Surveillance Using Multiple Unmanned Aerial Vehicles

THESIS

Christopher E. Booth, Captain, USAF

AFIT/GSS/ENY/09-M02

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the United States Government.

AFIT/GSS/ENY/09-M02

Surveillance Using Multiple Unmanned Aerial Vehicles

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 (Space Systems)

Christopher E. Booth, BSME, MSAE

Captain, USAF

March 2009

AFIT/GSS/ENY/09-M02

Surveillance Using Multiple Unmanned Aerial Vehicles

Christopher E. Booth, BSME, MSAE
Captain, USAF

Approved:

// signed //	13 Mar 2009
Dr. Richard G. Cobb (Chairman)	Date
// signed //	13 Mar 2009
Dr. David R. Jacques (Member)	Date
// signed //	13 Mar 2009
Lt Col Fred Harmon (Member)	Date

Abstract

This study examines the performance and limitations of a heuristic cooperative control (CC) surveillance algorithm for multiple unmanned aerial vehicles (UAVs) under both simulation and demonstration. The algorithm generates Dubin's based paths and provides velocity feedback to accomplish simultaneous arrival onto a surveillance orbit around the target and maintains position while orbiting. The performance was tested under multiple wind conditions in simulation and actual winds during flight testing. Both position accuracy and target visibility were examined.

The analysis covers three major topics: development of a closed loop model for a new airframe at AFIT for simulation purposes, development of the CC algorithm that interfaces with Procerus Technologies' Kestrel Autopilot, and achievable system performance analysis. The model assumes first-order responses to roll, pitch, and airspeed commands using time constants pulled from actual flight test data. The CC algorithm has two modes: one that generates commands to multiple UAVs for simultaneous arrival to a surveillance orbit, and one that maintains equal angular spacing about the orbit. In addition to positional performance metrics, percentage of target in-view time was also measured based on the UAV's side camera field of view (FOV). Simulation tested both modes under wind conditions of 0%, 10%, 25%, and 50% of the nominal airspeed (V_{nom}).

Results showed that the algorithm maintained UAV position with winds 25% of V_{nom}, but instabilities appeared at 50% where large overshoots appeared on the downwind side of the orbit. Target visibility was most impacted by crosstrack errors that steadily grew with increasing winds. Roll of the UAV showed the greatest impact on the FOV due to its coupling effect with crosstrack error. Overall target in-view time also improved with increasing numbers of UAVs for all wind conditions.

Acknowledgements

First and foremost, I would like to thank my thesis advisor Dr Cobb for his patience and great suggestions during the steep learning curve associated with this research. I owe many thanks to Major Adam Rutherford, Major Jonathan Taylor, and LCDR Ted Diamond for spending many hours learning and troubleshooting flight test hardware, preparing for review boards, and establishing rock solid test discipline. Both Don Smith and Jon McNeese, our resident experts, provided the tricks of the trade to solve many a problem. Capt Shannon Farrell provided a thorough second look at many of the routines and helped tremendously with keeping this complicated endeavor correct. Lt Jared Yeates saved me many hours figuring out the software/hardware interface. Thank you all.

Finally, I owe my greatest thanks to my family for their patience, love, and support to pull this off.

Christopher E. Booth

Table of Contents

Page

Page

List of Figures

List of Tables

List of Symbols

Symbol		Page
La	Latitude	41
Lo	Longitude	41
V_g	Ground Velocity	44
Ph	Phase Lag	56
AE	Angular Error	69
t_s	Settling Time	69
B	"The Box" Angle	94
$\dot{\omega}$	Angular Acceleration	116
T	Tension	116
m	Mass	116
g	Gravitational Acceleration	116
I	Moment of Inertia	116
α	Angle of Attack	117
δ_e	Elevon Deflection	119
C_l	Moment Coefficient-Roll	119
C_m	Moment Coefficient-Pitch	119
C_n	Moment Coefficient-Yaw	119
C_T	Thrust Coefficient	120
Ja	Advance Ratio	120
V_a	Airspeed	120
η	Propeller Rotation Speed (rev/s)	120
d	Diameter	120
L	Lift	120
D	Drag	120
Y	Side Force	120
$\bar{L}$	Moment about Roll Axis	120
M	Moment about Pitch Axis	120
N	Moment about Yaw Axis	120

List of Abbreviations

Abbreviation		Page
UAV	Unmanned Aerial Vehicle	1
AFIT	Air Force Institute of Technology	1
AFRL	Air Force Research Lab	2
BATCAM	Battlefield Air Targeting Camera Autonomous Micro UAV	2
FOV	Field of View	6
MAV	Micro Air Vehicle	7
USAF	United States Air Force	7
DOF	Degree of Freedom	7
HITL	Hardware in the Loop	7
ARA	Applied Research Associates	21
GPS	Global Positioning System	22
VC	Virtual Cockpit	23
RC	Radio Controlled	25
FAA	Federal Aviation Administration	25
TRB	Technical Review Board	25
SRB	Safety Review Board	25
AoA	Angle of Attack	28
ODE	Ordinary Differential Equation	29
NED	North/East/Down	35
DCM	Directional Cosine Matrix	35
CW	Clockwise	39
CCW	Counter Clockwise	39
WGS-84	World Geodetic System 1984	41
WP	Waypoint	45
CC	Cooperative Control	50
GUI	Graphical User Interface	50

Abbreviation		Page
TM	Telemetry	54
BC	BATCAM	89
AoA	Angle of Attack	117

Surveillance Using
Multiple Unmanned Aerial Vehicles

I. Introduction

1.1 Background

The Air Force is utilizing Unmanned Aerial Vehicles (UAVs) at an ever increasing pace. Small autonomous vehicles have sparked great interest in the military by providing an inexpensive system that increases capabilities and prevents placing personnel in dangerous situations. Autonomous platforms that fly have a unique appeal. They can traverse large distances quickly and provide a "bird's eye view" of the battlespace. Utilizing multiple vehicles enhances mission accomplishment with redundancy, robustness, and increased coverage when compared to a single platform. This research explores operating multiple UAVs for surveillance.

Multi-UAV surveillance holds many advantages over the other surveillance options in terms of proximity (close or far), speed (fast or slow), responsiveness, cost, and overall personnel risk. Manned surveillance is close and responsive but slow and places personnel at risk, traditional aircraft are fast and reasonably close but are expensive and also place personnel at risk. Space surveillance allows access to denied areas but is very expensive, limited by the orbit for timing and placement, and far from the target. Multi-UAV surveillance can reduce or remove personnel risk, be close to the target, provide persistence over the target, and can cost very little compared to manned aircraft and space options. Advances in the miniaturization of electronics aided this interest in UAVs. As cost and size decreased, capability increased for surveillance and autonomous technology. Consequently, research and development blossomed in both academia and the aerospace industry.

The Air Force Institute of Technology (AFIT) has conducted many UAV research projects, and vigorously continues to this day. The research conducted herein

continues an ongoing project at AFIT that focuses on utilizing UAVs for surveillance and target engagement missions. To fill the void in data for small aircraft, AFIT's research began in 2006 with the work of Nidal Jodeh [8] modelling a 9.16 foot wingspan radio controlled model airplane, the Sig Rascal, retrofitted with an autopilot. The stability and payload capacity of this airframe made it ideal for UAV research. Since the Sig Rascal became the primary demonstration aircraft, Jodeh's model became the base for many following projects. One UAV application focused on tracking and engaging a moving target with on-board video. At AFIT, this application became known as the "Fleeting Target Program." The problem was broken up into creating a path to the target in real time (Pathmaker) [19], flying the vehicle to the target using video feedback (Cursor On Target) [20], and integrating the hardware and software into a usable package (Fleeting Target Controller) [17]. The research described herein is the next iteration of the Fleeting Target Program.

For the current effort, the emphasis of research shifted away from target detection and engagement when Air Force Research Lab (AFRL) received an urgent need request from the warfighter to develop a route surveillance capability. AFRL was tasked to deliver a prototype system capable of monitoring many miles of road and revisiting any point at fixed intervals [2]. The proposed system consists of multiple UAVs with day and night sensors, a ground station with semi-autonomous control of the UAVs, and an anomaly detection system. The primary purpose was to surveil routes ahead of convoys to minimize risk to transportation operations. This need became the primary drive for this research.

To aid this research, AFRL provided AFIT with six new airframes, the Battlefield Air Targeting Camera Autonomous Micro UAVs (or BATCAMs for short). This airframe differed greatly from Jodeh's airframe: The BATCAM wingspan was 21" vs Jodeh's 9.1 ft, the BATCAM's propulsion was electric vs. Jodeh's gas engine, cameras were body fixed vs. gimballed, and control surfaces were a V-tail with no ailerons vs. the traditional aileron/rudder/elevator configuration. The old Sig Rascal

model no longer applied. By changing the demonstration aircraft, research using the BATCAM dictated development of a new model.

The scope of this work develops and assesses two main items: a model for the BATCAM, and an algorithm that controls multiple BATCAMs for surveillance.

1.2 Problem Statement

This research focuses on quantifying the abilities and limitations of a group of UAVs to monitor a fixed target. The primary focus is not target detection, but the control algorithm for the group that accomplishes persistent visual contact of the target. Quantifying the abilities and limitations of the UAV group are done from two perspectives: simulation, and flight demonstration.

The simulation portion requires construction of a model for the BATCAM, then controlling this model with an algorithm to accomplish surveillance. Using flight test data, this research constructs a representative closed-loop model for the BATCAM/autopilot system. The intent of this model is to capture the major handling characteristics of the system, then utilize an algorithm that focuses on UAV placement with respect to one another in-flight to surveil a target.

The flight demonstration portion utilizes only the algorithm to command the UAV, replacing the model with an actual BATCAM and autopilot. The algorithm will analyze the current state of multiple UAVs and create a command set to place all UAVs for surveillance. The algorithm accomplishes two distinct tasks: to place all UAVs into the surveillance orbit, and to maintain the surveillance orbit.

The goal of this research is to command multiple UAVs in real time to approach and maintain an orbit about a fixed target. The effects of wind on performance as well as the ability of the UAV group to reconfigure when individual UAVs are added/deleted will be determined.

1.3 Assumptions and Limitations

To keep a concise focus, certain bounds and simplifications must be made. Research of this nature is highly complex, and exhaustive treatment of a subject can quickly exceed the time and resources available. The limitations/bounds will apply to both simulation and demonstration, but the assumptions may only apply to the model/simulation. If one of the items applies to simulation or demonstration only, the list will specify the applicability.

The research bounds are:

- The "changing" conditions will be limited to wind scenarios, and adding and removing a UAV under wind.
- The maximum number of UAVs will be four.
- All UAVs will be the same.

The following lists the overall assumptions. Chapter III will elaborate on the reasoning behind these items.

- Each UAV closed-loop model will behave like a first order system (for simulation portion only).
- Wind vectors will be only in the horizontal plane (simulation only).
- The ground will be assumed planar and flat for sensor footprint projections.
- Communication is available to all UAVs at all times (simulation only).
- The flat earth model will be assumed "inertial" ignoring the rotation and curvature of the earth.

The intent of these assumptions is to make simplifications that are reasonably accurate to actuality but have the net benefit of decreasing complexity. The real time aspect of the algorithm relies on accurate yet timely information flow to and from UAVs. The objective of this research is to create a heuristic real-time algorithm that is robust

enough to handle environmental effects. Comparisons to truly optimal solutions will be accomplished in future research.

1.4 Preview

Chapter II presents past research from two areas: background research that this thesis builds upon, and related research that illustrates the different ways to control and optimize multiple UAVs under different scenarios. Chapter III presents the equipment used for flight demonstration, the modelling approach for simulation, and the heuristic algorithm. Chapter IV presents the performance results of the heuristic algorithm under simulation and flight demonstration. Chapter V concludes this research and makes recommendations for future research.

II. Literature Review

2.1 Overview

This survey draws from two areas: background research, and related research. The background research pulls together material that aids this thesis. Whereas the related research explores how various researchers approached the multi-UAV cooperative control problem. With a very different airframe from past Fleeting Target research, the first portion of background research aids in developing a new model. Jodeh [8] creates a framework for modelling small vehicles, then Kingston [9] from AFRL uses a simple closed-loop model for the route surveillance. The next area of background research is path planning. From the knowledge base here at AFIT, the works of Zollars and Terning illustrate two differing approaches: one optimal and one heuristic. Rysdyk [16] from the University of Washington also developed a nice closed form solution for keeping a target in the field of view (FOV) while orbiting. The related research section surveys some applications of multiple UAVs used cooperatively. Secondly, Beard [4] uses the broad area search technique to compare optimal verse sub-optimal solutions. Next, MIT's Richards and Bellingham [15] take the scenario of completing X tasks using UAVs, and increase the complexity to include differing UAV capabilities, time constraints between steps, and no fly zones. Ending this section is a brief overview of a powerful simulation tool called MultiUAV2. This product creates a realistic environment to test algorithms and highlights not only optimal path planning, but also key considerations for any fielded system like probability of detection and inter-UAV communication.

2.2 Background Research

Creating a good mathematical description of the BATCAM airframe is not a minor feat. Both the approach and the aerodynamic properties must be thorough for generating a representative model. The combined works of Jodeh [8] and Kingston [9] provide a solid base to build from.

2.2.1 Aircraft Modelling-Jodeh. Jodeh's [8] research created one of the first models of a micro air vehicle (MAV) at AFIT. He found that extensive research existed for modelling large airframes, but high fidelity small vehicle models seemed rare. Not surprisingly, he also found no research on UAV handling/stability characteristics or standards for handling/stability, both important to accomplish quality surveillance missions. The need for a modelling approach for small vehicles seemed apparent. The Sig Rascal airframe by Tower Hobbies was modelled. This is a stable airframe with the ability to handle small to medium payloads. A Piccolo autopilot gave the airframe autonomy.

This modelling approach did not use the wind tunnel, but used the United States Air Force (USAF) Stability and Control Digital Datcom software. This computer program was written for the Air Force under contract by McDonnell Douglas Astronautics Company. This program has the ability to output Lift, Drag, Moment, and Stability Derivatives (and many other items) when given desired flight conditions, attitudes, and physical geometry. With this output, Jodeh created a 6 Degree of Freedom (DOF) non-linear model in Matlab/Simulink. The model was compared to both flight testing and Hardware-in-the-loop (HITL) simulations.

To create a good comparison between the three sets of data, he used an elevator command which induced both a phugoid and short period motion. Comparing the flight test data to the 6 DOF model, the actual (flight test) oscillation had a period of 13s where the model predicted 10s. In the phugoid excitation, the model matched in period, but the amplitude differed: flight test values exceeded the model in both pitch rate and airspeed, but model amplitude values exceeded test values in altitude oscillations. A comparison of Hardware in the loop (HITL) to test showed larger frequencies and less damping for HITL. All difference were within a factor of two, but model values and HITL values differed from test values up to 25% in period and 50% in amplitude. For these types of measurements, being within a factor of two is reasonable and indicates that the model is fairly close. Small errors in the parameters of a non-linear model can manifest like differences shown above. Since MAVs have

small masses and moments in inertia, errors in aerodynamic forces and moments make state errors more pronounced.

