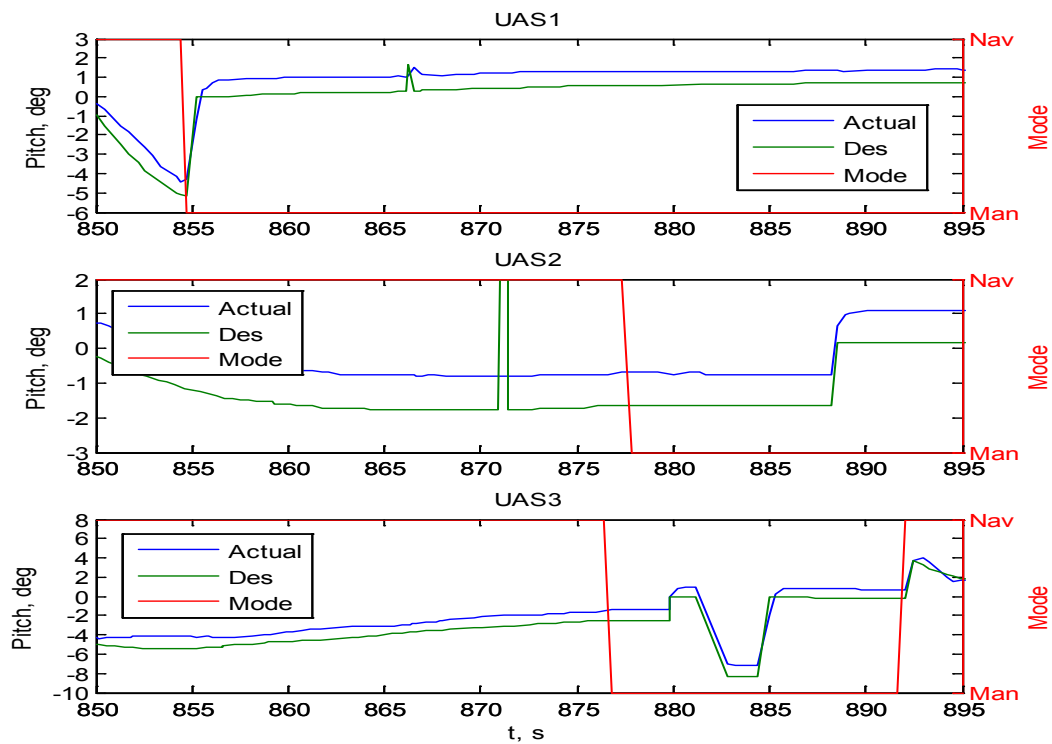**Figure 4-36: HIL Three-ship Simulation Turn Rate Avoidance Command**

Figure 4-37: HIL Three-ship Simulation Pitch Avoidance Command

UAS 1 achieved altitude separation from both UAS 2 and 3 because it immediately received and responded to its pitch command (see Figure 4-37). UAS 2 and 3 did not achieve altitude separation. This, again, is undoubtedly due to them receiving their pitch commands long after (estimated to be 10-20 sec) they were actually commanded. The steps in their commands after the Mode switch are uncharacteristic of the guidance commands processed by the avoidance algorithm. They should be smooth and continuous as seen in the UAS 1 command. The command steps are more evidence of communication problems, seemingly due to a communication "bottle-neck" that transmits commands sporadically.

Figure 4-38 and Figure 4-39 show the HIL three-ship range and altitude, respectively. The late Mode switches for UAS 2 and 3 are apparent in the range plot. UAS 1 altitude separation is evident in the altitude figure. It is curious that UAS 1 receives and responds to its Mode switch pitch command immediately, but does not receive or respond to its turn rate or altitude command. After further investigation, the order of command transmittals in the CA algorithm code is this: 1) mode switch, 2) airspeed, 3) pitch, 4) turn rate. The author makes the following hypothesis about these curious events. The mode switch happens immediately, and the airspeed command is transmitted, but is too small to be processed by the autopilot. The pitch command is then sent successfully. The turn rate command is added to the queue, along with commands for UAS 2 and 3, and the queue begins to build and delays subsequently accumulate. Further investigation into the workings of the Kestrel libraries and how command packets are transmitted may begin to shed light on these peculiarities.
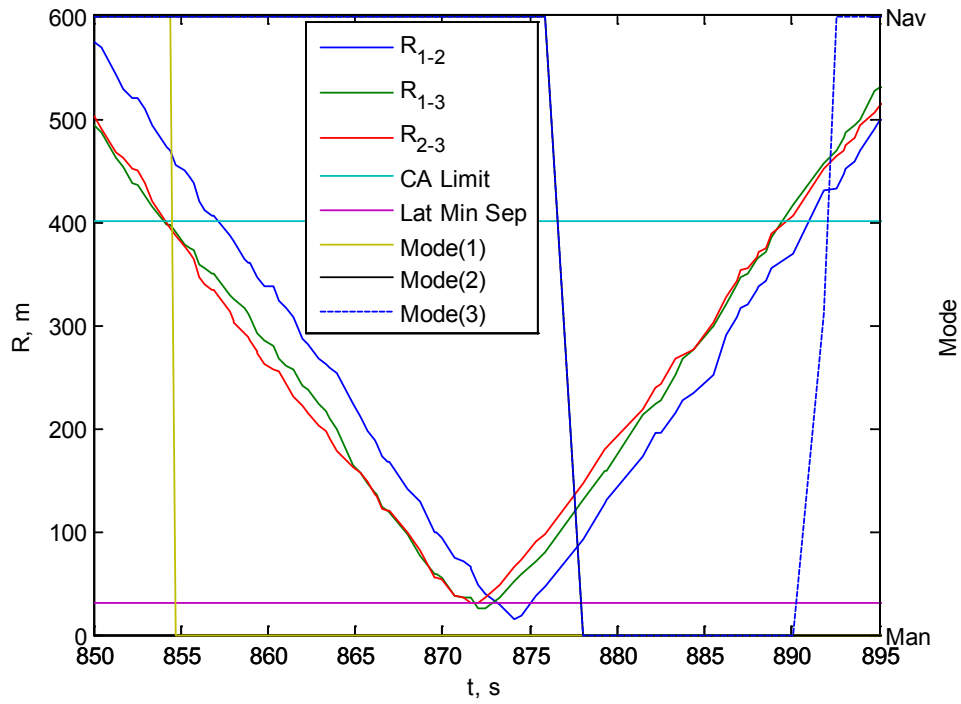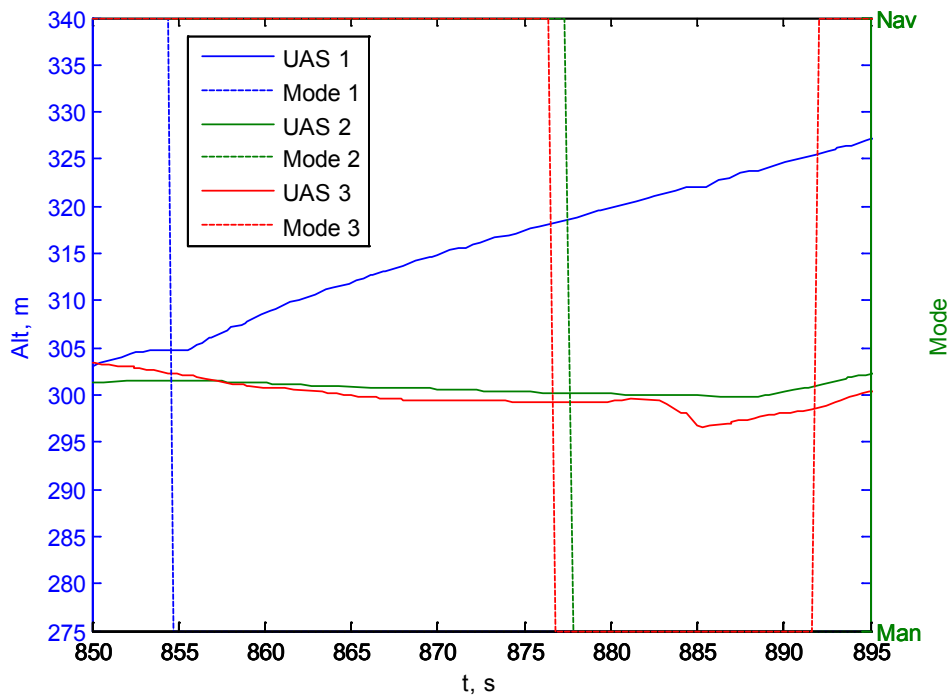
Figure 4-38: HIL Three-ship Simulation Range

Figure 4-39: HIL Three-ship Simulation Altitude

93

3. Flight Test Results

Flight tests were flown at Camp Atterbury, Indiana, a military installation with restricted airspace enabling autonomous flight in controlled airspace. An entire ground unit including the GCS, test support, and repair equipment was used as the test operations center. The GCS is equipped with the same laptop used in HIL tests along with additional communication and video equipment used with the BATCAM systems. The GCS test operations area, located in the front of the ground unit, is shown in Figure 4-40 and a test aircraft, BATCAM 1, is shown in Figure 4-41.

Figure 4-40: Flight Test Ground Control Station

Detailed flight test procedures, provided in Appendix G, were written and presented at AFIT to a safety and technical review board for test approval prior to launch. Safety considerations included the ballistic footprint of debris falling in the event of an actual collision. Emergency procedures were written specifically for CA testing in the case of unresponsive BATCAM aircraft under CA control. Safety precautions included 50 ft altitude separation warranting a relatively over-sized minimum separation volume

94

and waypoint placements that resulted in a theoretical collision point separated from the GCS by three times the ballistic debris footprint.

Figure 4-41: AFIT's BATCAM 1

Flight test procedures were written for two-ship and three ship encounters at the engagement angles used in SIL and HIL testing. Two-ship tests at two engagement angles were actually flown and will be reviewed in detail in later discussions. Wind conditions were a cause of changes to some of the flight test procedures. Waypoint patterns were constructed to produce the desired collision encounters in the test range area. These patterns were adjusted in orientation with respect to the Camp Atterbury runway to align the mean wind direction perpendicular to the flight paths in order to reduce discrepancies in ground speed. Also, for flight paths that could not be aligned perpendicular to the wind direction, speed adjustments were made in Navigation mode waypoint settings to account for head and tail winds. The Camp Atterbury runway and two-ship CA test waypoints are shown in Figure 4-42. Solid lines represent planned waypoints and dashed lines represent

actual test waypoints after, approximately, a 20 deg counter-clockwise adjustment for

winds. The total range between waypoints was also reduced in order to ensure the

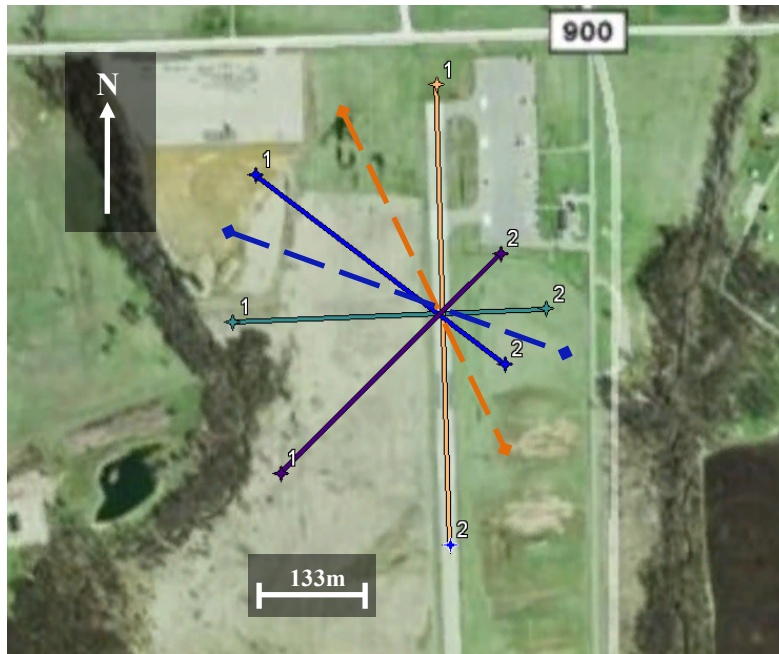BATCAMs were in-view throughout the entire test.

Figure 4-42: Two-Ship Flight Test Waypoints over Camp Atterbury

The BATCAMs in each collision encounter were separated in altitude by 50 ft.

This separation is included for two reasons: 1) safety, and 2) BATCAM altitude holding.

In previous flight tests of the BATCAMs, they were found to hold altitude only within

plus or minus 30 ft. To account for this, the vertical minimum separation in the algorithm

was set to 20 m (~65 ft) for all tests. The lateral minimum separation was also increased

to 60 m to excite the horizontal avoidance commands and flatten out the minimum

separation cylinder. It was found during the flight tests that significant "weaving"

occurred in navigation mode when transitioning to the line of sight between waypoints.

This introduced additional uncertainty in the tests for collision detection and command

generation and was partially removed by changing cross-track settings in Virtual Cockpit for Navigation mode. Unfortunately, these oscillatory flight patterns were not alleviated entirely and are apparent in flight test data.

3.1. Pre-flight Ground Testing

Ground tests were completed prior to BATCAM launch to verify two issues that cannot be determined in HIL testing. Test personnel physically walked with BATCAMs in-hand towards each other as though they were actually flying and activated CA. In the first ground-test, an R/C Manual mode exists in the Kestrel system that provides radio control of the BATCAMs for a safety pilot through the COMM BOX to the Kestrel autopilot. To safely proceed with testing, it had to be shown that R/C mode could over-ride CA commands in the event the aircraft become unresponsive during collision encounters. This was verified in ground tests. The R/C mode is limited to control over only one aircraft at a time, and a switch to another aircraft is made in Virtual Cockpit. It was found that even though R/C control will supersede CA in Manual mode, Virtual Cockpit will not switch R/C control to another aircraft when communication blockages are present in the communication channels. This limitation presents additional risk to recovering unresponsive aircraft and was a major contributor to reducing the number of completed tests. A second ground test was completed to ensure the autonomous mode switching in the CA algorithm could complete the entire CA process with all flight test equipment activated. It was verified that CA can detect a collision, switch to Manual mode, issue

guidance commands, and return to Navigation mode autonomously with the entire flight test configuration.

3.2. Flight Testing

Two engagement angles were flown in the flight test, Head-on and Approaching. Multiple encounters per engagement angle, four for Head-on and two for Approaching, were flown and each one resulted in a positive collision detection and avoidance maneuver transmission. The remaining two-ship encounters and the three-ship encounter were not flown because of communication delay problems allegedly caused by a packet transmission buffer processing at a slower rate than command generation. This has yet to be substantiated. The last test and, ironically, the most successful in terms of collision avoidance data collection, resulted in both BATCAMs becoming unresponsive and emergency procedures were executed. Both aircraft were successfully recovered. The unresponsiveness is caused by a build-up of avoidance commands that are slowly processed even after the CA application is terminated. A newer version of the Kestrel autopilot is available with a faster processor, but the precise location of the delay in the communication system should be found before any equipment is acquired.

The second Approaching encounter, and final test, resulted in the best encounter because of the geometric alignment at CA initiation, the aircrafts' response to commands, and data quality. The trajectories for this encounter are shown in Figure 4-43 and the range between the aircraft is shown in Figure 4-44.
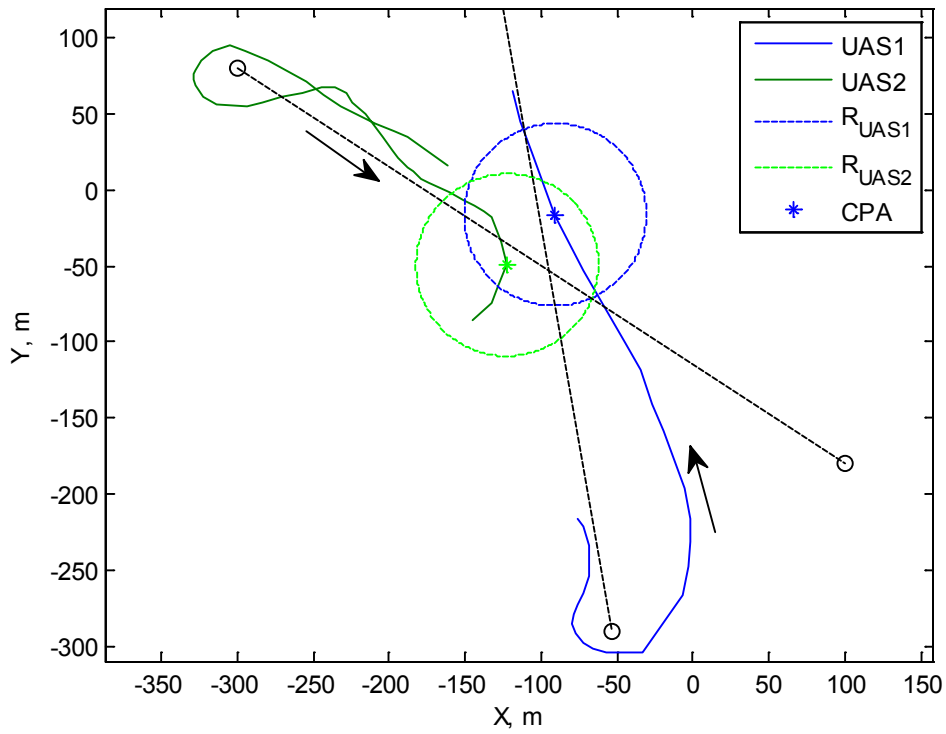
98

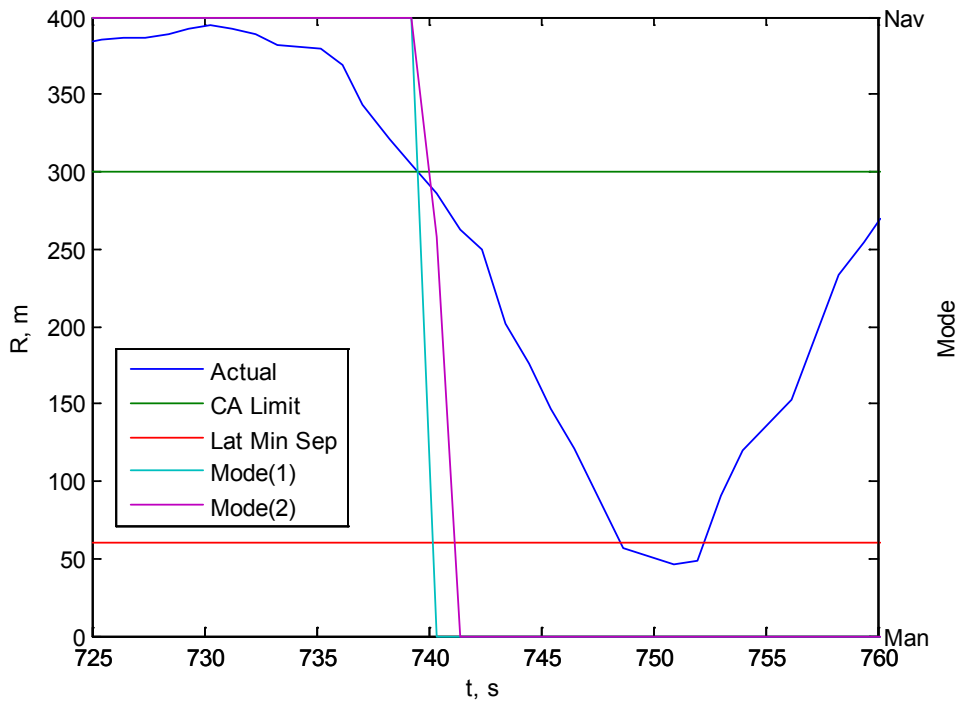**Figure 4-43: Flight Test Approaching Encounter 2 Trajectories**

Figure 4-44: Flight Test Approaching Encounter 2 Range

99

The mode switch to Manual mode for BATCAM 2 and BATCAM 1 CA was activated simultaneously at the maximum CA range as expected. The range at CPA for this encounter was approximately 47 m. The desired minimum separation of 60 m was not fully maintained, but appropriate avoidance maneuvers were commanded and did result in large separation distances in the presence of uncertainties. The mode switch and altitude are shown in Figure 4-45. It should be mentioned that even though the aircraft are supposed to be separated by 50 ft in altitude, it can clearly be seen that they hold altitude poorly in navigation mode and are nearly co-altitude at times.
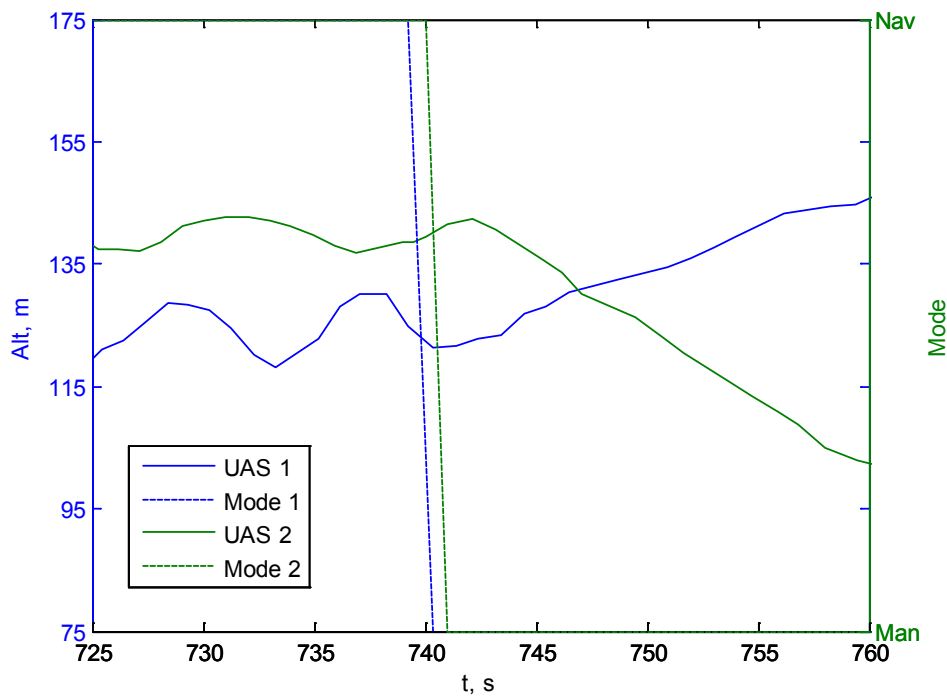

Figure 4-45: Flight Test Approaching Encounter 2 Altitude

It is peculiar that BATCAM 2 dives after its switch to CA control, but it is at a higher altitude than BATCAM 1. After further investigation into the vertical commands, BATCAM 2 was sent a positive pitch command to climb, but lost altitude

100

because of a decrease in angle of attack. The trim angle of attack was larger than the pitch command sent to the BATCAM. Despite the aircraft response, the algorithm was functioning properly and provided intuitive commands. As discussed in Chapter III, the interface between the CA algorithm and the Kestrel autopilot did not allow flight path angle commands and only offered orientation angle commands. A small angle of attack assumption and small dynamic delay between a pitch change and the resultant altitude change assumption were made for algorithm integration. The assumptions are shown to be inaccurate and will require changes in the algorithm interface to command different variables. The author has spoken with Procerus representatives, and climb rate commands are available in later versions of the Kestrel autopilot. Thus, integration improvements may be possible with updated equipment, although it is possible in newer versions the turn rate commands may have been removed. The pitch response is shown in Figure 4-46.

After CA initiation for BATCAM 1, it is commanded to decrease altitude. It pitches down but has a large steady state error and maintains a positive pitch angle. As a result, BATCAM 1 sustains a large angle of attack and actually begins to climb. Both BATCAM 1 and 2 exhibited altitude responses opposite to what was expected. One possible explanation is wind causing large differences in airspeed and subsequently angle of attack, resulting in the exact opposite reaction to what is expected and what was commanded. In navigation mode, large oscillations can be seen in pitch, and observed also in the altitude plot and result in poor altitude tracking. In CA mode, commands are smooth and relatively benign early in the

encounter. This is exactly the pattern one would want in collision situations and gives
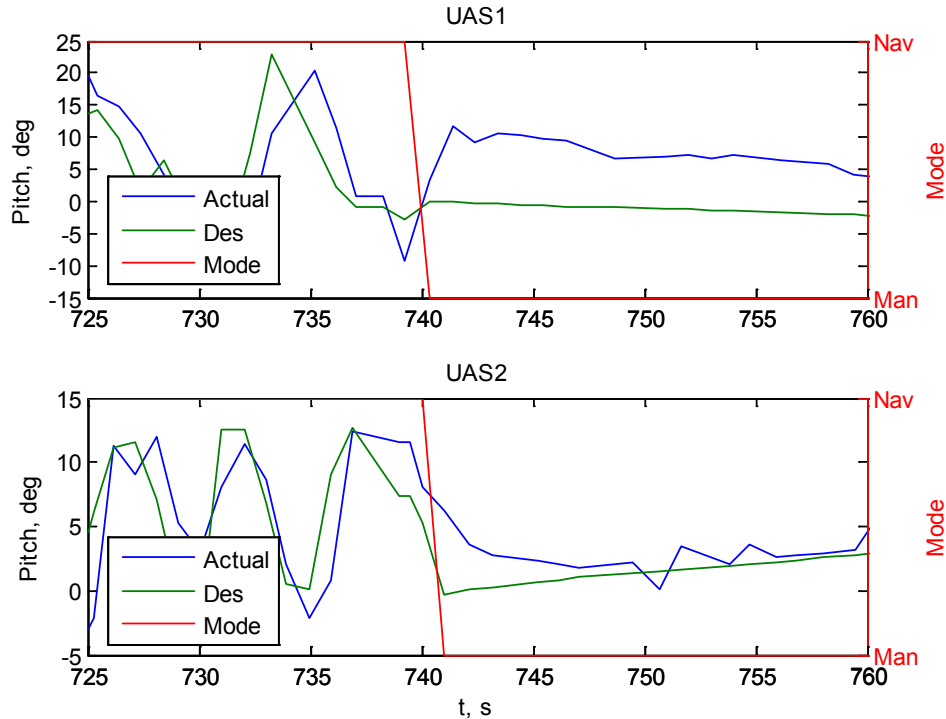assurance to the algorithm's design.

Figure 4-46: Flight Test Approaching Encounter 2 Pitch Response

The airspeed response is shown in Figure 4-47 and the turn rate response is in
Figure 4-48. BATCAM 1 is commanded to initially increase its airspeed and then
steadily decreases throughout the encounter. The commands to BATCAM 2 are
constant and at a slightly lower airspeed than in navigation mode. The BATCAM 1
response contains a steady-state error resulting in a slower airspeed, and the
BATCAM 2 response contains a steady-state error resulting in a faster airspeed.
These errors are caused by the BATCAMs turning into and away from the wind.
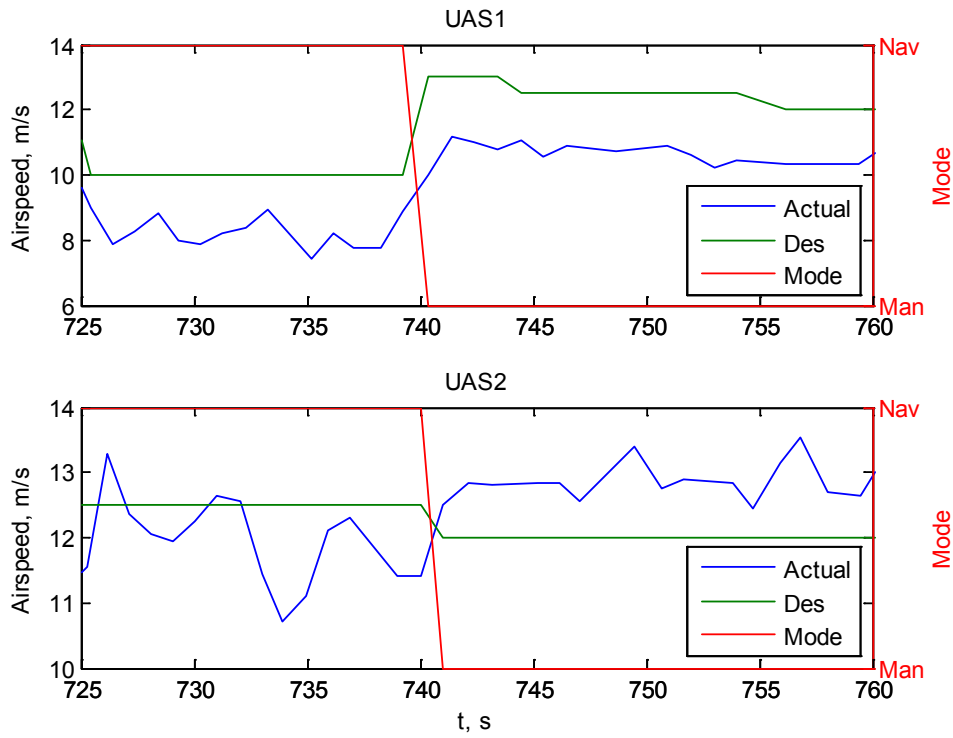
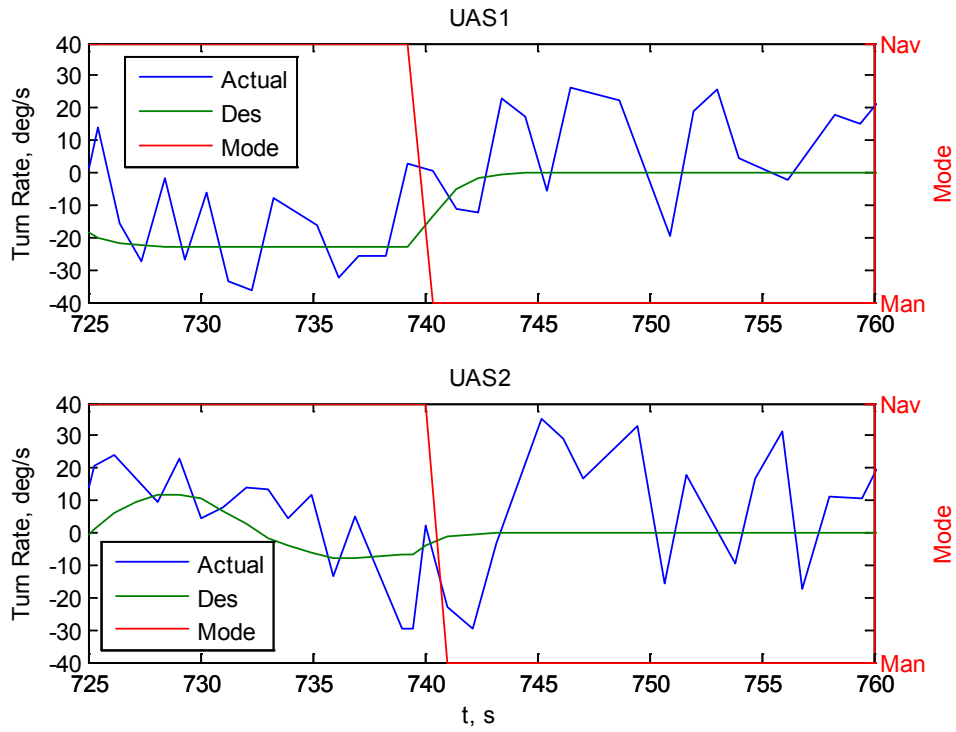**Figure 4-47: Flight Test Approaching Encounter 2 Airspeed Response**

Figure 4-48: Flight Test Approaching Encounter 2 Turn Rate Response

In SIL and HIL testing, it was not possible to record the turn rate command. However, in flight test, the turn rate command is recorded, but only in navigation mode. When in Manual mode, the command returns to zero, but not instantaneously. This is evidence of another autopilot control loop that generates turn rate commands is the one being recorded, and drifts to zero when not in navigation mode. Unfortunately, this still means CA turn rate commands are unavailable for recording, and only conjectures can be made about the response. One obvious conclusion about turn rate is that the measurements are extremely noisy. Only qualitative observations can be made, and no clear command pattern can be determined. By comparing to the trajectories in Figure 4-43, BATCAM 2 clearly receives a positive turn rate command, and BATCAM 1 also receives a positive command, though much less aggressive. These trends agree with those seen in Figure 4-48.
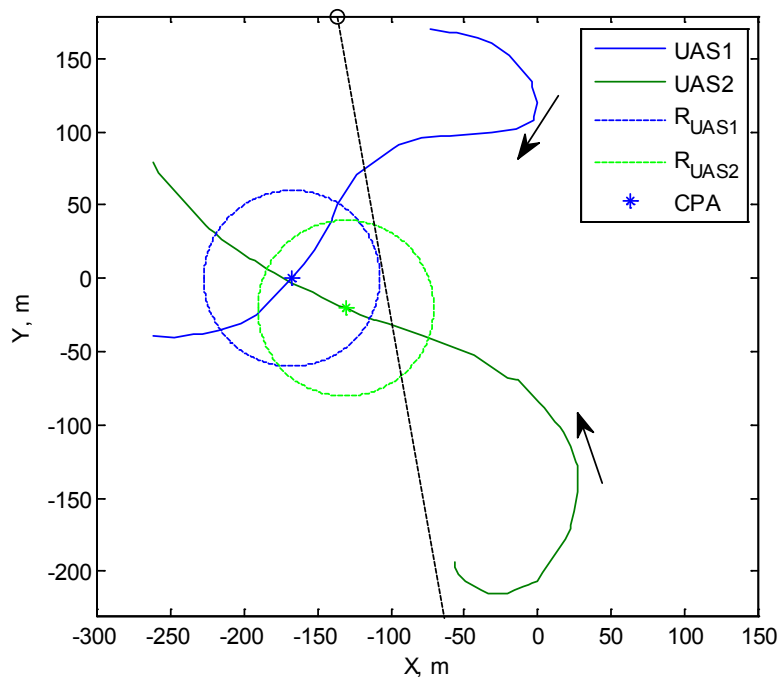

Figure 4-49: Flight Test Head-on Encounter 2 Trajectories

One of the most successful Head-on encounters was the second one flown. The BATCAMs were transitioning into their waypoint following routes and turned towards each other at a much smaller range (200 m) than the maximum CA range (300 m). Both BATCAMs switched to CA mode simultaneously after the CA algorithm immediately detected an imminent violation of minimum separation. The initial encounter through CPA was well behaved and demonstrated successful avoidance maneuvers. The range at CPA was approximately 45 m, and is about the same as the Approaching encounter discussed previously (47 m). The uncertainties in the system and environment caused consistent deviations from the desired minimum separation for both of these encounters. The trajectories for this encounter are in Figure 4-49. The range plot, in Figure 4-50, shows the CPA at approximately 785 sec and then additional undesired mode switching afterwards due to the communication delay and subsequent command build-up.

The excessive mode switching in Figure 4-50 after the CPA was caused by additional potential collision encounters. The communication delay did not allow a switch back to navigation mode and the BATCAMs continued processing old avoidance commands that introduced new encounters. After the command buffer was exhausted, the BATCAMs did return to navigation mode.

BATCAM 1 switched to CA mode and began to climb, even though it was at a lower altitude. This disagrees with the pitch command, which was correctly commanded by the algorithm. BATCAM 1 is initially commanded to pitch down but

because of a steady state error, it maintains a positive pitch angle and climbs. The altitude is in Figure 4-51 and the pitch response is in Figure 4-52.
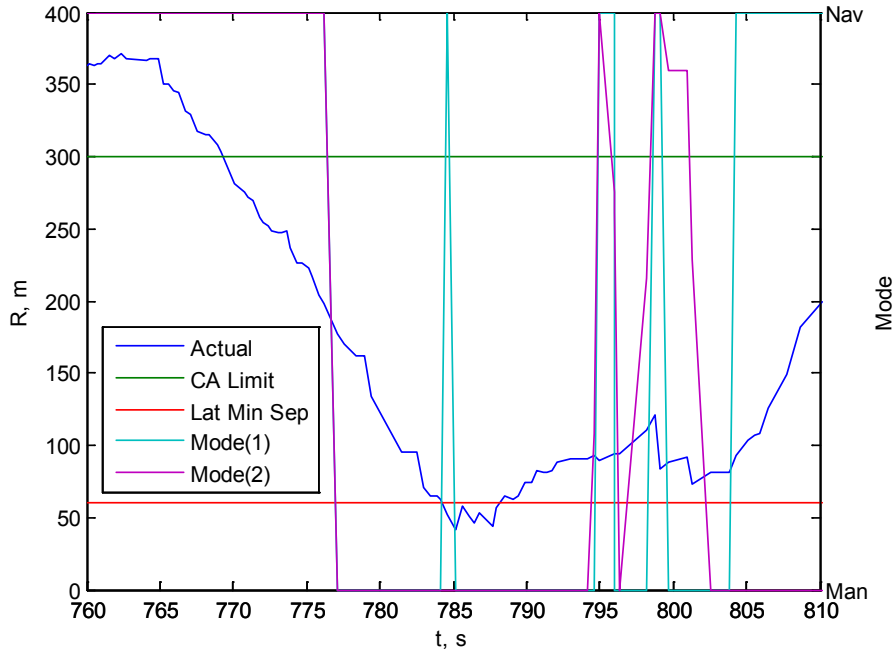

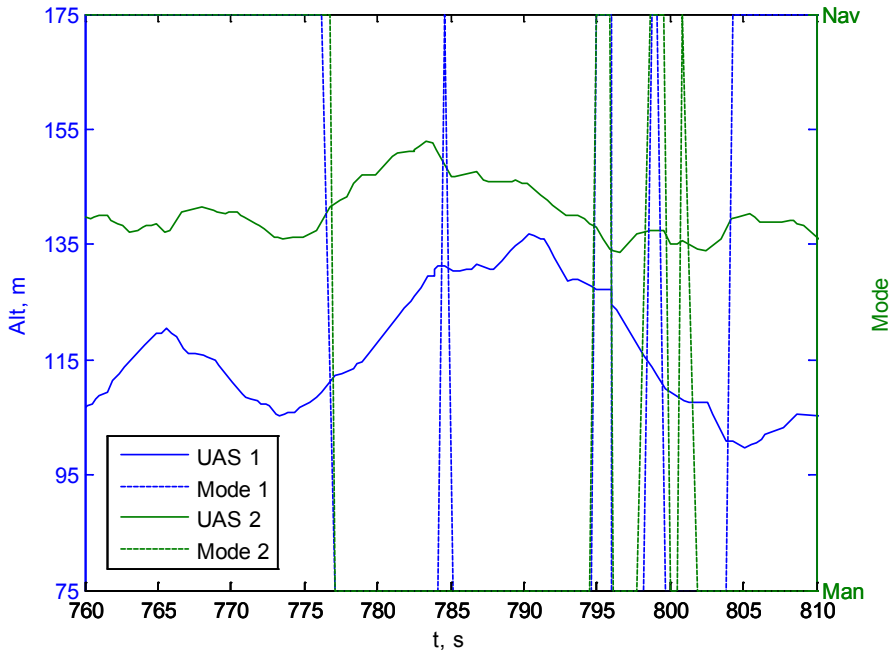**Figure 4-50: Flight Test Head-on Encounter 2 Range**

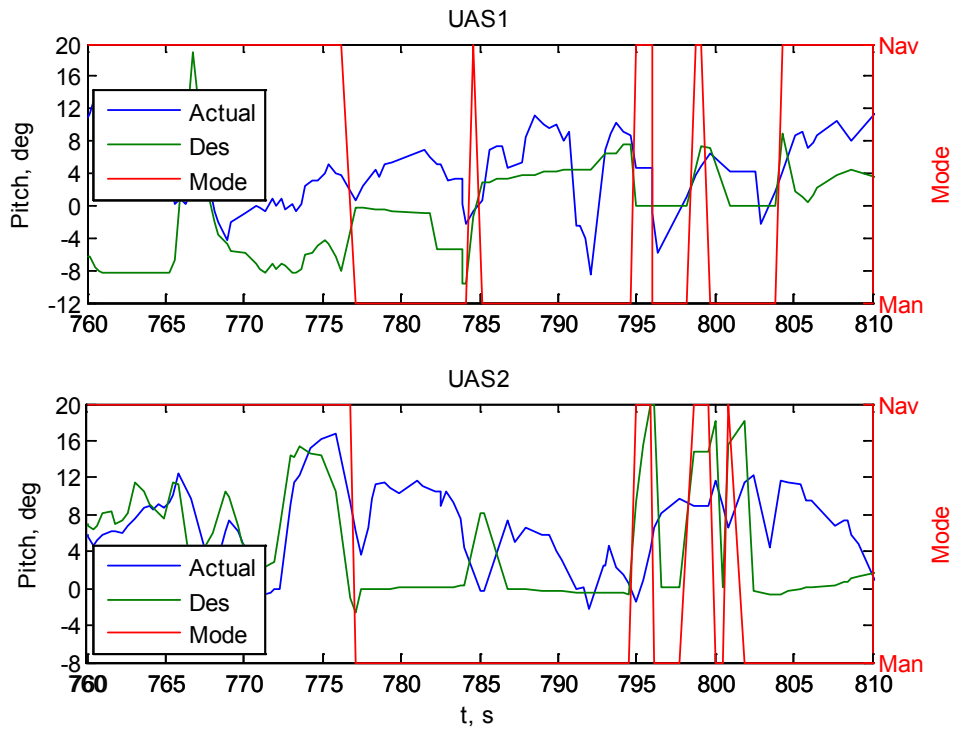**Figure 4-51: Flight Test Head-on Encounter 2 Altitude**

Figure 4-52: Flight Test Head-on Encounter 2 Pitch Response
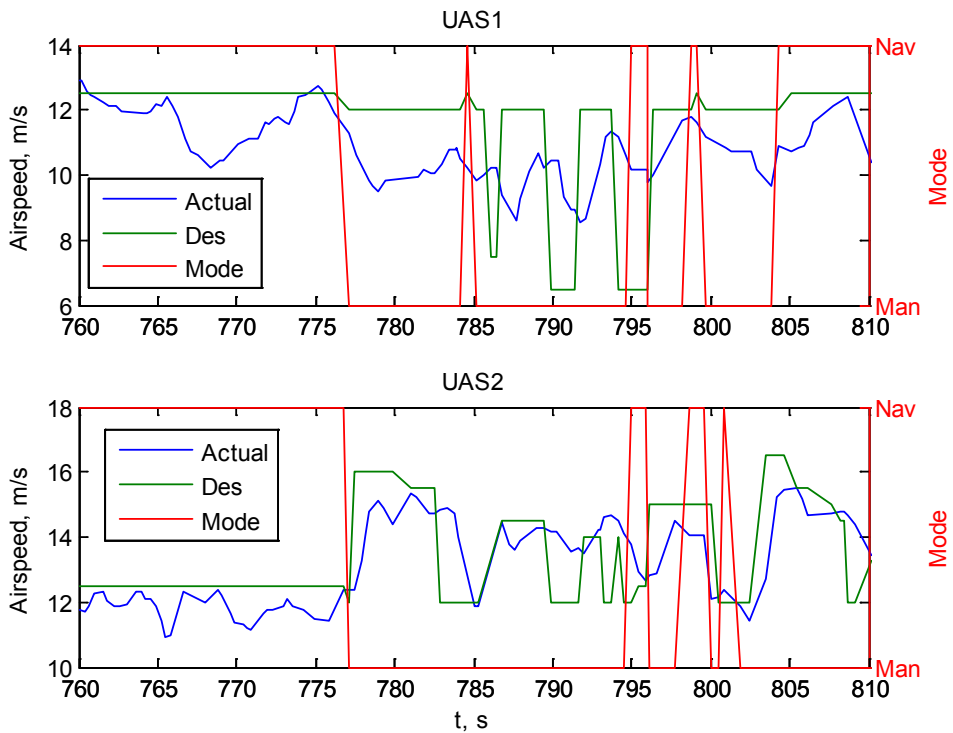

Figure 4-53: Flight Test Head-on Encounter 2 Airspeed Response

The airspeed commands, in Figure 4-53, from mode switch to CPA are well behaved. After the CPA, when CA should be turned off, they begin to look like a bang-bang type control. This is caused by the same problem seen in SIL testing where the CA airspeed commands are overwritten because of control loop settings beyond those adjusted for CA Manual mode. The turn rate commands, in Figure 4-54, behave as expected and exhibit commands positive in sign. As the aircraft approach CPA, positive turn rate commands would separate the aircraft.
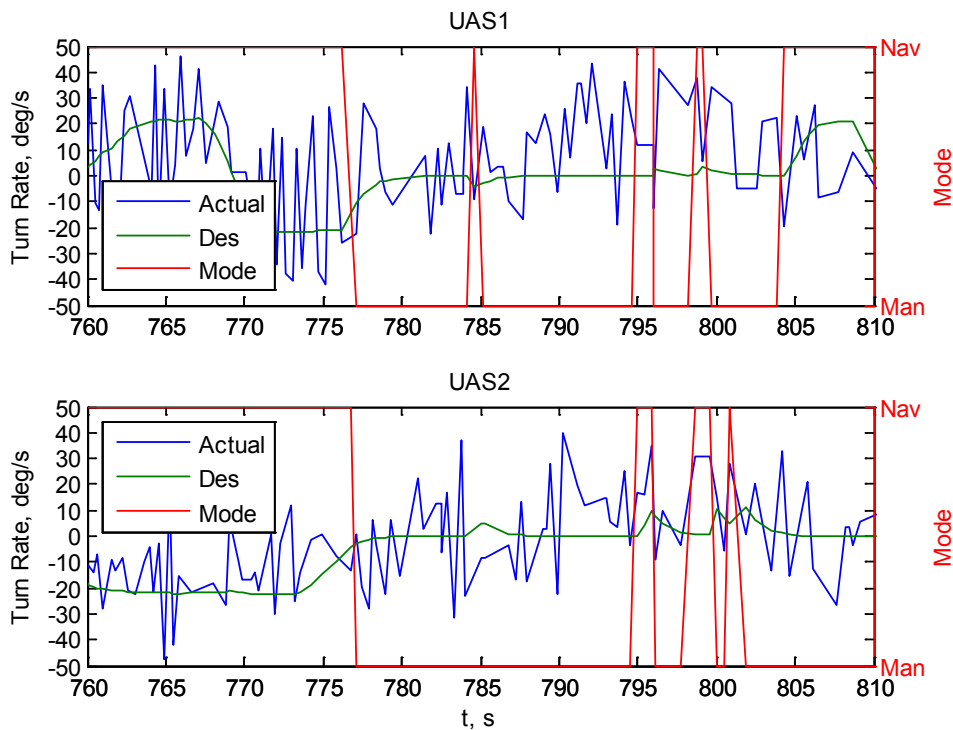

Figure 4-54: Flight Test Head-on Encounter 2 Turn Rate Response

An interesting Head-on encounter, not for its alignment or data quality, but for its stressing characteristics on CA is discussed next. BATCAM 2 was established in its transition from waypoint one to two but BATCAM 1 was lagging in the pattern and was merging into BATCAM 2's path in the fourth Head-on encounter. The CA

immediately detected the potential collision and initiated maneuvers for both aircraft while BATCAM 1 was in its turn. Sufficient separation (80 m) was still successfully maintained even with such close proximity prior to detection. This case shows that minimum separation or more can still be maintained with little time before the CPA by issuing aggressive commands. The trajectories for this encounter are shown in Figure 4-55.

The data rate for this encounter was quite poor. For the entire collision encounter, data was recorded once only every few seconds and explains the discontinuous telemetry plots.
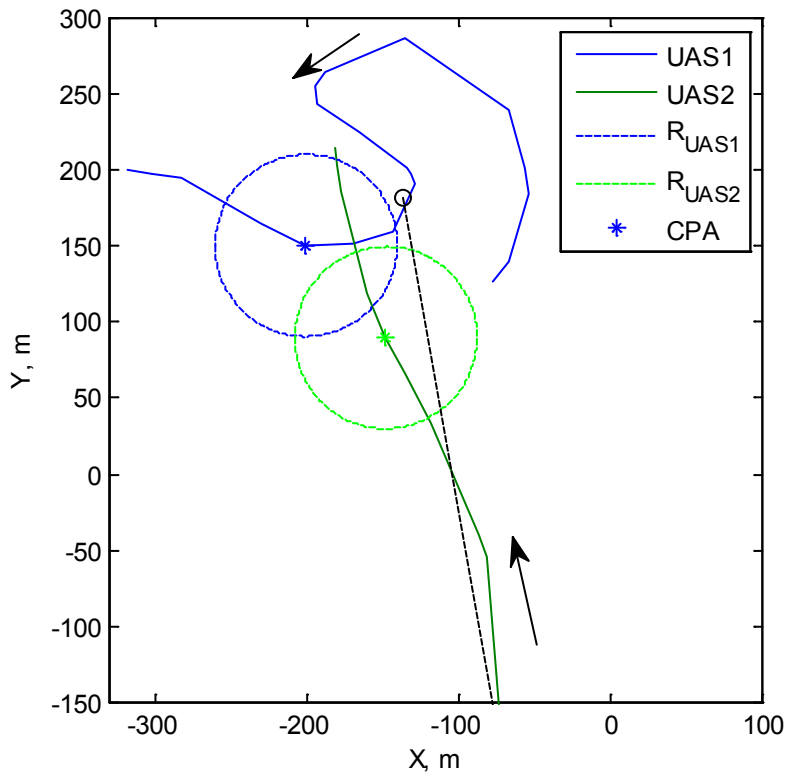

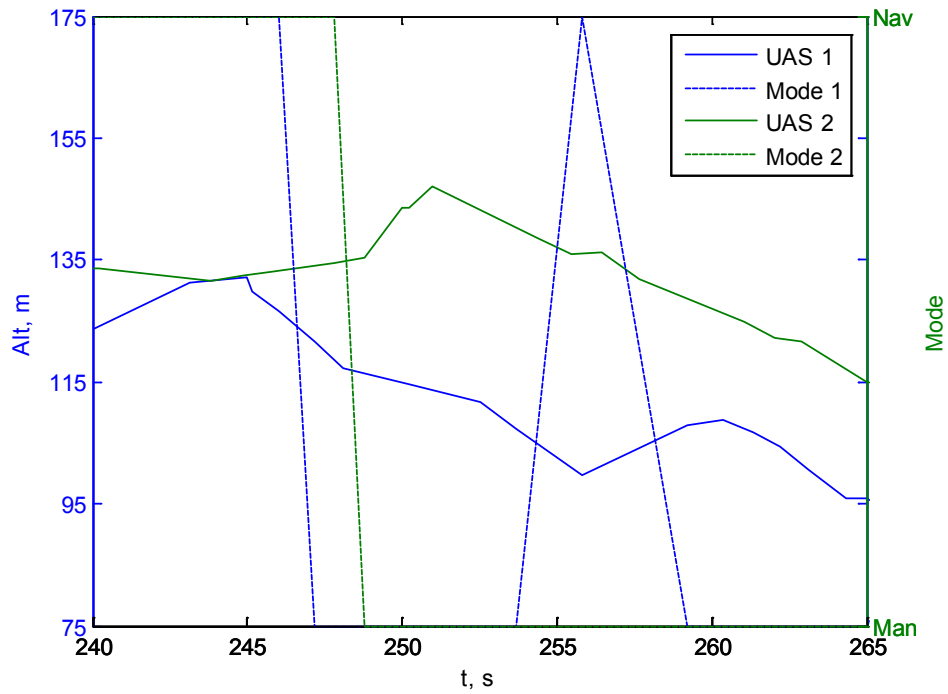**Figure 4-55: Flight Test Head-on Encounter 4 Trajectories**

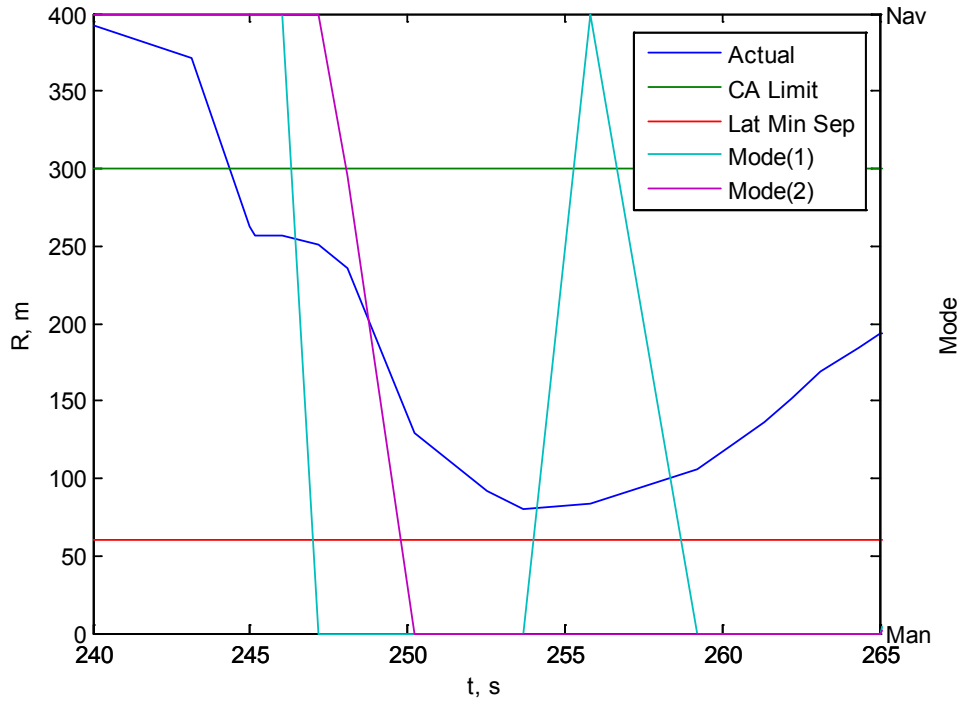**Figure 4-56: Flight Test Head-on Encounter 4 Altitude**

Figure 4-57: Flight Test Head-on Encounter 4 Range

110

Altitude and range are shown in Figures 4-56 and 4-57, respectively. The altitude responses display consistent patterns with their associated commands. BATCAM 1 behaves as expected; it is commanded to pitch down and responds by losing altitude. BATCAM 2 is commanded to pitch up and initially begins gaining altitude. However, it eventually loses altitude even though it maintains airspeed. Even with this loss, the BATCAMs are still separated in altitude by 33 m and successfully surpass the minimum 20 m of separation. The rapidly changing dynamics are difficult to characterize because data is recorded at a slow rate, as much as five seconds between points. The pitch and airspeed responses are shown in Figures 4-58 and 4-59, respectively.
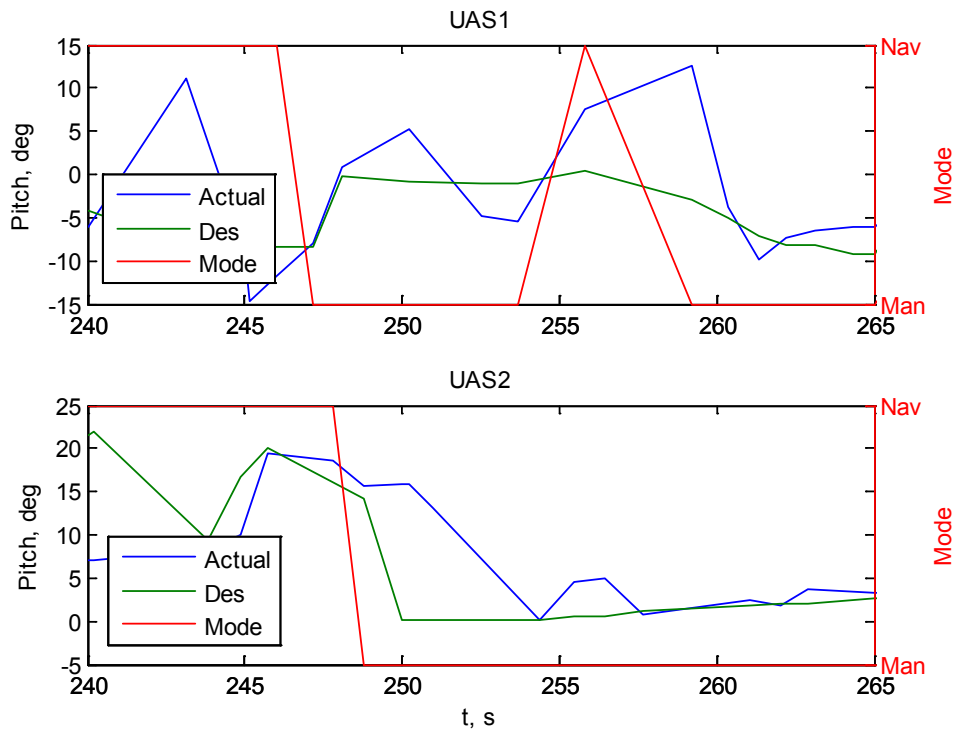

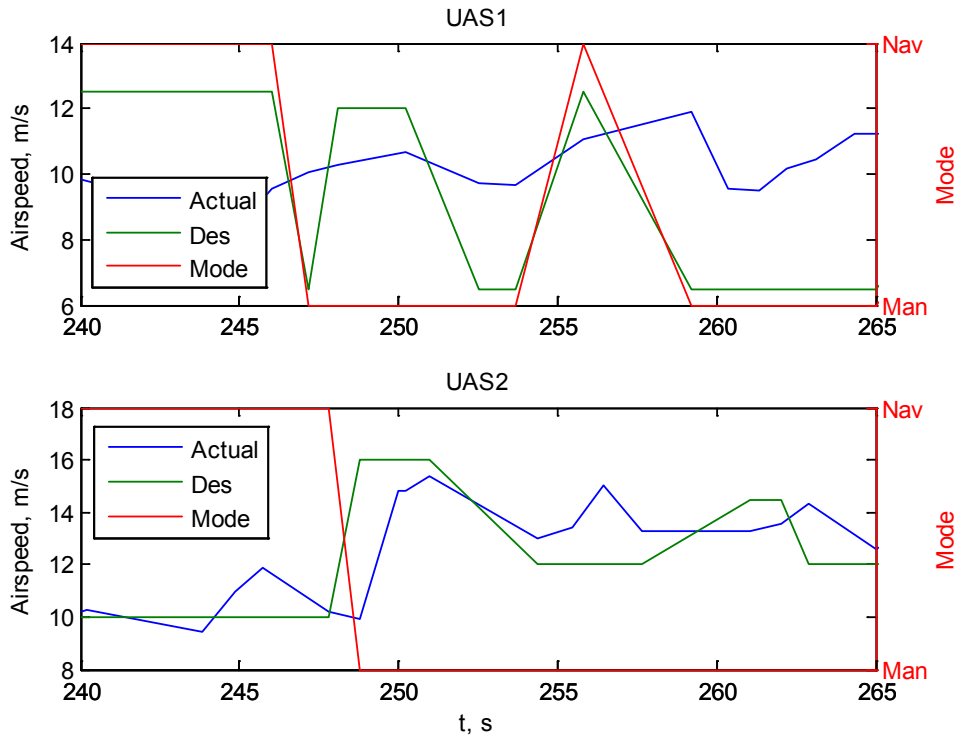**Figure 4-58: Flight Test Head-on Encounter 4 Pitch Response**

Figure 4-59: Flight Test Head-on Encounter 4 Airspeed Response
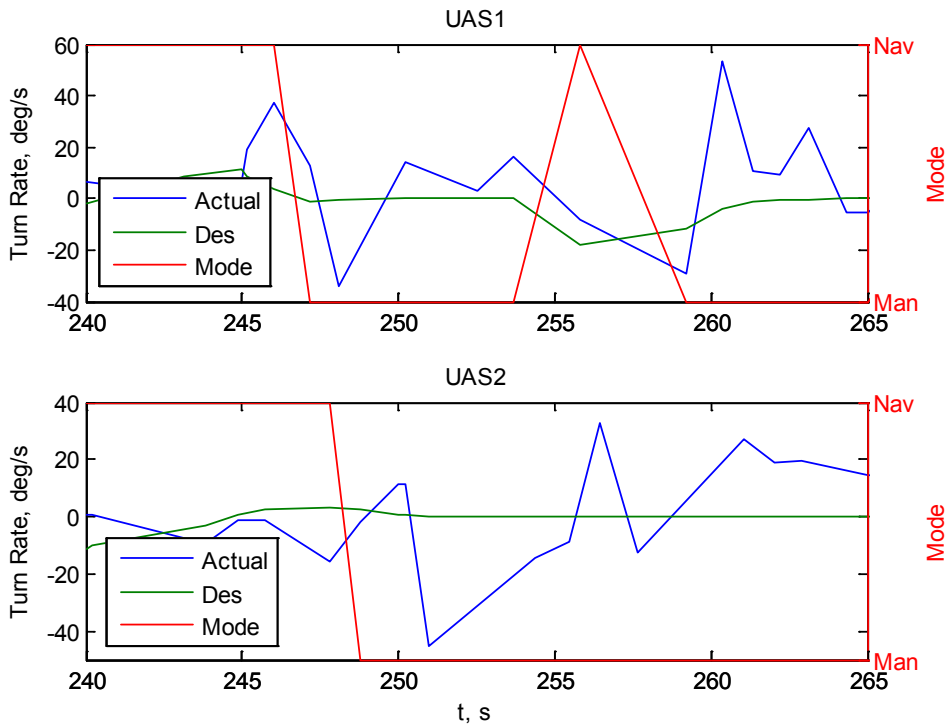

Figure 4-60: Flight Test Head-on Encounter 4 Turn Rate Response

112

The trajectories in Figure 4-55 clearly show both BATCAMs receive positive turn rate commands. Because the measurements for turn rate are obtrusively noisy and infrequent, this is not apparent in the turn rate data as shown in Figure 4-60.

Table 4-6: Flight Test Statistics

	Flight					
	Head-on				Approach	
Parameter	**1**	**2**	**3[1]**	**4**	**1**	**2**
Time of Minimum Lateral Separation	676.74 s	785.17 s	343.88 s	253.67 s	567.76 s	750.95 s
Minimum Lateral Separation	13.19 m	42.04 m	255.22 m	80.41 m	233.97 m	45.72 m
Time of Minimum Slant Range	676.74 s	785.17 s	343.88 s	253.67 s	567.76 s	750.95 s
Minimum Slant Range	24.24 m	45.11 m	255.34 m	86.98 m	236.83 m	47.32 m
UAS1 Alt at $T_{min,LS}$	117.00 m	130.50 m	124.33 m	107.17 m	101.83 m	134.50 m
UAS2 Alt at $T_{min,LS}$	137.33 m	146.83 m	132.20 m	140.33 m	138.54 m	122.28 m
UAS1 Alt at $T_{min,SR}$	117.00 m	130.50 m	124.33 m	107.17 m	101.83 m	134.50 m
UAS2 Alt at $T_{min,SR}$	137.33 m	146.83 m	132.20 m	140.33 m	138.54 m	122.28 m

1. Virtual Cockpit application displayed an error and closed resulting in data loss. Aircraft initiated "lost-comm" mode.

Statistics for the two encounters discussed above and the remaining encounters flown are in Table 4-6. Plots for the remaining flights are in Appendix F. Head-on, Flight 1, resulted in good geometry, but only BATCAM 1 entered CA mode. Flight 3 of the Head-on encounters showed promising geometry and both BATCAMs began performing avoidance maneuvers, but Virtual Cockpit returned an

error and shut down. This caused "lost comm" mode in the autopilots and they were commanded to return to their rally points. Only the initial maneuver and commands was recorded. This error was an anomaly, and has never before been seen in flight test or bench testing.

The first Approaching encounter was an excellent encounter for command generation and response, but the BATCAMs reacted much more aggressively than anticipated. BATCAM 1 turned nearly 180 deg and the aircraft never made a close approach at or near the minimum separation. These maneuvers can be partially attributed to wind conditions during the flight. Further investigation into the conversion of avoidance algorithm commands to Kestrel commands might alleviate some aggressive maneuvering for these particular aircraft. Additionally, the proportional navigation gains can be tuned for the aircraft if it is determined that they are accurately tracking the commands.

3.3. Flight Test Summary

Flight test validated the collision avoidance algorithm's ability to perform collision detection and avoidance maneuver generation. The BATCAM aircraft were able to respond to and avoid a minimum separation volume around each other as a result of the CA algorithm and in the presence of environmental and system uncertainty. The algorithm not only generated guidance commands, it provided system-specific flags that initiated autonomous mode switching with the human operator completely out of the loop. Flight test also confirmed successful integration of the algorithm into the BATCAM system, although improvements need to be made.

Flight test showed that turn rate and airspeed commands were effectively commanded and sufficiently tracked to provide lateral separation. Pitch commands that are responsible for altitude changes did not, however, provide vertical separation. Assumptions made in order to use pitch commands were not valid for this particular aircraft and resulted in undesirable vertical responses. Aircraft telemetry data was sufficient to detect potential losses of separation and issue successful avoidance commands, but was often excessively noisy and infrequent.

V. Conclusions and Recommendations

1. Chapter Overview

Chapter V provides a discussion of the collision avoidance system's development, application, and testing, as well as a "big picture" examination of current CA topics. Many aspects of this application and testing are specific to the available hardware and software, but considerations were always included throughout the design and evolution of the algorithm for other systems and missions. Military and civil applications were considered because, ultimately, the technologies related to this research will be far-reaching.

2. Conclusions of Research

A collision avoidance algorithm was developed and successfully implemented in a multi-vehicle miniature unmanned aircraft system. This algorithm was conceived after an extensive literature review of current conflict detection and resolution theories and methods with an attempt to capture the benefits of those methods and apply them in a single algorithm. The detection portion of the algorithm, based on geometric methods, has inherent simplifications (i.e. nominal trajectory projections) that are more robustly addressed in other approaches (e.g. probabilistic methods) but are overcome by its simplistic application to many systems and threat environments. No pre- or post-processing is required to represent the overall threat environment, and only tuning of navigation constants are required for different platforms. Novel developments include an approach to provide spatial awareness of all threats in a global sense to each vehicle in the cooperative system. Algorithm interfaces allow both cooperative and non-cooperative

inputs to be processed simultaneously and provide a truly global consideration of the environment only limited by sensing capability. The avoidance component of the algorithm is coupled with the detection algorithm to provide autonomous, continuous, and reactive commands to any collision encounter without a need for hard-coded threat prioritization or scripted maneuvers. Simple maneuver coordination logic is applied in the vertical dimension for the direction of the command. All other commands are completely autonomous and governed by the guidance law.

Effects of uncertainties in the environment and the host-system are mitigated by defining separation volumes that are sufficiently large and by commanding maneuvers in multiple dimensions for separation. Multi-dimensional commands provide a layer of redundancy in that the algorithm is always trying to achieve two independent separation distances. If one fails, the other is still active.

3. Significance of Research

This research resulted in the first known flight tests of a multiple-vehicle, global, three-dimensional CA algorithm. Miniature unmanned aircraft were placed in dynamic, real-world encounters and responded to autonomously generated avoidance commands. The algorithm provides an autonomous CA capability for any encounter geometry and is not limited to any particular system or sensing capability. This algorithm evolves from contemporary and prevalent research areas focused on conflict detection and resolution, sense and avoid, and CA of manned aircraft, robotics, and unmanned technologies. Advantages of certain methods were exploited and an amenable approach was taken while addressing limitations. A mixture of theoretical fields were combined to develop

117

the final algorithm: robotics (e.g. collision cone approach), homing guidance (e.g. proportional navigation), and airspace management (e.g. separation criteria). The significance of the results is not just an assessment of the effectiveness of this particular algorithm, but the consideration of all aspects of deconflicting unmanned systems, i.e. sensing and measurement requirements, system integration, control commands and tracking, and necessary test procedures and equipment to evaluate the effectiveness.

The ability to avoid obstacles and objects is of utmost concern for unmanned systems and the missions they are assigned. Weaponized unmanned aircraft will proliferate as technology advances and an alternative to endangering our warfighters becomes reliable and readily available. Their numbers and missions will expand and inevitably make a CA capability a system requirement. CA is already a requirement for any unmanned system requesting access to the NAS as stated in FAA regulations [35]. The specifics of this requirement are not defined and will not be defined for some time. As the author is writing this thesis, a bill has been introduced in the United States Congress with provisions for defining a timeline and requirements for unmanned system integration into the NAS [36]. The Federal Aviation Administration and the users requesting access to the NAS (Department of Defense, Department of Homeland Security, Customs and Border Protection, and Commercial and Private Users) are struggling to find a resolution between safety concerns and the desire for rapid integration and are willing to spend hundreds of millions of dollars to find a solution. CA is an integral part of unmanned system access to the NAS and a "sense and avoid" capability is arguably the most important, and unfortunately, ambiguous necessity for approval.

4. Recommendations for Action

Bench test and flight test activities and results have revealed several items pertaining to the CA implementation, not the algorithm itself, which require immediate attention. Communication delay issues resulting from a build-up of commands due to processing limitations, not from a constant transmission delay, need to be addressed. This is a UAS-specific characteristic external to the algorithm that needs to be precisely located in the communication chain. Previous testing with the ANT laboratory BATCAM system experienced similar problems and an application-specific solution was implemented [37]. This solution involves pulsing commands so a build-up of communication packets does not occur. This is not compatible with a CA algorithm that sends more commands to more aircraft. Due to the small amount of time available to deconflict the aircraft in a collision encounter, any attempt to clear unprocessed packets from a buffer would only increase the collision potential. Quantization is another option to alleviate increasing delays in the communication channels. This would reduce the number of packets sent, but would result in larger commands being generated later in the encounter because of smaller resultant command responses in the beginning of the encounter. The proportional navigation guidance commands are proportional to the rate of change of the collision cone boundaries, which are small at larger ranges and continuously increase as the range decreases. If the commands do not continuously grow proportional to the bound rates and are quantized based on a pre-determined delta value, deconfliction will occur later in the encounter timeline with larger magnitudes of resultant commands. Dynamic delta values in the quantization may lessen this effect, but is out of scope of this thesis.

One possible improvement to the increasing communication delay is aggregating the three commands for each aircraft into a single packet, thereby, reducing the number of packets sent. The current interface between the CA algorithm and the autopilot does not support this. Commands are currently sent separately to each BATCAM in a sequential manner. Action should be taken to determine if the Kestrel autopilot communication interfaces support this modification. It should also be determined whether or not the Kestrel autopilot and Virtual Cockpit GCS support separate communication links for telemetry and control packets. This would divide the two-way communication traffic and possibly hasten transmissions.

The small angle of attack assumption and the assumption that the dynamic delay between pitch changes and altitude changes were proven to be inaccurate in flight tests with environmental uncertainty. These assumptions were applied in order to convert the algorithm flight path angular rate command to the available Kestrel pitch command. The most current version of the Kestrel autopilot includes a climb rate command that can be more directly calculated from flight path rate commands. Regrettably, the reason turn rate command recording was not available is because turn rate commands may not be available in that same version of the autopilot as explained in discussions with Procerus engineers. It seems compatibility between all desired commands cannot be acquired simultaneously. This is an unfortunate side-effect of an evolving autopilot design and the users' attempt to develop their own applications.

5. Recommendations for Future Research

In any system with uncertainty, either inherent in the system or as a result of external inputs, there is no guarantee of desired results; there is only a probability that they will be achieved successfully. This applies to CA and the efforts to maintain minimum separation. The likelihood that this thesis' CA system will deconflict aircraft and maintain minimum separation could be quantitatively determined in Monte Carlo simulations prior to additional flight tests and implementation into more systems. Parametric studies should be completed to determine, based on the probabilities from each Monte Carlo run, what separation volume lateral and vertical distances would result in the highest likelihood of maintaining separation within some constraints. An independent variable in these studies is the required probability of maintaining separation. Depending on the system, the mission, and the user, should the separation be maintained 90%, 95% or 99% of the time? For inexpensive, expendable systems, a simple, less reliable CA system may be desired. For a complex, expensive system such as a Global Hawk UAS, an accurate and dependable CA system will be required.

Furthermore, in cases where separation is lost, what is the probability that the aircraft will actually collide? These statistics can be gleaned from the same Monte Carlo runs as for loss of separation studies, but more iterations may be required to have a statistically sufficient amount of encounters. These types of studies could be performed for a variety of algorithms or algorithm variations. Analyses such as these will be required by the FAA for candidate sense and avoid systems prior to unmanned system integration into the NAS.

The algorithm developed in this research was intentionally constructed in a generic manner for compatibility with many systems. However, when applied to a specific system, certain aspects of the algorithm can be tailored for enhanced performance. The most obvious variables were discussed in this thesis: the separation volume, the maximum CA range, and the proportional navigation constants. Further modifications could be made for increased probability of maintaining separation. For example, all commands in three dimensions are applied for every collision encounter. It may be determined in simulations or flight test that particular encounter geometries require only one type of command or require more benign commands. A command selection algorithm could be designed with this information and energy savings or less impact on the UAS mission could be achieved. Adaptive proportional gains could be derived that result in an optimal avoidance maneuver. Techniques such as these have been researched, and were discussed in Chapter II, but not for an algorithm such as this, i.e. global, three-dimensional, and depending on the optimality condition, cooperative. The effectiveness of any modification will depend on the aircraft and the threats it will encounter.

This algorithm relied on the Kestrel autopilot navigation mode to return the aircraft to waypoint following and their planned routes. Optimal trajectory generation could be used to return the aircraft to the planned routes before navigation mode is once again enabled. This would result in a CA and recovery algorithm that would both deconflict and restore the aircraft with minimal deviation from their original paths.

6. Summary

Fundamental theory was examined and further developed for application to the collision avoidance problem in this thesis, and an algorithm was designed, coded, and tested in ideal simulations. Application to an unmanned aircraft system, including algorithm development, user and system interface construction, and software-in-the-loop and hardware-in-the-loop testing was completed to validate the approach. Flight tests were conducted to assess the algorithm and system's performance in the presence of operational and environmental uncertainty. The results of all of the above efforts and events were discussed in detail along with high level discussions about current issues related to the collision avoidance topic.

Appendix A: Collision Avoidance Algorithm/Virtual Cockpit Interface

Inputs:

$$\begin{bmatrix} \vec{P}^i_{UAS} \\ V^i_{UAS} \\ \psi^i_{UAS} \\ \gamma^i_{UAS} \\ a^i_{UAS} \\ \dot{\psi}^i_{UAS} \\ \dot{\gamma}^i_{UAS} \end{bmatrix} \quad \text{where } i \text{ indicates the } i^{\text{th}} \text{ UAS}$$

$$\vec{P}^i_{UAS} = \begin{Bmatrix} X^i_{UAS} \\ Y^i_{UAS} \\ Z^i_{UAS} \end{Bmatrix} = \text{UAS Local-Level Referenced Position (can be derived from GPS}$$

Latitude, Longitude, and Altitude)

V^i_{UAS} = UAS Speed (preferably groundspeed from GPS, airspeed only if groundspeed not available)

ψ^i_{UAS} = UAS Ground Track (from GPS, heading is only acceptable if winds are accounted for)

γ^i_{UAS} = UAS Vertical Flight Path Angle (FPA) (can be derived, preferably, from velocity components, or assumed to equal, cautiously, aircraft pitch angle for small angle of attack)

a^i_{UAS} = UAS Translational Acceleration (can be assumed equal to the body X-axis acceleration for small angle of attack)

$\dot{\psi}^i_{UAS}$ = UAS Turn Rate

$\dot{\gamma}^i_{UAS}$ = UAS Rate of Change of Vertical Flight Path Angle (can be derived from climb acceleration, or from both γ and body Z-axis acceleration)

Table A-1: Algorithm to Kestrel Autopilot Variable Matrix, Virtual Cockpit 2.4

	ICD	Kestrel	
		Label	Variable Name
Position	X	F15 (F13)	GPS East Pos Estimate (Measured)
	Y	F16 (F14)	GPS North Pos Estimate (Measured)
	Z	F20	GPS Altitude
Speed	V	F18 (F17)	Ground Speed Estimate (Measured)
Ground Track	Ψ	F19 (F7)	GPS Heading (Filtered)
Vertical FPA	γ	(F2)	(Theta)
Acceleration	a	F52	Ax (accelerometer)
Turn Rate	ψ_{dot}	F33	Heading Rate
Rate of Change of FPA[1]	γ_{dot}	F54 and F18 (F17)	Az and Speed

Values in parentheses are alternates.