Jodeh's research lays a good foundation to follow for modelling small vehicles for this research. Not only does it provide a fairly complete list of items to model, but also illustrates that a model for a small vehicle can differ from reality within an order of magnitude from expected values.

2.2.2 UAV Closed Loop Model/Route Surveillance-Kingston. Members of AFRL Vehicles Directorate [9] presented both a simple UAV model and an algorithm that is inherently decentralized, convergent to optimal behavior in finite time, accounts for communication range limitations, and allows for changing perimeters.

Each UAV uses this simplified model derived from the 6 DOF model. The model assumes constant altitude and constant airspeed. The autopilot has also been tuned so that the closed-loop vehicle behaves like a first order system. The equations of motion become

$$\begin{aligned} \dot{p_N} &= V_a cos\psi + w_N \\ \dot{p_E} &= V_a sin\psi + w_E \\ \dot{\psi} &= \frac{g}{V_a} tan\phi \\ \dot{V_a} &= k_V(V_a^c - V_a) \\ \dot{\phi} &= k_\phi(\phi^c - \phi) \end{aligned} \tag{2.1}$$

where p_N and p_E are position; ψ, ϕ, V are yaw angle, roll angle, and airspeed; [w_N, w_E] is the wind vector; k 's are the first order parameters, and ^{c} denotes commanded quantities. This is a nice simple approach to capture the behavior of an airframe/autopilot closed-loop system. This equation will be modified to create the BATCAM model in Chapter III.

Kingston goes on to use this model to surveil a stretch of road. For N UAVs, the road is divided into N equal segments. With all vehicles on a single line, they travel back and forth along their assigned segment and meet at certain times. It is

possible for 2 (or more) vehicles to occupy the same position on the line. To handle communications, the analysis requires that UAVs physically meet along the line to exchange information. The base location is located at one end of the line (x=0), so when a UAV reaches the end, information reaches the base. As intuition would hold, by evenly spacing all vehicles, all information can reach any part of the line in minimum time.

Each UAV patrols the i^{th} segment back and forth. When the timing works perfectly, the two UAVs will briefly meet at the endpoints and exchange information. If the first UAV meets the second UAV early (or within its segment), the second UAV will turn around and be escorted back to the end of the segment, then turn around. If the second UAV is late, the first UAV will proceed into the other UAV's segment until they meet. At that time, the first UAV will turn around and be escorted back to the division point. As they contact, they exchange information, both surveillance info and 4 variables. These coordination variables are: P_R and P_L are the perimeter lengths right and left of the UAV; N_R and N_L are the number of UAVs right and left of the UAV. This information tells the UAV where it lies on the total perimeter P (= $P_R + P_L$) and what its segment is.

Let's take the worst case where the perimeter changes and the UAVs do not know how many total UAVs are patrolling the perimeter P, and initial UAV positions are random. If time T is P/V, then according to this research, all UAVs will reach the "low latency" configuration along P in less than 5T. It will take at most 3T for all UAVs to exchange information and get the correct perimeter and number of vehicles, then 2T more to reach the configuration where all UAVs are patrolling equal segments.

The algorithm was validated through flight testing using two UAVs with Kestrel Autopilots. The largest discrepancy between simulation and real data occurred in the turn around. The required U turn at the end of the segments took longer than predicted. The shared-border position of the two UAVs was approximately 60% of P,

compared to the predicted 50%. The wind that day was 35% of the airspeed of the UAVs.

Kingston's research utilizes a simple closed-loop model for a road surveillance algorithm. Comparing this model to flight test data, the simulation results compared fairly closely to flight test telemetry. This model is the base approach taken to create a model in this research. The control algorithm is one approach to decentralize the perimeter surveillance problem with enough robustness to handle both changing perimeter and number of UAVs. The major difference between Kingston's surveillance method to this one is the communication constraint. He accounts for the fact that not all UAVs are reachable at any given time, whereas this research assumes connectivity at all times to all vehicles. To extend the multi-UAV surveillance over large distances, this aspect must be addressed.

2.2.3 Path Planning-Zollars. Michael Zollars [21] analyzed optimal path planning for a single vehicle to a static point. The analysis assumed the aircraft is a point mass, aircraft airspeed is constant, and both wind heading and wind magnitude are constant. Important to note is that the wings were assumed level during turns, so the side sensor look angle did not vary. The wind varied from all directions and magnitudes were varied for windspeed/airspeed ratios up to 0.7 (wind is 70% of UAV's airspeed). Initial headings were constrained such that the UAV always took off into the wind, and the distance between the beginning UAV position and target was constant for all runs. Three problems were presented:

- Finding the optimal path minimizing flight time given an initial heading and position, and a final position and heading.
- Finding the optimal path minimizing flight time given through an urban canyon with the initial and final conditions from above.
- Finding the optimal path to both reach the target and orbit the target minimizing flight time and maximizing time in view while orbiting.

The first two problems yielded expected results. Flight times increased with increasing windspeed/airspeed ratios. Minimum flight time occurred with a tail wind (and vice versa). The interesting results came when the UAV orbited the target. If the sensor was fixed to the airframe and the sensor footprint was a point, the optimization scheme revealed it was physically impossible to track the target 100% of the time. When the footprint was modelled as a circle with diameter of 64% of the orbit radius, the target was in view for 82% of the orbit flight path.

This result highlights that even under constant wind conditions with many simplifying assumptions, orbiting under constant altitude has limits to keep the target in view. The windspeed/airspeed ratio was 0.2 for these results.

For multiple UAVs in wind, certain viewing directions may not be able to maintain the target in the field of view. To offset this another UAV must provide a different look angle to maintain persistence. For this research, the key result is that multiple UAVs orbiting a target has limitations in certain directions under windy conditions.

2.2.4 Path Planning-Terning. Building upon Zollars, Terning [19] created a heuristic approach to generate a path to engage a moving target in constant wind for real time use. This iterative approach uses the straight line distance to the target as an initial guess, then uses the calculated intercept time for the next iteration until the time for both projected paths (target and UAV) fall within a specified tolerance. This iteration was subject to both targets changing direction, and also latency in target information flow. The solution (called Pathmaker) is sub-optimal, but is fast enough for real time use.

The concept is quite simple and is illustrated in Figure 2.1. The first step finds the straight line distance to the moving target. The next step accounts for a maximum effort turn and finds a new distance. Using the time of travel for the new path, the projected target distance is recalculated (Step 3). The UAV refines the needed path in Step 4. Realize this assumes the UAV is a point mass, travelling at constant velocity

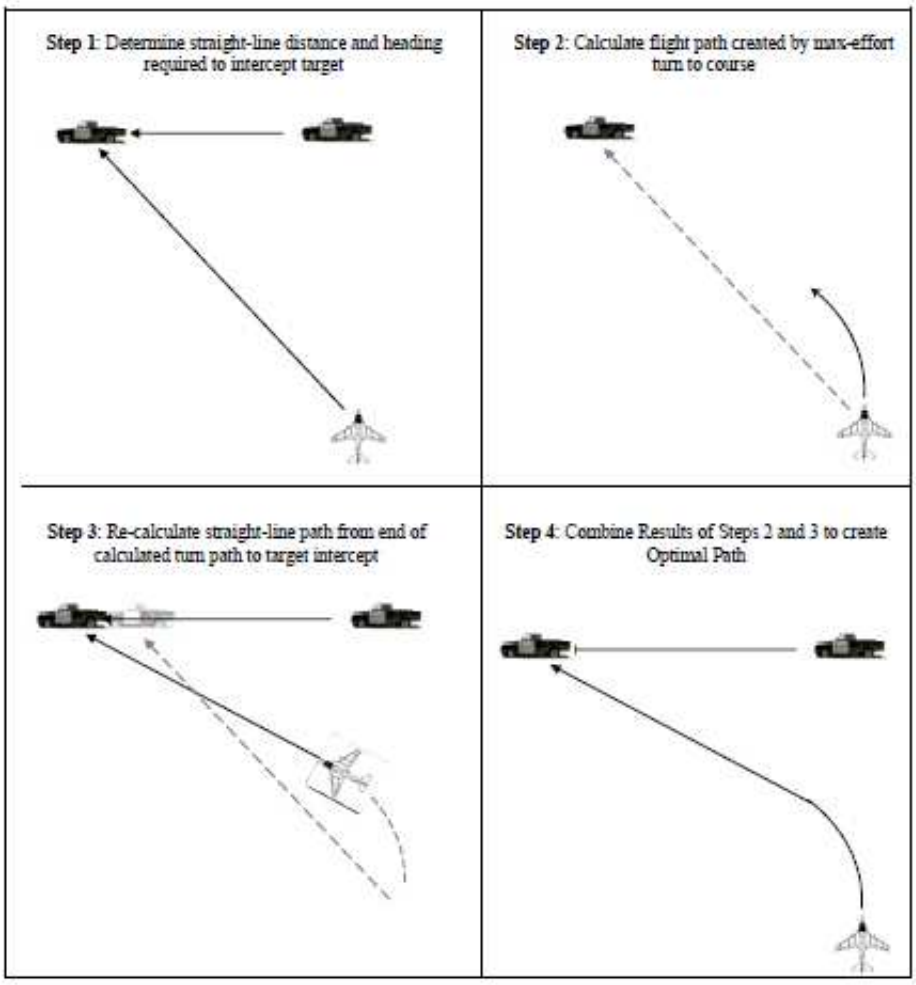

Figure 2.1: Pathmaker solution approach [19]

and altitude, with a constant wind vector and a constant target velocity along its heading.

Terning also showed this approach is rigorous enough to handle a target stopping and changing directions, as shown in Figure 2.2. The solid black line on the left is the path of the target, the blue line is the path of the UAV. The target moves north at 10 m/s, then east at 5 m/s, south at 20 m/s, and ending west at 15 m/s. Also in this simulation a 2 second lag is incorporated. As long as there is enough room spacially to track the target, the algorithm can accommodate changes in the target's path.

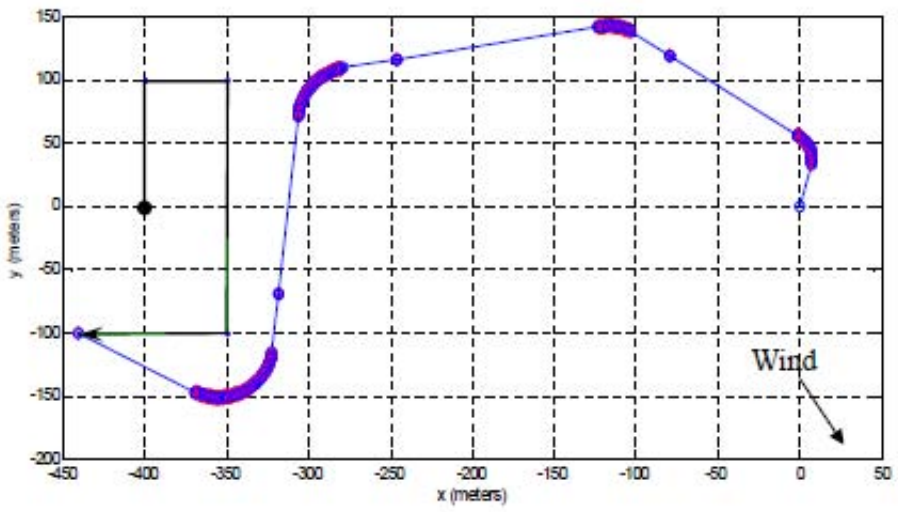

Figure 2.2: Pathmaker with Target Varying Trajectory [19]

Figure 2.3 shows there are instances where the algorithm has trouble intercepting the target. Again, the target path is black, the UAV path is blue. With the combination of both close proximity and a time lag, the algorithm can display oscillatory behavior and fail to get the target in the field of view of the sensor. This would be expected, especially with a UAV/target speed ratio greater than 2.

Unfortunately the winter weather of Ohio prevented Terning from rigorously flight testing this concept on actual hardware. He successfully integrated his algorithm to interface with Procerus Technology's Kestrel autopilot system (presented in Section 3.1.2), and showed in simulation that the concept worked. Although not truly optimal, the idea appears robust enough to handle communication delays (within limits) and quick enough to reach the changing target conditions.

This result shows that an iterative approach to changing conditions can be robust enough for real-time applications. This approach will be used in Chapter III when creating the cooperative control algorithm.

2.2.5 Path Planning-Rysdyk. The work of Rysdyk [16] presents a useful solution to keeping a target in sight of a sensor located on an air vehicle. The first part uses "helmsman behavior" control scheme to keep the vehicle on the desired

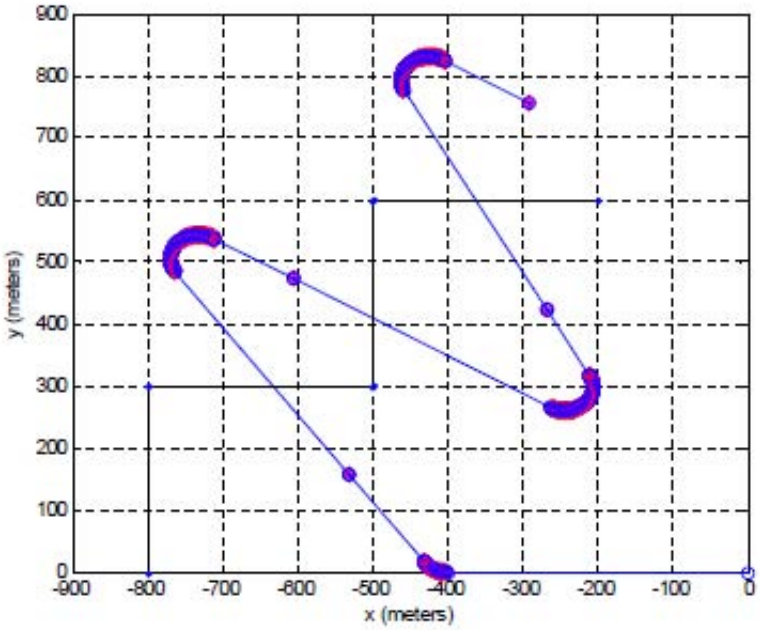

Figure 2.3: Pathmaker with Large Latency [19]

course. The second part develops an analytic solution to the correct orbit in the presence of wind to keep the target in view. "Helmsman behavior" determines an appropriate heading to get back on the desired path as function of the cross distance (y_s=perpendicular distance from the desired path) and the desired heading on the path (χ_s=the tangent of the desired path)(χ is the course/clock angle of the vehicle). The equation for the commanded heading becomes

$$\chi_c(y_s, \chi_s) = \sigma(y_s) + \chi_s \tag{2.2}$$

where the c subscript is the commanded heading and σ is a function that behaves like a spring (-kx) but saturates at the values $[-\tilde{\chi}_{icpt}, \tilde{\chi}_{icpt})]$ (χ_{icpt} denotes intercept heading, see Figure 2.4). The author chose the function

$$\sigma(y_s) = \tilde{\chi}_{icpt} \frac{e^{-ay_s/2} - 1}{e^{-ay_s/2} + 1} \tag{2.3}$$

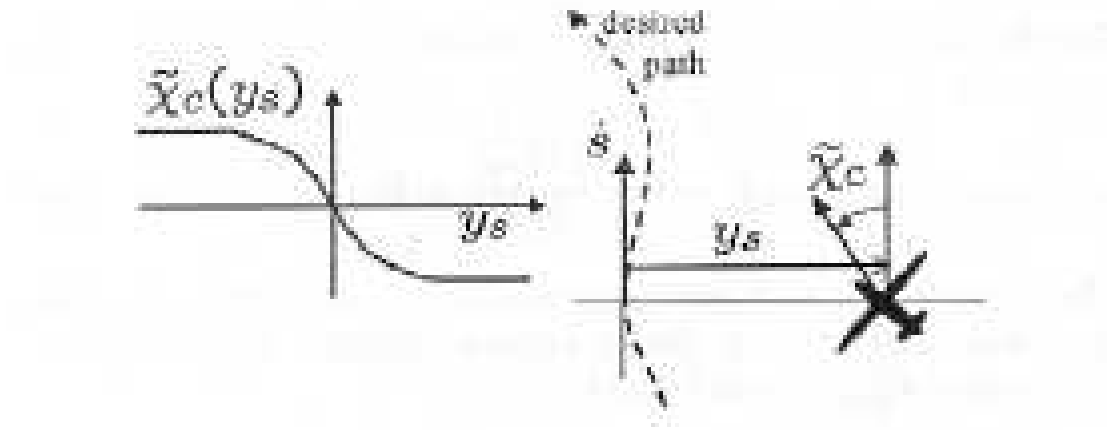

Figure 2.4: Cross Track Function [16]

where y_s is the crosstrack distance and tilde denotes an angle with respect to the desired heading (e.g. $\tilde{\chi_c} = \chi_c - \chi_s$). If you choose a PID control scheme, the closed-loop dynamics becomes

$$\dot{\tilde{\chi}} = \sigma_{y_s} V_g sin(\tilde{\chi}) + k_p(\tilde{\chi_c} - \tilde{\chi}) + k_i \int_0^t (\tilde{\chi_c} - \tilde{\chi}) d\tau \tag{2.4}$$

where V_g is the ground speed and

$$\sigma_{y_s} = \frac{d}{dy_s}\sigma(y_s) = -a\tilde{\chi}_{icpt}\frac{e^{-ay_s/2}}{(e^{-ay_s/2}+1)^2} \tag{2.5}$$

In a coordinated turn, bank angle is related to turn rate so

$$tan(\phi) = \frac{V_g}{g}\dot{\chi} \tag{2.6}$$

This can be used to provide the commanded turn rate using a set ground speed (V_g) and is a fairly simple approach to maintaining a desired path.

The above analysis assumes constant airspeed, altitude, and coordinated turns. The reason for coordinated turns means the rate of change of the heading is kinematically linked to the bank angle. The ideal solution makes the heading exactly

tangent to the desired path at the same moment the cross-distance becomes zero. This concept will be used to model closed-loop UAV behavior.

The concepts in both previous modelling research and the path planning background research will be utilized in Chapter III. Many ideas like the above (and much more) go into multi UAV control algorithms. The following "related research" is a quick survey of how others have approached the problem of controlling multiple UAVs for various missions.

2.3 Related Research

The following presents 4 different approaches to cooperative control of UAVs. The scenarios vary, but all illustrate the complexity of this area and also illustrate the pros and cons of optimality. Suitability of the algorithms for real-time is also presented.

2.3.1 Broad Area Search-Beard. Randal Beard [4] explored the affect of sub-optimal solutions on both performance and computational time. The problem was a broad area search with randomly generated opportunities and threats. The UAV team was subject to the constraints of vehicle dynamics, a lower distance limit for collision avoidance, and an upper distance limit for communications.

One example used 3 alike UAVs with a front-looking sensor with a footprint of width w. The globally optimal solution considered all possible paths which maximized the number of opportunities observed and minimized contact with the threats. In this particular scenario, there were 50^3 possible paths. This brute force solution sensed 10 targets and took 522 seconds to solve. By constraining the optimal feasible vehicle paths between the vehicle path on the right and the vehicle path on the left, 10 targets were observed and the computational time decreased to 13.3 seconds.

The sub-optimal solutions took two different approaches. The first approach lets the first vehicle take the "best leader" myopic path (not considering the other vehicles). The second vehicle also taking the next myopic path only constrained by

the first path. This repeats until all N paths are generated for the N vehicles. The second sub-optimal approach takes into account that the team may be better served by each individual taking sub-optimal paths. The search is limited to "pairwise" feasible paths. With the "best leader" approach, 9 targets were sensed and the algorithm found a solution in 1.2 seconds. The author did not run the second sub-optimal with the 3 UAV example, but did state for a cost function J,

$$J_{bestleader} \leq J_{pairwise} \leq J_{optimal} \tag{2.7}$$

This article is a good cost-benefit analysis between optimal and sub-optimal solutions, and similar results are expected for the heuristic approach developed herein.

2.3.2 Task Oriented Cooperative Control-Richards/Bellingham. Arthur Richards and John Bellingham [15] of MIT added a couple more layers of complexity to the problem. The problem was constructed to accomplish a mission in minimum time (the cost function). The "mission" became visiting all required waypoints and also adhering to any timing constraints (e.g. must visit A five minutes after D). Not all UAVs were alike, and only certain vehicles could visit certain waypoints depending on their capabilities. Superimposed on this was "no fly zones" –rectangular areas where trajectories could not penetrate.

The intent of this method was to find a globally optimal solution. Realizing the computational intensive nature of this solution, the authors suggest using this as a benchmark on which heuristic methods can be compared, and is not well suited for real-time applications. To illustrate the optimal/approximate differences, the solution is compared to a less intensive method that estimates the trajectory planning portion of the problem.

The UAV model was simplified to two dimensions. The vehicle itself was modelled as a point mass. The x-y position, x-y velocity, and yaw rate comprised the whole state. Vehicle dynamics were linearly modelled. The velocity is constrained to

a maximum magnitude, and the turn rate is also constrained by placing a limit on the lateral force magnitude. The optimization cost function minimizes not only time but also weights control effort of each UAV.

The approximate method simplifies the solution search. First, assignments that place a large number of waypoints (tasks) onto a single vehicle are eliminated. To find approximate completion times, a straight line approximation method is used. Once costs are calculated and an approximate minimum cost is found, then detailed trajectory analysis is performed that accounts for dynamics and collision avoidance.

Optimal and sub-optimal results were compared with a scenario involving three UAVs, 2 no fly zones, and 4 waypoints. Using CPLEX[1] optimization software with a 1GHz PC with 256MB of RAM, the computation took over eight minutes. With the same PC, CPLEX and MATLAB software, the approximate method found a solution in under 5 seconds. In this case, the approximate method found the globally optimal solution.

This is a good illustration that when approximations are used in the appropriate locations, solutions can be nearly optimal (maybe even optimal) at a fraction of the computer time. This research is a variation of Beard's approach, but requires that more than 1 UAV is needed to accomplish a task. When compared to optimal, the algorithm presented in this research will also be comparable to the "best leader" approach, since it also takes a myopic approach based on 1 UAV (See Chapter 3).

2.3.3 Cooperative Control Simulation-Rasmussen/Mitchell. To deploy a usable system that utilizes UAVs in a cooperative manner, many factors besides optimal path planning affect the performance. Whether the mission is a broad area search, surveillance/reconnaissance, or search and destroy, the necessary tasks may require UAVs to detect, assess, engage, assess again, and possibly engage again. Any one of these tasks are not trivial for an automated/unmanned system (let alone a manned

[1]A software package sold by ILOG(IBM) for business decision making, efficient resource utilization, and scheduling/planning. For more info: www.ilog.com/products/cplex

system). AFRL's Air Vehicles directorate created a simulation environment, MultiUAV2 [14], that attempts to provide an assessment tool for cooperative approaches, but is not intended for real-time use. The thought that went into this product illustrates the complexity of the problem, and highlights items to consider as an engineer or researcher.

The state of all targets is passed to all vehicles. Each target state includes position, and whether the team has detected, classified, attacked, and assessed the target. Messages are passed to all as a target changes state. Each vehicle computes the cost to accomplish the remaining tasks, and transmits to all. The cooperative control algorithm resides on all vehicles, the cost is assessed and assignments are given.

At the lowest level of logic is vehicle dynamics and path planning. All vehicles are modelled with a 6 degree of freedom aircraft model, inner control loops that govern the desired velocities and attitude, and outer control loops that govern desired altitude, heading, and waypoint tracking. One step higher in the hierarchy is the path. Unless specified otherwise, all paths calculated are optimal. The task at the end of the path could take many forms: a post-attack assessment, an attack, a second look, or possibly an anomaly that needs classification.

The detection portion combines probability and directional dependence. This accounts for the fact that even if a target is in the field of view, there is no guarantee that it will be detected. The probability of identification varies depending on which direction the sensor views the target. Using trigonometric functions, the probability maximizes at defined directions and decreases as the view angle deviates from the ideal angle. For example, in the default scenario, for a single pass, the maximum certainty is 0.8 and you need a certainty of 0.9 before you can attack. So the simulation forces multiple passes before the target can be positively identified.

At the highest level are the Cooperation algorithms. Each vehicle has these algorithms on-board, and all vehicles are "in sync" with their assigned tasks. Given the

state of the system (including all UAV states and all target states), tasks/assignments are assigned to each vehicle. The different algorithms to distribute the workload vary greatly, but fall into two general categories: Single Assignment Tour and Multiple Assignment Tour. As the name implies, Single Assignment Tour hands the vehicle only one task (e.g. search, or classify) at a time. It keeps complexity down, but can be very inefficient. The Multiple Assignment Tour accounts for the next step in the process and improves efficiency, but can lead to a "combinatorial explosion." Many algorithms can be implemented, both optimal and suboptimal. Brief explanations for each approach are included in [14].

This tool forces any designer of a Cooperative Control scheme to consider the necessary attributes needed to create a useful product to the warfighter. The design must include much more than path planning and should consider probability of detection, complexity vs efficiency, and inter UAV communication. A real-time autonomous application of this extends well beyond the scope of this research, and provides a road to follow for future real-time research.

2.4 Summary

As this chapter illustrates, using UAVs in a cooperative manner pulls knowledge from many areas and often becomes a very complex problem. To aid in tackling this task, this research pulls background information from past work in aircraft modelling and path planning. The works of Jodeh and Kingston will aid in creating a representative model. Also the works of Zollars and Terning establish a base to create a path planning approach. The related research section illustrates both the varied application of multi-UAV schemes and also the trade-offs of optimal vs sub-optimal solutions. This provides a good base to create a real-time multi vehicle algorithm.

III. Equipment and Analysis

3.1 Introduction

This chapter lays the foundation for modelling the BATCAM and creating the cooperative control algorithm. It begins with a description of the BATCAM and the equipment associated with flight testing. Pulling from research presented in Chapter II, a closed-loop model is created for the BATCAM. Next the algorithm is developed to simultaneously approach and maintain a surveillance orbit. The chapter ends with a brief overview of the cooperative control algorithm software that interacts with each UAV.

3.2 Equipment

3.2.1 BATCAM. Applied Research Associates (ARA) in conjunction with AFRL developed the BATCAM. The idea came from the need to develop a tactical air surveillance/reconnaissance tool with easy portability and very low logistical needs. The primary customer for this product are special operations forces. The BATCAM provides real time situational awareness and targeting information [3] and has many features well suited for field use (illustrated in Figure 3.1). Both its light (0.85 lb)

Figure 3.1: BATCAM UAV [3]

carbon composite airframe and flexible composite/fabric wings makes the body resilient to damage. Launching is done by hand (like a paper airplane) and is powered

Table 3.1: BATCAM Camera FOV Angles

Angle	Front Camera	Side Camera
Depression Angle	49°	39°
horizontal FOV	48°	48°
vertical FOV	40°	40°

by a quiet electric motor. The batteries can be recharged with a car cigarette lighter in less than an hour. Assembly of the wing airframe takes less than a minute, and removing a spent battery with a recharged battery also takes about a minute.

The surveillance capability consists of two small cameras hard mounted to the bottom of the fuselage. The first camera is a forward looking camera , the second camera is a side looking camera facing left (from the pilot's perspective). Neither camera is gimballed, so the attitude of the BATCAM dictates the FOV of the sensor. Table 3.1 provides the specifics on the FOVs. The side camera will be used for the algorithm as the UAV orbits the target counter-clockwise.

3.2.2 Kestrel Autopilot. Each BATCAM used for flight test contained Procerus Technology's Kestrel Autopilot. This small electronic device (see Figure 3.2) provides "autonomous flight control, Global Positioning System (GPS) waypoint navigation, autonomous take-off and landing." [1] The sensor suite contains 3-axis rate gyros, accelerometers, differential and absolute air pressure sensors for attitude estimation and altitude/airspeed estimation. A GPS receiver provides positional information to the flight computer. Extra serial ports provide the ability to dynamically communicate and execute commands to cameras or payload devices. The autopilot communicates with the ground station at 900 MHz with specialized TCP/IP packets. In addition to command packets, telemetry packets provide the aircraft state information at rates up to 25 Hz. The autopilot is just one component of the complete system.

The complete system requires a laptop for human interface, software for the laptop, a USB communication box to "talk" to the autopilot, the autopilot itself, and

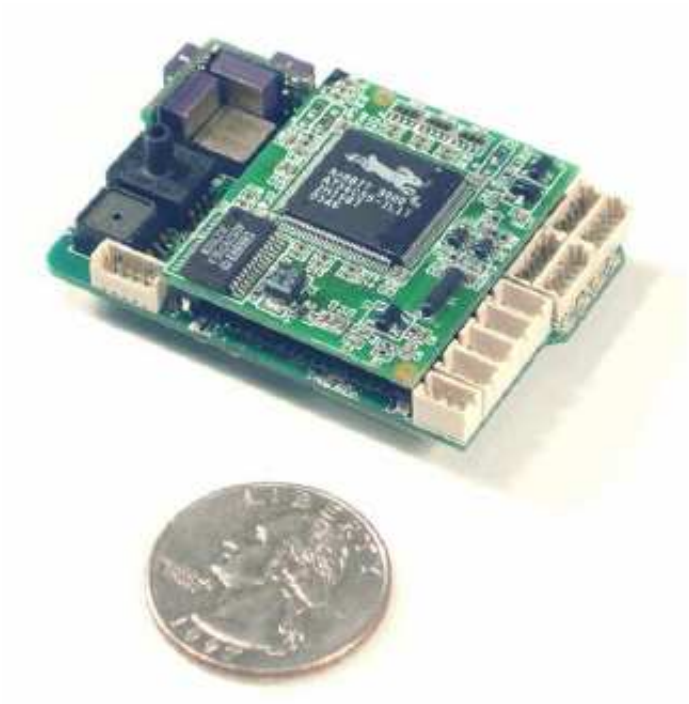

Figure 3.2: Kestrel Autopilot [1]

a Radio Controlled type model airplane retrofitted with a pitot tube. The software, named Virtual Cockpit, provides the user the ability to command a loiter, a set of waypoints, an airspeed, an attitude, and set up fail safes in the event of problems. VC can also toggle between manual and autonomous modes, monitor progress with detailed displays, and adjust many parameters in-flight. The software can communicate to multiple UAVs simultaneously via the communications box. The Virtual Cockpit (VC) software will be the primary means to command the autopilots in the air.