1. Assumption: $\dot{\gamma} \approx \dfrac{-A_z}{V}$

Table A-2: Algorithm to Kestrel Autopilot Packet Variable Matrix, Virtual Cockpit 2.4

ICD		Kestrel Communications			
Name		Packet	Index	Variable Name	Units
Position	X	248	14	GPS Lon	deg
		248	22	GPS Lon home	deg
	Y	248	8	GPS Lat	deg
		248	18	GPS Lat home	deg
	Z	248	2	GPS Alt	(m+1000)*6
Speed	V	248	0	GPS Velocity	(m/s+10)*20
Ground Track	ψ	248	4	GPS Heading	rad*1000
Vertical FPA	$\gamma = \sin^{-1}\left(\dfrac{\dot{h}}{V}\right)$	249	76	Actual Climb Rate	m/s*300
Acceleration	a	18	40	Ax	m/s^2*1000
Turn Rate	ψ_{dot}	249	10	Heading Rate	rad/s*1000
Rate of Change of FPA[1]	$\dot{\gamma} \approx \dfrac{-A_z}{V}$	18	44	Az	m/s^2*1000

1. Assumes Z-axis is down

Conversions:

X = (Lon - Lon Home) * dLon2m [Lon in deg]
Y = (Lat - Lat Home) * dLat2m [Lat in deg]

aLat = (Lat + Lat Home)/2
dLon2m = 111415.13*cos(aLat) – 94.55*cos(3*aLat)
dLat2m = 111132.09 – 566.05*cos(2*aLat) + 1.2*cos(4*aLat)

Outputs:

Algorithm:

$$\begin{Bmatrix} \dot{\psi}_{com} \\ \dot{\gamma}_{com} \\ a_{com} \end{Bmatrix} = \text{turn rate command, rate of change of vertical FPA command, acceleration}$$

command

Kestrel Autopilot:

1. Desired Turn Rate
2. Desired Pitch Angle
3. Desired Airspeed

$$\begin{Bmatrix} \dot{\psi}_{des} \\ \theta_{des} \\ V_{des} \end{Bmatrix} = \begin{Bmatrix} \dot{\psi}_{com} \\ \theta_{des} + \dot{\gamma}_{com} dt \\ V_{des} + a_{com} dt \end{Bmatrix} \text{ where small angle of attack is assumed, and } dt \text{ is the time step}$$

between command transmittals ***must be set according to command transmissions***

Table A-3: Algorithm to Kestrel Autopilot Command Packet Matrix, Virtual Cockpit 2.4

Set Desired Value (Packet 231)	Kestrel Communications (Section 3.58)		
Command	Byte Index	Variable Name	Units
$\dot{\psi}_{des}$	27	Desired turn rate	rad/s
θ_{des}	7	Desired pitch	rad
V_{des}	23	Desired airspeed	m/s

Appendix B: Collision Cone Boundary Rates

The following definitions and nomenclature are repeated here for convenience from [7] and [27]. The variables $\dot{\eta}_1$ and $\dot{\eta}_2$ are the rates of change of the collision cone boundaries with respect to the line of sight.

N_{21}, Case 1:

$$\dot{\eta}_1 = \dot{\eta}_2 = [null]$$

N_{21}, Case 2:

$$\dot{\eta}_1 = \frac{\dot{A}}{\cos(\eta_1 + \zeta)v} - \tan(\eta_1 + \zeta)\frac{\dot{v}}{v} - \dot{\zeta}$$

$$\dot{\eta}_2 = \frac{-\dot{A}}{\cos(\pi - \eta_2 - \zeta)v} + \tan(\pi - \eta_2 - \zeta)\frac{\dot{v}}{v} - \dot{\zeta}$$

N_{21}, Case 3:

$$\dot{\eta}_1 = \dot{\eta}_1 \big|_{case2}$$

$$\dot{\eta}_2 = \frac{-\dot{A}}{\cos(-\pi - \eta_2 - \zeta)v} + \tan(-\pi - \eta_2 - \zeta)\frac{\dot{v}}{v} - \dot{\zeta}$$

N_{21}, Case 4:

$$\dot{\eta}_1 = \dot{\eta}_2 = 0$$

N_{22}, Case 1:

$$\dot{\tilde{\eta}}_1 = \dot{\tilde{\eta}}_2 = [null]$$

N_{22}, Case 2:

$$\dot{\tilde{\eta}}_1 = \frac{\dot{\tilde{A}}}{\cos(\tilde{\eta}_1 + \tilde{\zeta})v} - \tan(\tilde{\eta}_1 + \tilde{\zeta})\frac{\dot{v}}{v} - \dot{\tilde{\zeta}}$$

$$\dot{\tilde{\eta}}_2 = \frac{-\dot{\tilde{A}}}{\cos(\pi - \tilde{\eta}_2 - \tilde{\zeta})v} + \tan(\pi - \tilde{\eta}_2 - \tilde{\zeta})\frac{\dot{v}}{v} - \dot{\tilde{\zeta}}$$

N_{22}, Case 3:

$$\dot{\tilde{\eta}}_1 = \dot{\tilde{\eta}}_1\Big|_{case2}$$

$$\dot{\tilde{\eta}}_2 = \frac{-\dot{\tilde{A}}}{\cos\left(-\pi - \tilde{\eta}_2 - \tilde{\zeta}\right)v} + \tan\left(-\pi - \tilde{\eta}_2 - \tilde{\zeta}\right)\frac{\dot{v}}{v} - \dot{\tilde{\zeta}}$$

N_{22}, Case 4:

$$\dot{\tilde{\eta}}_1 = \dot{\tilde{\eta}}_2 = 0$$

where

$$p = R\Big/\sqrt{r^2 - R^2}\ ;\quad \mu = \beta - \theta$$

$$A = \frac{p\cos\mu + \sin\mu}{\sqrt{p^2 + 1}};\quad \zeta = \sin^{-1}\left(\frac{p}{\sqrt{p^2 + 1}}\right);\quad v = \frac{V_{UAS}}{V_{INT}}$$

$$\tilde{A} = \frac{p\cos\mu - \sin\mu}{\sqrt{p^2 + 1}};\quad \tilde{\zeta} = \pi - \zeta$$

$$\dot{A} = \dot{p}\cos\zeta\left(\cos\mu - A\sin\zeta\right) + \dot{\mu}\cos\zeta\left(\cos\mu - p\sin\mu\right)$$

$$\dot{\tilde{A}} = \dot{p}\cos\tilde{\zeta}\left(\tilde{A}\sin\tilde{\zeta} - \cos\mu\right) + \dot{\mu}\cos\tilde{\zeta}\left(\cos\mu + p\sin\mu\right)$$

$$\dot{\zeta} = \dot{p}\cos^2\zeta$$

$$\dot{\tilde{\zeta}} = -\dot{\zeta}$$

Appendix C: Ideal Simulation Plots
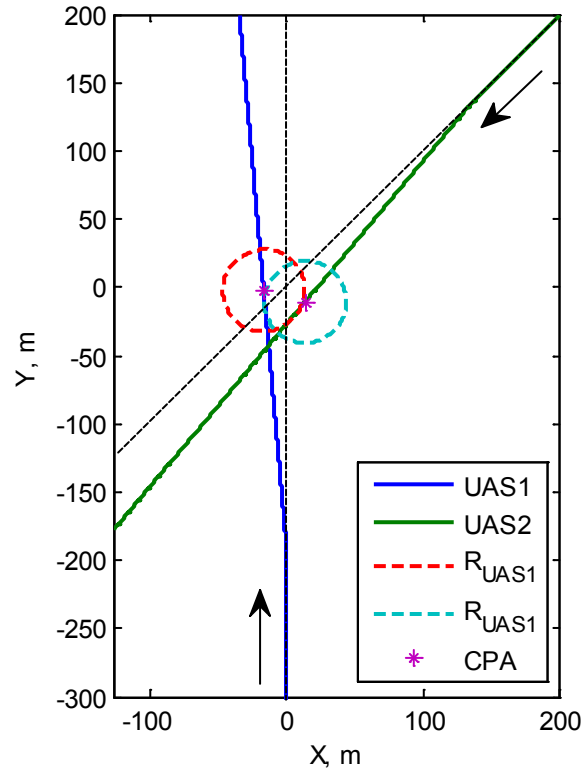
Figure C-1: Ideal Approaching Simulation Trajectories, CPA at 32.9 s
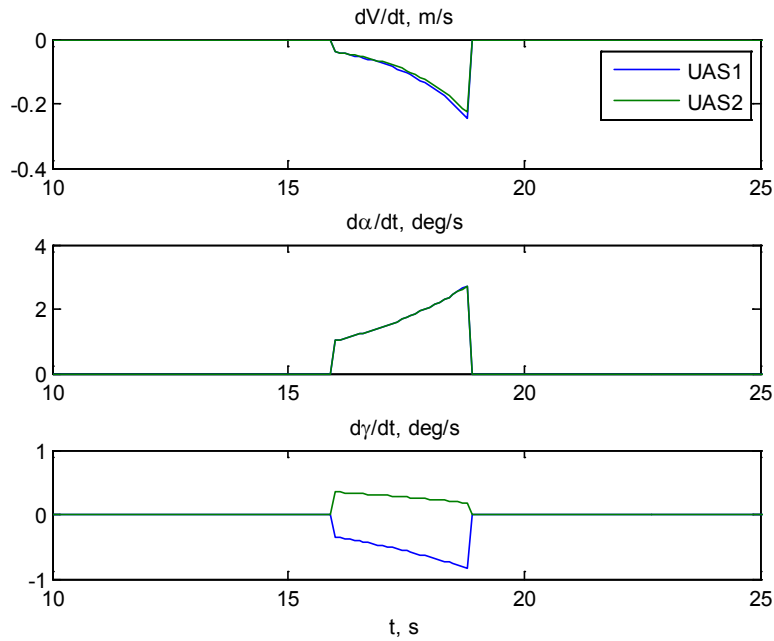
Figure C-2: Ideal Approaching Simulation Avoidance Algorithm Commands
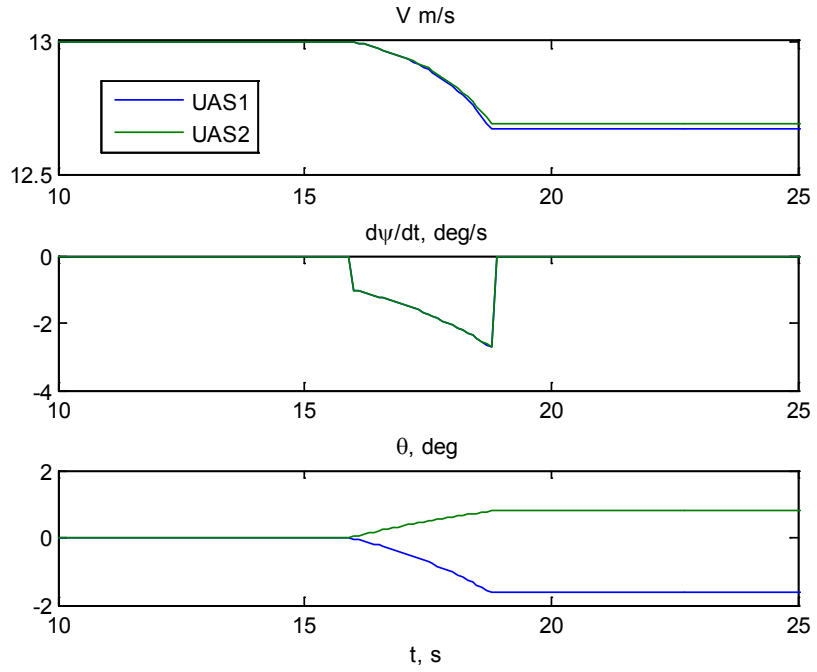
Figure C-3: Ideal Approaching Simulation Kestrel Commands
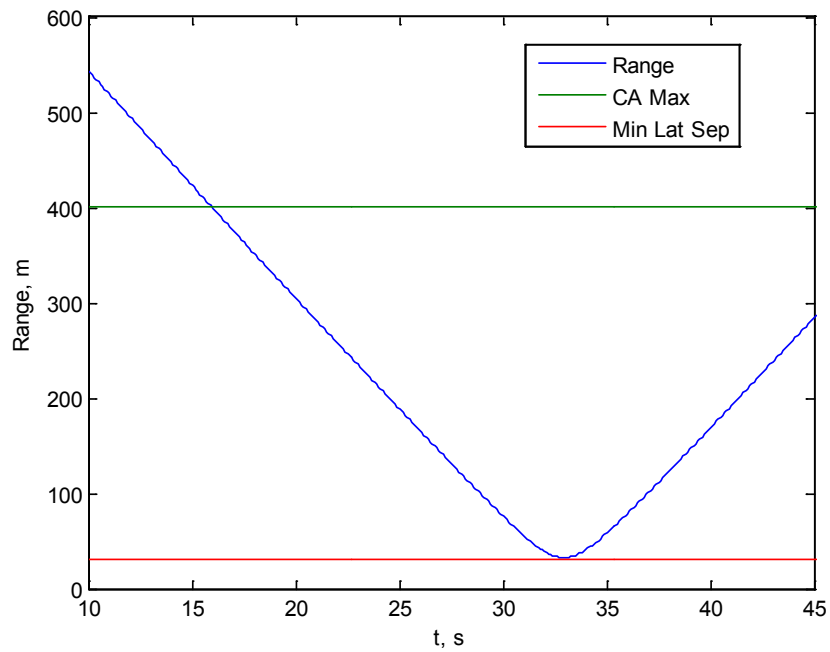
Figure C-4: Ideal Approaching Simulation Range
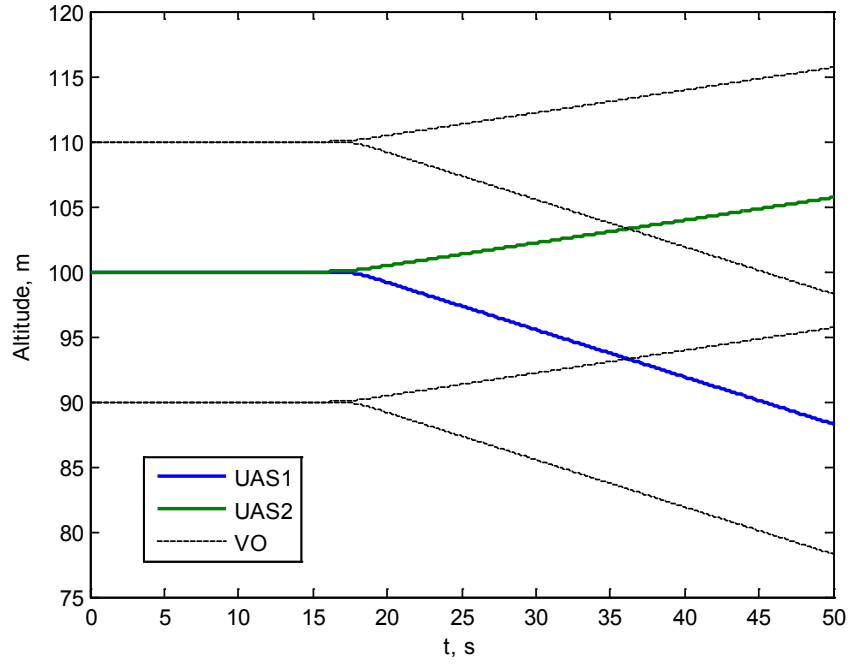
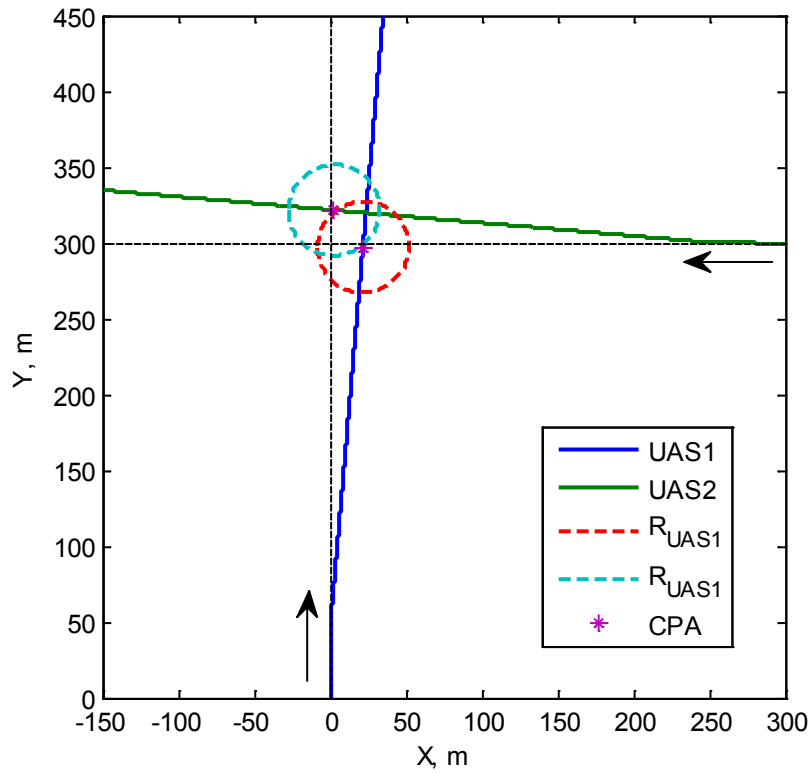**Figure C-5: Ideal Approaching Simulation Altitude**

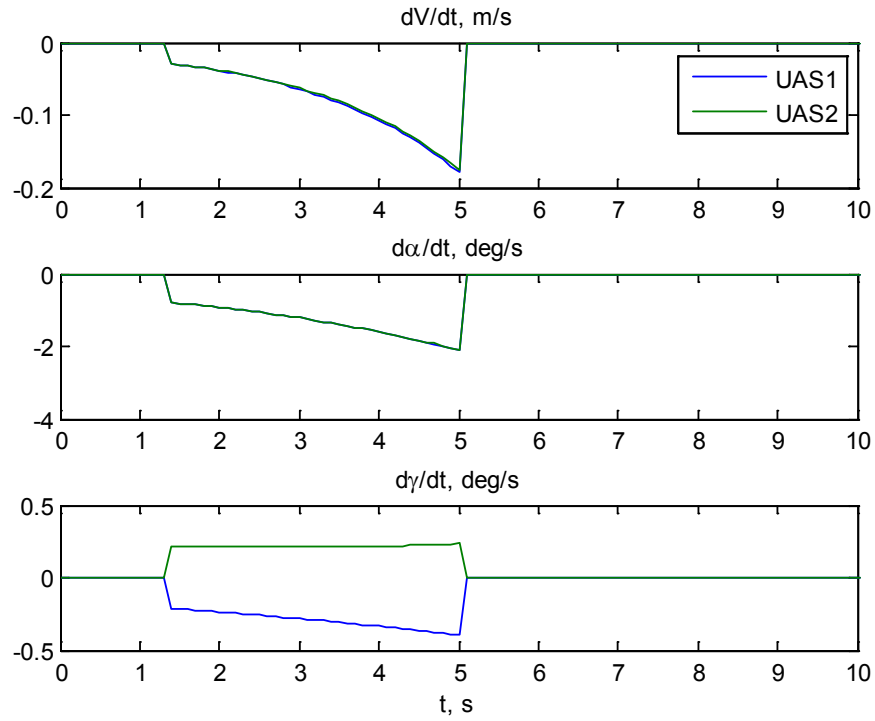**Figure C-6: Ideal Abeam Simulation Trajectories, CPA at 23.4 s**

131

Figure C-7: Ideal Abeam Simulation Avoidance Algorithm Commands
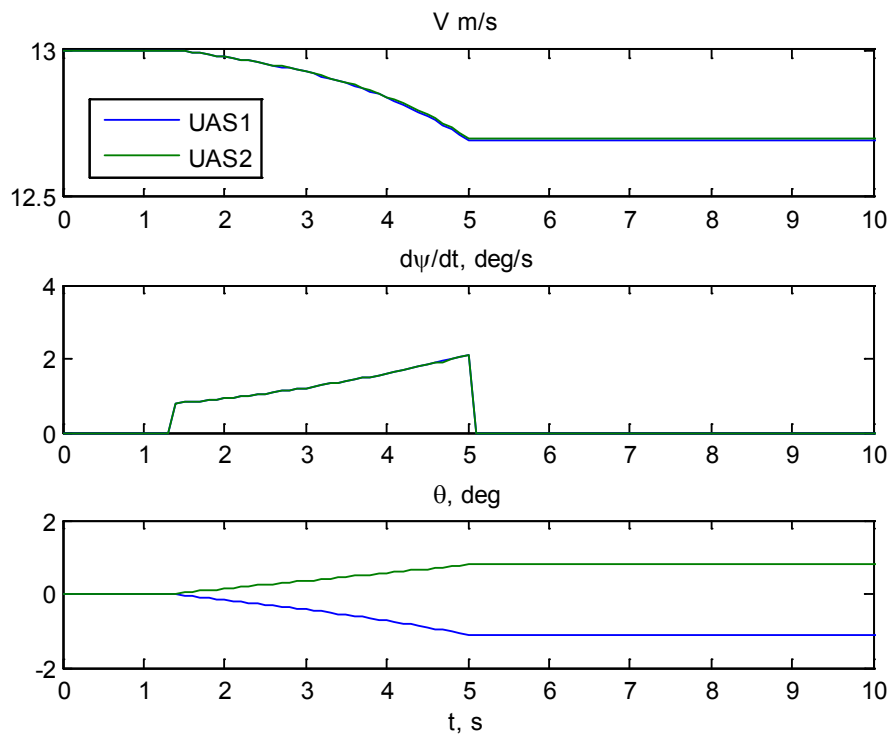
Figure C-8: Ideal Abeam Simulation Kestrel Commands
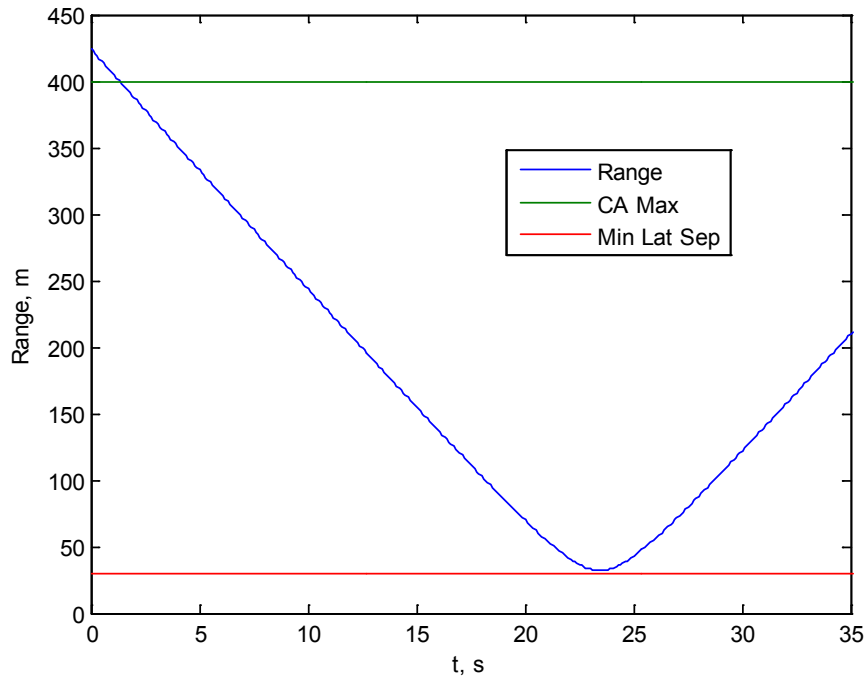
Figure C-9: Ideal Abeam Simulation Range
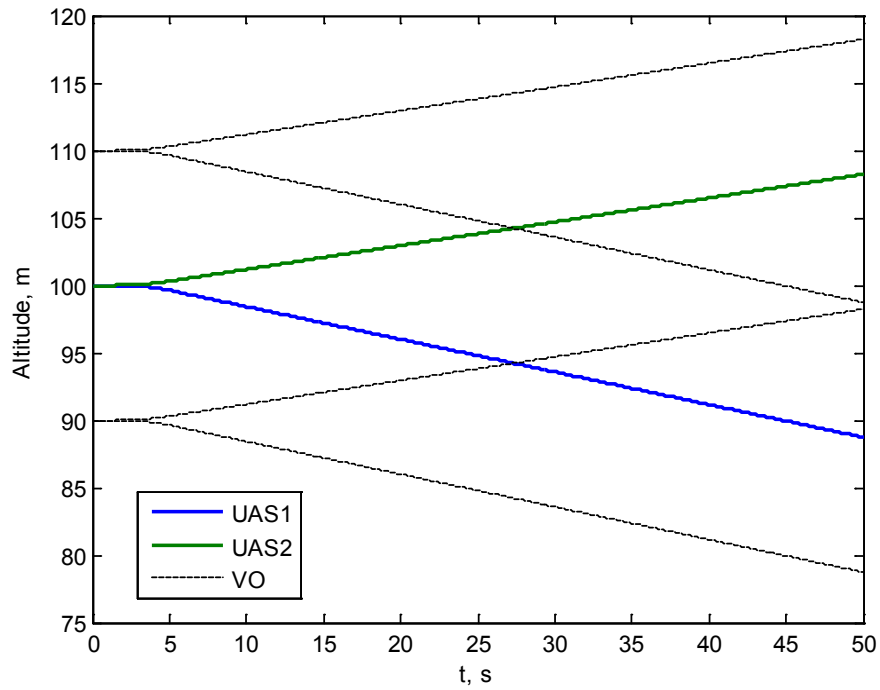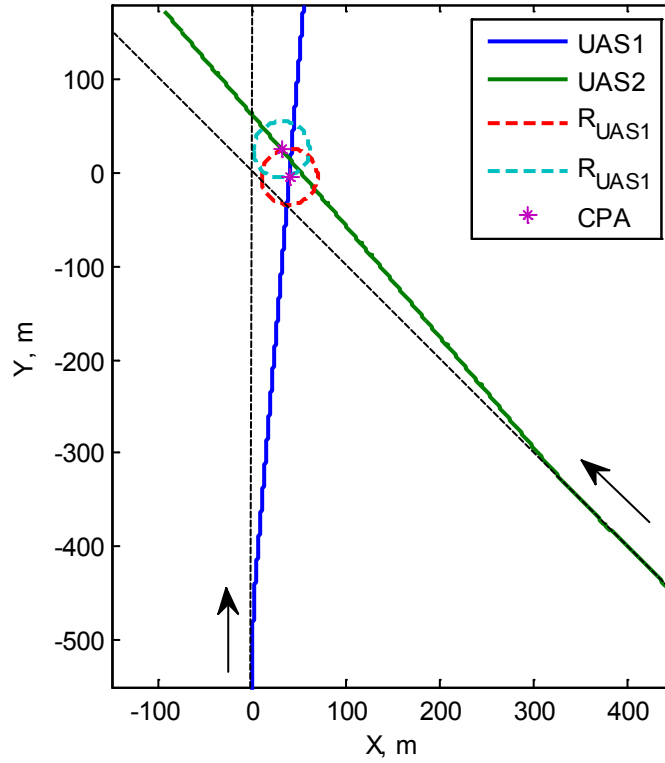

Figure C-10: Ideal Abeam Simulation Altitude

133

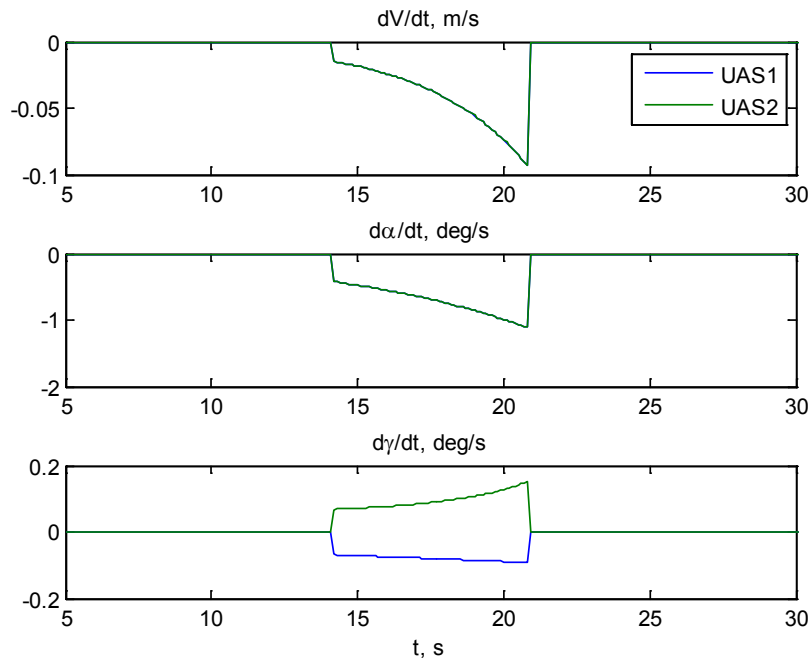**Figure C-11: Ideal Converging Simulation Trajectories, CPA at 55 s**

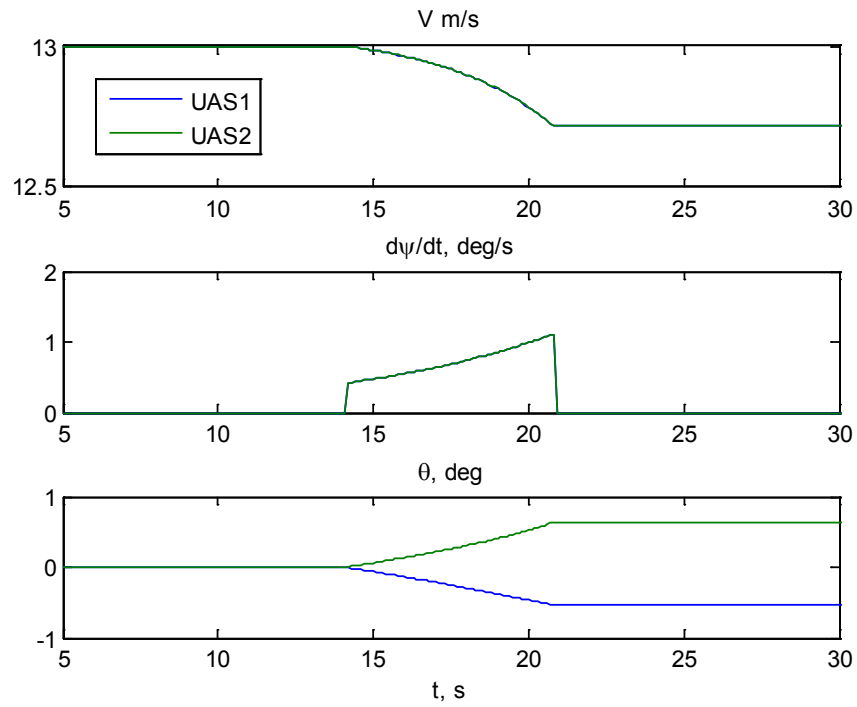**Figure C-12: Ideal Converging Simulation Avoidance Algorithm Commands**

134

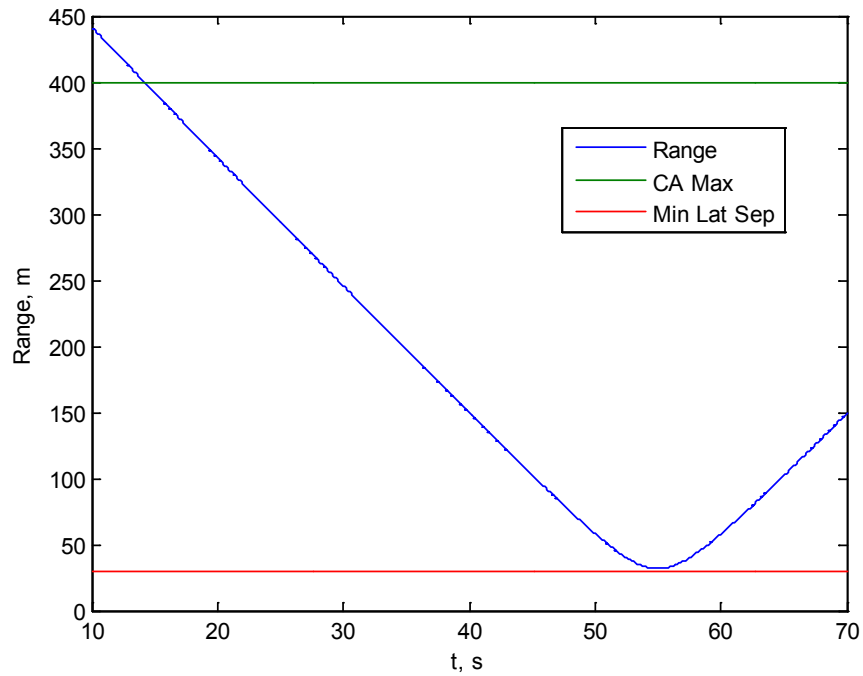**Figure C-13: Ideal Converging Simulation Kestrel Commands**

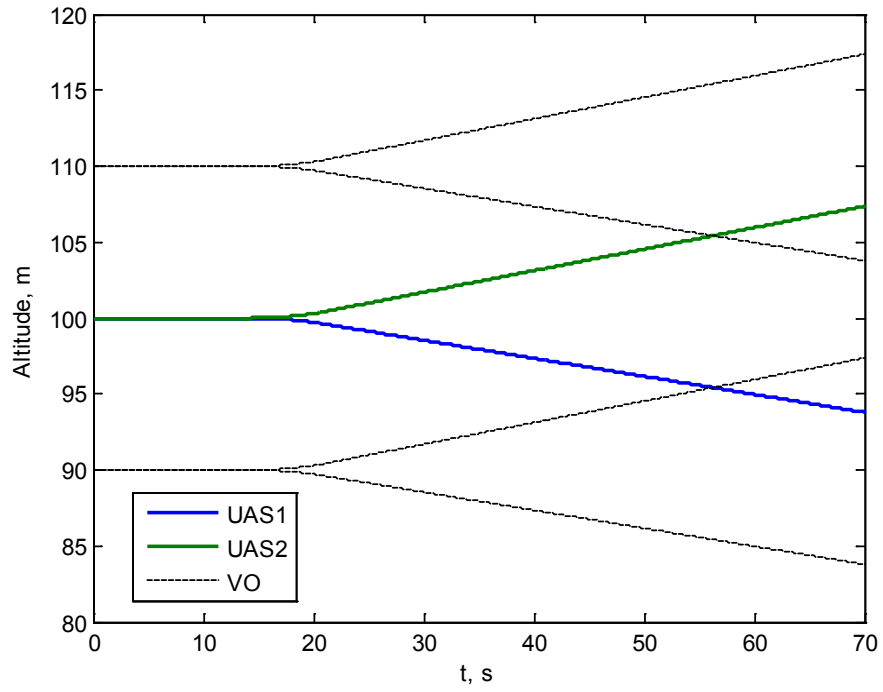**Figure C-14: Ideal Converging Simulation Range**

135

Figure C-15: Ideal Converging Simulation Altitude
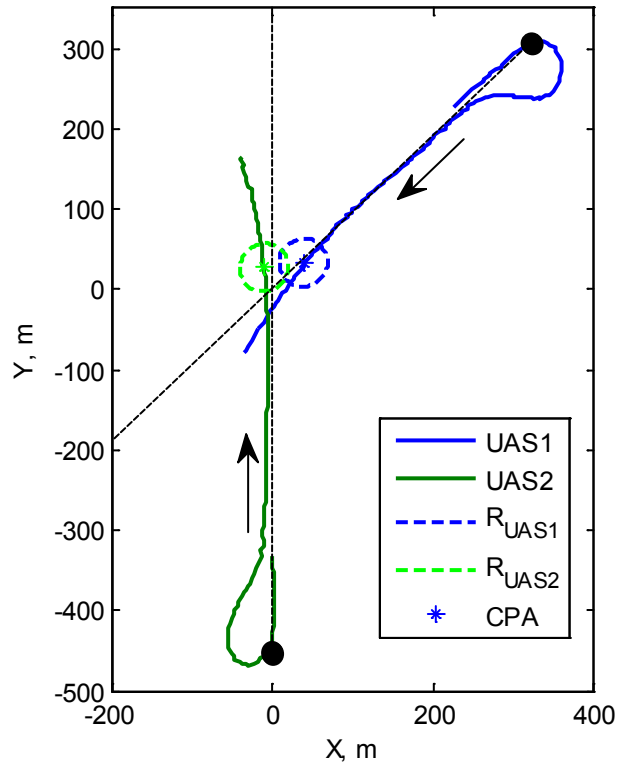
Appendix D: SIL Simulation Plots

Figure D-1: SIL Approaching Simulation Trajectories
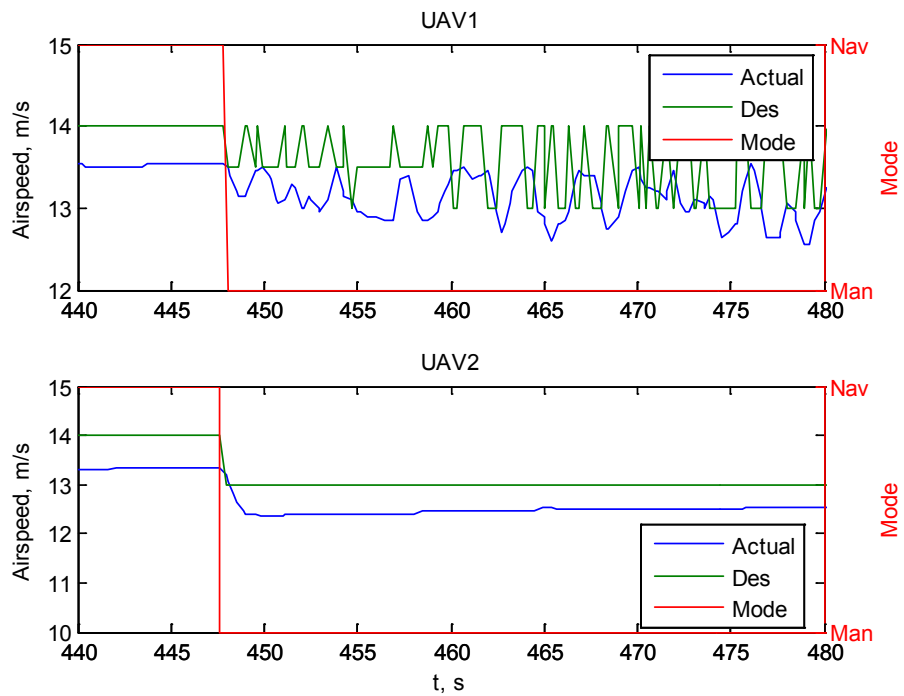

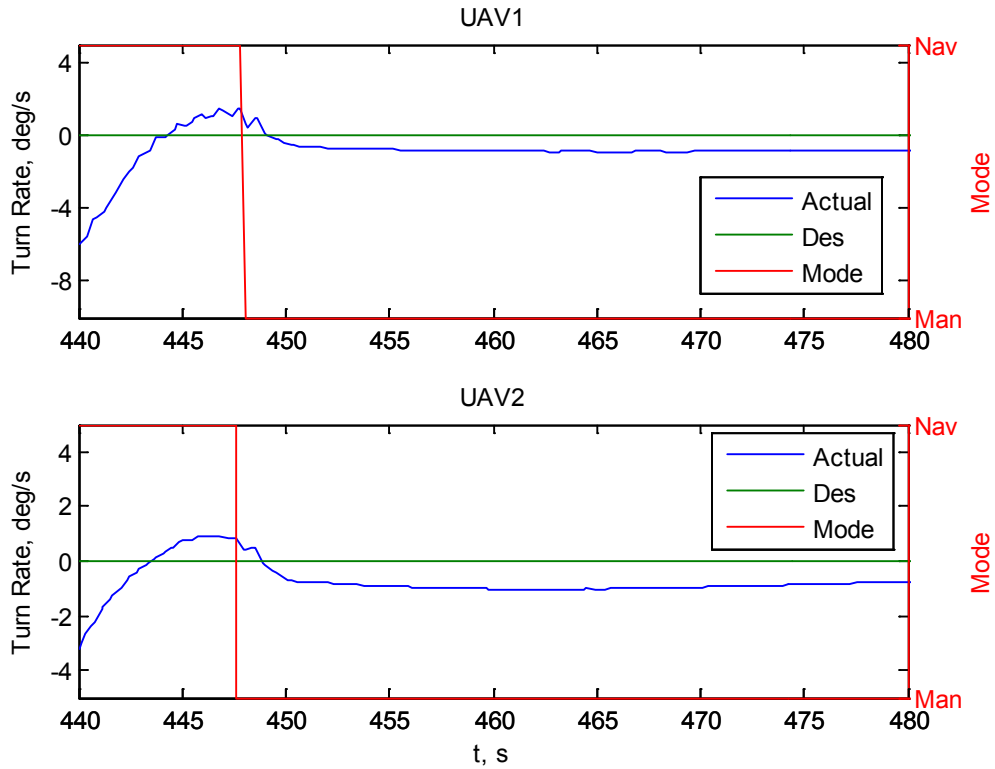**Figure D-2: SIL Approaching Simulation Airspeed**

Figure D-3: SIL Approaching Simulation Turn Rate
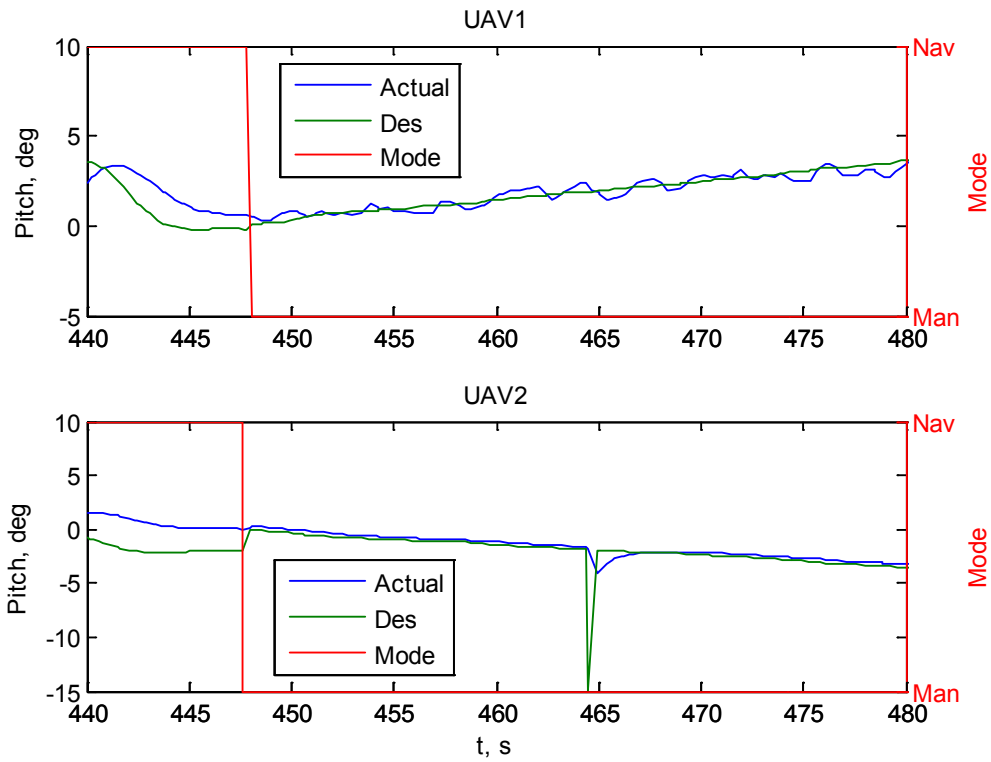
Figure D-4: SIL Approaching Simulation Pitch

138

Figure D-5: SIL Approaching Simulation Range

Figure D-6: SIL Approaching Simulation Altitude

139
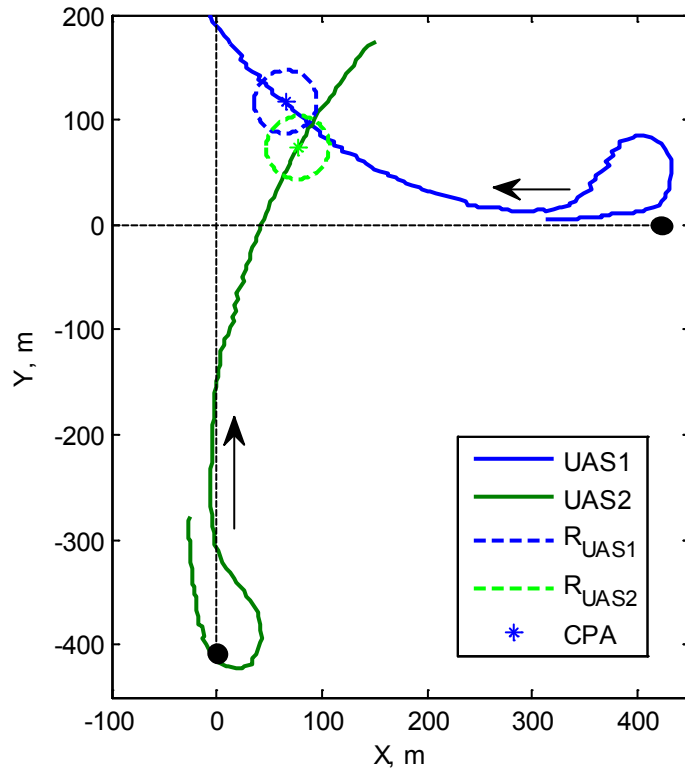
Figure D-7: SIL Abeam Simulation Trajectories
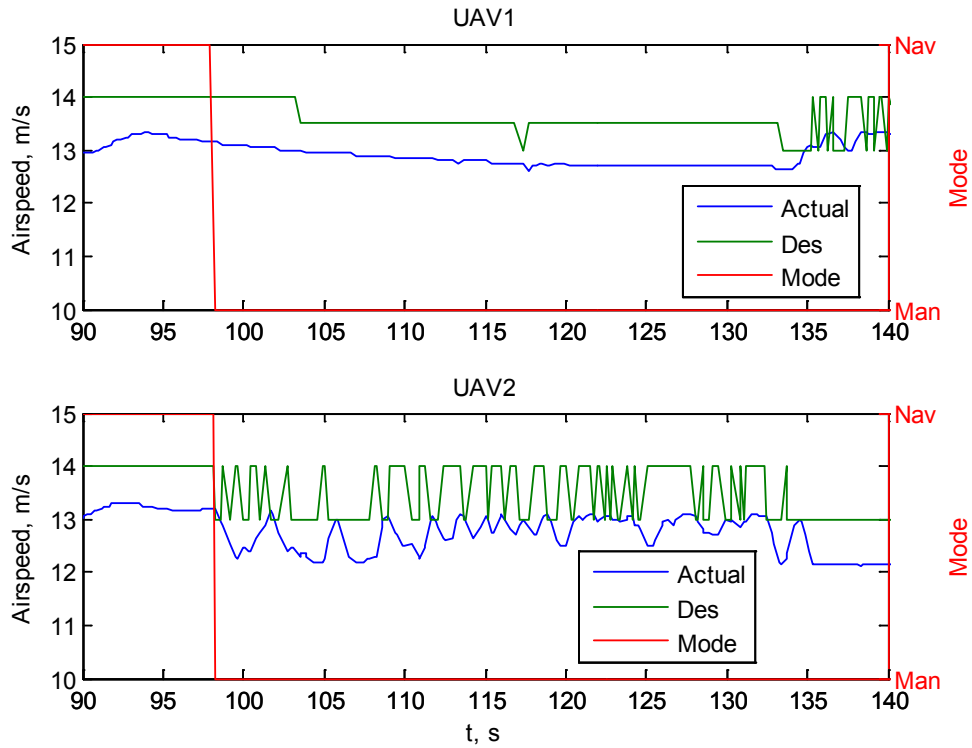
Figure D-8: SIL Abeam Simulation Airspeed

Figure D-9: SIL Abeam Simulation Turn Rate
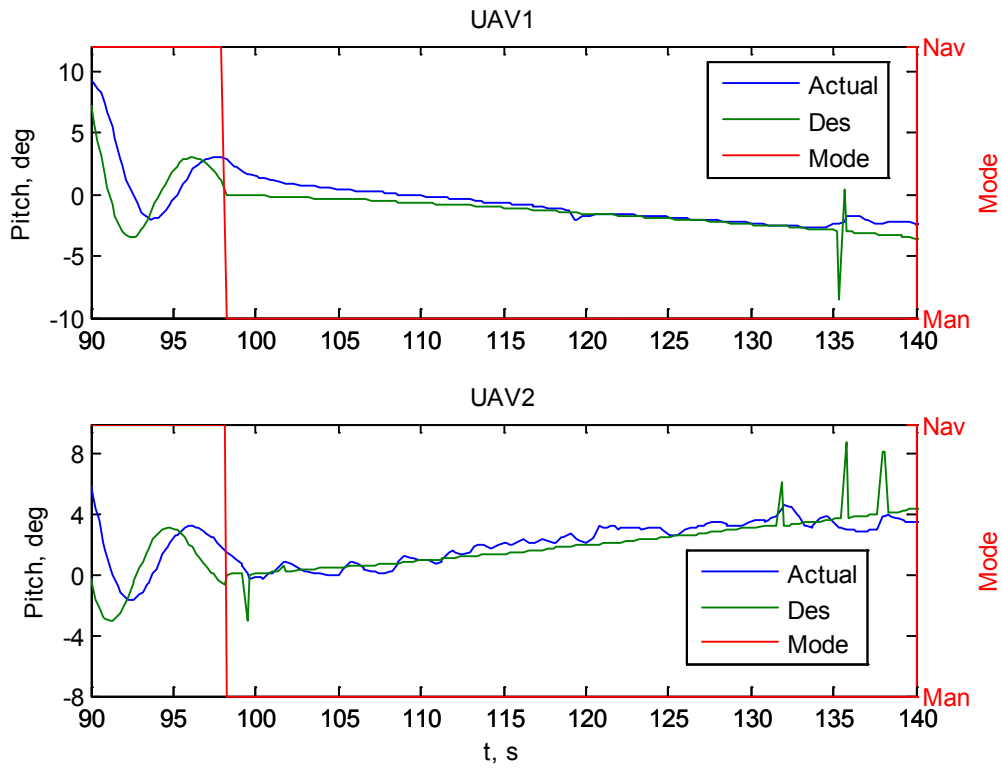
Figure D-10: SIL Abeam Simulation Pitch

Figure D-11: SIL Abeam Simulation Range
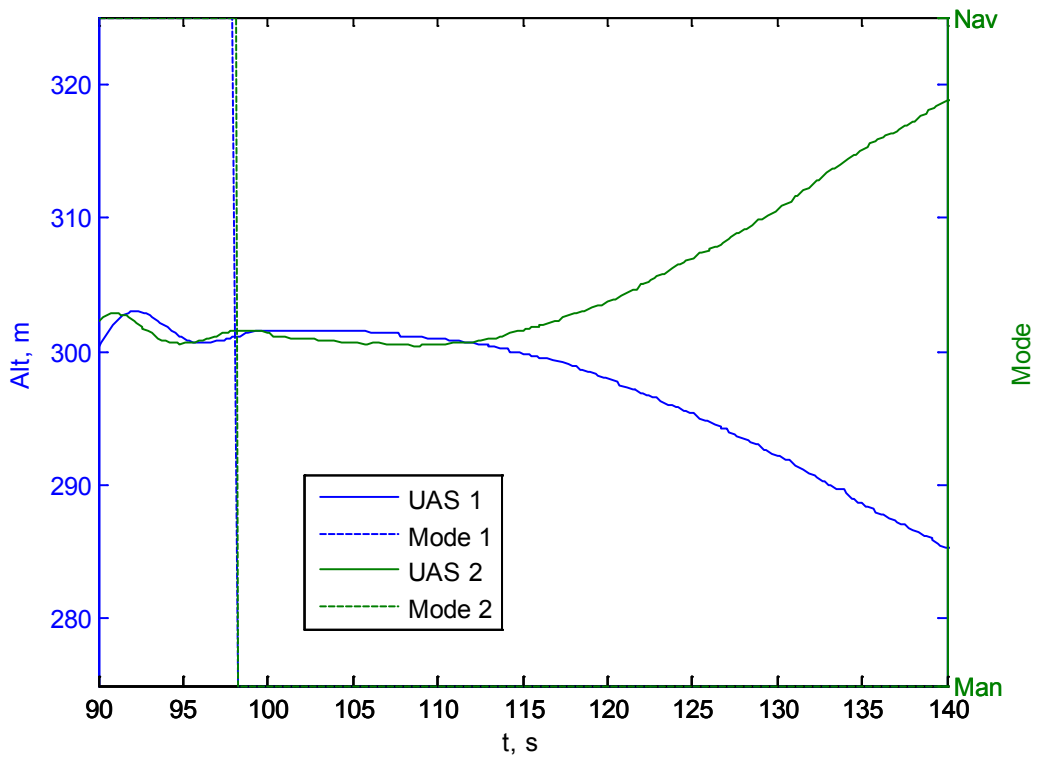
Figure D-12: SIL Abeam Simulation Altitude

142
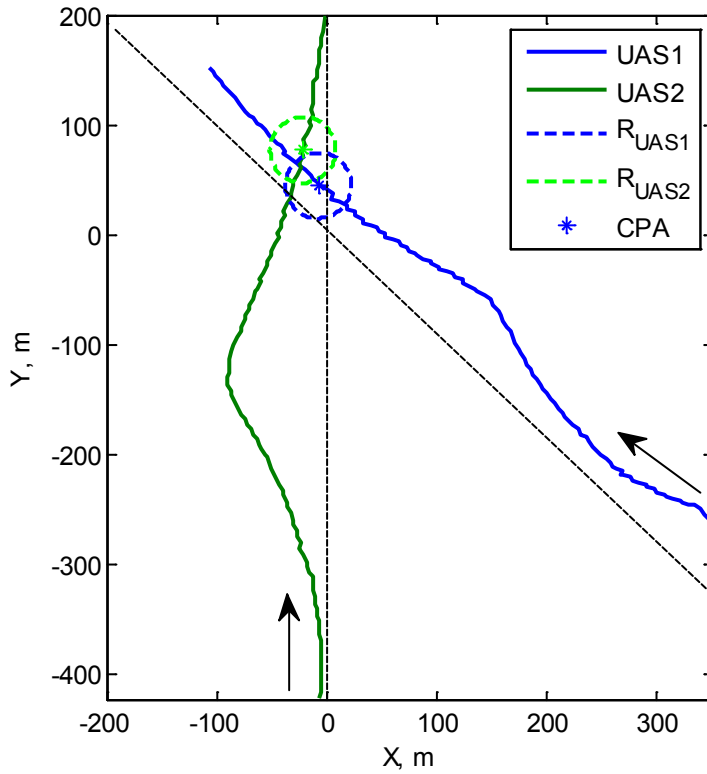
Figure D-13: SIL Converging Simulation Trajectories
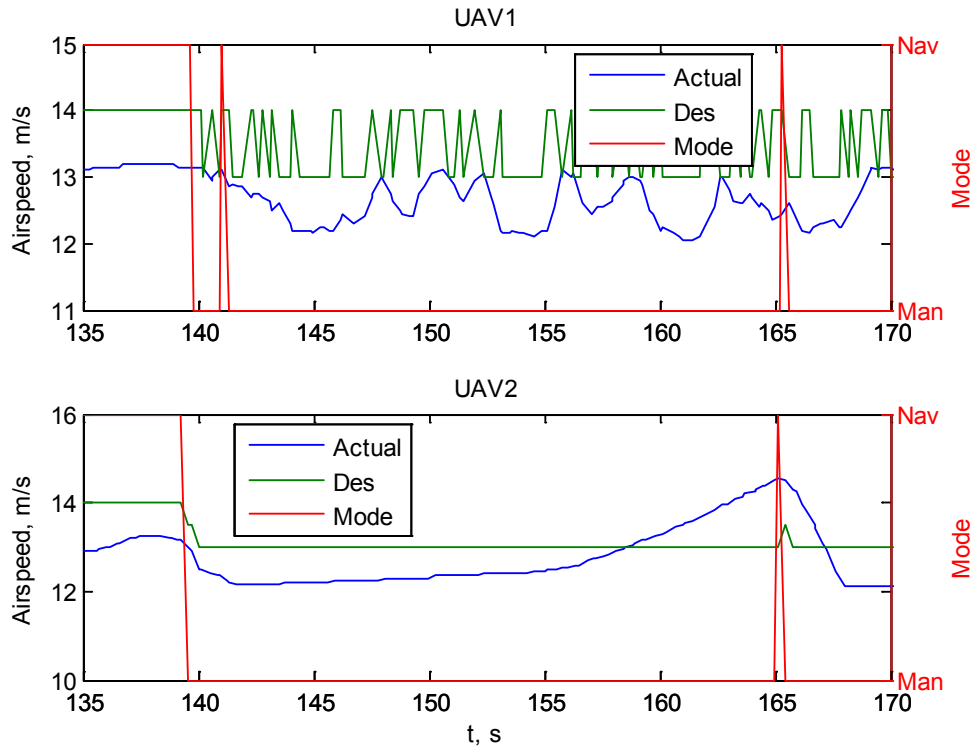
Figure D-14: SIL Converging Simulation Airspeed
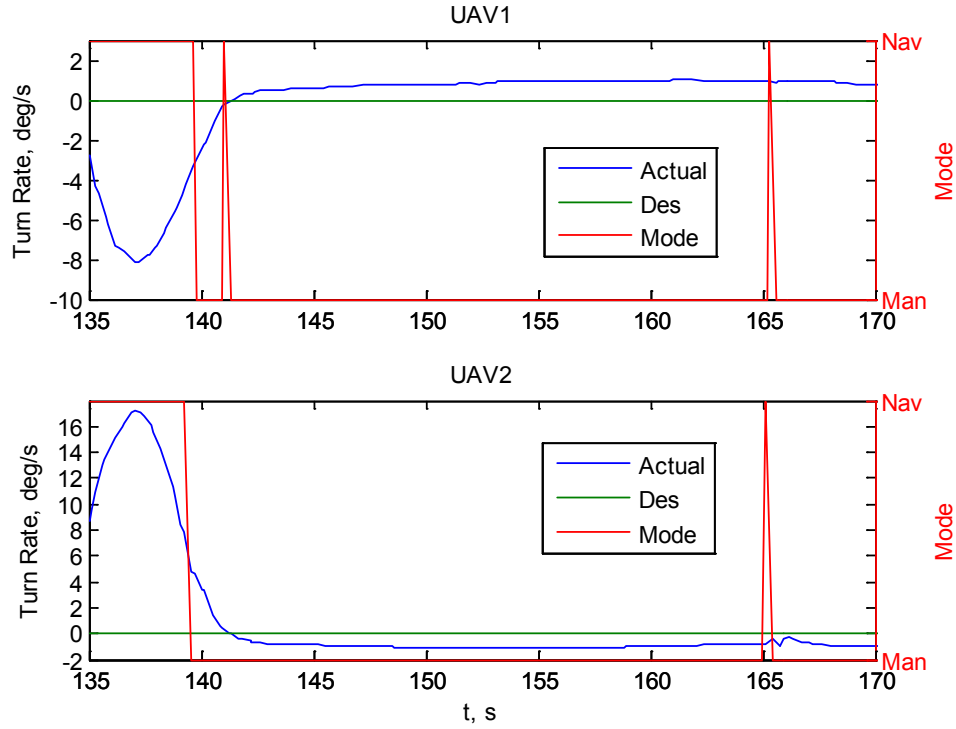
Figure D-15: SIL Converging Simulation Turn Rate
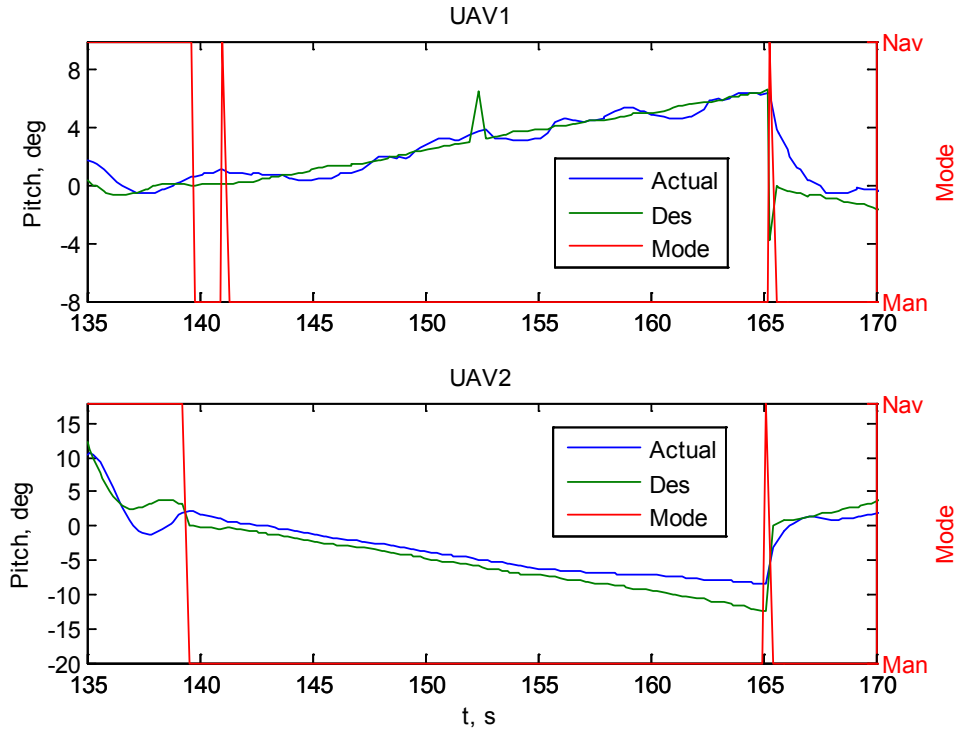
Figure D-16: SIL Converging Simulation Pitch

144
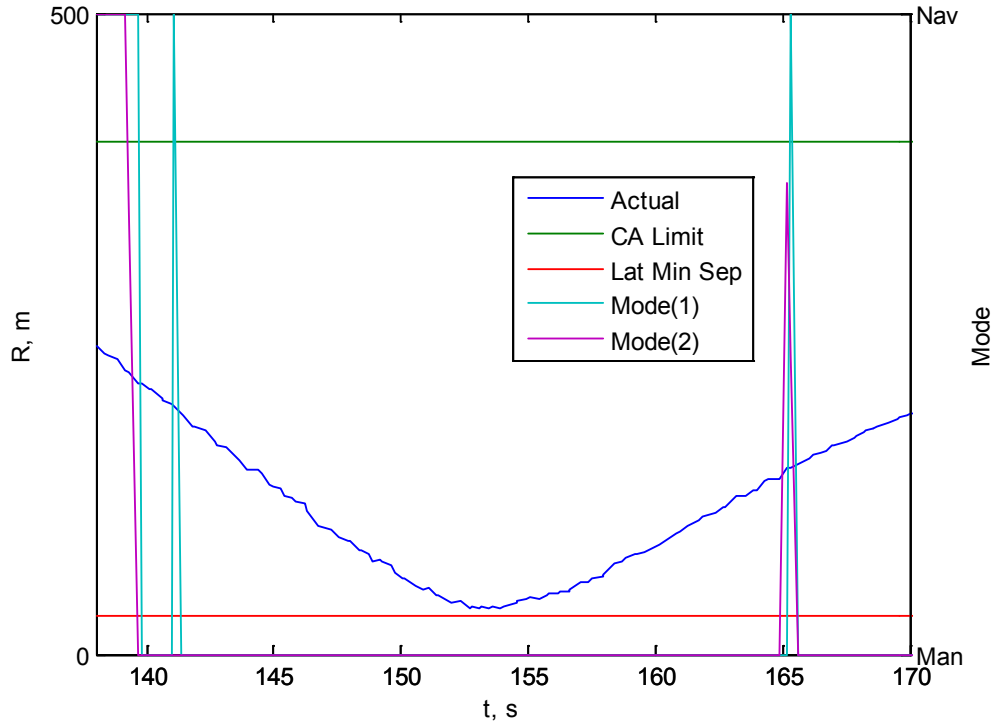
Figure D-17: SIL Converging Simulation Range
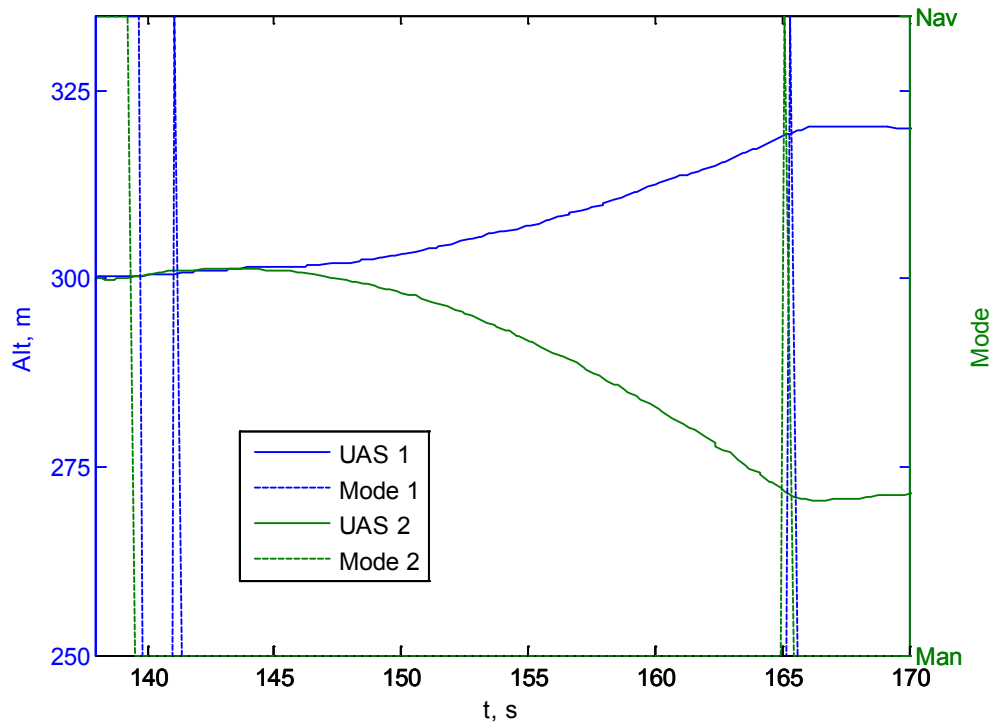
Figure D-18: SIL Converging Simulation Altitude

145

Appendix E: HIL Simulation Plots
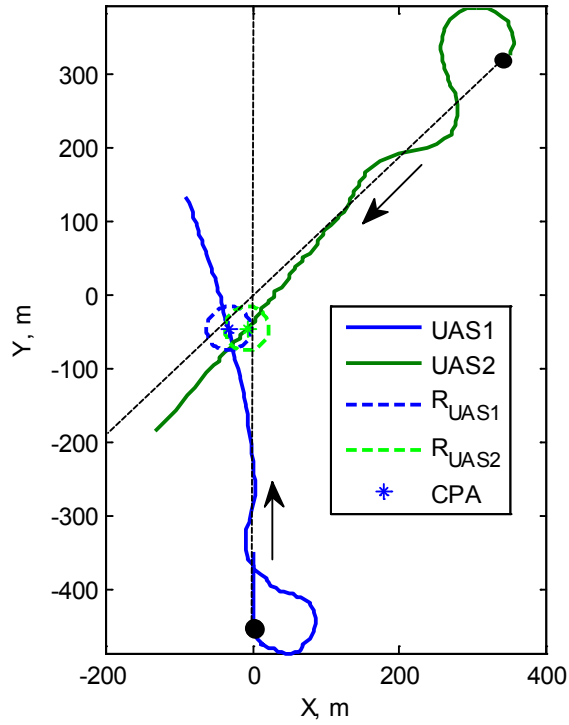
Figure E-1: HIL Approaching Simulation Trajectories
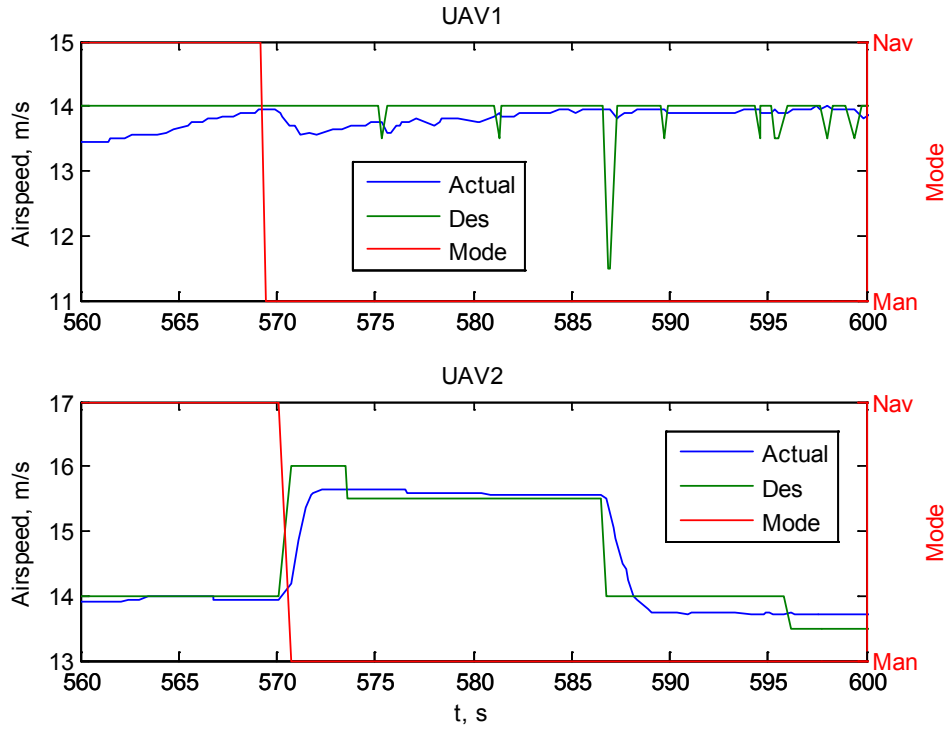
Figure E-2: HIL Approaching Simulation Airspeed Avoidance Command
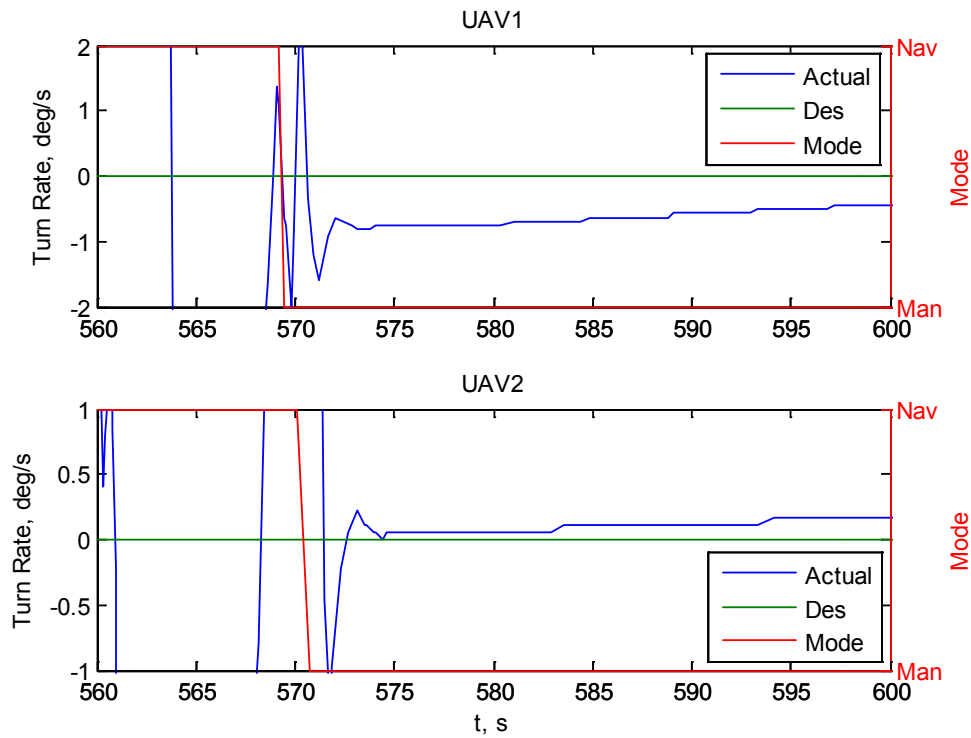
Figure E-3: HIL Approaching Simulation Turn Rate Avoidance Command
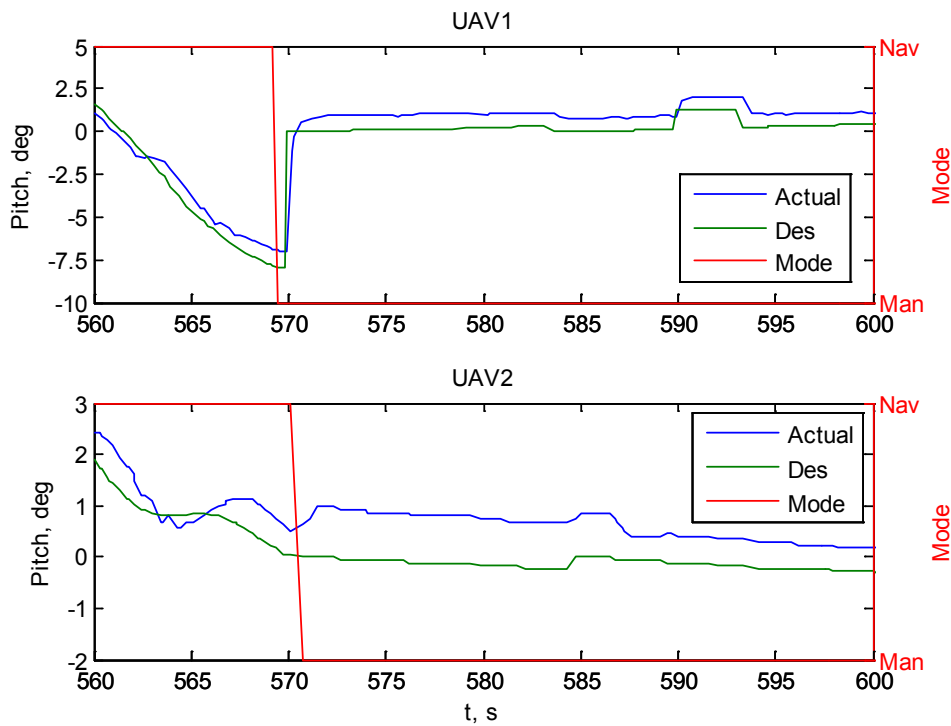
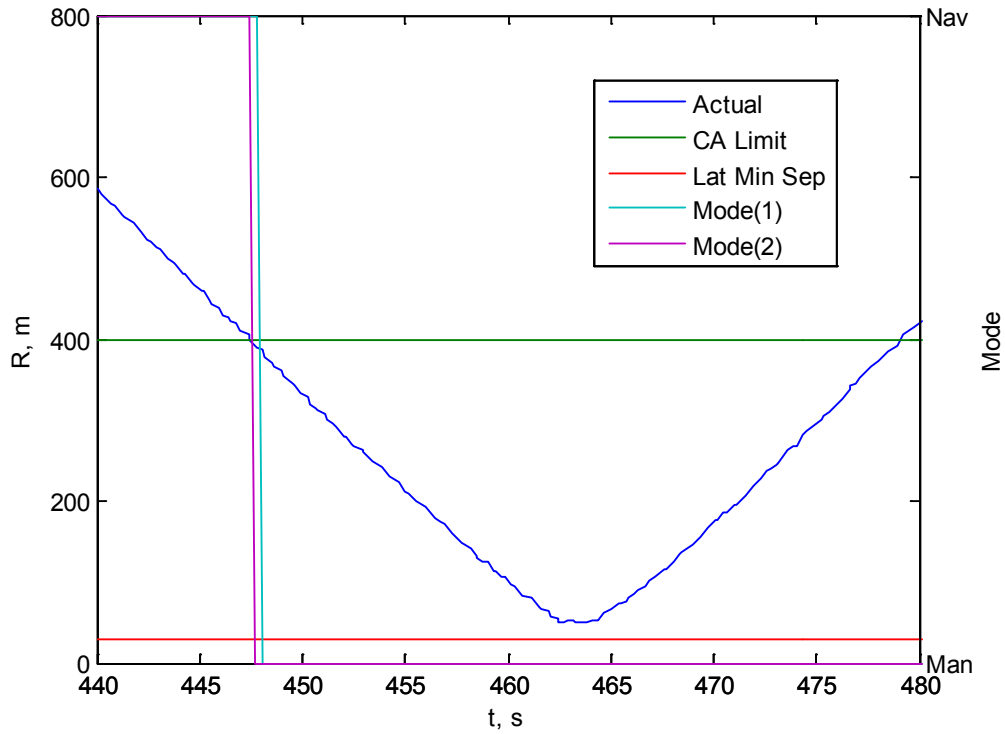**Figure E-4: HIL Approaching Simulation Pitch Avoidance Command**
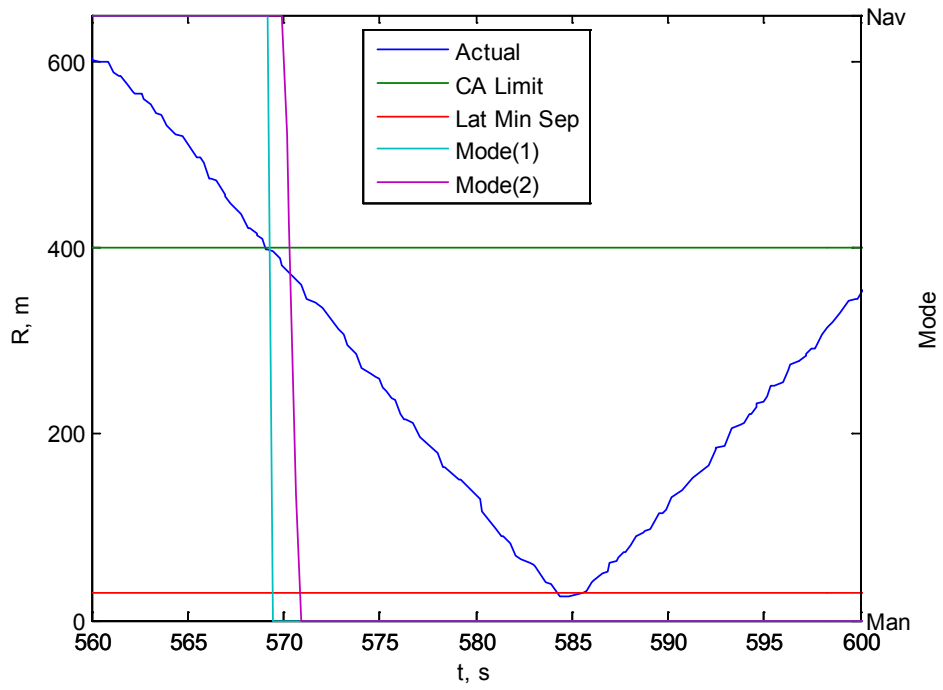
Figure E-5: HIL Approaching Simulation Range
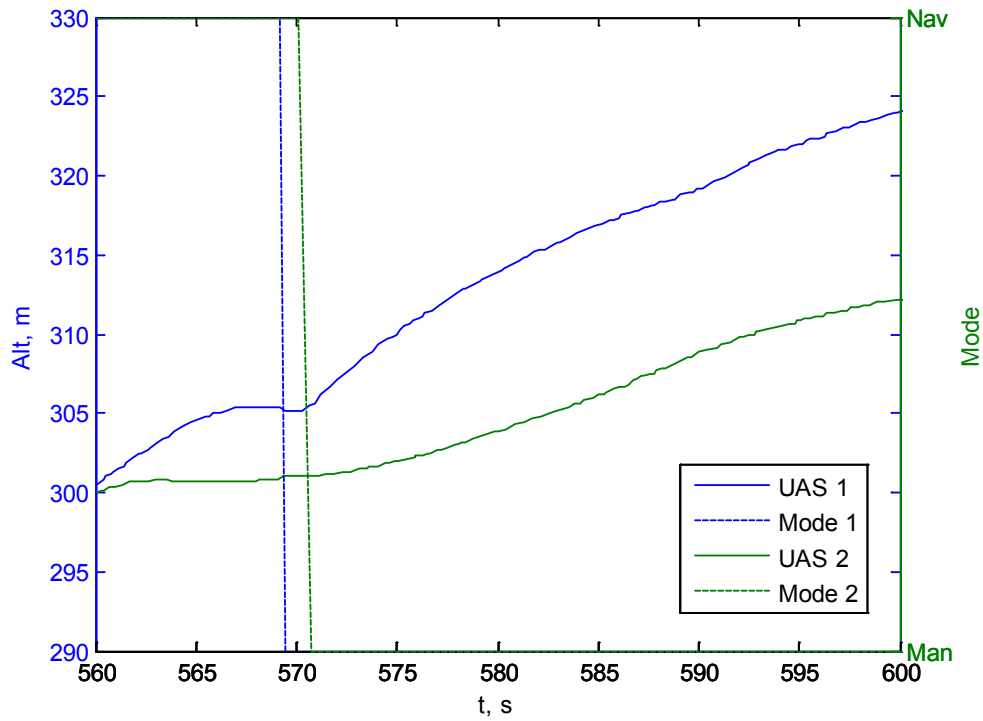
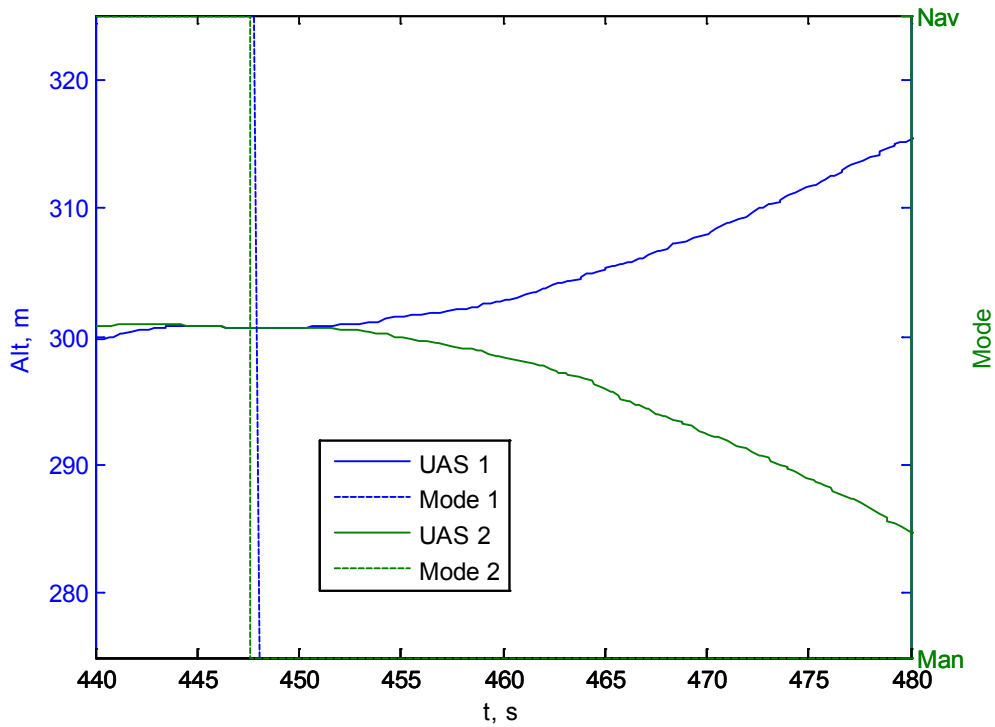**Figure E-6: HIL Approaching Simulation Altitude**