The software development kit for Virtual Cockpit creates an interface to create custom applications. Since the communication scheme is TCP/IP based, all information going to and from the UAVs is packet based. Many packet types are available, some that command the autopilot, and some contain UAV telemetry. Using the standard loop-back network capability, the custom application can run alongside Virtual Cockpit, receive all incoming packets, and pass command packets. This is the approach used to demonstrate the control scheme.

3.2.3 Flight Test Setup. The equipment used for flight testing is a self contained mobile trailer with all the necessary items for both flying and repair. The trailer has both an operations area and a maintenance/repair area (see Figure 3.3).

The operations area has all the necessary capability to both command and control the UAVs and display video. The trailer provides the operators and electronics protection from the elements. The laptop is used to command and control the UAVs and the video equipment provide the surveillance capability. The laptop passes commands and receives telemetry from the comm box using a serial to USB connection. This comm box is also made by Procerus Technologies (who makes the autopilot) and is a necessary component to communicate with the autopilot. The 900 MHz antenna is connected to the comm box. Video is received via two 2.1 GHz antennas. The antennas are configured to maximize coverage. Each signal is split into 4 and passed to 4 receivers, providing the capability to receive 4 separate feeds from 4 UAVs. Each receiver has two feeds, and chooses the one with the best quality for the output. The video switch takes the 4 feeds and provides the flat screen TV a customizable display. Possible displays include all 4 feeds on 1 screen, two chosen feeds, or just 1 feed from any available signal. Both the command, control and surveillance capabilities provide a good platform for UAV research.

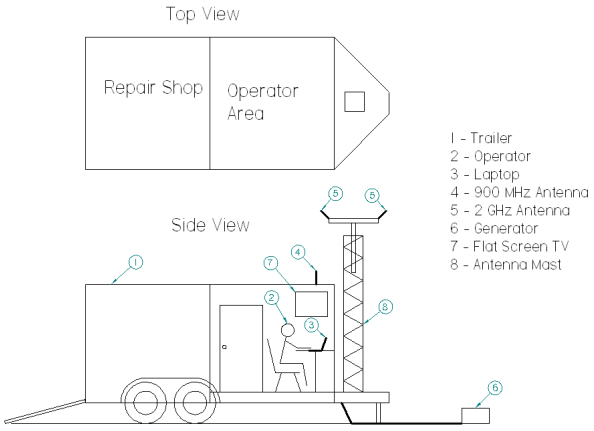

Figure 3.3: Flight Testing Setup

The maintenance/repair area helps sustain the UAVs. The toolbox contains a variety of basic tools (wrenches, screwdrivers, socket sets, glues) and specialized equipment for Radio Controlled (RC) airplanes (e.g. refuelling pumps, engine starters). The workbench provides an area to work and also has a strip of outlets for battery rechargers and power tools. Also on hand are repair materials like balsa wood, tubing, plastic, extra propellers, etc.

Flight Tests are conducted at Camp Atterbury, Indiana. Since the Federal Aviation Administration (FAA) requires that autonomous vehicles be only flown in Restricted Airspace, Camp Atterbury is the nearest facility to AFIT with this designation. The UAVs are flown at the airfield, and each flight is coordinated with the airfield.

Operations are conducted in typical Air Force style. Test objectives are created for each flight test and test cards are written. Each card contains detailed procedures necessary to accomplish each test. Before testing, the whole test is presented to a Technical Review Board/Safety Review Board (TRB/SRB) for approval. After incorporation of comments and approval from the presiding officer, the test is conducted.

Personnel are assigned specific roles to accomplish the test. The Test Conductor oversees the test operations. The Pilot is a certified UAV operator that keeps the vehicles in sight at all times and can manually fly them if necessary. The Ground Operator sits at the laptop and controls the UAVs using the VC software. The Data Recorder documents all information for each test card and also any anomalies noted. The Launch/Recovery personnel accomplish launch and recovery. The Safety Officer supervises the test and provides input to minimize injury and equipment damage. During flight, the Ground Operator and Pilot are in constant communication so that the Pilot understands what the UAV is suppose to be doing. The Pilot also provides feedback to the Operator to improve performance and prevent any undesired situations.

3.3 Airframe-Properties

The BATCAMs came to AFIT "as is." To make up for the lack of technical drawings and sparse documentation, some basic measurements were needed to begin the modelling process. Physical dimensions and mass were combined with Kingston's [9] closed-loop model to create a general airframe description.

3.3.1 Physical Dimensions. Airframes are typically geometrically complex, and the BATCAM is no exception. The desired approach needed to be simple, efficient, but also reasonably accurate and complete. For the purposes of this research, highly accurate measurements were not necessary. For dimensions not easily measured with a tape measure, a photographic technique was used. Digital pictures are taken of the airframe from each of the major axes (side, front, rear, top), with a reference measurement visible in the picture. To minimize parallax and perspective errors, the reference measurement must be placed at the same distance as the item of interest, the focal axis must be orthogonal to the image plane, and the distance between item and camera must be sufficient. This technique is regularly used in repair situations where the engineer is physically separated from the damaged part but must design a fix.

Figure 3.4 illustrates this technique. Using software comparable to GNU Image Manipulation Program (GIMP), pixel distances can be measured. In this example, the desired measurement is the overall length of the BATCAM. Three pixel measurements are required: the pixel distance of 1 inch(ppi), the pixel distance of the BATCAM(d_{pix}), and the angle of the bottom of the UAV (θ_{bot}) with respect to the picture. The approximate length (l) becomes

$$l = \frac{d_{pix}}{ppi \cos(\theta_{bot})} \tag{3.1}$$

In this case, the horizontal pixel distance is 1965 pixels, ppi is 98 pixels per inch, and the bottom of the BATCAM is tilted 1.54 degrees, yielding an overall length of 20.06

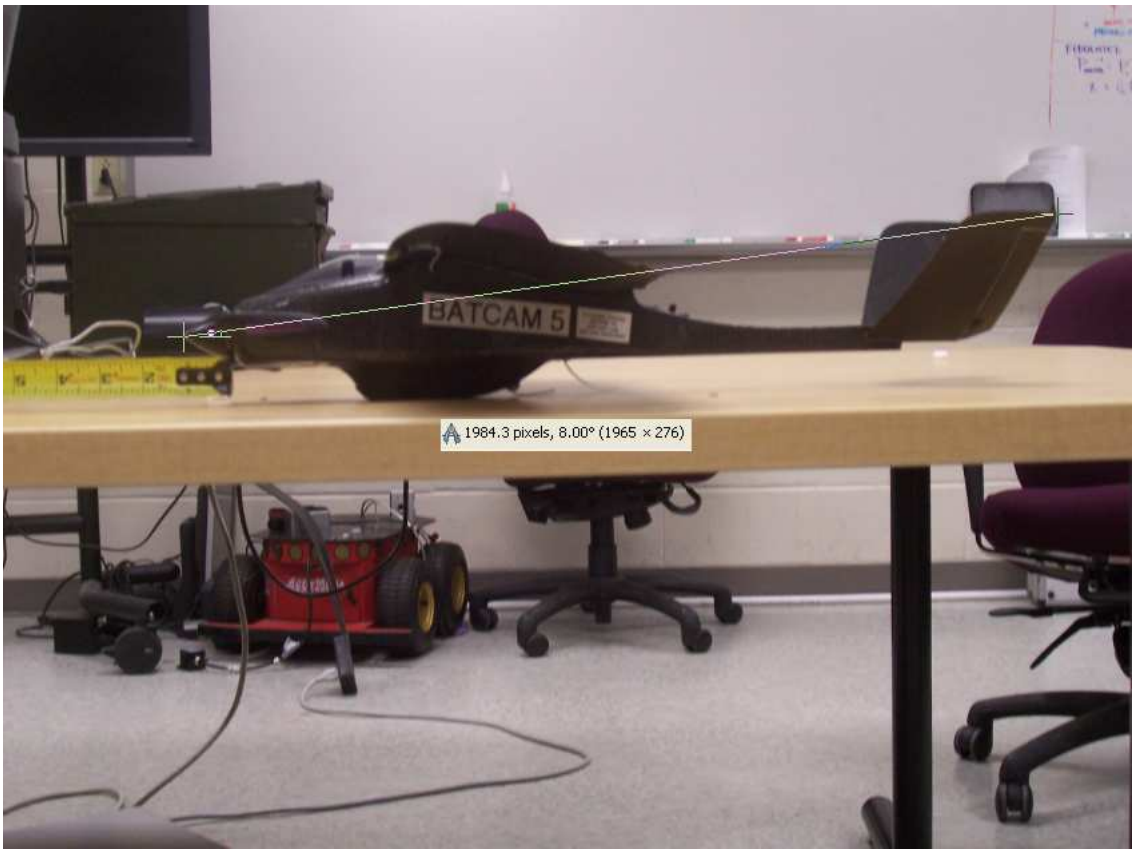

Figure 3.4: Length Measurements using Digital Pictures

inches. From the BATCAM users manual [3], the stated length is 20 inches. When done properly, the margin of error for this technique is estimated to be between .2 and 1%.

3.3.2 Mass, Aerodynamics, Results. This section presents the airframe properties required for the model in section 3.4.1. It includes a set of physical dimensions and areas needed for lift calculations and mass properties.

The mass is determined by simply placing the BATCAM on a scale. From basic physics, the weight is divided by the gravitational acceleration to find mass. The scale used was accurate to the nearest gram.

In Section 3.4.1, the only needed aerodynamic force for the model is lift. This research will assume a constant coefficient (C_L) of lift at 0 degrees angle of attack

Table 3.2: Dimensions and Mass Properties

Item	Value
Wing Reference Area (S)	103.7 in^2
Mass (m)	0.425 kg

(AoA) centered around the level steady flight conditions. From flight testing, level steady flight occurred at

$$\begin{aligned} V_{ao} &= 11.75 m/s \\ Throttle &= 50\% \end{aligned} \tag{3.2}$$

DeLuca [5] conducted an extensive aerodynamic analysis of the BATCAM airframe at airspeeds of 10, 20, 30, and 50 mph. Based on the steady flight conditions, the 20 mph data for a flexible was used to determine C_L. Using basic fluid mechanics [12], lift (L) for the BATCAM is

$$\begin{aligned} C_L &= 0.7 \\ L &= \frac{\rho}{2} V_a^2 S C_L \end{aligned} \tag{3.3}$$

where V_a is airspeed, ρ is the air density, and S is the wing reference area. All other aerodynamic forces are not omitted, but are rolled into the first order constants in the model. Section 3.4.1 will explain this further. Table 3.2 summarizes the needed values for the model.

3.4 Simplified Closed Loop model

This section creates a model for a single UAV that describes the closed loop system created by the UAV and autopilot. As stated in the assumptions, the response of some of the state is assumed to mimic a first-order ordinary differential equation (ODE), discarding higher order terms. Other variables in the state retain the non-linear properties.

3.4.1 Base Model. The simplified model is based upon Equation 2.1. To account for dynamics in the vertical direction, pitch (θ) and altitude (h) become part of the state, and Equation 2.1 is changed to

$$\begin{aligned}
\dot{p_N} &= V_a \cos\psi \cos\theta + w_N \\
\dot{p_E} &= V_a \sin\psi \cos\theta + w_E \\
\dot{V_a} &= k_V(V_a^c - V_a) \\
\dot{h} &= V_a \sin\theta \\
\ddot{h} &= \frac{L(V_a)\cos\phi\cos\theta}{m} - g \\
\dot{\phi} &= k_\phi(\phi^c - \phi) + w_\phi \\
\dot{\theta} &= k_\theta(\theta^c - \theta) \\
\dot{\psi} &= \frac{g}{V_a} tan\phi
\end{aligned} \tag{3.4}$$

For this set to be reasonably accurate, roll (ϕ) and pitch (θ) must remain "small". The proportional gains k_* will be empirically calculated from flight test results. Note that changing the heading depends on banking (roll) the UAV (i.e. a coordinated turn). The commanded pitch θ^c will be used to reach the desired altitude. The wind disturbance (w_ϕ) in the roll rate equation captures the impact of wind disturbances on the side camera of the BATCAM. Many of the aerodynamic forces (except L) are included in the k_* values. By using flight test data centered around the nominal cruise airspeed, the extracted k values should reasonably reflect BATCAM/autopilot system behavior.

The wind will be modelled using the Dryden wind turbulence model [10]. This representation adds turbulence to velocity spectra by passing band limited white noise through appropriate filters. These filters are mathematically described in MIL-HDBK 1797 [10]. The roll disturbance will be a scaled quantity of the wind magnitude. Conveniently, MATLAB's Simulink program has a prepackaged block that produces this wind model in the Aerospace blockset. The inputs to this block are wind speed and direction at 6 meter height, a probability scale for light/medium/heavy turbulence, scale height, UAV's airspeed, UAV's altitude, and the output is the North/East/Down components of wind. Only the North and East components are used in this research. The roll disturbance is a scaled value of the magnitude of the wind ($w_\phi = k_\phi |W|$). Figure 3.5 is a typical wind profile for a 1 m/s input.