148
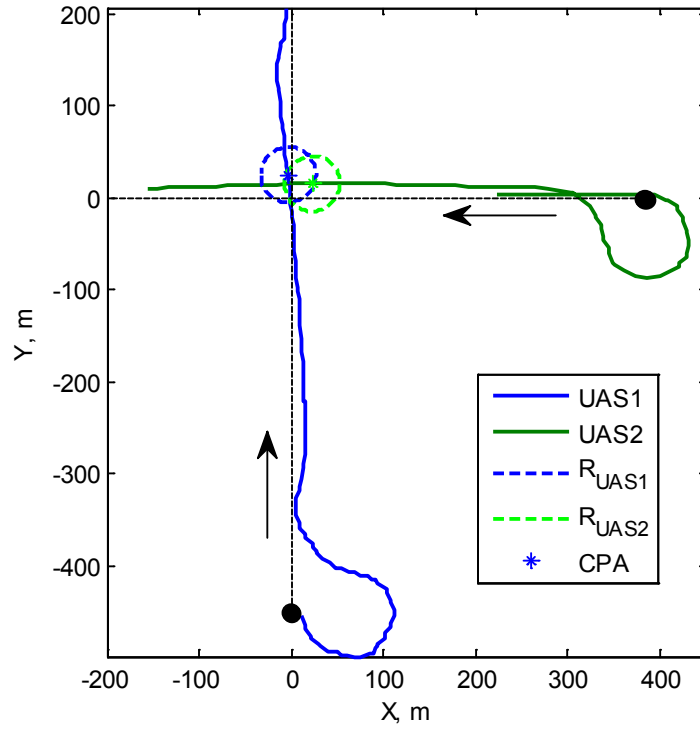
Figure E-7: HIL Abeam Simulation Trajectories
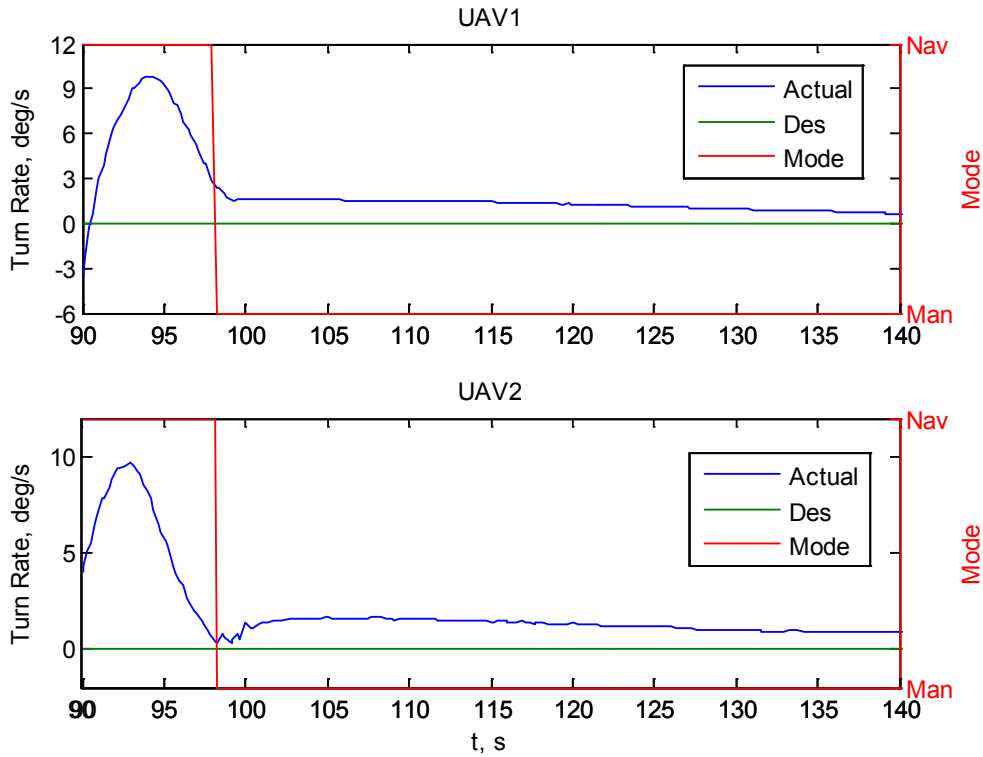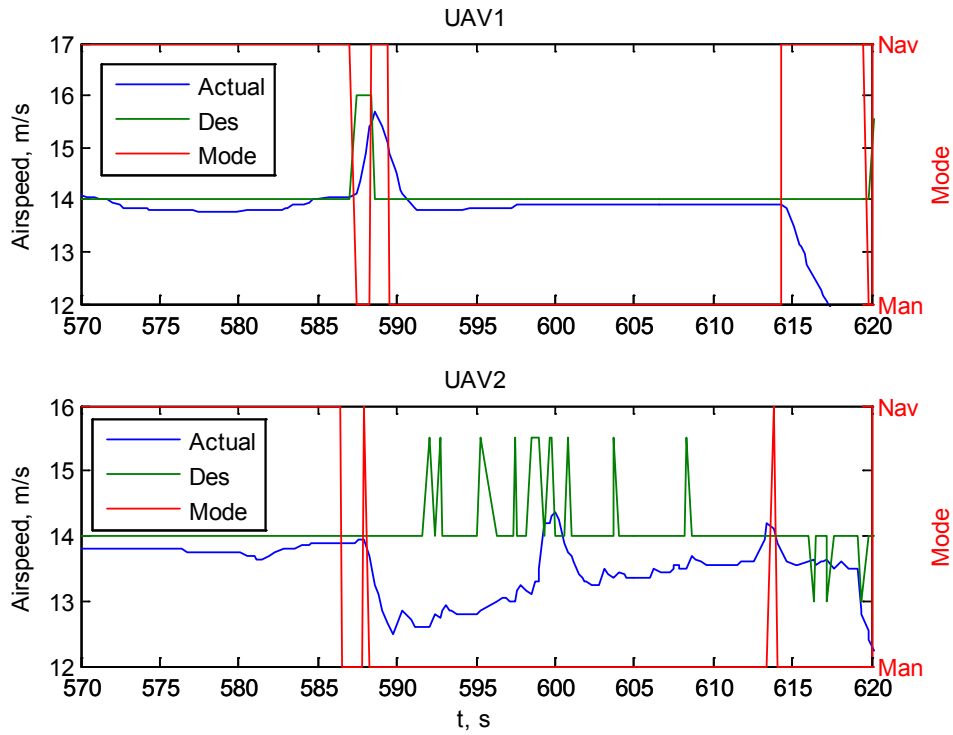

Figure E-8: HIL Abeam Simulation Airspeed Avoidance Command
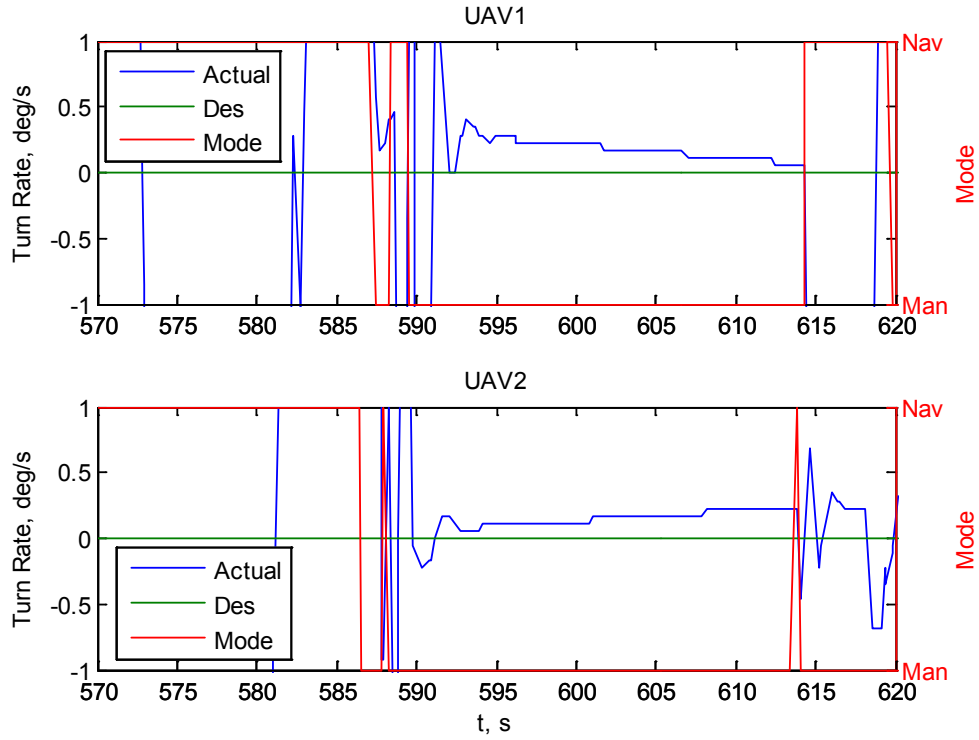
Figure E-9: HIL Abeam Simulation Turn Rate Avoidance Command
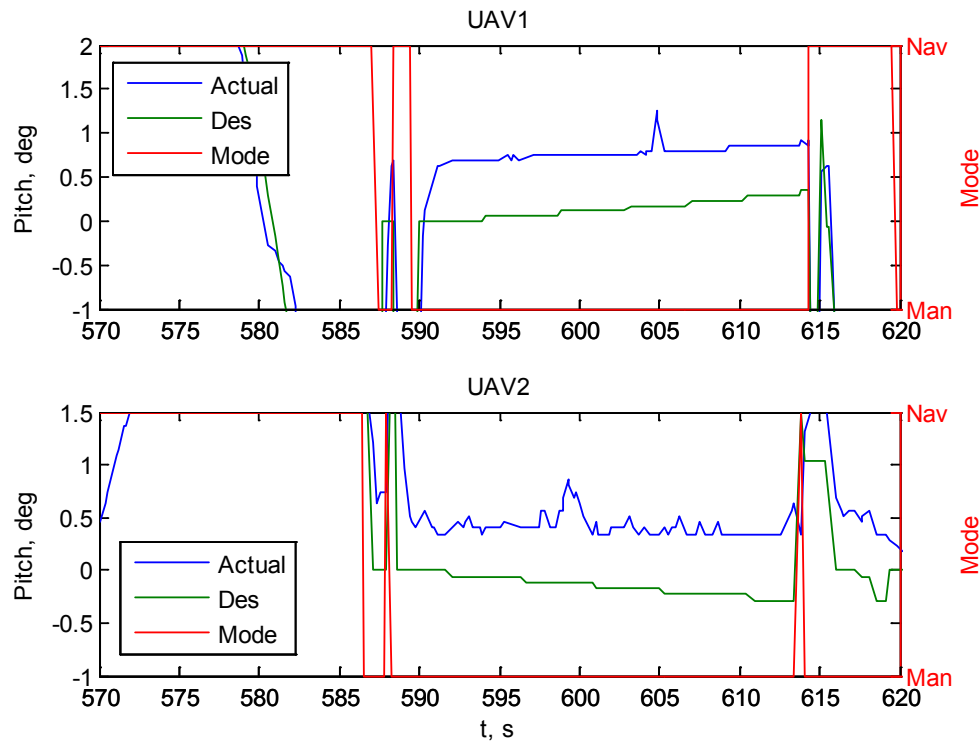
Figure E-10: HIL Abeam Simulation Pitch Avoidance Command
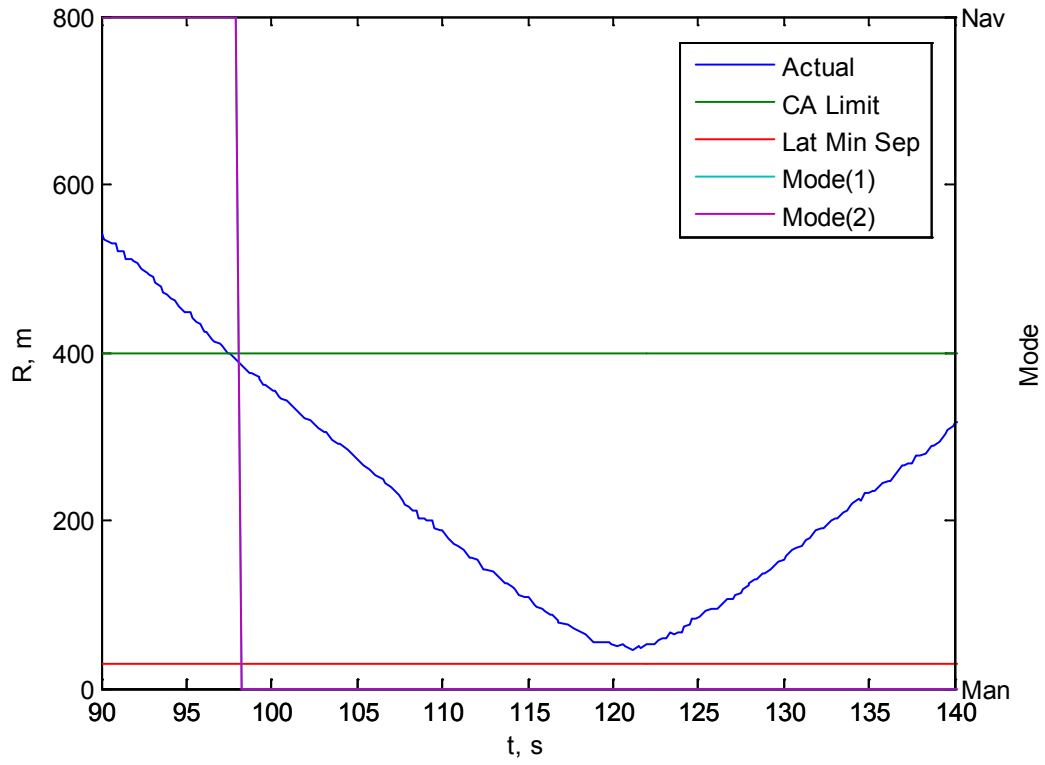
150
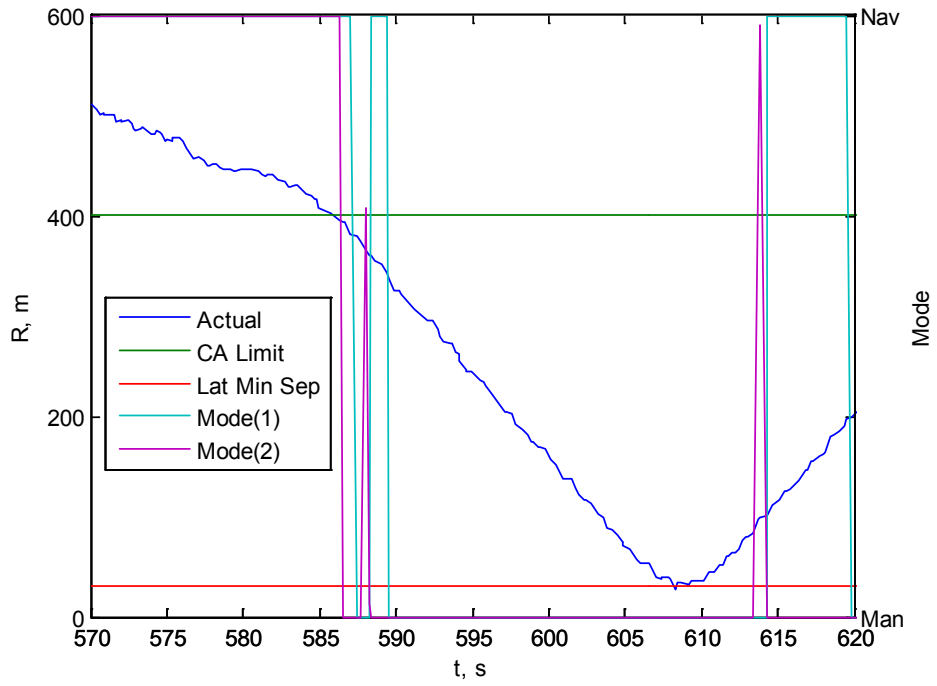
Figure E-11: HIL Abeam Simulation Range
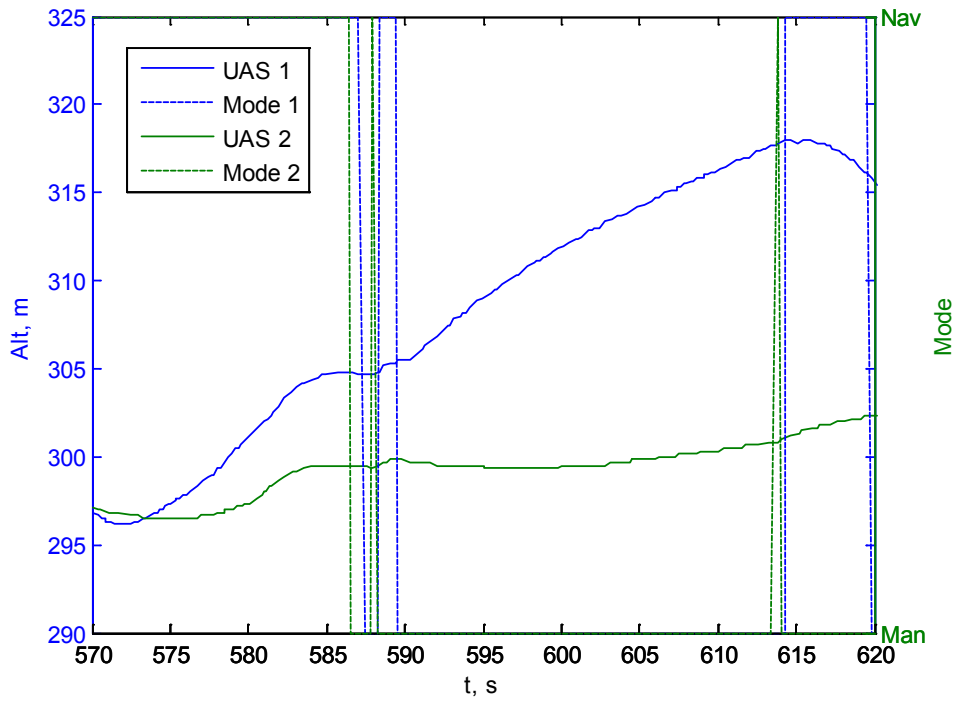
Figure E-12: HIL Abeam Simulation Altitude

151
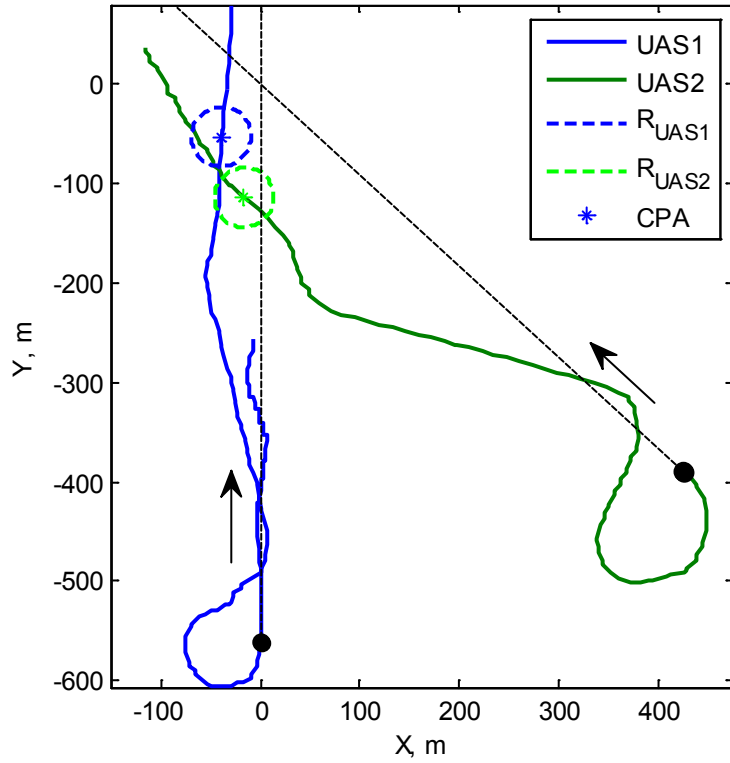
Figure E-13: HIL Converging Simulation Trajectories
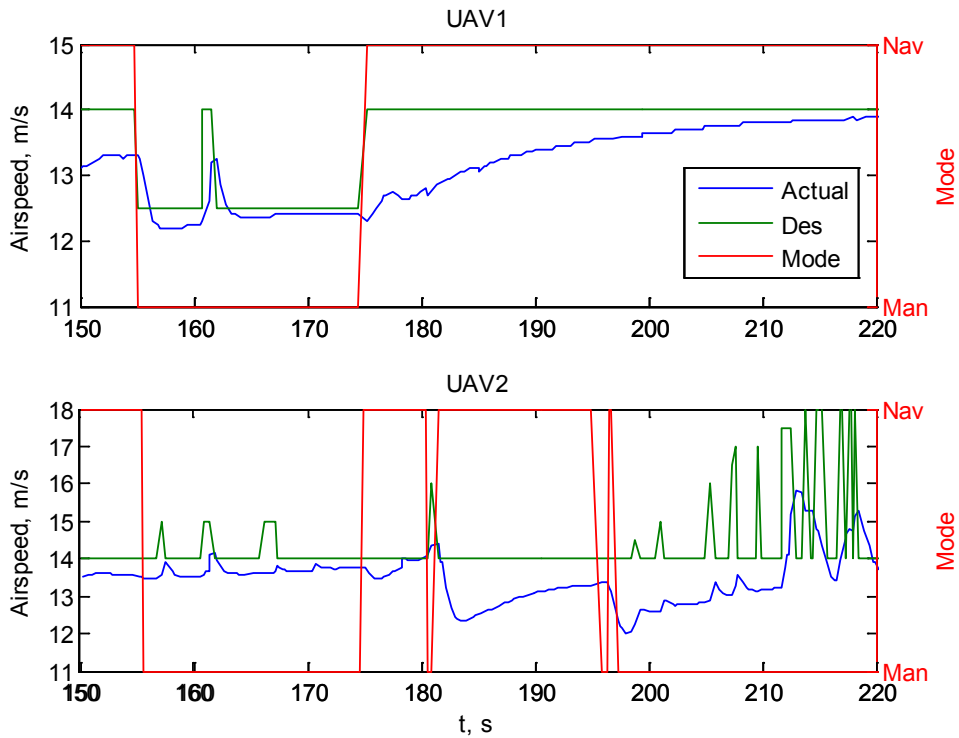
Figure E-14: HIL Converging Simulation Airspeed Avoidance Command

152
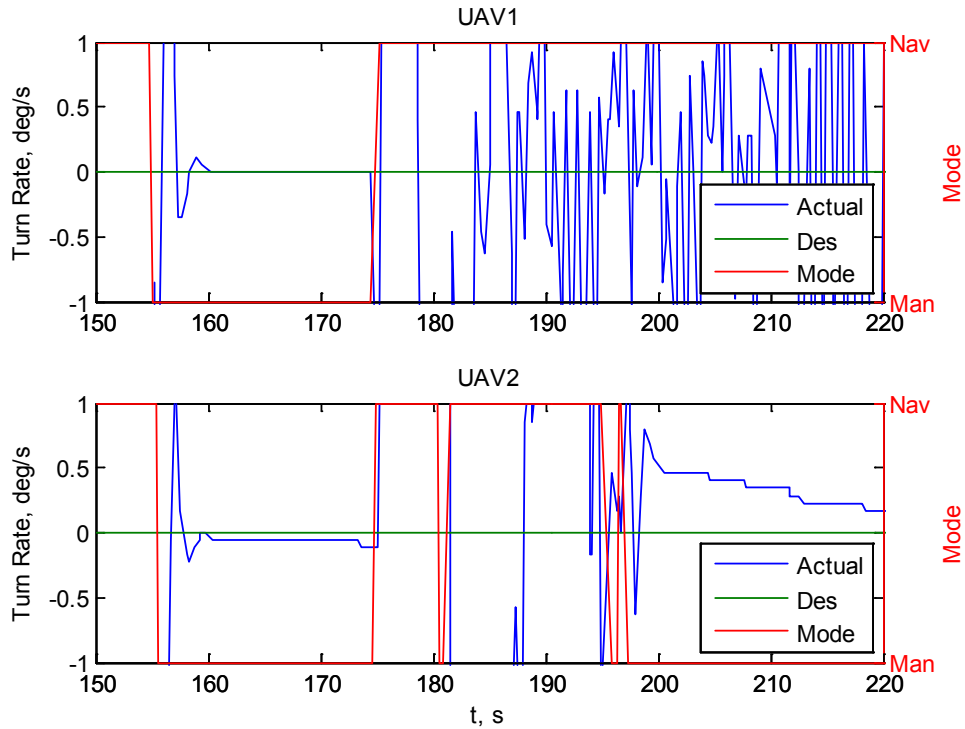
Figure E-15: HIL Converging Simulation Turn Rate Avoidance Command
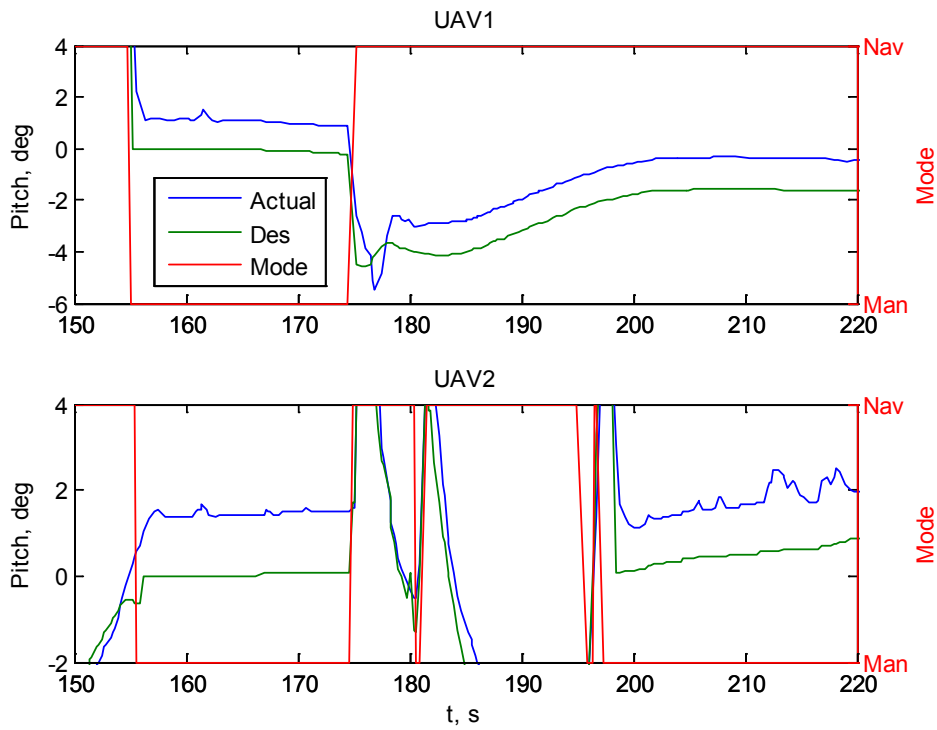
Figure E-16: HIL Converging Simulation Pitch Avoidance Command

153
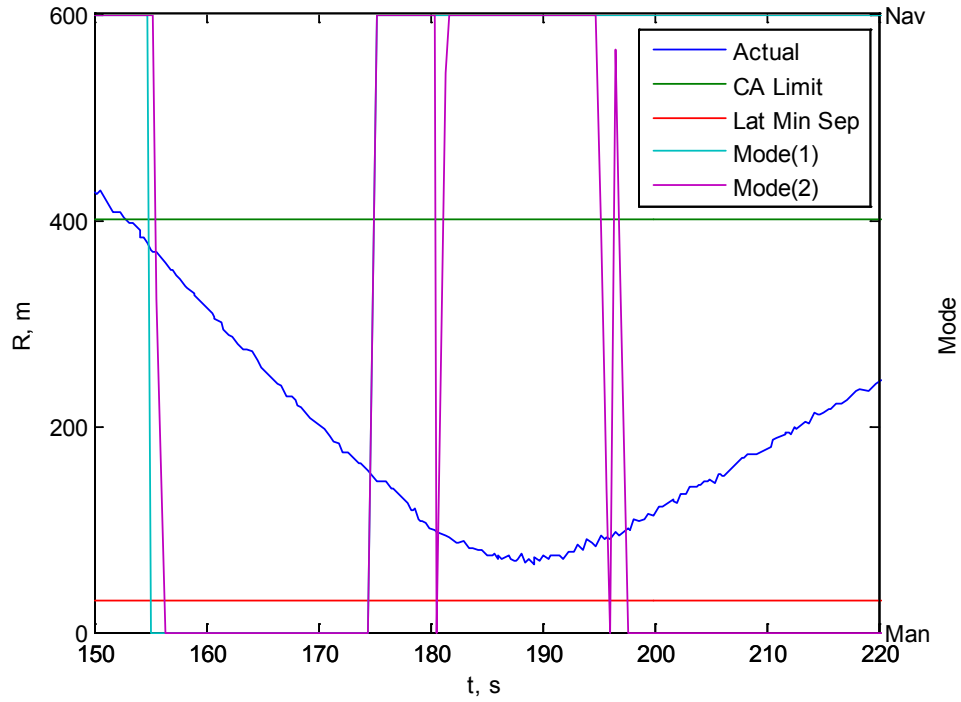
Figure E-17: HIL Converging Simulation Range
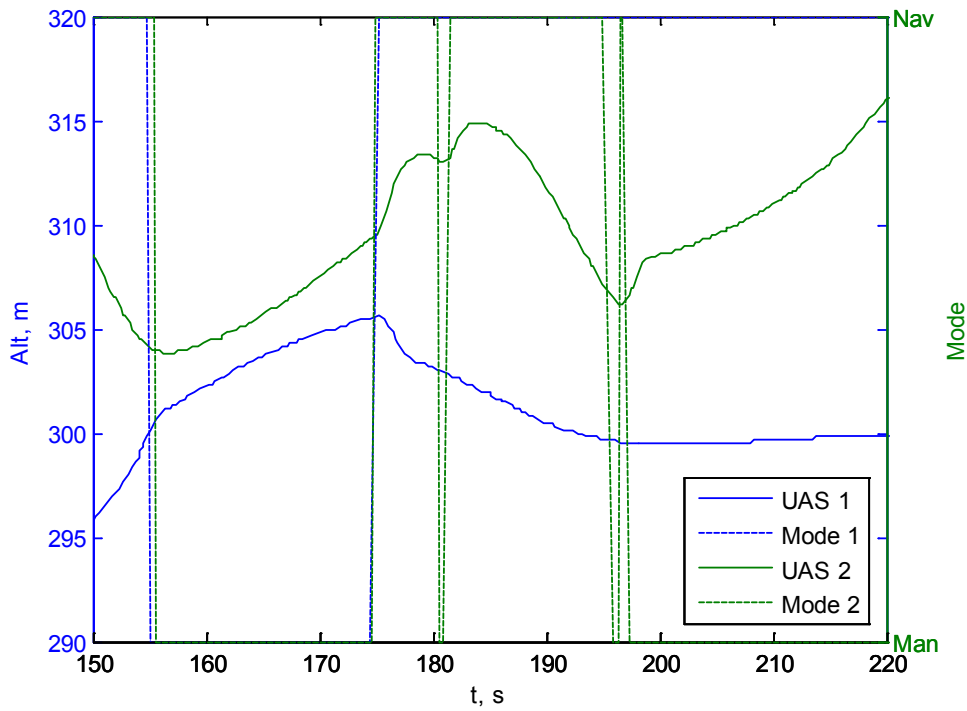
Figure E-18: HIL Converging Simulation Altitude

154

Appendix F: Flight Test Plots
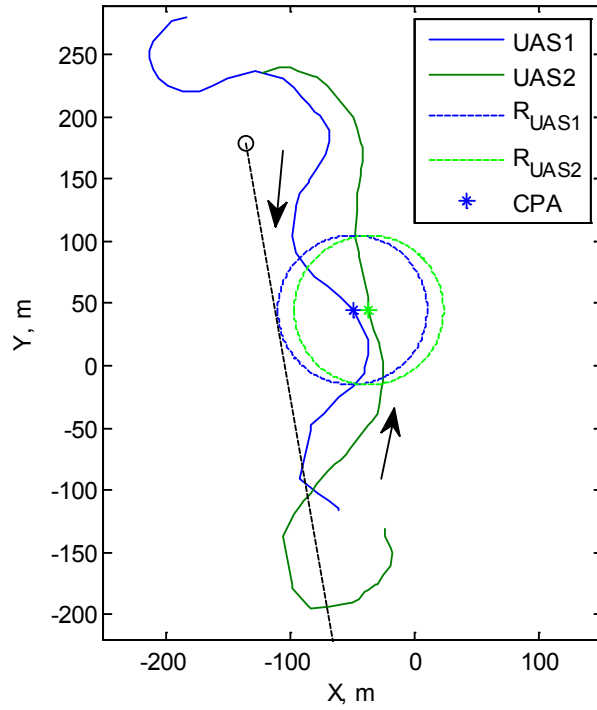
Figure F-1: Flight Test Head-on Encounter 1 Trajectories
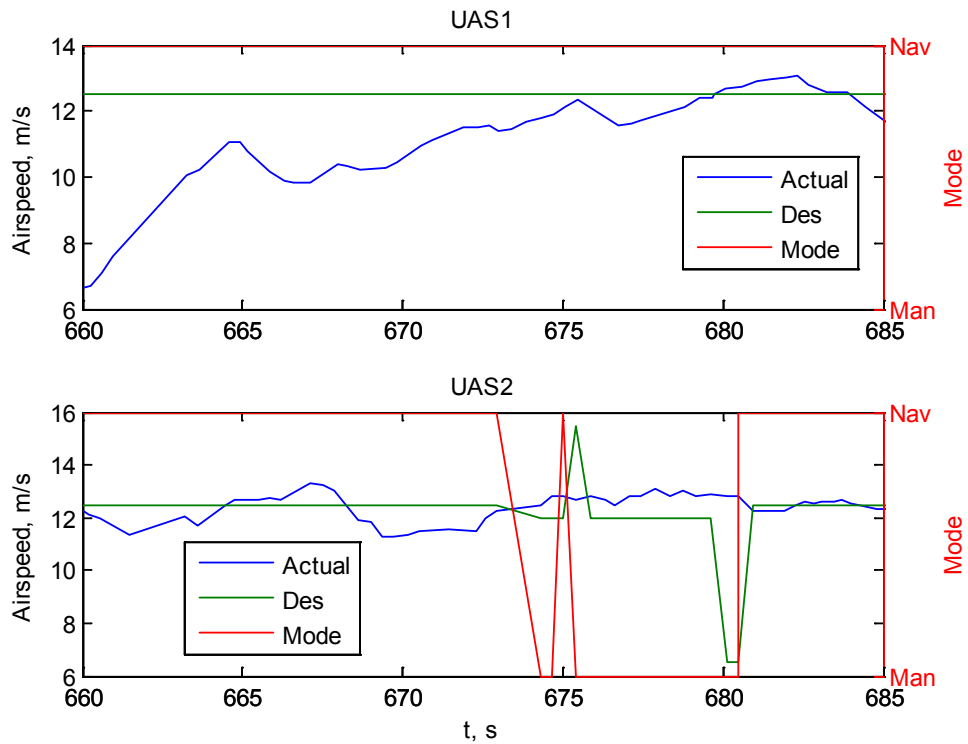
Figure F-2: Flight Test Head-on Encounter 1 Airspeed Response

155
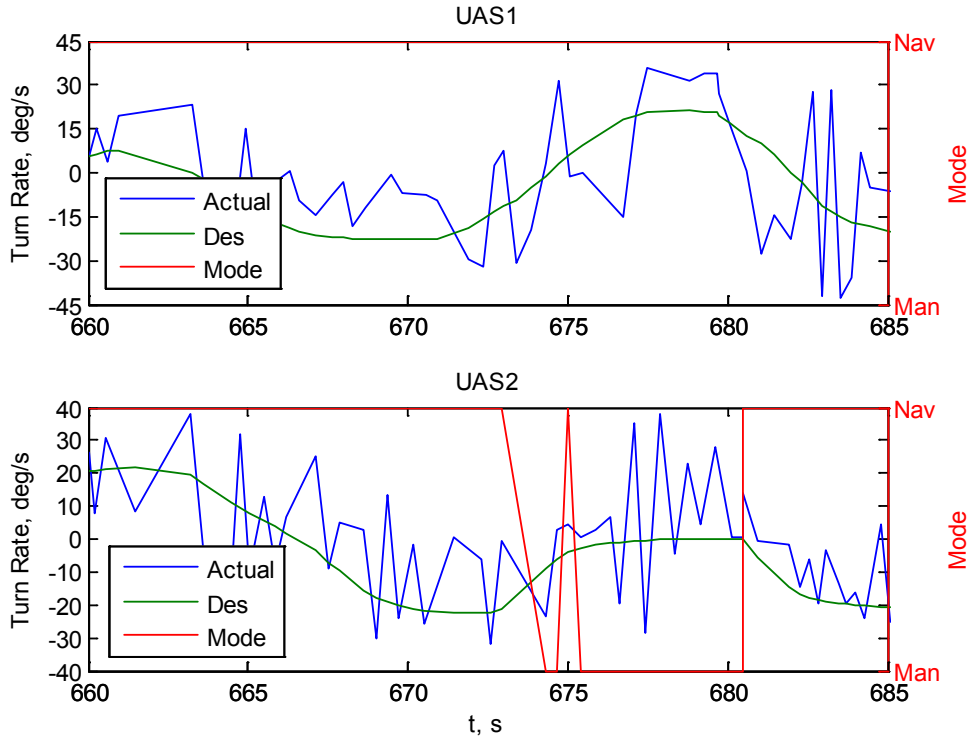
Figure F-3: Flight Test Head-on Encounter 1 Turn Rate Response
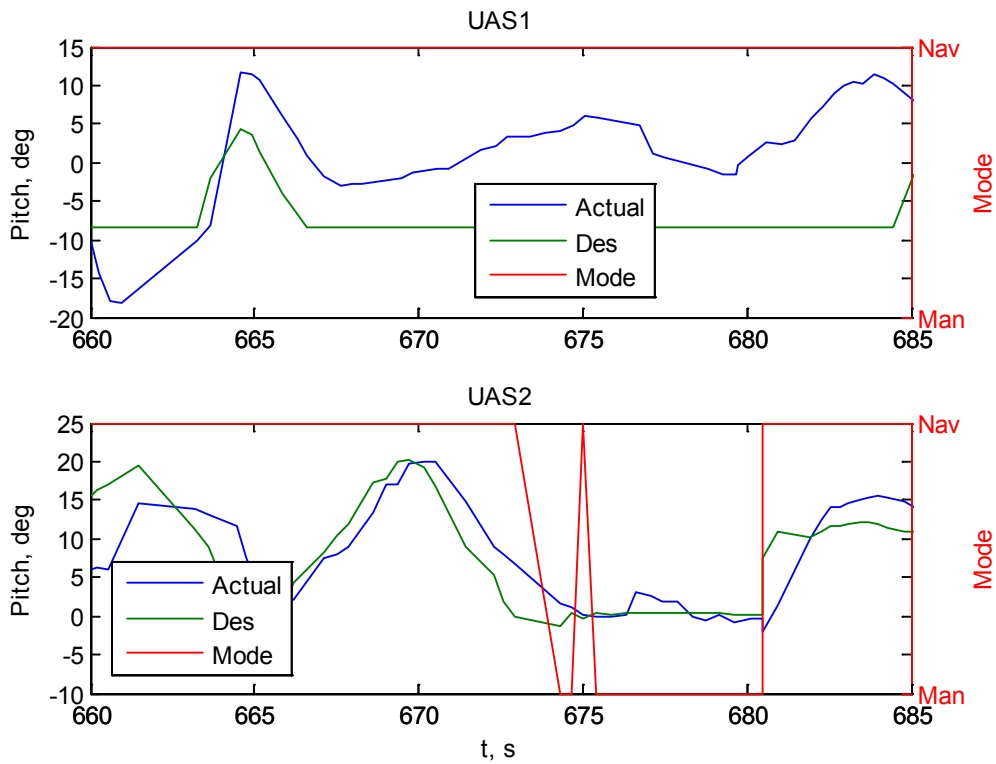
Figure F-4: Flight Test Head-on Encounter 1 Pitch Response
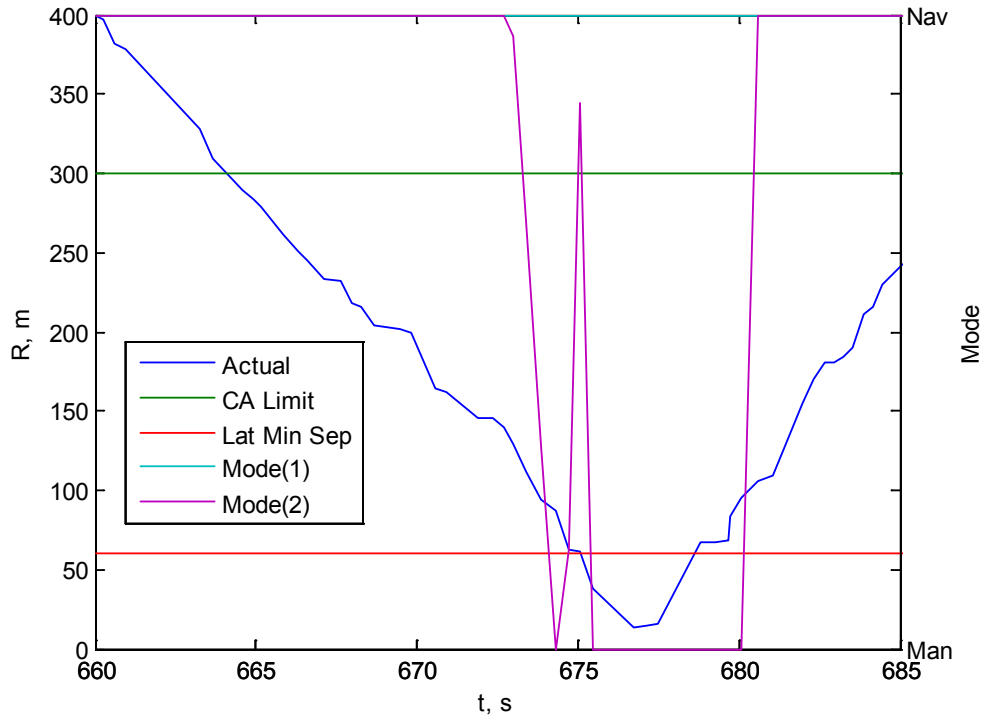
Figure F-5: Flight Test Head-on Encounter 1 Range
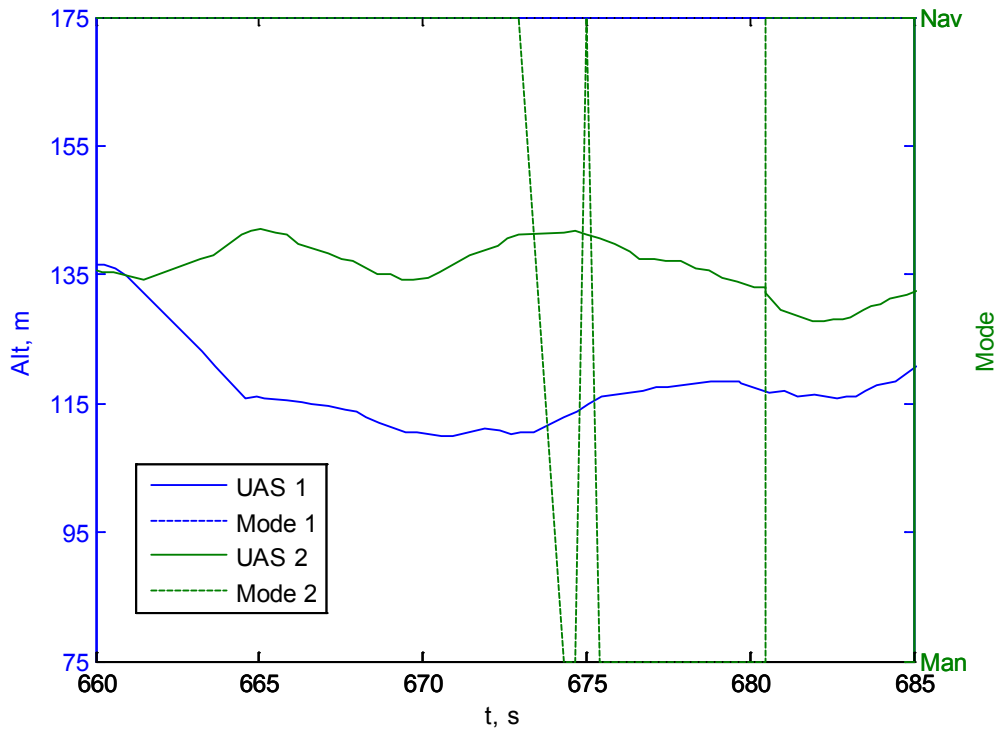
Figure F-6: Flight Test Head-on Encounter 1 Altitude

157
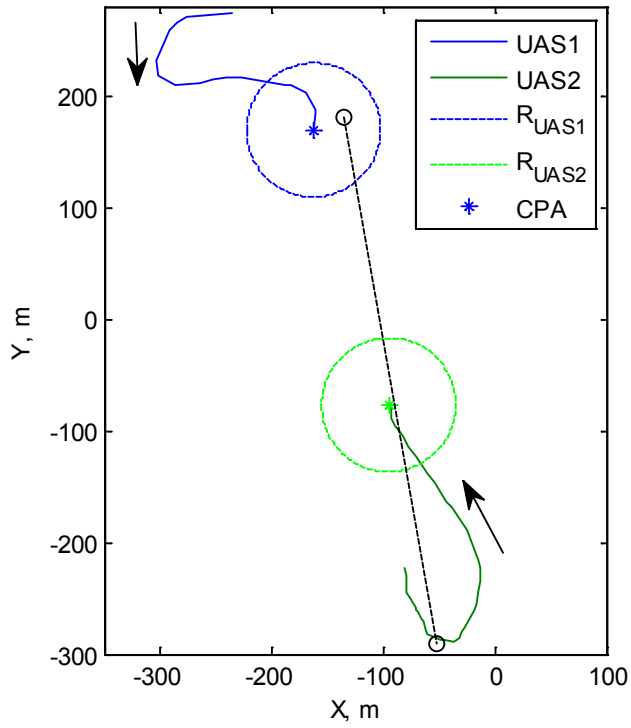
Figure F-7: Flight Test Head-on Encounter 3 Trajectories
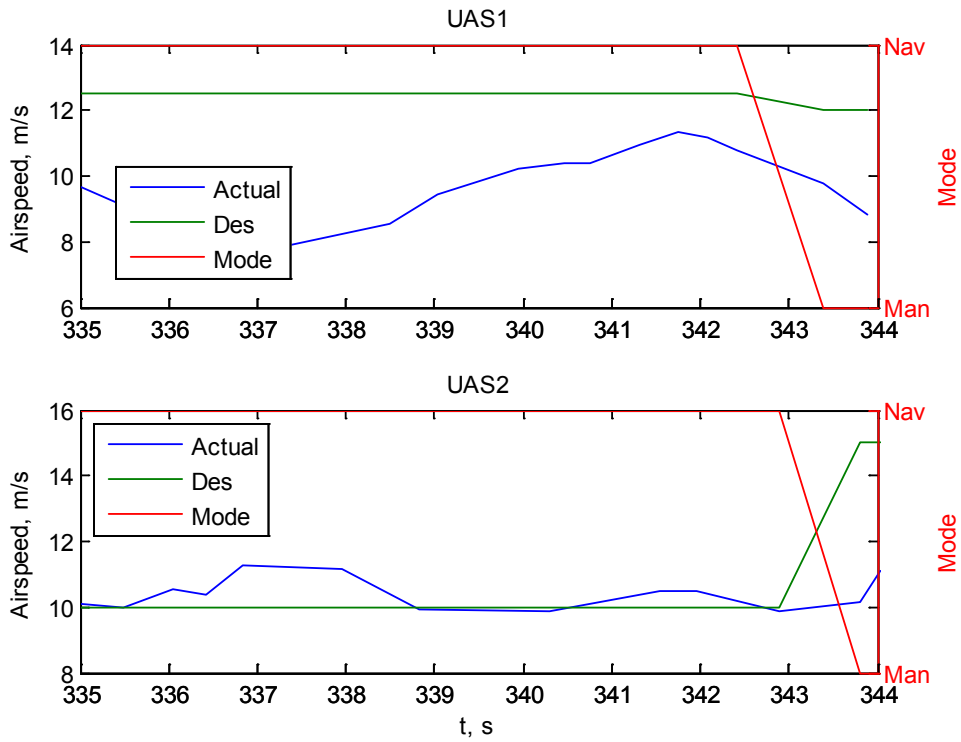
Figure F-8: Flight Test Head-on Encounter 3 Airspeed Response

158
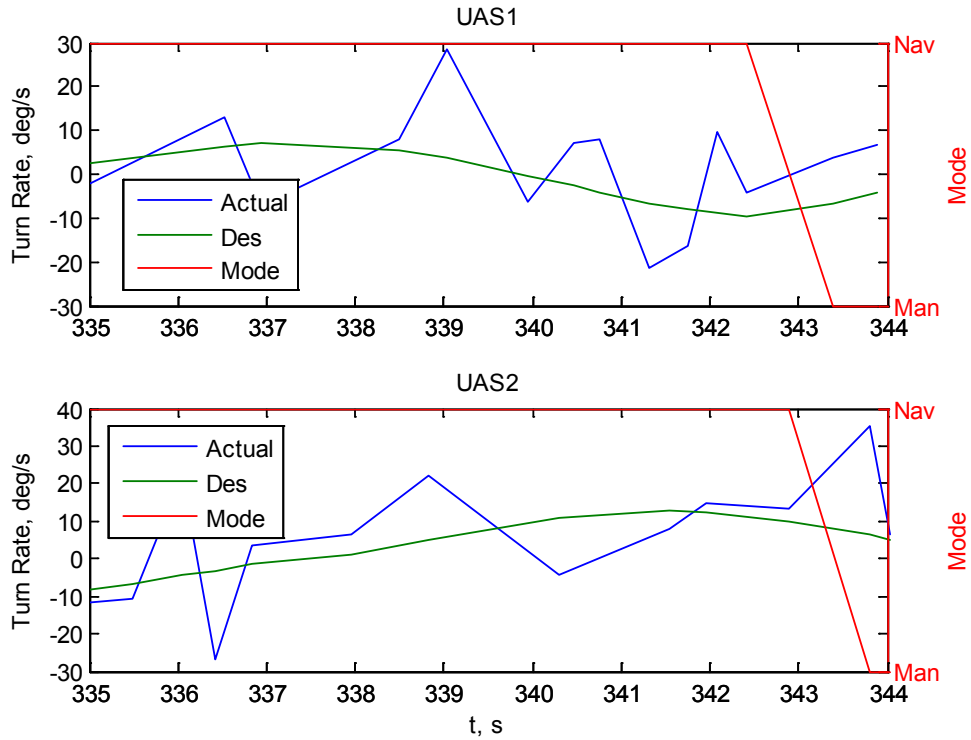
Figure F-9: Flight Test Head-on Encounter 3 Turn Rate Response
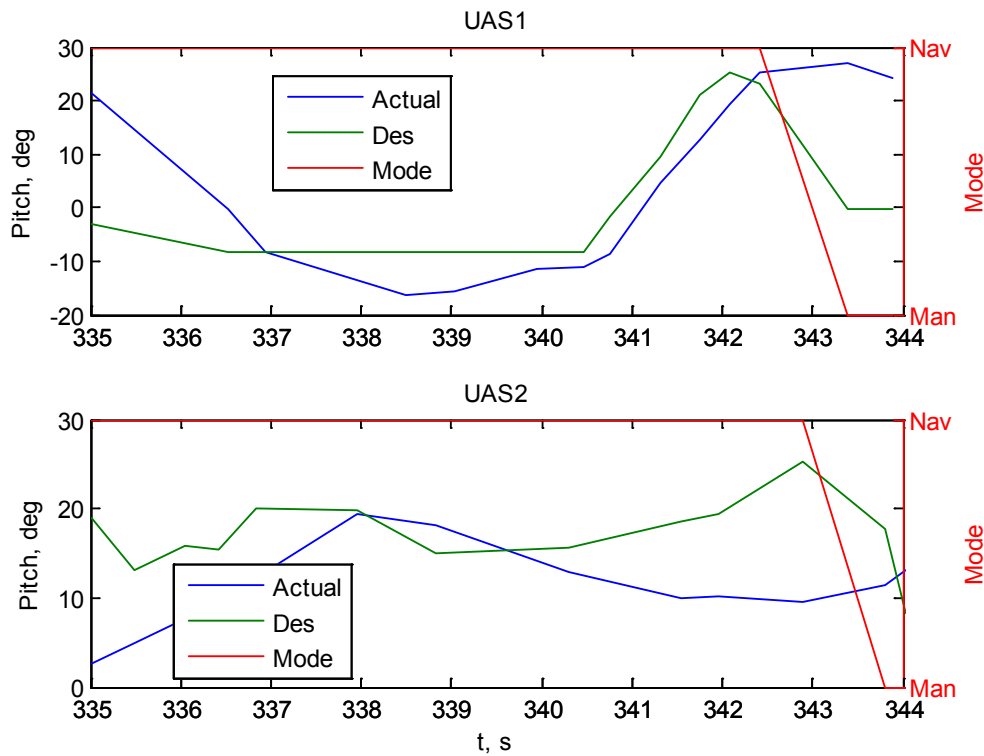
Figure F-10: Flight Test Head-on Encounter 3 Pitch Response
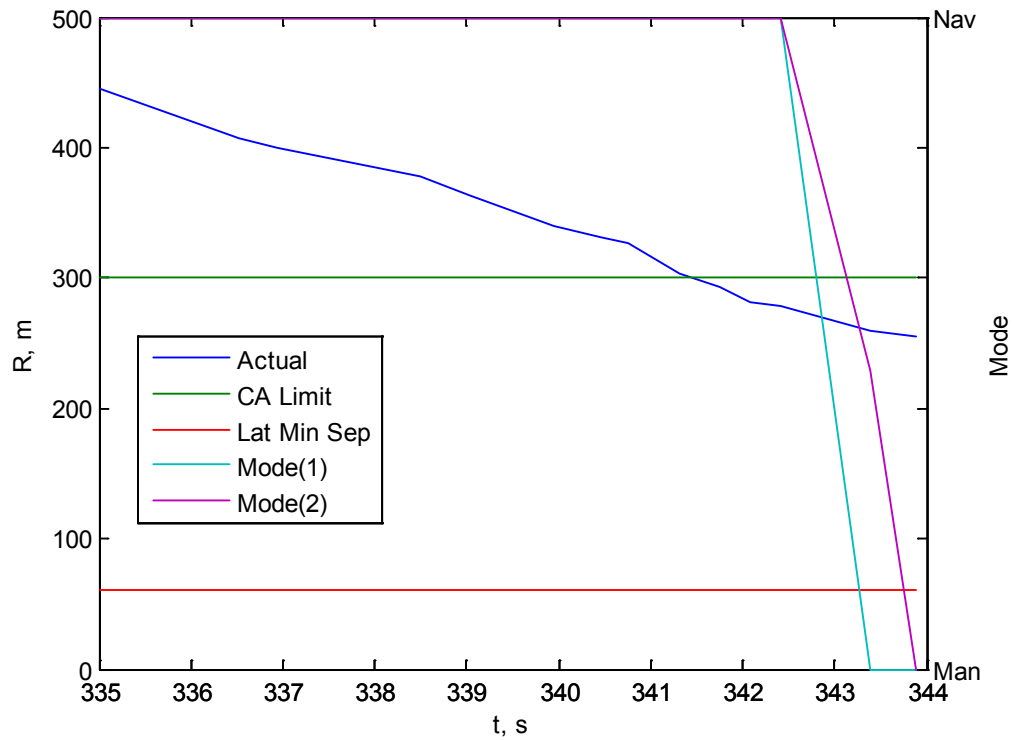
Figure F-11: Flight Test Head-on Encounter 3 Range
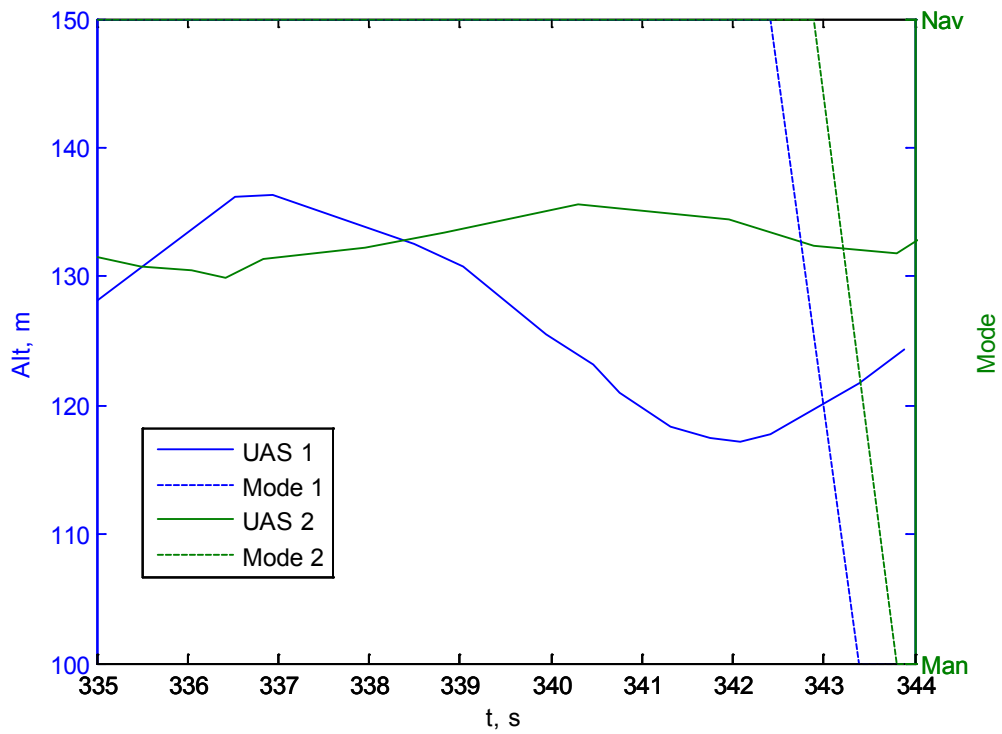
Figure F-12: Flight Test Head-on Encounter 3 Altitude
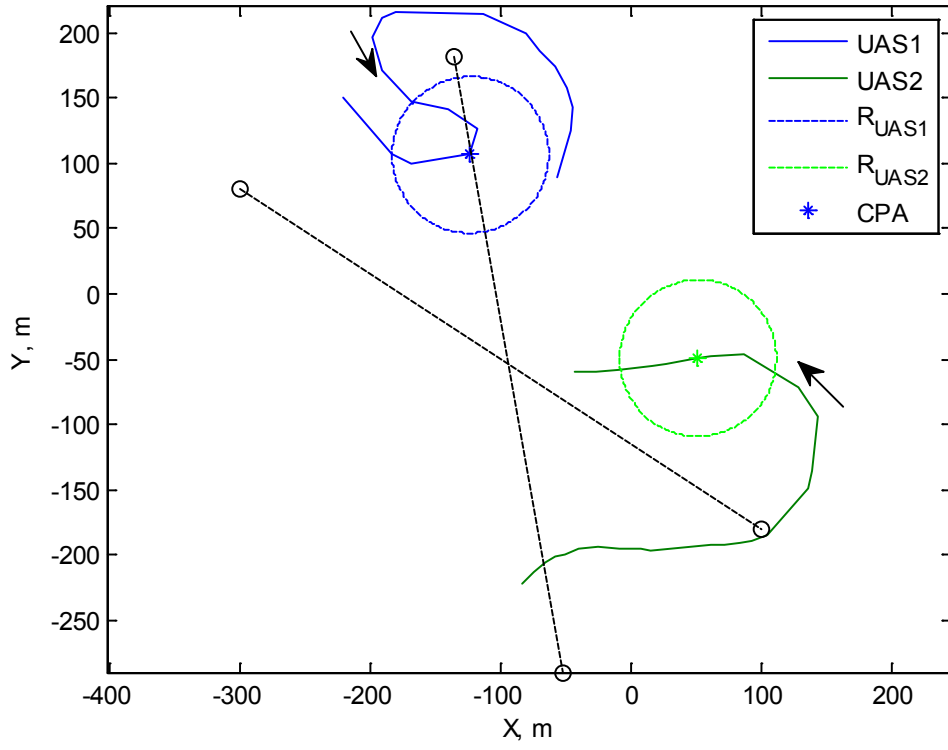
Figure F-13: Flight Test Approaching Encounter 1 Trajectories
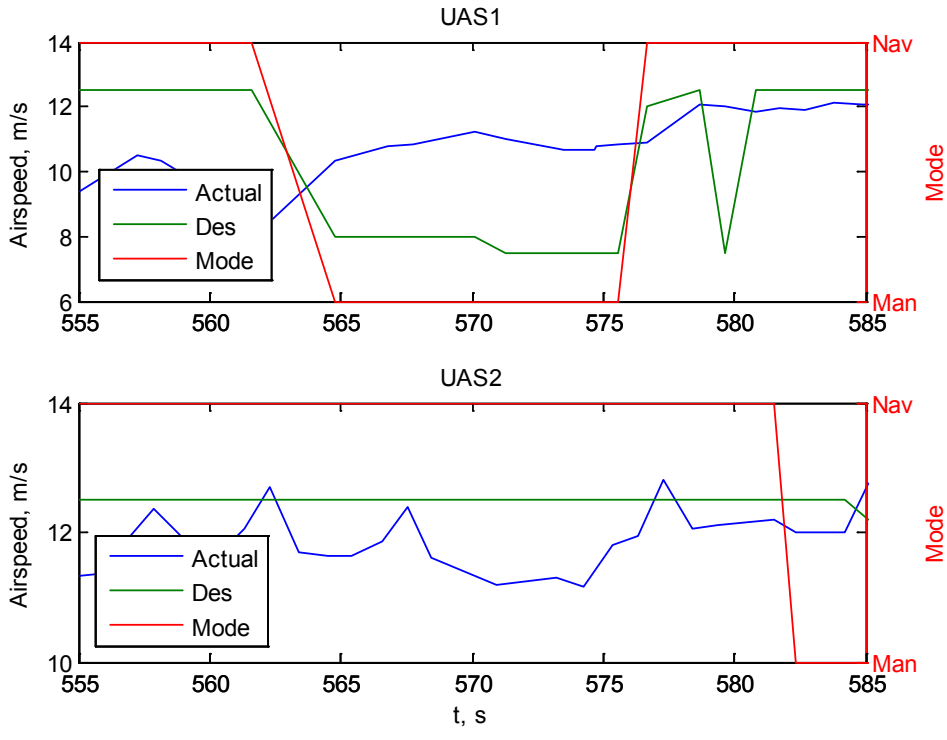
Figure F-14: Flight Test Approaching Encounter 1 Airspeed Response

161
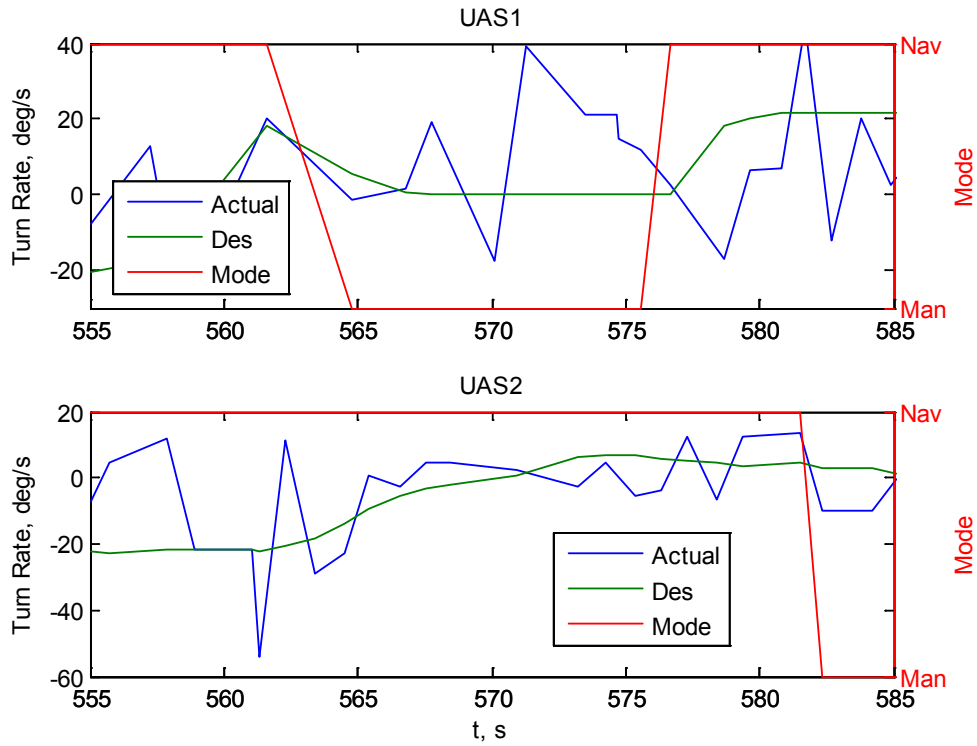
Figure F-15: Flight Test Approaching Encounter 1 Turn Rate Response
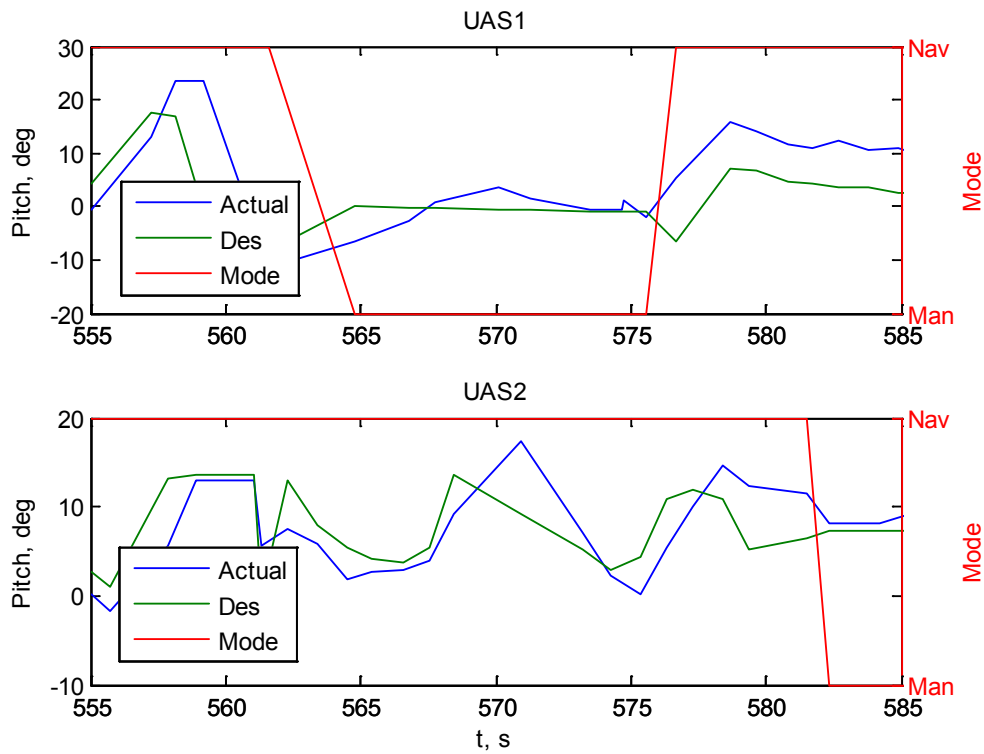
Figure F-16: Flight Test Approaching Encounter 1 Pitch Response
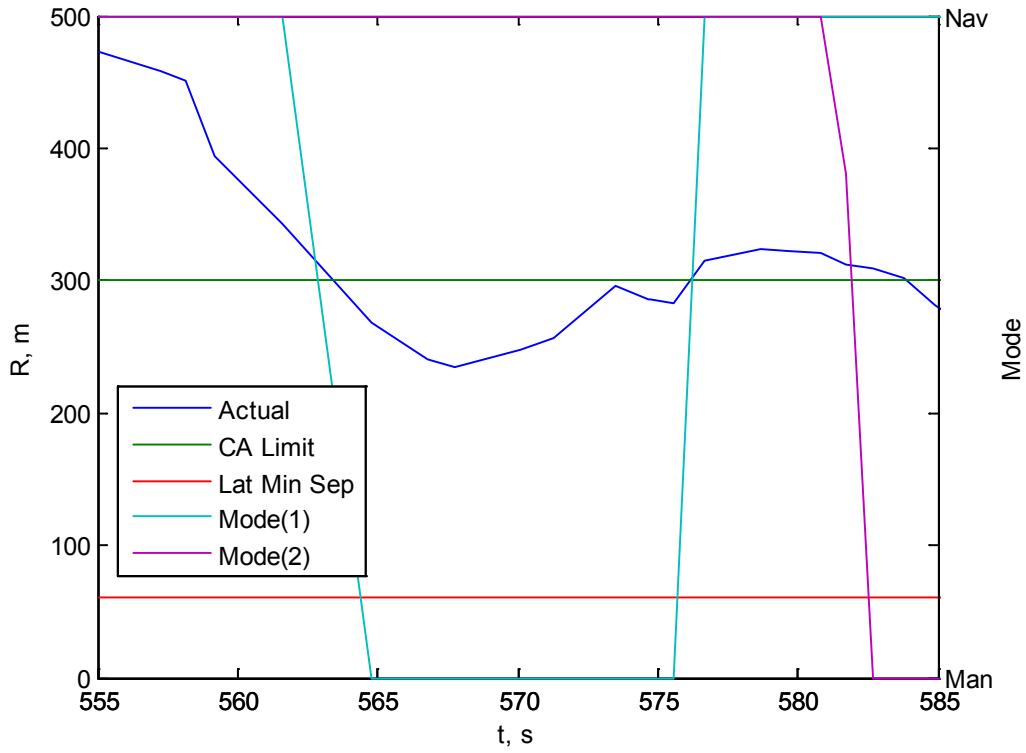
Figure F-17: Flight Test Approaching Encounter 1 Range
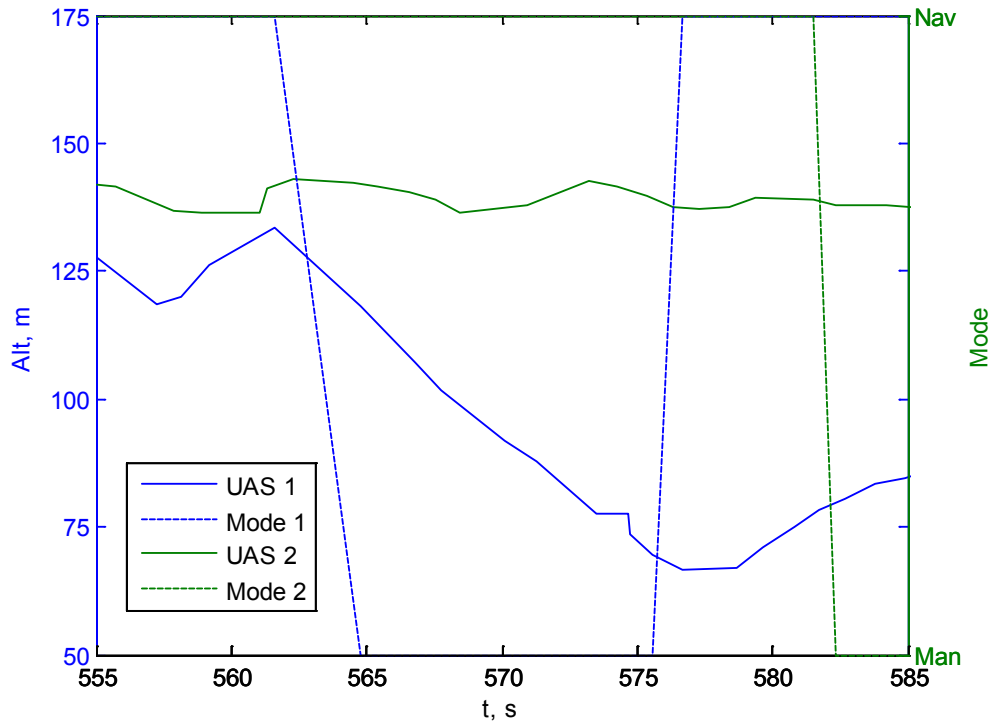
Figure F-18: Flight Test Approaching Encounter 1 Altitude