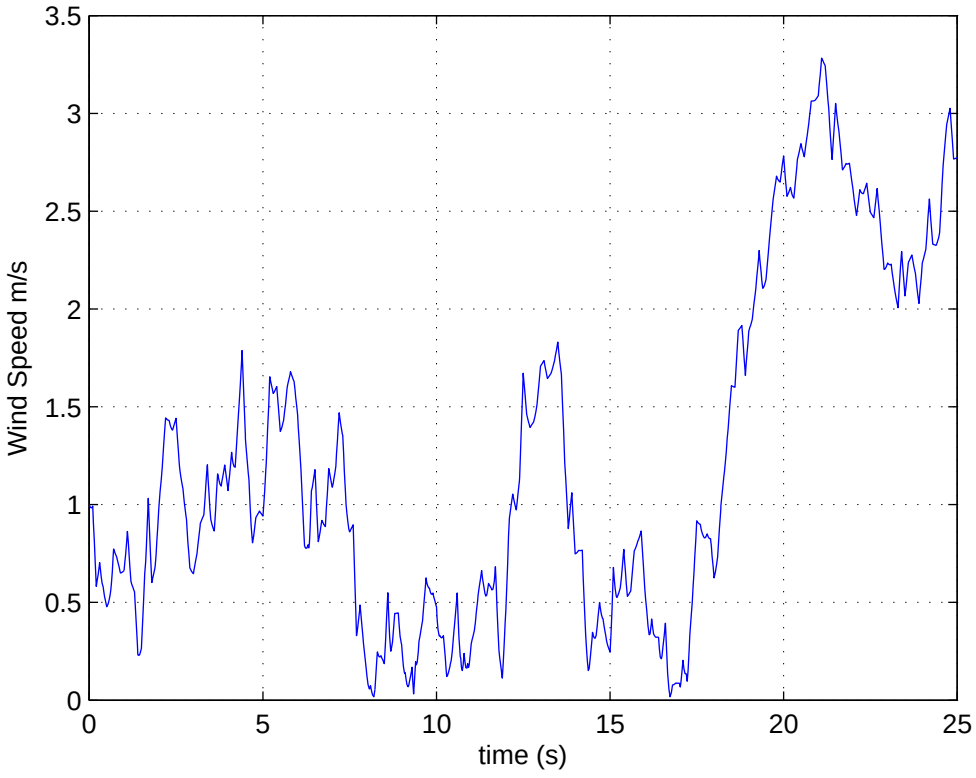

Figure 3.5: Dryden Wind Model-1 m/s Input Wind Profile

A full 6 DOF model based on measured data was initially attempted, but not yet completed. Appendix A contains this more rigorous model of the BATCAM (and

included MATLAB files) that includes linearized aerodynamic coefficients based on DeLuca's work [5].

3.4.2 Closed Loop Model Details. Both commanded heading (χ^c) and commanded altitude (h_c) are key parameters in controlling the UAV, but neither of these appear in the model above. Changes in heading depend on roll angle, and changes in altitude depend on airspeed and pitch angle. This section provides the details to bridge roll to heading, and altitude to pitch.

Cross distance (y_s) is the perpendicular distance of the UAV from the intended path (See Fig 2.4 or Fig 3.6). This is an important quantity for two reasons: 1-y_s will be used as a measure of performance for the algorithm; 2-y_s is used to determine the commanded heading χ^c. Since UAV placement is one of the key factors affecting sensor performance, Chapter IV will use y_s to quantify algorithm performance. y_s is also used to provide feedback and get the UAV back on the desired path. The following analysis will find the cross distance using basic vector math and then use Rysdyk's [16] approach presented in Section 2.2.5 to find the commanded heading to get the UAV back on the desired path. The error between commanded heading and UAV heading will feed into the roll feedback equation to cause a turn rate $\dot{\psi}$ (see Equation 3.2).

Figure 2.4 illustrates that the commanded heading depends on both the heading of the desired path(χ_s) and the cross distance from the desired path(y_s). The first step to find χ^c is to find y_s. Figure 3.6 illustrates the vectors necessary to find y_s. The waypoint location is at (P_{E2}, P_{N2})=$\vec{P}_2$, the desired straight line path starts at the origin to (P_{E2}, P_{N2}), and the current UAV location is at (P_E, P_N)=$\vec{P}$. The desired behavior is for the UAV to go to the waypoint along the desired path. From the definition of the cross product, the crosstrack distance (y_s) is

$$y_s = \frac{|\vec{P}_2 \times \vec{P}|}{|P_2|} \tag{3.5}$$

Realize that y_s is positive when $\theta_1 < \theta_2$ and y_s is negative when $\theta_1 > \theta_2$. To drive y_s to zero, the commanded heading equation (Eqn 2.2) becomes

$$\begin{aligned} \chi^c &= \sigma\left(\frac{|\vec{P_2} \times \vec{P}|}{|P_2|}\right) + \chi_s \\ \chi_s &= \frac{\pi}{2} - \arctan\left(\frac{P_{N2}}{P_{E2}}\right) \end{aligned} \tag{3.6}$$

The σ function was defined in Equation 2.3. Also note that the second equation converts an angle using the mathematical convention to a clock angle/heading angle. The commanded roll angle becomes

$$\phi^c = \begin{cases} k_\chi(\sigma(y_s) + \frac{\pi}{2} - \arctan\left(\frac{P_{N2}}{P_{E2}}\right) - \chi) & |\chi^c - \chi| \leq \chi_{max} \\ \phi^c = \phi_{max} & |\chi^c - \chi| > \chi_{max} \end{cases} \tag{3.7}$$

When the heading is "close" to the commanded heading, then a proportional gain will be used. When the heading difference is large enough, the commanded roll angle (which controls turn rate) is limited to a set value (ϕ_{max}). This is to keep the UAV stable. The value χ_{max} will determine this threshold.

The commanded pitch depends on the error between commanded altitude h^c and UAV altitude h. Bridging altitude to pitch will also use a proportional gain. The commanded pitch will be

$$\theta^c = \begin{cases} k_h(h^c - h) & |h^c - h| \leq h_{max} \\ \theta_{max} & |h^c - h| > h_{max} \end{cases} \tag{3.8}$$

Just like roll, commanded pitch will reach a maximum value when the altitude difference exceeds a certain value (h_{max}). Once again, this is to maintain positive control of the UAV.

Up to this point, the analysis has only described the BATCAM/autopilot model, and has not started the cooperative control algorithm. The inputs for this model are

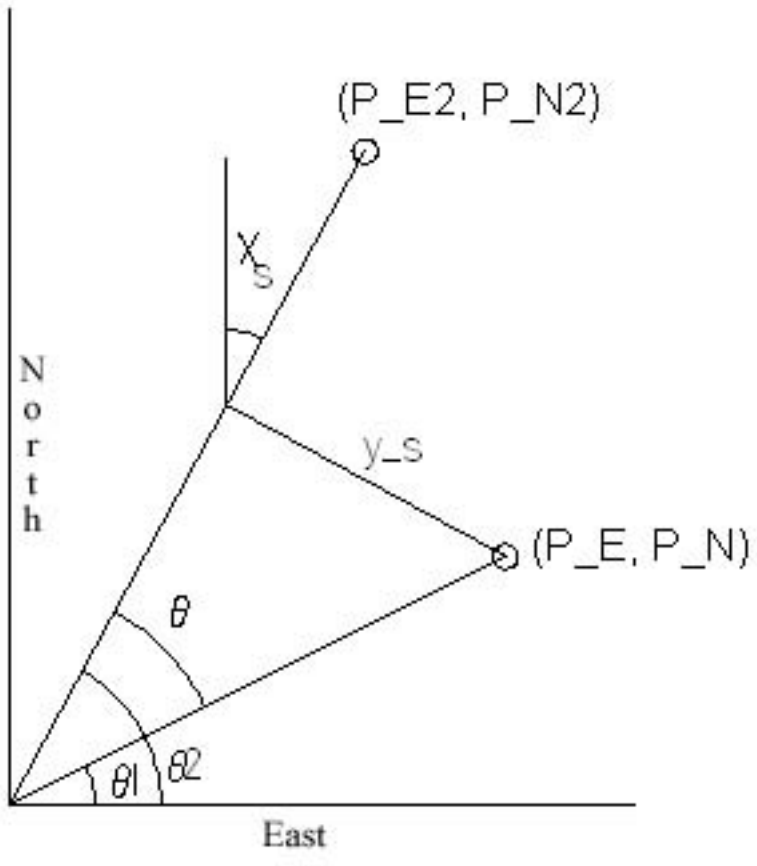

Figure 3.6: Illustration of Calculating Crosstrack Distance

the UAV state, waypoint location (p_{N2}, p_{E2}), the desired path heading χ_s, commanded altitude h^c, and commanded airspeed V_a^c. The cooperative control algorithm must provide these inputs (except the UAV state). Section 3.6 will describe how the algorithm will provide these inputs based on the state of all UAVs. One last section of background analysis remains for describing the individual UAV before the cooperative control algorithm, the Sensor Footprint.

3.5 Sensor Footprint Analysis

This section creates a projection of the FOV onto the ground given the UAV's attitude. As stated in Chapter I, this analysis assumes that the ground is flat and that the primary sensor is the side camera. This analysis provides the means to determine if a known target is visible, and also provides a way to determine the surveillance performance of the UAV. If the target lies within the FOV quadrilateral, then the target is "visible" to the camera. The measure of performance will find the percentage of time while orbiting that the target is visible.

The approach begins by creating vectors in the body frame and rotating them into the inertial frame via a directional cosine matrix. Then a parametric line will be drawn from the UAV position to the ground (z=0) in the direction of the rotated vector.

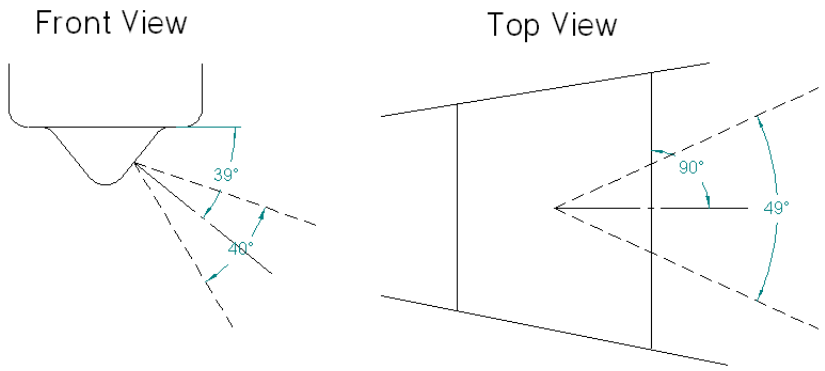

Figure 3.7: BATCAM's Field of View

3.5.1 FOV from body frame to inertial frame. Please refer to Figure 3.7. The BATCAM's FOV for the left facing side camera is square with the angles shown. Each corner of the FOV must be turned into a vector($\hat{e}$) in $\mathbb{R}^3$. If γ_d is the depression angle, FOV_H is the horizontal FOV, FOV_V is the vertical FOV, then the vectors for

all 4 corners of the FOV is

$$\hat{e}_1^b = \begin{pmatrix} \tan\frac{FOV_H}{2} \\ -1 \\ \tan\left(\gamma_d + \frac{FOV_V}{2}\right) \end{pmatrix} \quad \hat{e}_2^b = \begin{pmatrix} \tan\frac{FOV_H}{2} \\ -1 \\ \tan\left(\gamma_d - \frac{FOV_V}{2}\right) \end{pmatrix}$$
$$\hat{e}_3^b = \begin{pmatrix} -\tan\frac{FOV_H}{2} \\ -1 \\ \tan\left(\gamma_d + \frac{FOV_V}{2}\right) \end{pmatrix} \quad \hat{e}_4^b = \begin{pmatrix} -\tan\frac{FOV_H}{2} \\ -1 \\ \tan\left(\gamma_d - \frac{FOV_V}{2}\right) \end{pmatrix} \tag{3.9}$$

These vectors in the body frame use the standard body frame convention where the origin is at the center of gravity, the +x-axis extends out the nose of the UAV, the +y axis extends out the right wing (from the pilot's perspective), and +z-axis extends down. The ^{b} denotes body frame. Note the negative values in the y component to indicate a left facing camera. To rotate any of these body vectors into the geographic North/East/Down (NED) frame a directional cosine matrix (DCM)(C_b^n [18]) is used that utilizes the roll(ϕ), pitch(θ) and yaw(ψ) angles (Euler Angles). The vector in the NED frame is

$$\hat{e}_{1,2,3,4}^n = C_b^n \hat{e}_{1,2,3,4}^b \tag{3.10}$$

where (using the abbreviations c=cos, s=sin)

$$C_b^n = \begin{bmatrix} c\theta c\psi & -c\phi s\psi + s\phi s\theta c\psi & s\phi s\psi + c\phi s\theta c\psi \\ c\theta s\psi & c\phi c\psi + s\phi s\theta s\psi & -s\phi c\psi + c\phi s\theta s\psi \\ -s\theta & s\phi c\theta & c\phi c\theta \end{bmatrix} \tag{3.11}$$

The ^{n} denotes the geographic NED frame. Now the vectors are ready to extend to the ground.

3.5.2 Creating the Footprint. With the UAV at position (p_N, p_E, p_D) and attitude (ϕ, θ, ψ), lines that pass through the position with slopes $\hat{e}$ need to be created. Parametric representation is a convenient representation for this purpose. Using the

parameter (t), not to be confused with time, the vector representation of any point on the line is

$$\begin{pmatrix} x^n \\ y^n \\ z^n \end{pmatrix} = \hat{e}^n_{1,2,3,4} t + \begin{pmatrix} p_N \\ p_E \\ p_D \end{pmatrix} \tag{3.12}$$

These lines lie on each corner of the FOV and extend from the UAV position to the ground. There will be a total of 4 lines for the square FOV.

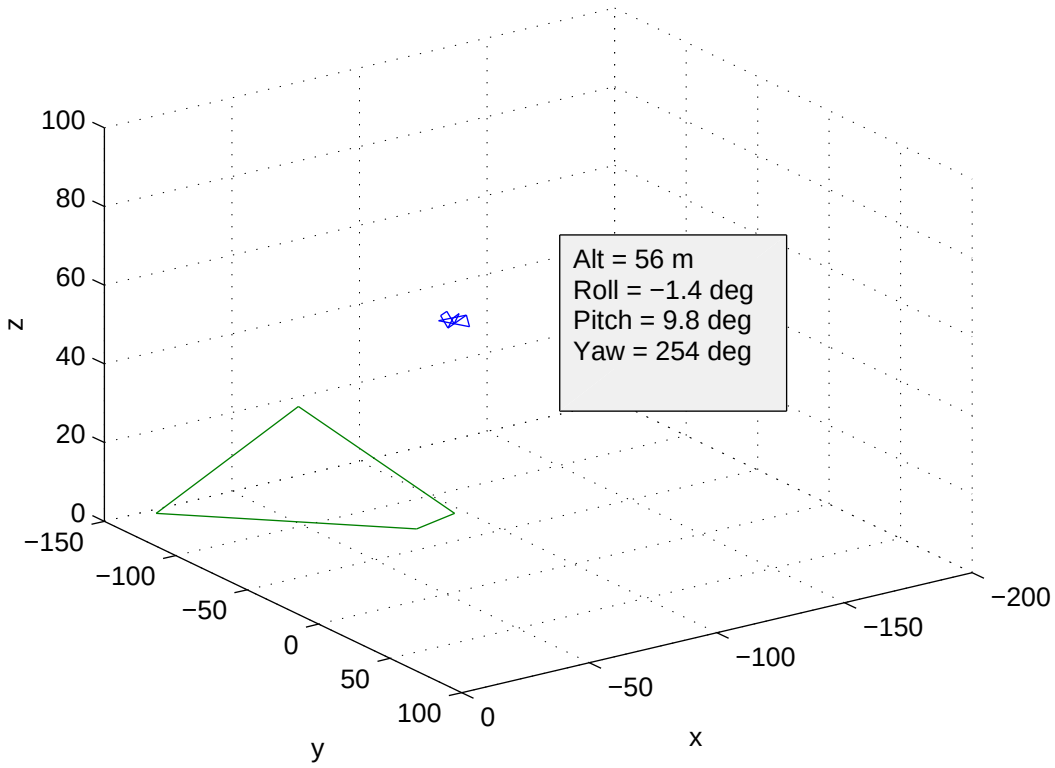

Figure 3.8: Sample Field of View Projected onto the Ground

It is important to remember that in the NED convention, a height above the ground shows up as a negative number since +z is down, therefore altitudes should be negative. Taking the z^n component and solving for t when z^n=0 provides the point on the line where the FOV intersects the ground. Each of the four $\hat{e}$ vectors will create a point on the ground (z=0 plane). By drawing a line between each of these points,

a quadrilateral is formed. Figure 3.8 illustrates this. If the target resides within this quadrilateral, the target is in the FOV. There are instances where the quadrilateral cannot be drawn. If the attitude of the UAV causes the FOV to be above the horizon, then a solution does not exist. The MATLAB routine written to create the FOV on the ground will check for this condition.

3.6 Cooperative Control Algorithm

This algorithm was written to command multiple UAVs to orbit a single target. The goals for the algorithm are to have all UAVs enter the surveillance orbit simultaneously, then maintain equal angular spacing during the orbit. This algorithm assumes the UAV is using a side camera and views the target while maintaining a CCW circular path above the target.

3.6.1 Overview. This CC approach creates Dubins Paths [6] for each UAV from their initial position and heading to a final position and heading that enters a surveillance orbit all tangent to the orbit and all in the same direction (CCW). To coordinate entering the surveillance orbit in an equally spaced manner, each path for each UAV is generated such that all have equal lengths, and are commanded such that the ground speed is equal for all. To aid each UAV so that they all enter the surveillance orbit at the same time, velocity feedback is incorporated as they traverse the path. As all UAVs enter the orbit, the algorithm switches modes and then tries to maintain equal angular spacing during orbit. It should be noted that this is just one of many possible solutions to cooperative surveillance, and will serve as a baseline for evaluating performance of alternative algorithms.

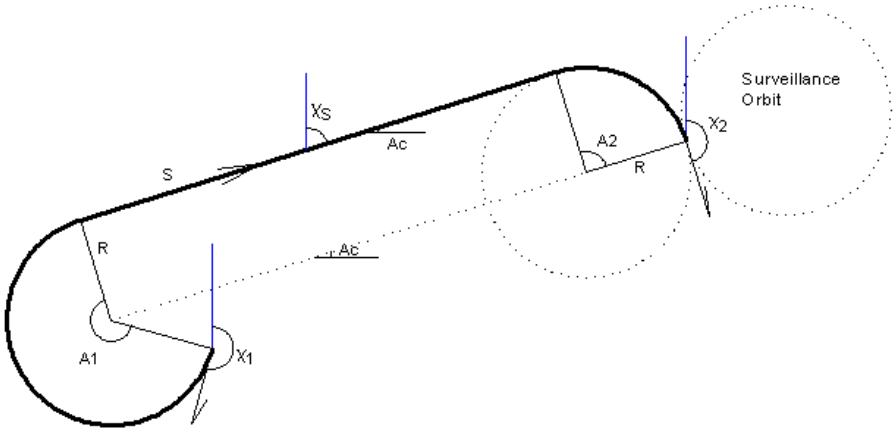

Figure 3.9: Dubins Path

3.6.2 Cooperative Dubins Paths. The Dubins path is composed of no more than three parts: two circular arcs and a straight path. In order to make the paths of all UAVs equal, a distance is needed for the sum of all three parts. From Figure 3.9, the path uses two clockwise(CW) circular arcs of equal radii(R) at each end of the straight section (S). The path length (Pl) is

$$Pl = S + A_1R + A_2R \tag{3.13}$$

where A_1 and A_2 are the angles swept (in radians) at each end. All paths for each UAV for the algorithm will be based on this path. There are other ways to construct this type of path by altering the rotation directions at each end and also having differing radii at each end. Equal radii is used for simplicity. The reason for CW rotations has to do with collision avoidance. The surveillance orbit must be counter clockwise (CCW) due to the left facing side camera. As multiple UAVs enter the surveillance orbit CW, they are forced to approach from the outside of the orbit, minimizing the chance of intersecting paths. No other method of collision avoidance was done as a part of this effort, although it is being investigated separately at AFIT.

Creating equal path lengths is done by altering the circular arcs. The airframe's maximum effort turn determines the minimum radius that can be used in the path. Using this radius, pathlengths for all UAVs are calculated. The UAV with the largest pathlength becomes the baseline. For all other UAVs, the radius is incrementally increased until all paths are equal. In some instances, there is not enough angular sweep to increase the path length to the desired length given the positions and headings. In this case, the algorithm will add a complete circle to one end and adjust the radius until the desired value is achieved.

Figure 3.10 is a quick illustration of the end product. This depicts 4 UAVs initially orbiting a racetrack on the right (in blue) then converging on a target orbit on the left. The algorithm sends each UAV a set of GPS waypoints (red X's) based on a path equal in length to the furthest UAV. Note that the UAV furthest from the

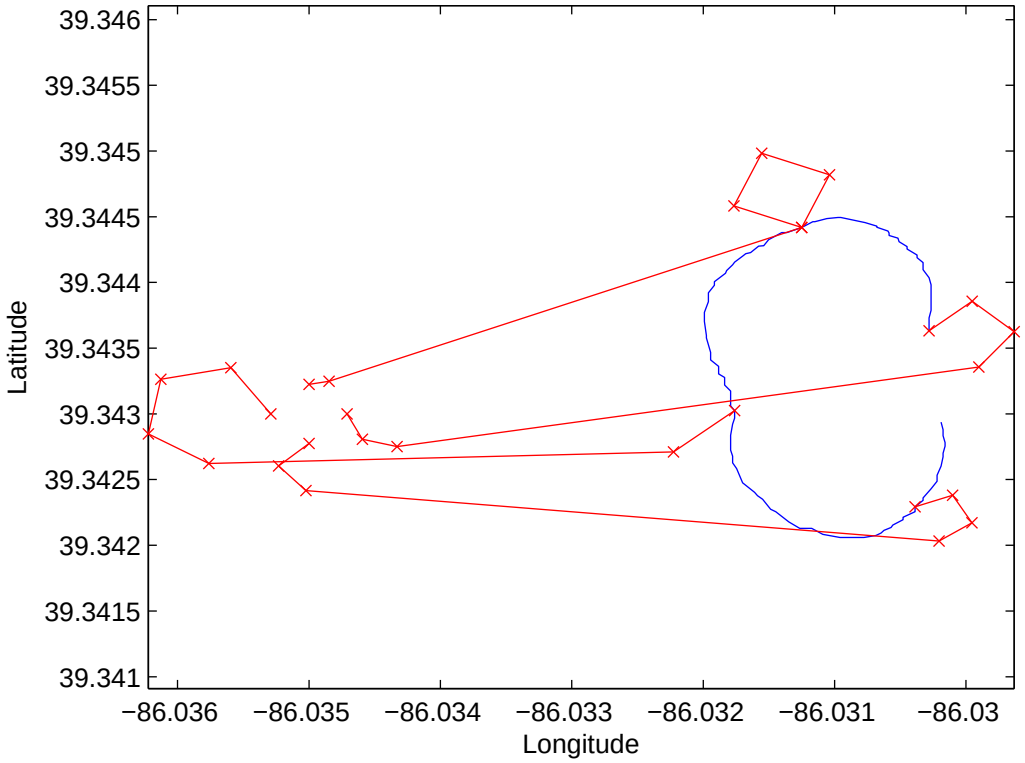

Figure 3.10: Waypoints based on Equal Path Lengths

target is on the lower right and has the smallest radius. Note that radii increase if the UAV is closer to the target. Also note that a full revolution is added to the upper right UAV due to lack of available angular sweep. The final headings at the target are determined by breaking the surveillance orbit into equal parts, in this case every 90 degrees. By keeping and maintaining the surveillance orbit, any angle of the target is kept in view at all times.

Simultaneous arrival simplifies many aspects of the approach to the surveillance orbit. First, it places the UAVs in the needed position as they arrive. This prevents the need for large adjustments after arrival and aids in collision avoidance. It also provides a nice simple approach to path generation and velocity feedback. If the paths are all equal, and all UAVs travel at the same ground speed, then arrival should be the same. Usually the real situation alters these conditions, so velocity feedback is used.

This is an arbitrary constraint, and simultaneous arrival is not the only approach, but this approach is simple and easy to implement. Next, the details of creating the paths is described.

Procerus Technologies Kestrel Autopilot and the accompanying VC software use GPS to exchange position telemetry, thus arises the need to translate Cartesian coordinates to Latitude (La) and Longitude (Lo), and vice versa. Using the World Geodetic System 1984 (WGS-84) ellipsoid [13] and a small angle approximation, a change in position can be converted from La/Lo/h into p_N, p_E, p_D using [13]

$$\begin{aligned} p_N &= (R_m + h)\Delta La \\ p_E &= (R_p + h)\cos La \Delta Lo \\ p_D &= \Delta h \end{aligned} \tag{3.14}$$

where

$$\begin{aligned} R_m &= \frac{a(1-e^2)}{(1-e^2\sin^2 La)^{3/2}} \\ R_p &= \frac{a}{(1-e^2\sin^2 La)^{3/2}} \end{aligned}$$

a is the semi-major axis of the Earth, e is the eccentricity of the WGS-84 ellipsoid. This equation set is valid only for distances that are "small" compared to the size of the Earth. Rearranging the above and solving for ΔLa and ΔLo, the conversion from Cartesian to La/Lo/h is

$$\begin{aligned} La &= La_o + \frac{\Delta p_N}{R_m + h} \\ Lo &= Lo_o + \frac{\Delta p_E}{(R_p + h)\cos La_o} \\ h &= h_o + \Delta p_D \end{aligned} \tag{3.15}$$

This equation requires that the La/Lo/h of the Cartesian origin is known ($La_o/Lo_o/h_o$).

The heading angle (CW starting from due north) also needs a conversion to conform with the standard mathematical conventions (CCW starting from the x-

axis). Let the * denote the converted heading angle. With angles in radians, the conversions to and from are

$$\begin{aligned} \chi &= \frac{\pi}{2} - \chi^* \\ \chi^* &= \frac{\pi}{2} - \chi \end{aligned} \tag{3.16}$$

With the position in Cartesian coordinates and heading angle using standard mathematical conventions, the first step to creating the path is finding the centers of the end circles (See Fig 3.9) given the initial and final positions/headings. Subscript 1 denotes quantities in the initial circle, and Subscript 2 denotes quantities in the final circle. Since the paths run CW, the center of the circle will always lie to the right of the UAV, from the pilot's perspective. Consequently the angle from the UAV to the center of the circle will be $\chi_c^* = \chi_{1,2}^* - \pi/2$. If the UAV positions are $(p_{E1,2}, p_{N1,2})$, the positions of the circle centers are

$$\begin{pmatrix} p_{Ec1,2} \\ p_{Nc1,2} \end{pmatrix} = \begin{pmatrix} p_{E1,2} + R\cos(\chi_{1,2}^* - \pi/2) \\ p_{N1,2} + R\sin(\chi_{1,2} - \pi/2) \end{pmatrix} \tag{3.17}$$

With the circle centers, χ_S can be found. The heading in the straight section is parallel to the line passing through the circle centers since the radii are equal. The straight path heading is

$$\chi_S^* = \arctan\left(\frac{P_{Nc2} - P_{Nc1}}{P_{Ec2} - P_{Ec1}}\right) \tag{3.18}$$

From this, A1 and A2 can be calculated:

$$\begin{aligned} A1 &= \chi_1^* - \chi_S^* \\ A2 &= \chi_S^* - \chi_2^* \end{aligned} \tag{3.19}$$

A1 and A2 must always be positive to generate a positive length. When a negative quantity occurs, 2π must be added to A1/A2.

The position of the two tangency points is needed to find the length of S. Using the circle center positions, the tangency points are located at

$$\begin{pmatrix} p_{Et1,2} \\ p_{Nt1,2} \end{pmatrix} = \begin{pmatrix} p_{Ec1,2} + R\cos(\chi_{S1,2}^{*} + \pi/2) \\ p_{Nc1,2} + R\sin(\chi_{S1,2}^{*} + \pi/2) \end{pmatrix} \tag{3.20}$$

Then the straight section distance is

$$S = \sqrt{(P_{Nt2} - P_{Nt1})^2 + (P_{Et2} - P_{Et1})^2} \tag{3.21}$$

Some simplifying geometry occurs when S>2R. In this case, the distance between circle centers is the same as S eliminating the need to find the tangency points.

Now Equation 3.22 can provide the total path length. The next step is to find the appropriate R for each UAV such that all path lengths are equal. Finding the necessary R's is an iterative process. Once the path lengths are calculated using the minimum radius, the largest path length (Pl_{base}) becomes the baseline for the rest. Figure 3.11 illustrates the loop used to find the correct R for each UAV. As the top decision block implies, the path lengths are not exactly the same, but are found within a defined tolerance. After the path length is calculated, the algorithm checks to see whether that value has exceeded the baseline. If so, dr changes direction and is halved. If Pl_i has not exceeded the baseline, R is incremented by dr. The next decision block checks whether the solution is converging. Non-convergence occurs when there is not enough angular sweep in the given path. In this case, R is reset to the minimum radius and a full circle is added to the initial arc in the path. This approach is not mathematically elegant, but yielded stable results under all circumstances. The algorithm repeats until all UAVs have the same path length.

With a solution for all desired paths, the information is converted back into GPS format for the autopilot. The autopilot needs a discrete set of GPS waypoints. This algorithm is configured to provide a waypoint for every 90 degrees of sweep in

the arcs, in addition to a waypoint at each end of the straight section. With all paths defined, the next item to address are the UAV velocities.

3.6.3 Handling Wind with Feedback. For all UAVs to arrive into the surveillance orbit at the same time and phased equally around the orbit, the ground velocity (V_g) must also be the same for all. In the presence of wind, maintaining this ground velocity depends on the heading of the UAV. To find the necessary airspeed (V_a), vector addition states that

$$\vec{V_g}^c = \vec{V_a}^c + \vec{V}_W \tag{3.22}$$

Using the above, the commanded values for V_a^c and χ^{c*} become

$$\begin{aligned} \chi^{c*} &= \arctan\left(\frac{V_g^c \sin\chi_g^* - V_W \sin\chi_W^*}{V_g^c \cos\chi_g^* - V_W \cos\chi_W^*}\right) \\ V_a^c &= \frac{V_g^c \cos\chi_g^* - V_W \cos\chi_W^*}{\cos\left(\arctan\left(\frac{V_g^c \sin\chi_g^* - V_W \sin\chi_W^*}{V_g^c \cos\chi_g^* - V_W \cos\chi_W^*}\right)\right)} \end{aligned} \tag{3.23}$$

The Kestrel autopilot provides a wind estimate in the telemetry from each UAV. The commanded ground velocity will come from cooperative control algorithm (Eqns 3.33 and 3.36). Each time the algorithm issues a ground speed command, it will be transformed into an airspeed command using the above equation before forwarding to the UAV autopilot. However, the feedback is actually done in terms of ground speed. During flight testing, the autopilot commands the heading, so the first equation in 3.32 is not used.