Appendix G: Flight Test Procedures

FT-1: *Collision Avoidance Algorithm with Two BATCAMs*

Objectives:
1. Demonstrate successful two-ship UAS collision avoidance divert maneuvers resulting from collision encounter
   a. Head-on encounter
   b. Approaching encounter
   c. Abeam encounter
   d. Converging encounter

FT-1: PROCEDURES	Notes: Dur: X min
1. Preflight BATCAM 1 and 2	
2. **O:** Configure Autopilot Manual Mode settings according to CA specifications for each BATCAM	2.
3. **O:** Upload Configuration / Update Flash as necessary for each BATCAM	
4. **O:** Set "Lost Comm" to 3-5 seconds	
5. **O:** Upload CA Test waypoints (a.) to BATCAM 1 and 2	
6. **P:** Disable RC mode on the controller (Ch 5)	
7. **O:** Select RC checkbox for lowest altitude BATCAM	
8. **O:** Enable Altitude Override and set 300' for BATCAM 1	
9. **L:** Prepare to launch BATCAM 1	
10. **O:** Select "Takeoff Waypoint" mode	
11. **L:** Launch BATCAM 1	
12. **O:** Confirm BATCAM 1 climbs to 300' in Takeoff orbit	
13. **O:** Confirm BATCAM is established in Takeoff orbit	
14. **O:** Enable Altitude Override and set 200' for BATCAM 2	
15. **L:** Prepare to launch BATCAM 2	
16. **O:** Select "Takeoff Waypoint" mode	
17. **L:** Launch BATCAM 2	
18. **O:** Confirm BATCAM 2 climbs to 200' in Takeoff orbit	

Notes section (item 2):

Name	Setting
Level 1 Loops	
Roll	Checked
Roll Rate	Checked
Pitch	Checked
Pitch Rate	Checked
Yaw Rate	Unchecked
Throttle	"Airspeed"
Level 2 Loops	
Pitch Dynamic Input	"Fixed"

4. 50 ft altitude separation

   20m vertical, 60m lateral separation parameters (2 times simulation settings)

19. **O:** Confirm BATCAM is established in Takeoff orbit
20. **O:** Run CA GUI application
21. **O:** Ensure proper initialization and UAS packet acquisition
22. **O:** Close application
23. **O:** Select "Nav" mode for BATCAM 1
24. **O:** Select "Nav" mode for BATCAM 2
25. **O:** Align UAS for CA encounter in waypoint following mode
26. **O:** Run CA GUI application
27. **O:** Type separation parameters into CA GUI
28. **O:** Upload parameters to CA algorithm (press GUI button)
29. **O:** Verify BATCAMs approach in desired geometry
30. **O:** Record time of encounter
31. **O:** Verify BATCAMs switch to Manual mode and perform CA maneuver
32. **O:** Verify BATCAM return to NAV mode
    - **Yes: proceed to step 32**
    - **No: initiate emergency procedures**
33. **O:** Close CA GUI
34. **O:** Upload new two-ship CA test waypoints
35. **O:** Repeat steps 24-33 for waypoint sets (b.) to (d.)
36. **O:** Repeat test points as necessary
37. **O:** Select "Land" mode
38. **O:** Verify BATCAMs land normally
39. **O:** Assess CA performance for progression to three-ship encounter
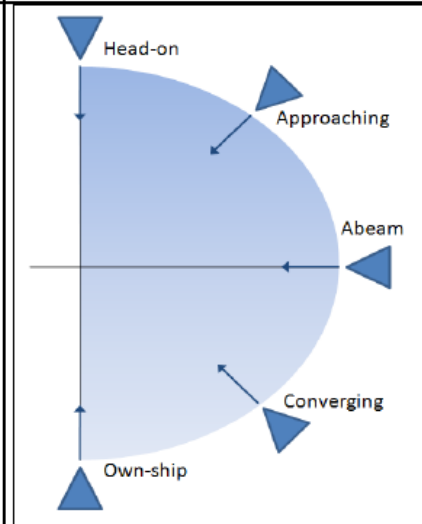
EMERGENCY PROCEDURES for NON-RESPONSIVE BATCAMs in CA MODE

1. **O:** Verify BATCAMs are beyond closest point of approach
2. **O:** Manually press NAV mode button
3. **O:** Verify BATCAMs return to NAV mode
    - **Yes: proceed to normal procedures**
    - **No: proceed to step 4**
4. **O:** Switch to RC mode for lowest altitude BATCAM
5. **O:** Verify RC mode switch
   P: Verify pilot control
    - **Yes: proceed to step 6**
    - **No: proceed to step 11**
6. **P:** Return aircraft to center of test area and maintain holding pattern
7. **O:** Switch to RC mode for remaining BATCAM
8. **P:** Return aircraft to center of test area and maintain holding pattern
9. **O:** Switch BATCAM to NAV mode
10. **O:** Verify BATCAMs return to NAV mode
    - **Yes: proceed to normal procedures**
    - **No: proceed to step 11**
11. **O:** Turn **OFF** COMM Box
12. **O:** Verify BATCAMs proceed to "Lost Comm" point
13. **O:** Turn **ON** COMM Box
14. **O:** Switch BATCAMs to NAV mode
15. **O:** Verify BATCAMs return to NAV mode

FT-2: *Collision Avoidance Algorithm with Three BATCAMs*

Objectives:
1. Demonstrate successful three-ship UAS collision avoidance divert maneuvers resulting from collision encounter
   a. 120 degree angular spacing

FT-2: PROCEDURES	Notes:	Dur: Y min
1. Preflight BATCAM 1, 2, and 3		
2. **O:** Configure Autopilot Manual Mode settings according to CA specifications for each BATCAM	2.	
3. **O:** Upload Configuration / Update Flash as necessary for each BATCAM		
4. **O:** Set "Lost Comm" to 3-5 seconds		
5. **O:** Upload CA Test waypoints (a.) to BATCAM 1, 2, and 3		
6. **P:** Disable RC mode on the controller (Ch 5)		
7. **O:** Select RC checkbox for lowest altitude BATCAM		
8. **O:** Enable Altitude Override and set 400' for BATCAM 1		
9. **L:** Prepare to launch BATCAM 1		
10. **O:** Select "Takeoff Waypoint" mode		
11. **L:** Launch BATCAM 1		
12. **O:** Confirm BATCAM 1 climbs to 400' in Takeoff orbit		
13. **O:** Confirm BATCAM is established in Takeoff orbit		
14. **O:** Enable Altitude Override and set 300' for BATCAM 2		
15. **L:** Prepare to launch BATCAM 2		
16. **O:** Select "Takeoff Waypoint" mode		
17. **L:** Launch BATCAM 2		
18. **O:** Confirm BATCAM 2 climbs to 300' in Takeoff orbit		
19. **O:** Confirm BATCAM is established in Takeoff orbit		
20. **O:** Enable Altitude Override and set 200' for BATCAM 3		
21. **L:** Prepare to launch BATCAM 3		

Notes for step 2:

Name	Setting
Level 1 Loops	
Roll	Checked
Roll Rate	Checked
Pitch	Checked
Pitch Rate	Checked
Yaw Rate	Unchecked
Throttle	"Airspeed"
Level 2 Loops	
Pitch Dynamic Input	"Fixed"

4. 50 ft altitude separation

20m vertical, 60m lateral separation parameters (2 times simulation settings)
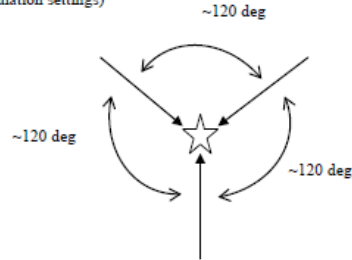
FT-2: PROCEDURES	Notes:	Dur: Y min
22. **O:** Select "Takeoff Waypoint" mode		
23. **L:** Launch BATCAM 3		
24. **O:** Confirm BATCAM 3 climbs to 200' in Takeoff orbit		
25. **O:** Confirm BATCAM is established in Takeoff orbit		
26. **O:** Run CA GUI application		
27. **O:** Ensure proper initialization and UAS packet acquisition		
28. **O:** Close application		
29. **O:** Select "Nav" mode for BATCAM 1		
30. **O:** Select "Nav" mode for BATCAM 2		
31. **O:** Select "Nav" mode for BATCAM 3		
32. **O:** Align UAS for CA encounter in waypoint following mode		
33. **O:** Run CA GUI application		
34. **O:** Type separation parameters into CA GUI		
35. **O:** Upload parameters to CA algorithm (press GUI button)		
36. **O:** Verify BATCAMs approach in desired geometry		
37. **O:** Record time of encounter		
38. **O:** Verify BATCAMs switch to Manual mode and perform CA maneuver		
39. **O:** Verify BATCAM return to NAV mode		
40. **O:** Close CA GUI		
41. **O:** Select "Land" mode		
42. **O:** Verify BATCAMs land normally		

Appendix H: Collision Avoidance Algorithm MATLAB Code

MULT_UAS_AA

```matlab
function [alphadotc gammadotc vdotc pdcKOUT tcKOUT vcKOUT...
    r rh rv rdot conflict cflct_array cflct_arrayv] =...

mult_uas_aa(Rl,VO,DMAX,dt,V_ALL,x_ALL,y_ALL,z_ALL,alpha_ALL,gamma_ALL,...
    vdot_ALL,alphadot_ALL,gammadot_ALL,varargin)

persistent violation evasion evasionv flag flagv
persistent pdcK tcK vcK

Nuas=length(V_ALL);
if Nuas<=1,
    alphadotc=0; %alphadot_ALL;
    gammadotc=0; %gammadot_ALL;
    vdotc=0; %vdot_ALL;
    pdcKOUT=0; %-alphadot_ALL;
    tcKOUT=0; %gamma_ALL;
    vcKOUT=0; %V_ALL;
    r=0.0;
    rh=0.0;
    rv=0.0;
    rdot=0.0;
    conflict=0.0;
    cflct_array=0.0;
    cflct_arrayv=0.0;
    return;
else
    % Pre-allocate some outputs
    conflict=zeros(1,Nuas);
    pdcKOUT=zeros(1,Nuas);
    tcKOUT=zeros(1,Nuas);
    vcKOUT=zeros(1,Nuas);
    vdotc=zeros(1,Nuas);
    gammadotc=zeros(1,Nuas);
    alphadotc=zeros(1,Nuas);
end

if isempty(varargin),
    Nnoncoop=0;
else
    Nnoncoop=length(varargin{1});
end

if isempty(violation),
    violation=zeros(1,Nuas);
    evasion=zeros(1,Nuas);
    evasionv=zeros(1,Nuas);
    flag=zeros(1,Nuas);
    flagv=zeros(1,Nuas);
```

```matlab
elseif length(violation)<Nuas,
    dim_viol=length(violation);
    dim_addv=Nuas-dim_viol;
    violation=[violation zeros(1,dim_addv)];
    evasion=[evasion zeros(1,dim_addv)];
    evasionv=[evasionv zeros(1,dim_addv)];
    flag=[flag zeros(1,dim_addv)];
    flagv=[flagv zeros(1,dim_addv)];
end

if isempty(pdcK),
    pdcK=-alphadot_ALL;
    tcK=gamma_ALL;
    vcK=V_ALL;
elseif length(pdcK)<Nuas,
    dim_pdcK=length(pdcK);
    dim_add=Nuas-dim_pdcK;
    pdcK=[pdcK' zeros(1,dim_add)]';
    tcK=[tcK' gamma_ALL(dim_pdcK+1:end).*ones(1,dim_add)]';
    vcK=[vcK' V_ALL(dim_pdcK+1:end).*ones(1,dim_add)]';
end

if Nnoncoop>0,
    V_F_non=varargin{1};
    x_F_non=varargin{2};
    y_F_non=varargin{3};
    z_F_non=varargin{4};
    beta_non=varargin{5};
    chi_non=varargin{6};
    vfdot_non=varargin{7};
    betadot_non=varargin{8};
    chidot_non=varargin{9};
else
    V_F_non=[];
    x_F_non=[];
    y_F_non=[];
    z_F_non=[];
    beta_non=[];
    chi_non=[];
    vfdot_non=[];
    betadot_non=[];
    chidot_non=[];
end

for jj=1:Nuas,

    V_O=V_ALL(jj);
    x_O=x_ALL(jj);
    y_O=y_ALL(jj);
    z_O=z_ALL(jj);
    alpha=alpha_ALL(jj);
    gamma=gamma_ALL(jj);
    vdot=vdot_ALL(jj);
    if jj==1,
```

```matlab
        indexF=jj+1:Nuas;
    elseif jj==Nuas,
        indexF=1:Nuas-1;
    else
        indexF=[1:jj-1 jj+1:Nuas];
    end
    indexFall=[indexF -1:-1:-length(V_F_non)];
    V_F=[V_ALL(indexF);V_F_non];
    x_F=[x_ALL(indexF);x_F_non];
    y_F=[y_ALL(indexF);y_F_non];
    z_F=[z_ALL(indexF);z_F_non];
    beta=[alpha_ALL(indexF);beta_non];
    chi=[gamma_ALL(indexF);chi_non];
    vfdot=[vdot_ALL(indexF);vfdot_non];
    betadot=[alphadot_ALL(indexF);betadot_non];
    chidot=[gammadot_ALL(indexF);chidot_non];

    %Calculate velocity components
    Vx_O = V_O*cos(alpha)*cos(gamma);
    Vy_O = V_O*sin(alpha)*cos(gamma);
    Vz_O = V_O*sin(gamma);
    Vx_F = V_F.*cos(beta).*cos(chi);
    Vy_F = V_F.*sin(beta).*cos(chi);
    Vz_F = V_F.*sin(chi);

    % Range
    r(1+(jj-1)*(Nuas-1+Nnoncoop):jj*(Nuas-1+Nnoncoop))    = sqrt((z_F-
z_O).^2+(y_F-y_O).^2+(x_F-x_O).^2);
    rdot(1+(jj-1)*(Nuas-1+Nnoncoop):jj*(Nuas-1+Nnoncoop)) = ((Vy_F-
Vy_O).*(y_F-y_O)+(Vx_F-Vx_O).*(x_F-x_O)+(Vz_F-Vz_O).*(z_F-z_O))./...
        sqrt((z_F-z_O).^2+(y_F-y_O).^2+(x_F-x_O).^2);
    rh(1+(jj-1)*(Nuas-1+Nnoncoop):jj*(Nuas-1+Nnoncoop))   = sqrt((y_F-
y_O).^2+(x_F-x_O).^2);
    rv(1+(jj-1)*(Nuas-1+Nnoncoop):jj*(Nuas-1+Nnoncoop))   = sqrt((z_F-
z_O).^2);

    % Define Intruder Inputs for ones in Sensor Volume
    x_F_in=x_F;
    y_F_in=y_F;
    z_F_in=z_F;
    V_F_in=V_F;
    vfdot_in=vfdot;
    beta_in=beta;
    chi_in=chi;
    betadot_in=betadot;
    chidot_in=chidot;

    %Reset CCAA flags
    Fv=violation(jj);
    Fe=evasion(jj);
    Fev=evasionv(jj);
    Ff=flag(jj);
    Ffv=flagv(jj);
    Fsv=zeros(1,Nuas-1+Nnoncoop);
```

```matlab
    %Process inputs
    psiO=-alpha+pi/2;
    psiF=-beta_in+pi/2;
    gammaO=gamma;
    gammaF=chi_in;
    psidF=-betadot_in;
    gammadF=chidot_in;

    %Call CCAA
    [psidotc gamdotc acc violation(jj) evasion(jj) evasionv(jj)
flag(jj) flagv(jj) sensvol(jj,:)...
        conflict(jj) cflct_int cflct_intv]=...
        cc_pn_aa(x_O,y_O,z_O,V_O,vdot,psiO,gammaO,...
        x_F_in,y_F_in,z_F_in,V_F_in,vfdot_in,psiF,...
        gammaF,psidF,gammadF,Rl,VO,Fv,Fe,Fev,Ff,Ffv,Fsv);

    seeint=1:Nuas-1+Nnoncoop;
    cflct_see_int=[];cflct_true_int=[];
    cflct_see_intv=[];cflct_true_intv=[];
    %Determine conflict properties, if any
    if ~isempty(cflct_int),
        cflct_see_int=seeint(cflct_int);
        cflct_true_int=indexFall(cflct_see_int);
    end
    if ~isempty(cflct_intv),
        cflct_see_intv=seeint(cflct_intv);
        cflct_true_intv=indexFall(cflct_see_intv);
    end
    cflct_array(1+(jj-1)*(Nuas-1+Nnoncoop):jj*(Nuas-
1+Nnoncoop))=[cflct_true_int';zeros(Nuas-1+Nnoncoop-
length(cflct_true_int),1)];
    cflct_arrayv(1+(jj-1)*(Nuas-1+Nnoncoop):jj*(Nuas-
1+Nnoncoop))=[cflct_true_intv';zeros(Nuas-1+Nnoncoop-
length(cflct_true_intv),1)];

    %Process outputs
    alphadotc_=-psidotc;
    gammadotc_=gamdotc;
    vdotc_=acc;

    vdotc(jj)=vdotc_;
    gammadotc(jj)=gammadotc_;
    alphadotc(jj)=alphadotc_;

    %Kestrel Commands
    if conflict(jj)>=1,
        pdcK(jj)=-alphadotc(jj);
        tcK(jj)=tcK(jj)+gammadotc(jj)*dt;
        vcK(jj)=vcK(jj)+vdotc(jj)*dt;
        %Range Check
        csum=0;
        for uu=1+(jj-1)*(Nuas-1):jj*(Nuas-1),
            r_uu=r(uu);
```

```matlab
            if ((cflct_array(uu)~=0) && (cflct_arrayv(uu)~=0) &&
r_uu<DMAX);
                csum=1;
                break;
            end
        end
        if csum==0,
            conflict(jj)=0;
            vdotc(jj)=0;
            alphadotc(jj)=0;
            gammadotc(jj)=0;
            vcK(jj)=V_ALL(jj);
            pdcK(jj)=-alphadotc(jj);
            tcK(jj)=gamma_ALL(jj);
            violation(jj)=0;
            evasion(jj)=0;
            evasionv(jj)=0;
            flag(jj)=0;
            flagv(jj)=0;
        end
    else
        pdcK(jj)=-alphadotc(jj);
        tcK(jj)=gamma_ALL(jj);
        vcK(jj)=V_ALL(jj);
    end

end

%Coordinate Commands
cflct_idx=find(conflict>=1);
cflct_idx_len=length(cflct_idx);
if cflct_idx_len>1,
    for jj=1:cflct_idx_len-1,
        primary=cflct_idx(jj);
        others=cflct_idx(jj+1:end);
        for kk=1:length(others),
            if sign(gammadotc(primary))==sign(gammadotc(others(kk))),
                gammadotc(others(kk))=-gammadotc(others(kk));

tcK(others(kk))=tcK(others(kk))+2*gammadotc(others(kk))*dt;
            end
        end
    end
end

pdcKOUT=pdcK;
tcKOUT=tcK;
vcKOUT=vcK;
```

171

CC_PN_AA

```
function [psidotc gamdotc acc violation evasion evasionv flag flagv
sensvol...
    conflict cflct_int cflct_intv]=...

cc_pn_aa(x_O,y_O,z_O,V_O,vdot,psiO,gammaO,x_F,y_F,z_F,V_F,vfdot,psiF,...
    gammaF,psidF,gammadF,Rl,VO,Fv,Fe,Fev,Ff,Ffv,Fsv)

%Minimum Separation Parameters
R=Rl;
Hv=VO;

%Determine number if intruders
Ni=length(x_F);

%Convert aircraft parameters to Collision Cone parameters
alpha=wrap_mpi2pi(-psiO+pi/2);
beta(:,1)=wrap_mpi2pi(-psiF+pi/2);
gamma=wrap_mpi2pi(gammaO);
chi(:,1)=wrap_mpi2pi(gammaF);

betadot=-psidF;
chidot=gammadF;

%Calculate velocity components
Vx_O = V_O*cos(alpha)*cos(gamma);
Vy_O = V_O*sin(alpha)*cos(gamma);
Vz_O = V_O*sin(gamma);
Vx_F = V_F.*cos(beta).*cos(chi);
Vy_F = V_F.*sin(beta).*cos(chi);
Vz_F = V_F.*sin(chi);

%Initialize Flags
violation = Fv;
evasion = Fe;
evasionv = Fev;
flag = Ff;
flagv = Ffv;
sensvol = Fsv;

%Calculate necessary parameters
% range
r = sqrt((z_F-z_O).^2+(y_F-y_O).^2+(x_F-x_O).^2);
rh = sqrt((y_F-y_O).^2+(x_F-x_O).^2);
% line of sight angle
th=atan2(y_F-y_O,x_F-x_O);
phii_=atan2(z_F-z_O,cos(th).*(x_F-x_O)+sin(th).*(y_F-y_O));
phii=atan2(z_F-z_O,cos(alpha).*(x_F-x_O)+sin(alpha).*(y_F-y_O));
% relative velocity along LOS
vri=V_F.*cos(beta-th).*cos(chi-phii_)-V_O.*cos(alpha-th).*cos(gamma-
phii_);
```

```matlab
% relative velocity perp to LOS
vthi=V_F.*sin(beta-th).*cos(chi-phii_)-V_O.*sin(alpha-th).*cos(gamma-
phii_);
vphi=V_F.*sin(chi-phii_)-V_O.*sin(gamma-phii_);
% relative velocity magnitude
vrel = sqrt((V_O*cos(alpha)*cos(gamma)-V_F.*cos(beta).*cos(chi)).^2+...
    (V_O*sin(alpha)*cos(gamma)-V_F.*sin(beta).*cos(chi)).^2+...
    (V_O*sin(gamma)-V_F.*sin(chi)).^2);
% relative heading
psirel = atan2((V_O*sin(alpha)-V_F.*sin(beta)),(V_O*cos(alpha)-
V_F.*cos(beta)));
phirel = atan2((V_O*sin(gamma)-V_F.*sin(chi)),(V_O*cos(gamma)-
V_F.*cos(chi)));
% Relative azimuth and elevation calculations
xrfeh = cos(alpha)*(x_F-x_O)+sin(alpha)*(y_F-y_O);
yrfeh = -sin(alpha)*(x_F-x_O)+cos(alpha)*(y_F-y_O);
relaz = atan2(-yrfeh,xrfeh);
rrfev = sqrt(xrfeh.^2+yrfeh.^2)*cos(gamma)+(z_F-z_O)*sin(gamma);
zrfev = -sqrt(xrfeh.^2+yrfeh.^2)*sin(gamma)+(z_F-z_O)*cos(gamma);
relel = atan2(zrfev,rrfev);


%Vertical collision circle calculations
% vertical line of sights to hockey-puck corners
rRv=sqrt((z_F-z_O).^2+(cos(alpha).*(x_F-x_O)+sin(alpha).*(y_F-
y_O)).^2);
for oo = 1:Ni,
    if (z_F(oo)-z_O) >= (Hv),
        philow(oo) = atan2(rRv(oo).*sin(phii(oo))-
(Hv),rRv(oo).*cos(phii(oo))+R);
        phihig(oo) =
atan2(rRv(oo).*sin(phii(oo))+(Hv),rRv(oo).*cos(phii(oo))-R);
    elseif (z_F(oo)-z_O) < (Hv) && (z_F(oo)-z_O) > -(Hv),
        philow(oo) = atan2(rRv(oo).*sin(phii(oo))-
(Hv),rRv(oo).*cos(phii(oo))-R);
        phihig(oo) =
atan2(rRv(oo).*sin(phii(oo))+(Hv),rRv(oo).*cos(phii(oo))-R);
    else
        philow(oo) = atan2(rRv(oo).*sin(phii(oo))-
(Hv),rRv(oo).*cos(phii(oo))-R);
        phihig(oo) =
atan2(rRv(oo).*sin(phii(oo))+(Hv),rRv(oo).*cos(phii(oo))+R);
    end
end
psiv = phihig-philow;
Rv = rRv.*sin(psiv'/2);
% center of vertical circle
x_ccv = rRv.*abs(cos(philow' + psiv'/2)).*cos(alpha).*sign(xrfeh); %ALS
- ABS value function
y_ccv = rRv.*abs(cos(philow' + psiv'/2)).*sin(alpha).*sign(xrfeh); %ALS
- ABS value function
z_ccv = rRv.*sin(philow' + psiv'/2);
% LOS and relative velocities to vertical circle
phic_=atan2(z_ccv,cos(th).*(x_ccv)+sin(th).*(y_ccv));
phic=atan2(z_ccv,cos(alpha).*(x_ccv)+sin(alpha).*(y_ccv));
```

```matlab
vrc=V_F.*cos(beta-alpha).*cos(chi-phic)-V_O.*cos(alpha-
alpha).*cos(gamma-phic);
vthc=V_F.*sin(beta-alpha).*cos(chi-phic)-V_O.*sin(alpha-
alpha).*cos(gamma-phic);
vphc=V_F.*sin(chi-phic)-V_O.*sin(gamma-phic);
% rate of change of collision avoidance vector angle
gam = asin(R./rh);
thdot = ((Vy_F-Vy_O).*(x_F-x_O)-(Vx_F-Vx_O).*(y_F-y_O))./...
    ((x_F-x_O).^2+(y_F-y_O).^2);
gamv = asin(Rv./rRv);
phi = phic;
h_Phi = sqrt((x_F-x_O).^2+(y_F-y_O).^2);
Vh_Phi = sqrt((Vx_F-Vx_O).^2+(Vy_F-Vy_O).^2);
pdden = (h_Phi.^2+(z_F-z_O).^2);
for pd_i=1:length(pdden),
    pdden(pd_i)=max([1e-6 pdden(pd_i)]);
end
phidot = ((Vz_F-Vz_O).*h_Phi-Vh_Phi.*(z_F-z_O))./pdden;

%Collision Cone parameters
mu = beta - th;
nu = (V_O*cos(gamma))./(V_F.*cos(chi));
p = R./sqrt(rh.^2-R^2);
%muv = chi - phi;
for kl=1:Ni,
    if sign(xrfeh(kl))>0 && cos(alpha-beta(kl))<0,
        muv(kl) = (pi-chi(kl)) - phi(kl);
    else
        muv(kl) = chi(kl) - phi(kl);
    end
end
nuv = (V_O)./(V_F.*abs(cos(beta-alpha))); %ALS - ABS value function
pv = Rv./sqrt(rRv.^2-Rv.^2);

pdot = vri.*(-p.^3.*rh/(R^2));
mudot = betadot-thdot;
pdotv = vrc.*(-pv.^3.*rRv./(Rv.^2));
mudotv = chidot-phidot;
nudot = (vdot*cos(gamma))./(V_F.*cos(chi)) - nu.*vfdot./V_F; %ALS
nudotv = vdot./(V_F.*abs(cos(beta-alpha))) - nuv.*vfdot./V_F; %ALS

%Collision Check
% initialize cone variables as empty
acount=[];
alpha_up=[];alpha_dn=[];alpha_up_dot=[];alpha_dn_dot=[];
acountv=[];
alpha_upv=[];alpha_dnv=[];alpha_up_dotv=[];alpha_dn_dotv=[];
% check each Intruder
for kk = 1:Ni,
    %Check for intruder in sensor volume
    %Check for miss distance violation
    violation1=zeros(1,Ni);
    if r(kk) <= R && z_O>z_F(kk)-Hv && z_O<z_F(kk)+Hv,
        violation1(kk) = 1;
        alpha_up=[];alpha_dn=[];alpha_up_dot=[];alpha_dn_dot=[];
```

174

```matlab
        alpha_upv=[];alpha_dnv=[];alpha_up_dotv=[];alpha_dn_dotv=[];
        break;
    elseif violation == 1,
        alpha_up=[];alpha_dn=[];alpha_up_dot=[];alpha_dn_dot=[];
        alpha_upv=[];alpha_dnv=[];alpha_up_dotv=[];alpha_dn_dotv=[];
        break;
    else
        %Call Collision Cone
        [eta_up,eta_dn,eta_up_dot,eta_dn_dot] = ...
f_collisioncone4(mu(kk),nu(kk),p(kk),mudot(kk),nudot(kk),pdot(kk));
        [eta_upv,eta_dnv,eta_up_dotv,eta_dn_dotv] = ...
f_collisioncone4(muv(kk),nuv(kk),pv(kk),mudotv(kk),nudotv(kk),pdotv(kk)...
);
        %Dispose of invalid cones
        deta=abs(eta_up-eta_dn);
        detav=abs(eta_upv-eta_dnv);
        ide=find(deta>1e-4 & deta<2*pi-1e-4);
        idev=find(detav>1e-4 & detav<2*pi-1e-4);
        eta_up=eta_up(ide);
        eta_up_dot=eta_up_dot(ide);
        eta_dn=eta_dn(ide);
        eta_dn_dot=eta_dn_dot(ide);
        eta_upv=eta_upv(idev);
        eta_up_dotv=eta_up_dotv(idev);
        eta_dnv=eta_dnv(idev);
        eta_dn_dotv=eta_dn_dotv(idev);
        %Define Angular Limits of cones
        alpha_up = [alpha_up; eta_up' + th(kk)];
        alpha_dn = [alpha_dn; eta_dn' + th(kk)];
        alpha_up_dot = [alpha_up_dot; eta_up_dot' + thdot(kk)];
        alpha_dn_dot = [alpha_dn_dot; eta_dn_dot' + thdot(kk)];

        alpha_upv = [alpha_upv; eta_upv' + phi(kk)];
        alpha_dnv = [alpha_dnv; eta_dnv' + phi(kk)];
        alpha_up_dotv = [alpha_up_dotv; eta_up_dotv' + phidot(kk)];
        alpha_dn_dotv = [alpha_dn_dotv; eta_dn_dotv' + phidot(kk)];

        %Record the number of cones
        acount = [acount length(eta_up)];
        acountv = [acountv length(eta_upv)];
    end
end
violation = max(violation1);

%Check and correct for horizontal cone overlap
overlap_reg=[];
if ~isempty(alpha_up),
    alpha_up_P = wrap_pos(alpha_up);
    alpha_dn_N = wrap_neg(alpha_dn);
    alpha_up_N = wrap_neg(alpha_up);
    alpha_dn_P = wrap_pos(alpha_dn);
    [alfs aidx]=sort(alpha_up_P);
    alpha_up_P=alpha_up_P(aidx);
    alpha_dn_N=alpha_dn_N(aidx);
    alpha_up_N=alpha_up_N(aidx);
```

175

```matlab
    alpha_dn_P=alpha_dn_P(aidx);
    alpha_up=alpha_up(aidx);
    alpha_dn=alpha_dn(aidx);
    if sum(acount)>1,
        for nn = 1:sum(acount)-1,
            mmsweep=nn+1:sum(acount);
            for mm = mmsweep,
                %Correct for Quadrant 1 and 4 overlap/non-overlap and
                % zero boundary
                if alpha_dn_N(nn)>alpha_up_N(nn) &&
alpha_dn_N(nn)~=alpha_up_N(mm),
                    alpha_dn_N(nn)=alpha_dn_N(nn)-2*pi;
                    alpha_dn_P(nn)=alpha_dn_P(nn)-2*pi;
                end
                if alpha_dn_N(mm)>alpha_up_N(mm) &&
alpha_dn_N(mm)~=alpha_up_N(nn),
                    alpha_dn_N(mm)=alpha_dn_N(mm)-2*pi;
                    alpha_dn_P(mm)=alpha_dn_P(mm)-2*pi;
                end
                if alpha_up_N(nn)>alpha_dn_N(mm) &&
alpha_dn_P(mm)<alpha_up_P(nn),
                    overlap_reg=[overlap_reg;nn mm];
                    [temp loc]=max([alpha_up_P(nn) alpha_up_P(mm)]);
                    if loc==1,
                        alpha_up(mm)=alpha_up(nn);
                        alpha_up_dot(mm)=alpha_up_dot(nn);
                        %%ALS
                        alpha_up_P(mm)=alpha_up_P(nn);
                        alpha_up_N(mm)=alpha_up_N(nn);
                        %%
                    elseif loc==2,
                        alpha_up(nn)=alpha_up(mm);
                        alpha_up_dot(nn)=alpha_up_dot(mm);
                        %%ALS
                        alpha_up_P(nn)=alpha_up_P(mm);
                        alpha_up_N(nn)=alpha_up_N(mm);
                        %%
                    end
                    [temp1 loc1]=min([alpha_dn_N(nn) alpha_dn_N(mm)]);
                    if loc1==1,
                        alpha_dn(mm)=alpha_dn(nn);
                        alpha_dn_dot(mm)=alpha_dn_dot(nn);
                        %%ALS
                        alpha_dn_N(mm)=alpha_dn_N(nn);
                        alpha_dn_P(mm)=alpha_dn_P(nn);
                        %%
                    elseif loc1==2,
                        alpha_dn(nn)=alpha_dn(mm);
                        alpha_dn_dot(nn)=alpha_dn_dot(mm);
                        %%ALS
                        alpha_dn_N(nn)=alpha_dn_N(mm);
                        alpha_dn_P(nn)=alpha_dn_P(mm);
                        %%
                    end
                end
```