The feedback routine that maintains ground speed while approaching the surveillance orbit does not use time directly to accomplish simultaneous arrival. By using the mean path length ($\bar{Pl}_r$) remaining for all UAVs from the orbit, it creates a floating reference for proportional feedback. The commanded ground speed for the ith UAV

becomes

$$V_{gi}^{c} = V_{nom} - k_1(\bar{Pl}_r - Pl_{ri})$$
$$\text{where} \tag{3.24}$$
$$\bar{Pl}_r = \frac{1}{n}\sum_i Pl_{ri}$$

where Pl_{ri} is the remaining path length for the ith UAV, and k_1 is a proportional gain. Finding the remaining path length uses the waypoints. Figure 3.12 illustrates how to find this distance. When the path is calculated, an array is created that stores the remaining distance in the path at each waypoint (WP). For example, d_5 is the distance from WP 5 to the surveillance orbit, and d_{toWP5} is the straight line distance from the UAV to WP 5. Then $Pl_r = d_{toWP5} + d_5$. A whole new Dubins path is not created when the UAV is off the desired path or is not at the needed ground speed, but uses the initial path and adjusts the heading and airspeed accordingly.

Once the UAVs reach the surveillance orbit, we need to find the correct geometry of this orbit to maximize the sensor orientation and FOV. Please refer to Figure 3.13. Assuming a coordinated turn and constant altitude, the lift of the airplane provides both the centripetal acceleration (V_g^2/r) of the orbit and counters gravity. Trigonometry also relates the bank angle, sensor depression angle (γ_d), and altitude(h) to the orbit radius (r). This creates two equations, with two unknowns (r, ϕ)

$$\begin{aligned} \frac{V_g^2}{r} &= g\tan\phi \\ \tan(\gamma_d + \phi) &= \frac{r}{h} \end{aligned} \tag{3.25}$$

If you assume that ϕ is small, then after a little algebra a quadratic equation with respect to r appears. Taking the solution that maximizes r (and minimizes ϕ), the solution to r is

$$r = \frac{1}{2}\left(h\tan\phi + \frac{V_g^2}{g}\tan\phi + \sqrt{\tan^2\phi\left(h + \frac{V_g^2}{g}\right)^2 - 4\left(\frac{hV_g^2}{g}\right)^2}\right) \tag{3.26}$$

This result will be the orbit radius used in the surveillance orbit. A more rigorous treatment of this orbit is provided in [7].

Once the UAVs enter the surveillance orbit, the next feedback scheme tries to maintain the angular spacing of the orbit with respect to the target position. Figure 3.14 illustrates the algorithm. One of the vehicles is designated as the reference vehicle (UAV 1). The orbit is divided into equal pieces with respect to UAV 1, and the i^{th} UAV is assigned a reference angle A_{refi} (in this case A_{ref} is every 90 degrees for 4 UAVs). The commanded ground speed for the i^{th} UAV in this mode is

$$V_{gi}^{c} = \begin{cases} V_{nom} + k_2(A_{refi} - A_i) & |A_{refi} - A_i| > tol \\ V_{nom} & |A_{refi} - A_i| \leq tol \end{cases} \tag{3.27}$$

Note that the commanded speed is centered around a nominal velocity, and that there is a buffer zone around a tolerance. For the case of UAV 3, the commanded velocity would be greater than the nominal since the angle difference $A_{refi} - A_i$ is greater than the tolerance. Conversely, UAV 4 would be commanded a velocity lower than nominal.

In a flight testing situation, the Kestrel Autopilot takes care of certain functions. When given a set of waypoints, the autopilot controls the commanded heading so that the vehicle always heads toward the current WP. The lower level control loops dictate the control surface positions and attitude, so typically there is no need to provide servo or roll/pitch/yaw commands. The commands primarily left to the user are waypoints, loiter points, altitude, and airspeed. As the two feedback equations suggest, airspeed control is a central part of the algorithm.

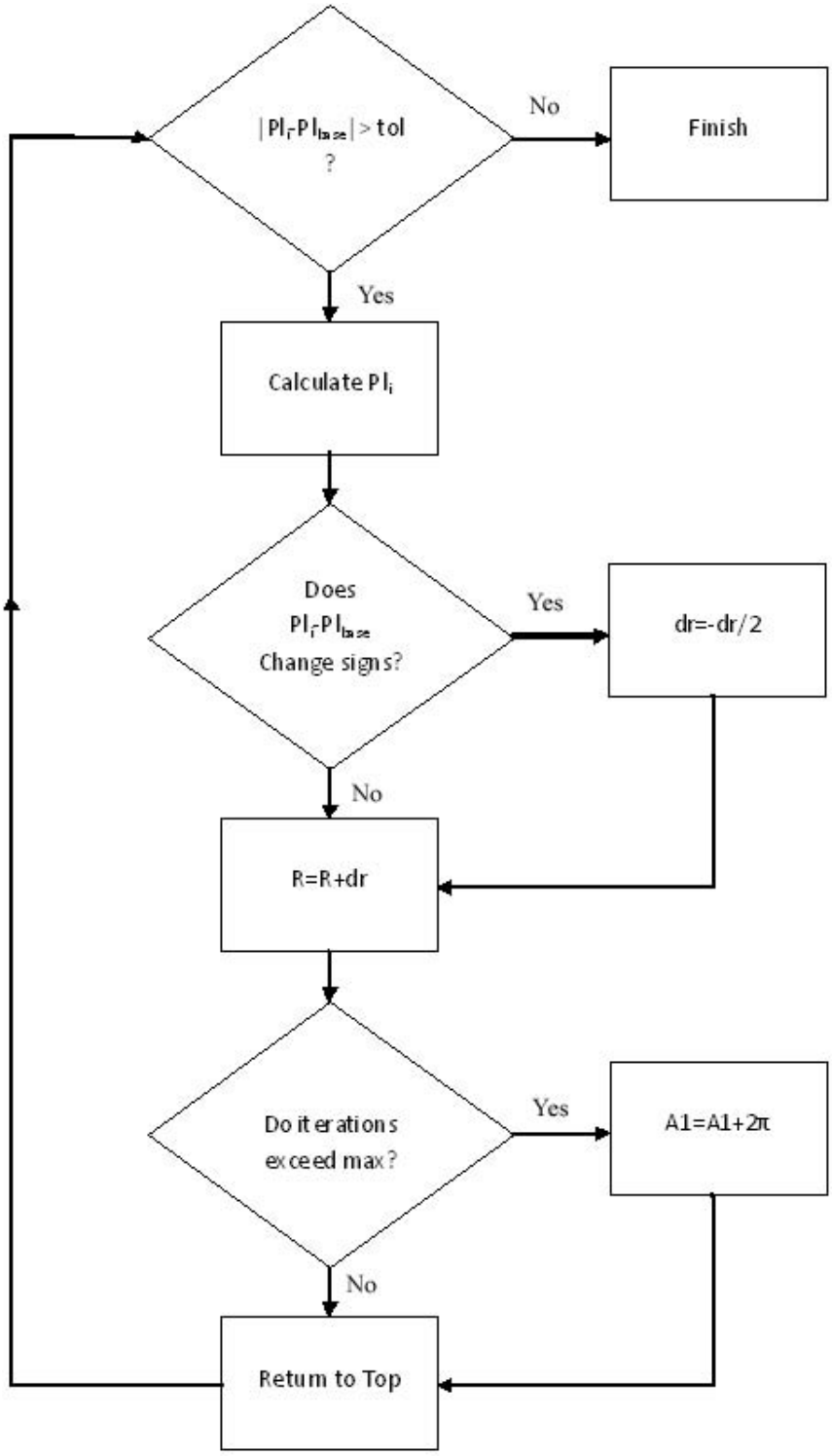

Figure 3.11: Equate Pathlengths Algorithm

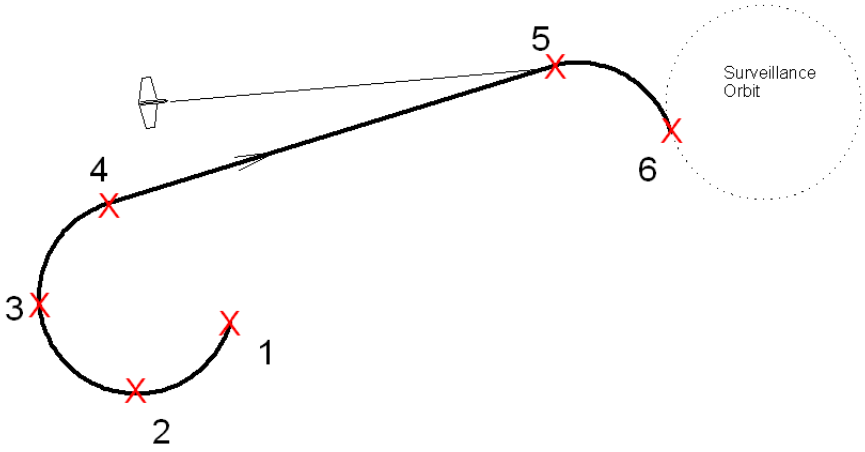

Figure 3.12: Calculating Remaining Path Length

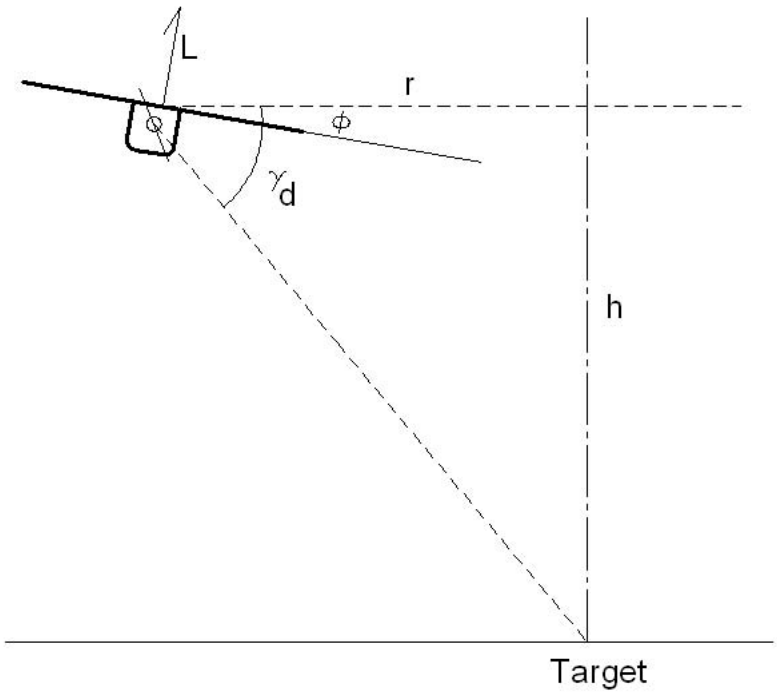

Figure 3.13: Relating Bank Angle to Orbit Radius

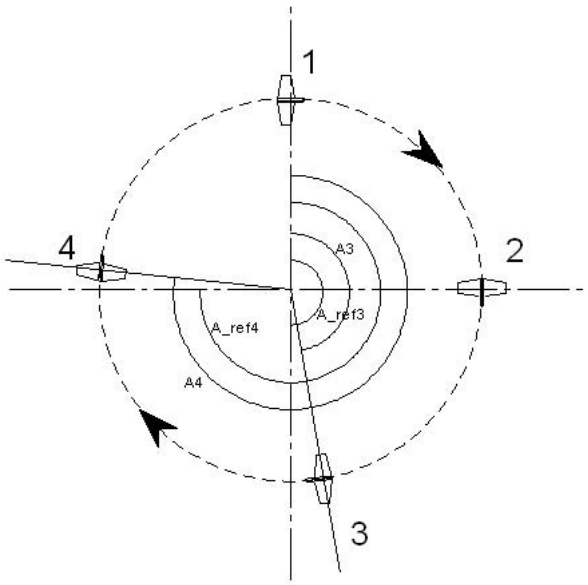

Figure 3.14: Maintaining Orbit Around the Target

3.6.4 Software Overview. The CC algorithm software tries to maximize the higher level capabilities of the Kestrel autopilot. The three basic capabilities used are waypoint sets, loiter points, and airspeed control. As stated above, waypoints provide the capability to follow a defined path. Loiter commands create a circular orbit about a given GPS location at a given altitude. Airspeed control gets to and holds a commanded airspeed. All of these commands (and more) are contained in the PC based VC software that comes with the Kestrel Autopilot [1].

The cooperative control (CC) algorithm will be a program that runs simultaneously with VC. Since VC communication with both the UAVs and external software is TCP/IP based, all commands are network packets sent to the UAVs via VC. Also, all telemetry packets from the UAVs are forwarded from VC to the CC program using network loopback. The CC software is written in C++, but utilizes custom header files from Procerus that enable communication with VC, and also special header files from Mathworks that enable MATLAB routines to be utilized in C++. Using Visual Studio 2005 for development, the software is a Windows based and event driven with a graphical user interface (GUI).

The MATLAB capabilities handled the complex calculations needed to create the final positions and headings in the Dubins paths, to create the waypoint sets and provide airspeed feedback both approaching the surveillance orbit and in the surveillance orbit. The following functions were written as a part of this effort and are used in the software:

- mlfEnterOrbit - Using the target location, nominal airspeed, and desired altitude, this routine calculates the orbit radius based on Equation 3.35 and returns the GPS coordinates and headings of the endpoints of the Dubins paths that are tangent to the surveillance orbit.
- mlfCreateOrbit - Using the current UAV locations and the GPS coordinates/-headings entering the surveillance orbit, this routine returns a set of GPS way-

points with Dubins paths of equal length. The algorithm shown in Figure 3.12 generates the WPs.

- mlfUpdateGrndSpd - Using the current position of the UAV and the current waypoint the UAV is headed to, this routine returns a set of ground speeds for each UAV to all enter the surveillance orbit simultaneously. Equation 3.33 provides the ground speeds.
- mlfAirspeed - Using the results of either mlfUpdateGrndSpd or mlfMaintainOrbit and the current wind estimate, returns a set of commanded airspeeds using Equation 3.32.
- mlfMaintainOrbit - Using the current UAV locations and target location, this routine generates a set of commanded ground speeds using Equation 3.36.