176

```matlab
                    end
                end
            end
end
%Check and correct for vertical cone overlap
overlapv_reg=[];
if ~isempty(alpha_upv),
    alpha_up_Pv = wrap_pos(alpha_upv);
    alpha_dn_Nv = wrap_neg(alpha_dnv);
    alpha_up_Nv = wrap_neg(alpha_upv);
    alpha_dn_Pv = wrap_pos(alpha_dnv);
    [alfsv aidxv]=sort(alpha_up_Pv);
    alpha_up_Pv=alpha_up_Pv(aidxv);
    alpha_dn_Nv=alpha_dn_Nv(aidxv);
    alpha_up_Nv=alpha_up_Nv(aidxv);
    alpha_dn_Pv=alpha_dn_Pv(aidxv);
    alpha_upv=alpha_upv(aidxv);
    alpha_dnv=alpha_dnv(aidxv);
    if sum(acountv)>1,
        for nn = 1:sum(acountv)-1,
            mmsweep=nn+1:sum(acountv);
            for mm = mmsweep,
                %Correct for Quadrant 1 and 4 overlap/non-overlap and
                % zero boundary
                if alpha_dn_Nv(nn)>alpha_up_Nv(nn) &&
alpha_dn_Nv(nn)~=alpha_up_Nv(mm),
                    alpha_dn_Nv(nn)=alpha_dn_Nv(nn)-2*pi;
                    alpha_dn_Pv(nn)=alpha_dn_Pv(nn)-2*pi;
                end
                if alpha_dn_Nv(mm)>alpha_up_Nv(mm) &&
alpha_dn_Nv(mm)~=alpha_up_Nv(nn),
                    alpha_dn_Nv(mm)=alpha_dn_Nv(mm)-2*pi;
                    alpha_dn_Pv(mm)=alpha_dn_Pv(mm)-2*pi;
                end
                if alpha_up_Nv(nn)>alpha_dn_Nv(mm) &&
alpha_dn_Pv(mm)<alpha_up_Pv(nn),
                    overlapv_reg=[overlapv_reg;nn mm];
                    [temp loc]=max([alpha_up_Pv(nn) alpha_up_Pv(mm)]);
                    if loc==1,
                        alpha_upv(mm)=alpha_upv(nn);
                        alpha_up_dotv(mm)=alpha_up_dotv(nn);
                        %%ALS
                        alpha_up_Pv(mm)=alpha_up_Pv(nn);
                        alpha_up_Nv(mm)=alpha_up_Nv(nn);
                        %%
                    elseif loc==2,
                        alpha_upv(nn)=alpha_upv(mm);
                        alpha_up_dotv(nn)=alpha_up_dotv(mm);
                        %%ALS
                        alpha_up_Pv(nn)=alpha_up_Pv(mm);
                        alpha_up_Nv(nn)=alpha_up_Nv(mm);
                        %%
                    end
                    [temp1 loc1]=min([alpha_dn_Nv(nn)
alpha_dn_Nv(mm)]);
```

```matlab
                    if loc1==1,
                        alpha_dnv(mm)=alpha_dnv(nn);
                        alpha_dn_dotv(mm)=alpha_dn_dotv(nn);
                        %%ALS
                        alpha_dn_Nv(mm)=alpha_dn_Nv(nn);
                        alpha_dn_Pv(mm)=alpha_dn_Pv(nn);
                        %%
                    elseif loc1==2,
                        alpha_dnv(nn)=alpha_dnv(mm);
                        alpha_dn_dotv(nn)=alpha_dn_dotv(mm);
                        %%ALS
                        alpha_dn_Nv(nn)=alpha_dn_Nv(mm);
                        alpha_dn_Pv(nn)=alpha_dn_Pv(mm);
                        %%
                    end
                end
            end
        end
    end
end
alpha_up_p=[];alpha_dn_p=[];alpha_up_pv=[];alpha_dn_pv=[];
alpha_up_p = alpha_up;
alpha_dn_p = alpha_dn;
alpha_up_pv = alpha_upv;
alpha_dn_pv = alpha_dnv;

%horizontal evasive maneuver
for jj=1:length(alpha_up),
    alpha_up_wrap = wrap_mpi2pi(alpha_up(jj));
    alpha_dn_wrap = wrap_mpi2pi(alpha_dn(jj));
    alpha_up_norm = alpha_up_wrap-alpha;
    alpha_dn_norm = alpha_dn_wrap-alpha;
    if alpha_up_norm<alpha_dn_norm,
        alpha_dn_norm=alpha_dn_norm-2*pi;
    end
    %
    if flag == 0,
        if alpha_up_dot(jj)>=0 && alpha_dn_dot(jj)<=0,
            if abs(alpha_up_dot(jj))<=abs(alpha_dn_dot(jj)),
                angle=alpha_dn(jj);
                angle_dot = alpha_dn_dot(jj); flag=2;
            else
                angle=alpha_up(jj);
                angle_dot = alpha_up_dot(jj); flag=1;
            end
        elseif alpha_up_dot(jj)<=0 && alpha_dn_dot(jj)>=0,
            if abs(alpha_up_dot(jj))<=abs(alpha_dn_dot(jj)),
                angle=alpha_dn(jj);
                angle_dot = alpha_dn_dot(jj); flag=2;
            else
                angle=alpha_up(jj);
                angle_dot = alpha_up_dot(jj); flag=1;
            end
        elseif alpha_up_dot(jj)>=0 && alpha_dn_dot(jj)>=0,
            if abs(alpha_up_dot(jj))<=abs(alpha_dn_dot(jj)),
```

```matlab
                angle=alpha_up(jj);
                angle_dot = alpha_up_dot(jj); flag=1;
            else
                angle=alpha_dn(jj);
                angle_dot = alpha_dn_dot(jj); flag=2;
            end
        elseif alpha_up_dot(jj)<=0 && alpha_dn_dot(jj)<=0,
            if abs(alpha_up_dot(jj))<=abs(alpha_dn_dot(jj)),
                angle=alpha_up(jj);
                angle_dot = alpha_up_dot(jj); flag=1;
            else
                angle=alpha_dn(jj);
                angle_dot = alpha_dn_dot(jj); flag=2;
            end
        end
    elseif flag==1,
        angle=alpha_up(jj);
        angle_dot = alpha_up_dot(jj);
    elseif flag==2,
        angle=alpha_dn(jj);
        angle_dot = alpha_dn_dot(jj);
    end
    angle_dot_vec=angle_dot;
    %
    if ((alpha_up_norm >=0.0) && (alpha_dn_norm<=0)) ||...
            ((alpha_up_norm <= -3*pi/2) && (alpha_dn_norm <=-2*pi)) ||...
            ((alpha_up_norm >= 2*pi) && (alpha_dn_norm >=3*pi/2)),
        evasion = 1;
        %
        % proportional gain
        N = 2;
        % acceleration command
        a = N*V_O*angle_dot;
        % UAS accleration command components
        vdot(jj) = -a * sin(angle - alpha);
        alphadot(jj) = -a/V_O * cos(angle - alpha);

    else
        alphadot(jj) = 0.0;
        vdot(jj) = 0.0;
    end

end

%vertical evasive maneuver
for jj=1:length(alpha_upv),
    alpha_up_wrapv = wrap_mpi2pi(alpha_upv(jj));
    alpha_dn_wrapv = wrap_mpi2pi(alpha_dnv(jj));
    alpha_up_normv = alpha_up_wrapv-gamma;
    alpha_dn_normv = alpha_dn_wrapv-gamma;
    if alpha_up_normv<alpha_dn_normv,
        alpha_dn_normv=alpha_dn_normv-2*pi;
    end
    %
```

```matlab
if flagv == 0,
    if alpha_up_dotv(jj)>=0 && alpha_dn_dotv(jj)<=0,
        if abs(alpha_up_dotv(jj))<=abs(alpha_dn_dotv(jj)),
            anglev=alpha_upv(jj);
            angle_dotv = alpha_up_dotv(jj); flagv=1;
        else
            anglev=alpha_dnv(jj);
            angle_dotv = alpha_dn_dotv(jj); flagv=2;
        end
    elseif alpha_up_dotv(jj)<=0 && alpha_dn_dotv(jj)>=0,
        if abs(alpha_up_dotv(jj))<=abs(alpha_dn_dotv(jj)),
            anglev=alpha_dnv(jj);
            angle_dotv = alpha_dn_dotv(jj); flagv=2;
        else
            anglev=alpha_upv(jj);
            angle_dotv = alpha_up_dotv(jj); flagv=1;
        end
    elseif alpha_up_dotv(jj)>=0 && alpha_dn_dotv(jj)>=0,
        if abs(alpha_up_dotv(jj))<=abs(alpha_dn_dotv(jj)),
            anglev=alpha_dnv(jj);
            angle_dotv = alpha_dn_dotv(jj); flagv=2;
        else
            anglev=alpha_upv(jj);
            angle_dotv = alpha_up_dotv(jj); flagv=1;
        end
    elseif alpha_up_dotv(jj)<=0 && alpha_dn_dotv(jj)<=0,
        if abs(alpha_up_dotv(jj))<=abs(alpha_dn_dotv(jj)),
            anglev=alpha_upv(jj);
            angle_dotv = alpha_up_dotv(jj); flagv=1;
        else
            anglev=alpha_dnv(jj);
            angle_dotv = alpha_dn_dotv(jj); flagv=2;
        end
    end
elseif flagv==1,
    anglev=alpha_upv(jj);
    angle_dotv = alpha_up_dotv(jj);
elseif flagv==2,
    anglev=alpha_dnv(jj);
    angle_dotv = alpha_dn_dotv(jj);
end
angle_dot_vecv=angle_dotv;
%ALS
anglev=wrap_mpi2pi(anglev);
%
if (alpha_up_normv >=0.0) && (alpha_dn_normv<=0),
    evasionv = 1;
    %
    % proportional gain
    N = 2;
    % acceleration command
    a = N*V_O*angle_dotv;
    % UAS accleration command components
    vdotv(jj) = -a * sin(anglev - gamma);
    gammadot(jj) = -a/V_O * cos(anglev - gamma);
```

```matlab
    else
        gammadot(jj) = 0.0;
        vdotv(jj) = 0.0;
    end

end

if isempty(alpha_up),
    alphadot=[];
    vdot=[];
end
if isempty(alpha_upv),
    gammadot=[];
    vdotv=[];
end

%Choose command from vector of potential commands
if evasion == 1 && evasionv ==1 && norm([alphadot vdot gammadot
vdotv])~=0,
    %Conflict Properties
    conflict = 1;
    cflct_cone = find(alphadot~=0);
    cflct_int = [];
    for ww=1:length(cflct_cone),
        for xx=1:length(acount),
            if cflct_cone(ww)<=sum(acount(1:xx)),
                if isempty(cflct_int) || xx~=cflct_int(end),
                    cflct_int=[cflct_int xx]; %record index of intruder
                    break;
                end
            end
        end
        if cflct_int(end)==length(acount),
            break; %break loop if there is already a conflict with the
last intruder
        end
    end
    cflct_conev = find(gammadot~=0);
    cflct_intv = [];
    for ww=1:length(cflct_conev),
        for xx=1:length(acountv),
            if cflct_conev(ww)<=sum(acountv(1:xx)),
                if isempty(cflct_intv) || xx~=cflct_intv(end),
                    cflct_intv=[cflct_intv xx]; %record index of
intruder
                    break;
                end
            end
        end
        if cflct_intv(end)==length(acountv),
            break; %break loop if there is already a conflict with the
last intruder
        end
```

181

```matlab
    end

    %Commands
    sad=sum(alphadot);
    ssad=sign(sad);
    sadv=sum(gammadot);
    ssadv=sign(sadv);
    if ssad>0,
        i1=find(alphadot>0);
        [mad,imad]=min(alphadot(i1));
        i11=i1(imad);
        vdotc1 = vdot(i11);
        alphadotc = alphadot(i11);
    elseif ssad<0,
        i2=find(alphadot<0);
        [mad2,imad2]=max(alphadot(i2));
        i22=i2(imad2);
        vdotc1 = vdot(i22);
        alphadotc = alphadot(i22);
    else
        vdotc1 = 0.0;
        alphadotc = 0.0;
        flag=0;
        evasion=0;
    end
    if ssadv>0,
        i1=find(gammadot>0);
        [mad,imad]=min(gammadot(i1));
        i11=i1(imad);
        if ~isempty(i11)
            vdotc2 = vdotv(i11);
            gammadotc = gammadot(i11)-vdotc1/V_O*sin(gamma);
        end
    elseif ssadv<0,
        i2=find(gammadot<0);
        [mad2,imad2]=max(gammadot(i2));
        i22=i2(imad2);
        if ~isempty(i22)
            vdotc2 = vdotv(i22);
            gammadotc = gammadot(i22)-vdotc1/V_O*sin(gamma);
        end
    else
        vdotc2 = 0.0;
        gammadotc = -vdotc1/V_O*sin(gamma);
        flagv=0;
        evasionv=0;
    end
    vdotc = vdotc1*cos(gamma) + vdotc2;
else
    conflict = 0;
    cflct_int = [];
    cflct_intv = [];
    vdotc = 0.0;
    alphadotc = 0.0;
    gammadotc = 0.0;
```

```
        evasion = 0;
        evasionv = 0;
        flag=0;
        flagv=0;
end


% cc_plot(Ni,r,x_O,y_O,z_O,V_O,alpha,gamma,x_F,y_F,z_F,V_F,...
%
beta,chi,R,Rv,VO,violation,x_ccv,y_ccv,z_ccv,alpha_up_p,alpha_dn_p,...
%      alpha_up_pv,alpha_dn_pv,sensvol,acount,acountv)

%Convert Collision Cone commands to Aircraft commands
psidotc=-alphadotc;
gamdotc=gammadotc;
acc=vdotc;
```

F_COLLISIONCONE4

```
function [eta_up,eta_dn,eta_up_dot,eta_dn_dot] =
f_collisioncone4(mu,nu,p,mudot,nudot,pdot)

% FUNCTION f_collisioncone2
%
% [alpha_up,alpha_dn] = f_collisioncone2(mu,nu,p,mudot,nudot,pdot)
%
% INPUTS:
%  mu = beta - theta0
%  nu = V_A/V_B
%  p  = R/sqrt(r0^2-R^2) - For Circular Object
%     = abs(tan(psi0/2)) - For Irregular Shaped Object
%  mudot = betadot - theta0dot
%  nudot = A_A/V_B - nu*A_F/V_F
%  pdot = -Vr*p^3*r0/(R^2)
%
% OUTPUTS:
%  eta_up - upper angular collision cone limit
%  eta_dn - lower angular collision cone limit
%
% This function calculates the angular limits of the collision cone
from
% the point A to the object B. There can be a double collision cone
% depending on the geometry of the encounter. In this case, there will
be
% two sets of angular limits.

A = (p*cos(mu) + sin(mu))/sqrt(p^2+1);
zeta = asin(p/sqrt(p^2+1));
Atilde = (p*cos(mu) - sin(mu))/sqrt(p^2+1);
zetatilde = pi - zeta;

Adot = pdot*(cos(zeta)*cos(mu) -
cos(zeta)^2*sin(zeta)*(p*cos(mu)+sin(mu))) + ...
```

183

```matlab
    mudot*cos(zeta)*(cos(mu)-p*sin(mu)));
zetadot = pdot*cos(zeta)^2;
Atildedot = pdot*(-cos(zetatilde)*cos(mu) -
cos(zetatilde)^2*sin(zetatilde)*(p*cos(mu)-sin(mu))) + ...
    mudot*cos(zetatilde)*(cos(mu)+p*sin(mu));
zetatildedot = -zetadot;

% Collision Cone Boundaries
% Satisfy (Vr0 < 0)
if cos(mu)/nu >= 1,
    N1_up = [];
    N1_dn = [];
    %no cone
    eta_up = [];
    eta_dn = [];
    eta_up_dot = [];
    eta_dn_dot = [];
    return;
elseif (cos(mu)/nu >= -1 && cos(mu)/nu < 1),
    N1_up = acos(cos(mu)/nu);
    N1_dn = -acos(cos(mu)/nu);
elseif cos(mu)/nu < -1,
    N1_up = 2*pi;
    N1_dn = 0;
end
% Satisfy (Vth0^2 <= p^2*Vr0^2) -> (Vth0 <= -p*Vr0)
if A/nu > 1,
    N21_up = [];
    N21_dn = [];
    N21_up_dot = [];
    N21_dn_dot = [];
elseif (A/nu >= 0 && A/nu <= 1),
    N21_up = pi - asin(A/nu) - zeta;
    N21_dn = asin(A/nu) - zeta;
    N21_up_dot = tan(pi-N21_up-zeta)*nudot/nu-Adot/(cos(pi-N21_up-
zeta)*nu)-zetadot;
    N21_dn_dot = Adot/(cos(N21_dn+zeta)*nu)-tan(N21_dn+zeta)*nudot/nu-
zetadot;
elseif (A/nu > -1 && A/nu < 0),
    N21_up = -pi - asin(A/nu) - zeta;
    N21_dn = asin(A/nu) - zeta;
    N21_up_dot = tan(-pi-N21_up-zeta)*nudot/nu-Adot/(cos(-pi-N21_up-
zeta)*nu)-zetadot;
    N21_dn_dot = Adot/(cos(N21_dn+zeta)*nu)-tan(N21_dn+zeta)*nudot/nu-
zetadot;
elseif A/nu <= -1,
    N21_up = 2*pi;
    N21_dn = 0;
    N21_up_dot = 0;
    N21_dn_dot = 0;
end
% Satisfy (Vth0^2 <= p^2*Vr0^2) -> (p*Vr0 <= Vth0)
if Atilde/nu > 1,
    N22_up = [];
    N22_dn = [];
```

```matlab
    N22_up_dot = [];
    N22_dn_dot = [];
elseif (Atilde/nu >= 0 && Atilde/nu <= 1),
    N22_up = pi - asin(Atilde/nu) - zetatilde;
    N22_dn = asin(Atilde/nu) - zetatilde;
    N22_up_dot = tan(pi-N22_up-zetatilde)*nudot/nu-Atildedot/(cos(pi-
N22_up-zetatilde)*nu)-zetatildedot;
    N22_dn_dot = Atildedot/(cos(N22_dn+zetatilde)*nu)-
tan(N22_dn+zetatilde)*nudot/nu-zetatildedot;
elseif (Atilde/nu > -1 && Atilde/nu < 0),
    N22_up = -pi - asin(Atilde/nu) - zetatilde;
    N22_dn = asin(Atilde/nu) - zetatilde;
    N22_up_dot = tan(-pi-N22_up-zetatilde)*nudot/nu-Atildedot/(cos(-pi-
N22_up-zetatilde)*nu)-zetatildedot;
    N22_dn_dot = Atildedot/(cos(N22_dn+zetatilde)*nu)-
tan(N22_dn+zetatilde)*nudot/nu-zetatildedot;
elseif Atilde/nu <= -1,
    N22_up = 2*pi;
    N22_dn = 0;
    N22_up_dot = 0;
    N22_dn_dot = 0;
end
% N21 |-| N22
if ((A/nu > 1) || (Atilde/nu >1)),
    N2_up = [];
    N2_dn = [];
    N2_up_dot = [];
    N2_dn_dot = [];
elseif (A/nu > -1 && A/nu <= 1) && (Atilde/nu > -1 && Atilde/nu <= 1),
    if (nu >= 1) && (round(zeta*1e10)/1e10 >=
round((0.5*abs(asin(A/nu)+asin(Atilde/nu)))*1e10)/1e10),
        N2_up = N22_up;
        N2_dn = N21_dn;
        N2_up_dot = N22_up_dot;
        N2_dn_dot = N21_dn_dot;
    elseif (nu < 1) && ((zeta >= 0) && (zeta <=
0.5*abs(asin(A/nu)+asin(Atilde/nu)))),
        N2_up = [N22_up N21_up];
        N2_dn = [N21_dn N22_dn];
        N2_up_dot = [N22_up_dot N21_up_dot];
        N2_dn_dot = [N21_dn_dot N22_dn_dot];
    else
        N2_up = [];
        N2_dn = [];
        N2_up_dot = [];
        N2_dn_dot = [];
    end
else
    if (A/nu <= -1) && ((Atilde/nu <= 1) && (Atilde/nu > -1)),
        N2_up = N22_up;
        N2_dn = N22_dn;
        N2_up_dot = N22_up_dot;
        N2_dn_dot = N22_dn_dot;
    elseif (Atilde/nu <= -1) && ((A/nu <= 1) && (A/nu > -1)),
        N2_up = N21_up;
```

```
            N2_dn = N21_dn;
            N2_up_dot = N21_up_dot;
            N2_dn_dot = N21_dn_dot;
        elseif (A/nu <= -1) && (Atilde/nu <= -1),
            N2_up = 2*pi;
            N2_dn = 0;
            N2_up_dot = 0;
            N2_dn_dot = 0;
        end
end


eta_up = N2_up;
eta_dn = N2_dn;
eta_up_dot = N2_up_dot;
eta_dn_dot = N2_dn_dot;
```

WRAP_MPI2PI

```
function [angleOUT] = wrap_mpi2pi(angleIN)
for ii=1:length(angleIN),
    if angleIN(ii)>pi,
        angleOUT(ii) = angleIN(ii) - 2*pi;
    elseif angleIN(ii)<-pi,
        angleOUT(ii) = angleIN(ii) + 2*pi;
    else
        angleOUT(ii) = angleIN(ii);
    end
end
```

WRAP_NEG

```
function [angleOUT] = wrap_neg(angleIN)
angleOUT=angleIN;
for ii=1:length(angleIN),
    while angleOUT(ii)>0,
        angleOUT(ii) = angleOUT(ii) - 2*pi;
    end
end
```

WRAP_POS

```
function [angleOUT] = wrap_pos(angleIN)
angleOUT=angleIN;
for ii=1:length(angleIN),
    while angleOUT(ii)<0,
        angleOUT(ii) = angleOUT(ii) + 2*pi;
    end
end
```

CC_PLOT

```matlab
function cc_plot(Ni,r,x_O,y_O,z_O,V_O,alpha,gamma,x_F,y_F,z_F,V_F,...

beta,chi,R,Rv,VO,violation,x_ccv,y_ccv,z_ccv,alpha_up_p,alpha_dn_p,...
    alpha_up_pv,alpha_dn_pv,sensvol,acount,acountv)

persistent counter
global Ni_loop
if isempty(counter),
    figure(1)
    clf
    counter=1;
end
if counter<=Ni_loop,
    figure(1)
    hold on
    counter=counter+1;
else
    figure(1)
    clf
    hold on
    counter=2;
end

%Calculate velocity components
Vx_O = V_O*cos(alpha)*cos(gamma);
Vy_O = V_O*sin(alpha)*cos(gamma);
Vz_O = V_O*sin(gamma);
Vx_F = V_F.*cos(beta).*cos(chi);
Vy_F = V_F.*sin(beta).*cos(chi);
Vz_F = V_F.*sin(chi);

anglecircle = linspace(0,2*pi,200);

xcircle =
x_F*ones(1,length(anglecircle))+R*ones(Ni,1)*cos(anglecircle);
ycircle =
y_F*ones(1,length(anglecircle))+R*ones(Ni,1)*sin(anglecircle);
zcircle = z_F*ones(1,length(anglecircle));
xcirclev =
(x_ccv+x_O)*ones(1,length(anglecircle))+Rv.*ones(Ni,1).*cos(alpha)*cos(
anglecircle);
ycirclev =
(y_ccv+y_O)*ones(1,length(anglecircle))+Rv.*ones(Ni,1).*sin(alpha)*cos(
anglecircle);
zcirclev =
(z_ccv+z_O)*ones(1,length(anglecircle))+Rv.*ones(Ni,1)*sin(anglecircle)
;

llcount=1;
llcountv=1;
% figure(1)
% clf
```

```matlab
% hold on
view(-30,30)
%view(0,90)
%view(0,0)
for kk = 1:Ni,
    if (~isempty(alpha_up_p) || ~isempty(alpha_up_pv)) && violation ==
0,
        plot3([x_O x_F(kk)],[y_O y_F(kk)],[z_O z_F(kk)],'b-*')
        quiver3([x_O x_F(kk)],[y_O y_F(kk)],[z_O z_F(kk)],[Vx_O
Vx_F(kk)],[Vy_O Vy_F(kk)],[Vz_O Vz_F(kk)],'AutoScale','off')
        if sensvol(kk) == 1;

hdl1=plot3(xcircle(kk,:),ycircle(kk,:),zcircle(kk,:)+VO,'k');set(hdl1,'
Color',[212 208 200]/255)
            hdl2=plot3(xcircle(kk,:),ycircle(kk,:),zcircle(kk,:)-
VO,'k');set(hdl2,'Color',[212 208 200]/255)

hdl3=plot3(xcirclev(kk,:),ycirclev(kk,:),zcirclev(kk,:),'k');set(hdl3,'
Color',[212 208 200]/255)
        else
            plot3(xcircle(kk,:),ycircle(kk,:),zcircle(kk,:)+VO,'k')
            plot3(xcircle(kk,:),ycircle(kk,:),zcircle(kk,:)-VO,'k')
            plot3(xcirclev(kk,:),ycirclev(kk,:),zcirclev(kk,:),'k')
        end
        plot3(x_ccv+x_O,y_ccv+y_O,z_ccv+z_O,'c*')
        for ll=llcount:acount(kk)+llcount-1,
            plot3([x_O r(kk)*cos(alpha_dn_p(ll))+x_O],[y_O
r(kk)*sin(alpha_dn_p(ll))+y_O],[z_O z_O],'r')
            plot3([x_O r(kk)*cos(alpha_up_p(ll))+x_O],[y_O
r(kk)*sin(alpha_up_p(ll))+y_O],[z_O z_O],'r')

quiver3(r(kk)/2.*cos(alpha_up_p(ll))+x_O,r(kk)/2.*sin(alpha_up_p(ll))+y
_O,z_O,sin(alpha_up_p(ll)),-
cos(alpha_up_p(ll)),0,'g','AutoScale','off')

quiver3(r(kk)/2.*cos(alpha_dn_p(ll))+x_O,r(kk)/2.*sin(alpha_dn_p(ll))+y
_O,z_O,-
sin(alpha_dn_p(ll)),cos(alpha_dn_p(ll)),0,'g','AutoScale','off')
            llcount=llcount+1;
        end
        for ll=llcountv:acountv(kk)+llcountv-1,
            plot3([x_O r(kk)*cos(alpha_dn_pv(ll))*cos(alpha)+x_O],[y_O
r(kk)*cos(alpha_dn_pv(ll))*sin(alpha)+y_O],[z_O
r(kk)*sin(alpha_dn_pv(ll))+z_O],'r')
            plot3([x_O r(kk)*cos(alpha_up_pv(ll))*cos(alpha)+x_O],[y_O
r(kk)*cos(alpha_up_pv(ll))*sin(alpha)+y_O],[z_O
r(kk)*sin(alpha_up_pv(ll))+z_O],'r')

quiver3(r(kk)/2.*cos(alpha_up_pv(ll))*cos(alpha)+x_O,r(kk)/2.*cos(alpha
_up_pv(ll))*sin(alpha)+y_O,r(kk)/2.*sin(alpha_up_pv(ll))+z_O,sin(alpha_
up_pv(ll))*cos(alpha),sin(alpha_up_pv(ll))*sin(alpha),-
cos(alpha_up_pv(ll)),'g','AutoScale','off')

quiver3(r(kk)/2.*cos(alpha_dn_pv(ll))*cos(alpha)+x_O,r(kk)/2.*cos(alpha
_dn_pv(ll))*sin(alpha)+y_O,r(kk)/2.*sin(alpha_dn_pv(ll))+z_O,-
```

```matlab
sin(alpha_dn_pv(ll))*cos(alpha),-
sin(alpha_dn_pv(ll))*sin(alpha),cos(alpha_dn_pv(ll)),'g','AutoScale','o
ff')
                llcountv=llcountv+1;
            end

    elseif violation == 1,
        plot3([x_O x_F(kk)],[y_O y_F(kk)],[z_O z_F(kk)],'r-*')
        quiver3([x_O x_F(kk)],[y_O y_F(kk)],[z_O z_F(kk)],[Vx_O
Vx_F(kk)],[Vy_O Vy_F(kk)],[Vz_O Vz_F(kk)],'AutoScale','off')
        plot3(xcircle(kk,:),ycircle(kk,:),zcircle(kk,:)+VO,'r')
        plot3(xcircle(kk,:),ycircle(kk,:),zcircle(kk,:)-VO,'r')
        plot3(xcirclev(kk,:),ycirclev(kk,:),zcirclev(kk,:),'r')
        plot3(x_ccv+x_O,y_ccv+y_O,z_ccv+z_O,'c*')

    else
        plot3([x_O x_F(kk)],[y_O y_F(kk)],[z_O z_F(kk)],'b-*')
        quiver3([x_O x_F(kk)],[y_O y_F(kk)],[z_O z_F(kk)],[Vx_O
Vx_F(kk)],[Vy_O Vy_F(kk)],[Vz_O Vz_F(kk)],'AutoScale','off')
        if sensvol(kk) == 1;

hdl1=plot3(xcircle(kk,:),ycircle(kk,:),zcircle(kk,:)+VO,'k');set(hdl1,'
Color',[212 208 200]/255)
            hdl2=plot3(xcircle(kk,:),ycircle(kk,:),zcircle(kk,:)-
VO,'k');set(hdl2,'Color',[212 208 200]/255)

hdl3=plot3(xcirclev(kk,:),ycirclev(kk,:),zcirclev(kk,:),'k');set(hdl3,'
Color',[212 208 200]/255)
        else
            plot3(xcircle(kk,:),ycircle(kk,:),zcircle(kk,:)+VO,'k')
            plot3(xcircle(kk,:),ycircle(kk,:),zcircle(kk,:)-VO,'k')
            plot3(xcirclev(kk,:),ycirclev(kk,:),zcirclev(kk,:),'k')
        end
        plot3(x_ccv+x_O,y_ccv+y_O,z_ccv+z_O,'c*')

    end
end
hold off
plotlim=9;
%axis([x_O-plotlim x_O+plotlim y_O-plotlim*8/10 y_O+plotlim*8/10 z_O-
plotlim/3 z_O+plotlim/3])
axis equal
```

Bibliography

[1] Department of Defense, "Unmanned Systems Roadmap 2007-2032," 2007.

[2] James C Neidhoefer, Christopher S Gibson, Maithilee Kunda, and Eric N Johnson, "Deteminism in Autonomy for Applications in the National Airspace System (NAS)," *AIAA Infotech@Aerospace*, Rohnert Park, CA, 2007.

[3] Derek Kingston, Randal Beard, and Ryan Holt, "Decentralized Perimeter Surveillance Using a Team of UAVs," *Transactions on Robotics*, vol. 24, no. 6, December 2008.

[4] James K Kuchar and Lee C Yang, "Survey of Conflict Detection and Resolution Modeling Methods," *AIAA Guidance, Navigation, and Control Conference*, New Orleans, LA, 1997.

[5] James K Kuchar and Lee C Yang, "A Review of Conflict Detection and Resolution Modeling Methods," *Transactions on Intelligent Transportation Systems*, vol. 1, no. 4, December 2000.

[6] Gilles Dowek and Cesar Munoz, "Conflict Detection and Resolution for 1,2.N Aircraft," *7th Aviation Technology, Integration, and Operations Conference*, Belfast, Northern Ireland, 2007.

[7] Animesh Chakravarthy and Debasish Ghose, "Obstacle Avoidance in a Dynamic Environment: A Collision Cone Approach," *Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 28, no. 5, September 1998.

[8] Karl D Bilimoria, "A Geometric Optimization Approach to Aircraft Conflict Resolution," *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Denver, CO, 2000.

[9] Jennifer Goss, Rahul Rajvanshi, and Kamesh Subbarao, "Aircraft Conflict Detection and Resolution using Mixed Geometric and Collision Cone Approaches," *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Providence, RI, 2004.

[10] C Carbone, U Ciniglio, F Carraro, and S Luongo, "A Novel 3D Geometric Algorithm for Aircraft Autonomous Collision Avoidance," *Proceedings of the 45th Conference on Decision & Control*, San Diego, CA, 2006.

[11] Karin Sigurd and Jonathan How, "UAV Trajectory Design Using Total Field Collision Avoidance," *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Austin, TX, 2003.

[12] Amirreza Rahmani, Kunihiko Kosuge, Takashi Tsukamaki, and Mehran Mesbahi, "Multiple UAV Deconfliction via Navigation Functions," *AIAA Guidance, Navigation and Control Conference and Exhibit*, Honolulu, HI, 2008.

[13] Maria Prandini, Jianghai Hu, John Lygeros, and Shankar Sastry, "A Probablistic Approach to Aircraft Conflict Detection," *Transactions on Intelligent Transportation Systems*, vol. 1, no. 4, December 2000.

[14] Maria Prandini, John Lygeros, Arnab Nilim, and Shankar Sastry, "Randomized Algorithms for Probablistic Aircraft Conflict Detection," *Proceedings of the 38th Conference on Decision & Control*, Phoenix, AZ, 1999.

[15] Brad D Kelly and Solomon D Picciotto, "Probability Based Optimal Collision Avoidance Maneuvers," *Space 2005*, Long Beach, CA, 2005.

[16] Kwang-Yeon Kim, Jung-Woo Park, and Min-Jea Tahk, "UAV Collision Avoidance Using Probablistic Method in 3-D," *International Conference on Control, Automation, and Systems*, Seoul, Korea, 2007.

[17] Chang-Su Park, Min-Jea Tahk, and Hyochoong Bang, "Multiple Aerial Vehicle Formation Using Swarm Intelligence," *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Austin, TX, 2003.

[18] Craig W Reynolds, "Flocks, Herds, and Schools: A Distributed Behavioral Model," *Computer Graphics*, vol. 21, no. 4, July 1987.

[19] Craig W Reynolds, "Steering Behaviors for Autonomous Characters," *Proceedings of Game Developers Conference*, San Jose, CA, 1999, pp. 763-782.

[20] Yu Gu, Girish K Sagoo, Brad Seanor, Giampiero Campa, and Marcello R Napolitano, "Curvature-Velocity-Orientation Method for UAV Collision Avoidance," *AIAA Guidance, Navigation and Control Conference and Exhibit*, Honolulu, HI, 2008.

[21] Todd Farley and Heinz Herzbanger, "Fast-Time Simulation Evaluation of a Conflcit Resolution Algorithm Under High Air Traffic Demand," *Proceedings of the 7th USA/Europe Air Traffic Management R&D Seminar*, Barcelona, Spain, 2007.

[22] Russell A Paielli, "Tactical Conflict Resolution Using Vertical Maneuvers in En Route Airspace," *Journal of Aircraft*, vol. 45, no. 6, 2008.

[23] Yoko Watanabe, Anthony J Calise, Eric N Johnson, and Johnny H Evers, "Minimum-Effort Guidance for Vision-Based Collision Avoidance," *AIAA Atmospheric Flight Mechanics Conference and Exhibit*, Keystone, CO, 2006.

[24] Yoko Watanabe, Anthony J Calise, and Eric N Johnson, "Vision-Based Obstacle Avoidance for UAVs," *AIAA Guidance, Navigation and Control Conference and Exhibit*, Hilton Head, SC, 2007.

[25] Giancarmine Fasano et al., "Multisensor based Fully Autonomous Non-Cooperative Collision Avoidance System for UAVs," *AIAA Infotech@Aerospace*, Rohnert Park, CA, 2007.

[26] Eric Portilla, Alex Fung, Won-Zon Chen, Omid Shakernia, and Thomas Molnar, "Sense and Avoid (SAA) & Traffic Alert and Collision Avoidance System (TCAS) Integration for Unmanned Aerial Systems (UAS)," *AIAA Infotech@Aerospace*, Rohnert Park, CA, 2007.

[27] Austin L Smith, Dennis M Coulter, and Christopher S Jones, "UAS Collision Encounter Modeling and Avoidance Algorithm Development for Simulating Collision Avoidance," *AIAA Modeling and Simulation Technologies Conference and Exhibit*, Honolulu, HI, 2008.

[28] Su-Cheol Han and Hyochoong Bang, "Proportional Navigation-Based Optimal Collision Avoidance for UAVs," *2nd International Conference on Autonomous Robots and Agents*, Palmerston North, New Zealand, 2004.

192

[29] Su-Cheol Han, "Proportional Navigation-Based Optimal Collision Avoidance for UAVs," Korea Advanced Institute of Science and Technology, Thesis MAE 20033668, 2005.

[30] U. S. Shukla and P. R. Mahapatra, "The Proportional Navigation Dilemma - Pure or True?," *Transactions on Aerospace and Electronics Systems*, vol. 26, no. 2, March 1990.

[31] James B Engle, "Fiscal Year 2006 Air Force Science and Technology," Office of the Deputy Assistant Secretary of the Air Force for Science, Technology, and Engineering, Presentation 2005.

[32] Department of Defense, "Unmanned Systems Roadmap 2005-2030," 2005.

[33] Defense Update. [Online]. http://www.defense-update.com/products/n/nighthawk.htm

[34] Procerus Technologies, Kestrel User Guide, 2008.

[35] Federal Aviation Administration, "Unmanned Aircraft Systems Operation in the U.S. National Airspace System," Unmanned Aircraft Program Office AIR-160, Interim Operational Approval Guidance 08-01 2008.

[36] Rep James L. Oberstar, "FAA Reauthorization Act of 2009," HOUSE OF REPRESENTATIVES, Bill H.R. 915, 2009.

[37] Troy H Vantrease, "Development and Employment of a Semi-Autonomous Cursor on Target Navigation System for Micro Air Vehicles," Air Force Institute of Technology, Wright-Patterson AFB, Thesis AFIT/GAE/ENY/08-J06, 2008.

Vita

Austin L. Smith was born in Lafayette, IN. He attended Purdue University and graduated With Distinction in May of 2005 with a Bachelor of Science Degree in Aeronautical and Astronautical Engineering. Austin started his career as an engineer in the Air Force Research Laboratory's Air Vehicles Directorate. He is currently an engineer with Modern Technology Solutions, Inc., and is involved in UAS airspace access efforts and sense and avoid developments.

1. REPORT DATE *(DD-MM-YYYY)* 26-03-2009	2. REPORT TYPE Master's Thesis	3. DATES COVERED *(From – To)* March 2008 – March 2009
4. TITLE AND SUBTITLE UAS Collision Avoidance Algorithm That Minimizes The Impact On Route Surveillance		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Smith, Austin L.		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 Hobson Way, Building 640 WPAFB OH 45433-8865		8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/GAE/ENY/09-M18
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory, Sensors Directorate, Dr. Alok Das, 2241 Avionics Circle, WPAFB OH 45433		10. SPONSOR/MONITOR'S ACRONYM(S) AFRL/RY
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.		
13. SUPPLEMENTARY NOTES		

14. ABSTRACT

A collision avoidance algorithm is developed and implemented that is applicable to different types of unmanned aerial systems ranging from a single platform with the ability to perform all collision avoidance functions independently to multiple vehicles performing functions as a cooperative group with collision avoidance commands computed at a ground station. The collision avoidance system is exercised and tested using operational hardware and platforms and is demonstrated in representative missions similar to those planned for operational systems. The results presented are the first known flight tests of a global, three-dimensional, geometric collision avoidance system on an unmanned aircraft system. Novel developments using an aggregated collision cone approach allows each unmanned aircraft to detect and avoid collisions with two or more other aircraft simultaneously. The collision avoidance system is implemented using a miniature unmanned aircraft with an onboard autopilot. Various test cases are used to demonstrate the algorithms robustness to different collision encounters. Two-ship encounters at various engagement angles are flight tested. The flight test results are compared with ideal, software-in-the-loop, and hardware-in-the-loop tests.

15. SUBJECT TERMS

collision avoidance, unmanned aircraft, conflict detection and resolution, autonomous guidance

16. SECURITY CLASSIFICATION OF: Unclassified			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON Frederick G. Harmon, Lt Col, USAF
a. REPORT U	b. ABSTRACT U	c. THIS PAGE U	UU	213	19b. TELEPHONE NUMBER *(Include area code)* (937) 255-3636, ext 7478 (Frederick.Harmon@afit.edu)