The accompanying CD contains the C++ source code and Appendix B contains the supporting MATLAB routines. Since the CC software is event driven, four events drive this program: pushing the "Calc Final Points" button, pushing the "Upload to VC" button, an arrival of a telemetry packet, and a timer interrupt. Figure 3.15 illustrates the sequence of operations for each event. The first event occurs when the "Calc Final Points" button is pressed. This utilizes the mlfEnterOrbit function to create endpoints for the Dubins paths at the surveillance orbit. The second event, push the "Upload to VC" button, creates WPs based on equal paths, checks them against a predefined box of limits, and begins the process of uploading all WPs with the UploadCommands() subroutine. This routine controls the three distinct modes of the program (See Figure 3.16): Generating and Uploading WPs (Mode 1), arriving simultaneously (Mode 2), and maintaining orbit (Mode 3). The sole purpose is to govern the transition from one mode to another. The reason for the first mode is due to the Kestrel Autopilot. It will not start executing the WP set until the whole set is uploaded. WPs are only uploaded one at a time, and the next WP is not sent until an "Acknowledge"(ACK) packet is received back from the UAV autopilot. Once all WPs are uploaded to all UAVs, the program transitions to mode 2. Airspeed

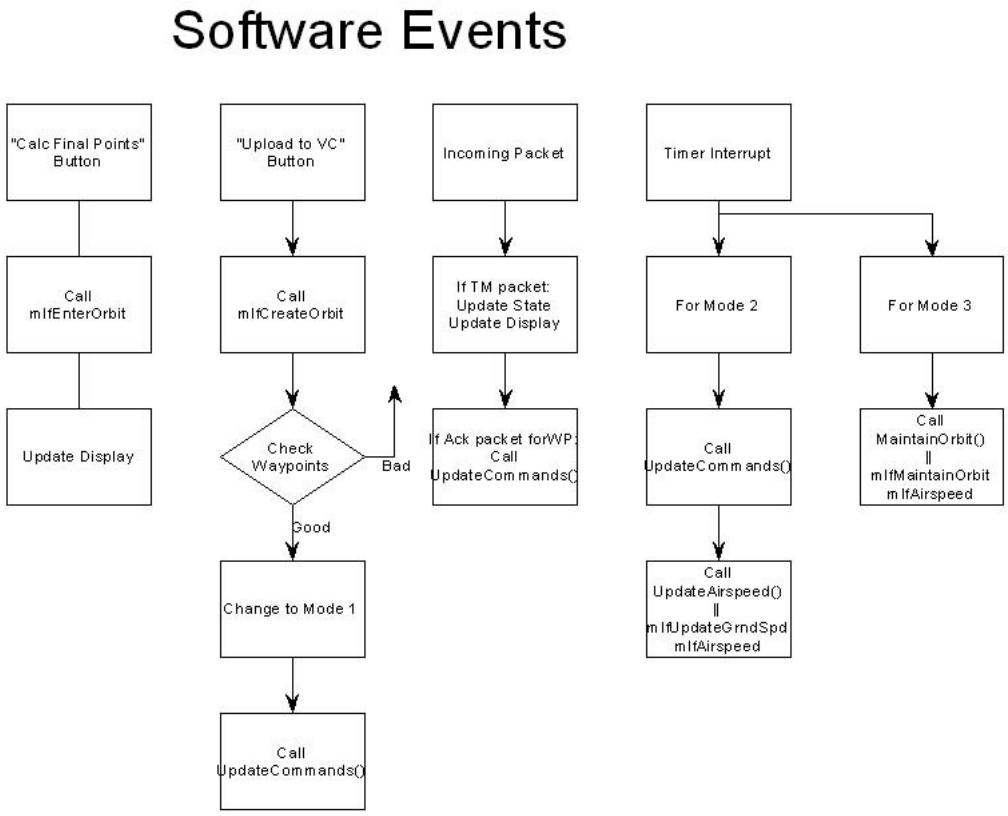

Figure 3.15: Software Events

commands are sent every second to aid simultaneous arrival. Once all UAVs arrive on the surveillance orbit, then the program switches to mode 3 which adjusts airspeed every second based on angular position with respect to the target.

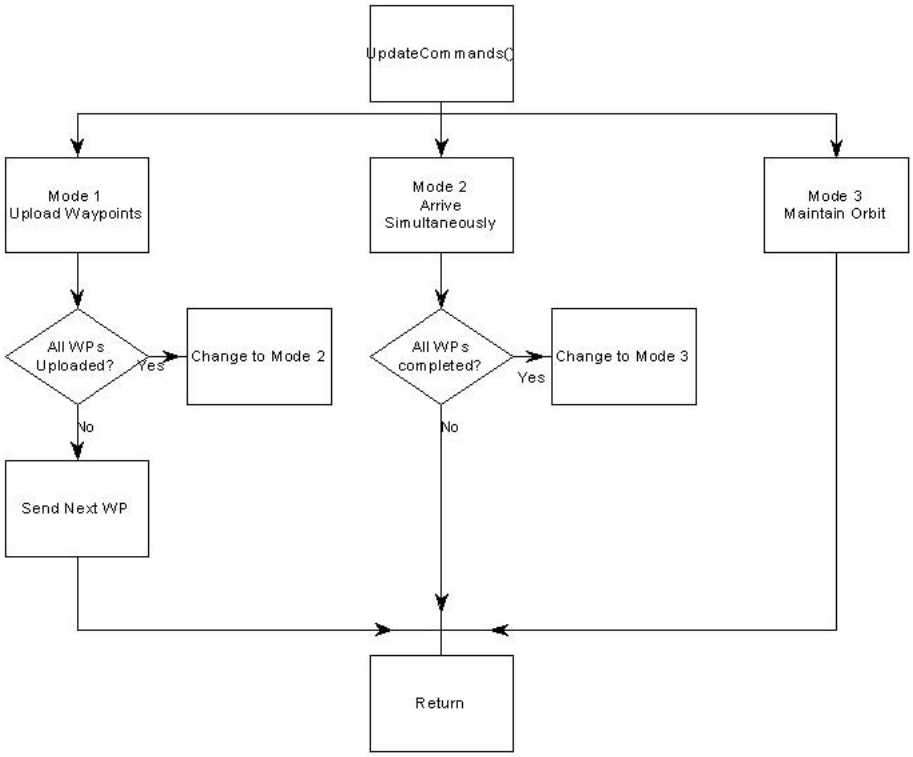

Figure 3.16: Logic Flow of UploadCommands()

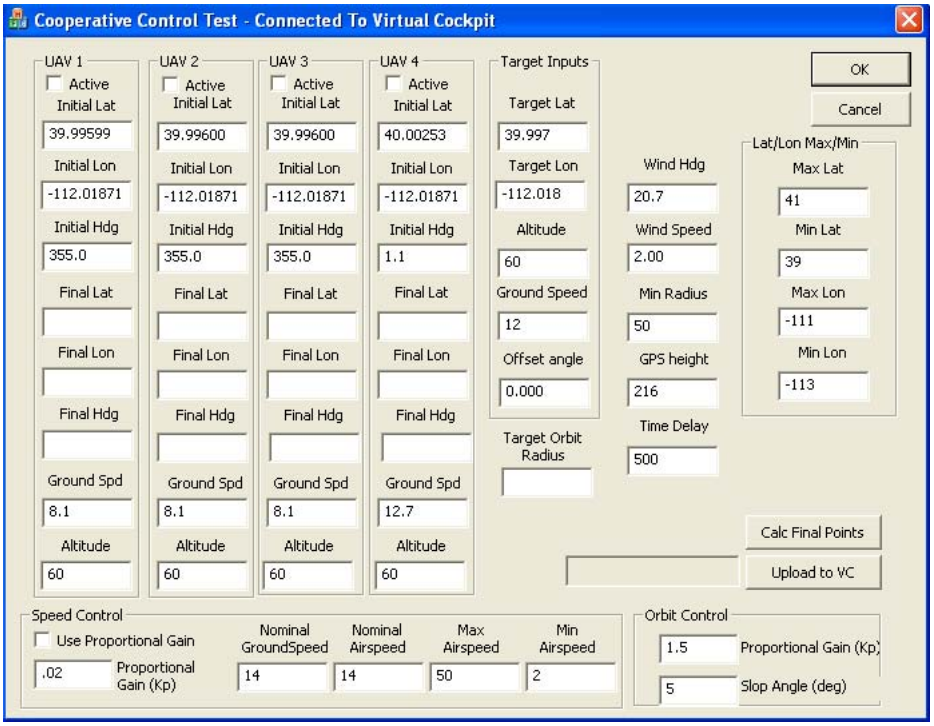

Figure 3.17: CC Algorithm GUI

Referring back to Figure 3.15, the third event(incoming telemetry packet) is key to the success of the CC algorithm. An incoming telemetry packet(TM) causes the program to update the state variables. Also, when the program sees an ACK packet from a waypoint command, then UpdateCommands() can send the next waypoint up. The fourth and final event occurs in modes 2 and 3. Once mode 1 completes, a timer is set for every other second. Based on the state of all UAVs, commands are sent to adjust the airspeed. The only way the program can transition from mode 3 back to mode 1 is to push the "Upload to VC" button. If the user so desires, he/she can enter a new target and the UAVs will begin the algorithm again. A sample screen shot of the GUI is shown in Figure 3.17.

The user must enter certain parameters into the GUI. In the "Target Inputs" group, the program needs target position, surveillance orbit altitude, desired ground speed, and an offset angle that alters the position of first UAV position in the orbit. An offset angle of zero makes the first UAV position due East of the target, so changing this angle rotates this position counter-clockwise about the target. The "Calc Final Points" button uses this target information and populates the final lat/lon/heading for each UAV. The user also inputs the maximum and minimum allowed latitude/-longitude for waypoints in the "Lat/Lon Max/Min" Group. The "Speed Control" group contains the parameters for mode 2 velocity feedback, and the "Orbit control" group in the GUI contains the parameters for mode 3. As telemetry is received, UAV information is automatically updated and also the wind information.

The CD accompanying this thesis contains the C++ source code of the CC software.

3.7 Summary

This chapter covered the approach and analysis of modelling the UAV and creating a CC algorithm for surveillance. The chapter began with an overview of both the BATCAM airframe, the Kestrel Autopilot, and the flight testing setup. The next

section modified a first-order model from Kingston [9] to create a closed-loop model of the BATCAM/autopilot system. Using a Dubins based path, the CC algorithm provides a method to have all UAVs arrive at the surveillance orbit simultaneously, and then maintain the angular spacing during the orbit. Software written in C++ to interface to the VC software utilizes the analytic power of MATLAB to create a real-time solution to implement the CC algorithm. The next chapter will summarize the BATCAM model, in addition to how well this approach worked both in simulation and in flight testing.

IV. Results

4.1 Introduction

This chapter presents the results of modelling and the performance of the algorithm. Using actual flight test data, time constants for the first order responses are extracted and woven into Equation 3.4. Using Simulink, the Cooperative Control algorithm is connected to the model and compared to flight test data under different wind conditions. Finally, the achieved coverage of the target is analyzed for both simulation and flight test results.

4.2 Airframe Model and Performance

The best set of data to extract model parameters occurred on the last flight of testing on September 2, 2008. The time of day was near sunset, and the winds lessened to their lowest point that whole day (< 5 knots). Flight telemetry contained less disturbances than previous flights that day, and was considered the cleanest presentation of control inputs to UAV response.

4.2.1 Closed-Loop Property Results. The model contains three first-order time constants: roll, pitch, and airspeed. Using the time lag between control input and response, a time constant can be extracted. The plots of control input and response are not clean step responses. Figures 4.1 and 4.2 show a small section of the roll and pitch commands, respectively, and the airframe response. Both plots display some periodicity, and also display a phase difference between command and response. These are low level control loops, so all commands originate in the on-board autopilot. This is important because without ground station involvement, the delay between when the command is issued and when it is executed is "small" compared to the period shown in the figures. Assuming negligible delay, from basic control theory the phase lag (Ph) due to a first-order system subject to a periodic forcing function

is [11]

$$\begin{aligned} Ph &= -\arctan \omega T \\ T &= \frac{\tan Ph}{\omega} \\ k &= \frac{1}{T} \end{aligned} \tag{4.1}$$

where ω is the frequency of the input, and T is the time constant of the first order system. The plots contain both Ph and ω, so the k values in the model are available by finding T. ω is calculated finding the time between peaks of either command signal or response signal, and Ph is calculated measuring the time between command peak and response peak using the period of ω. For Roll (Fig 4.1), the period of the

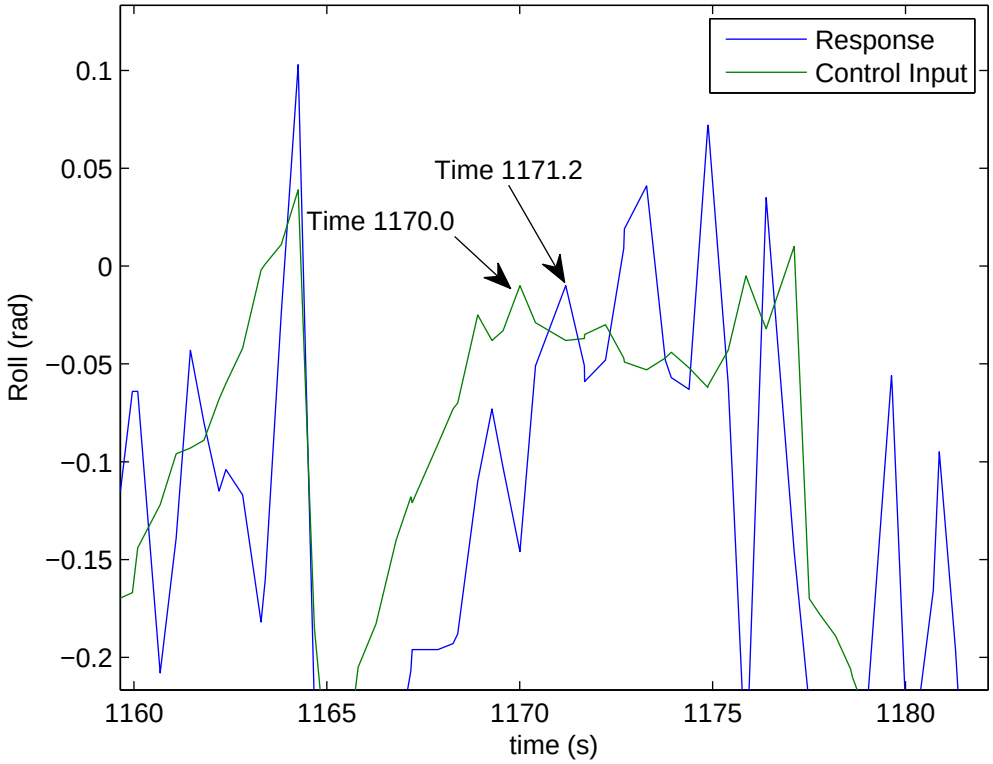

Figure 4.1: Typical Roll Response to Command

commanded signal was 2.3 seconds, and the response peak lagged the command peak by 1.2 seconds. This yielded T=0.434 and k_ϕ=2.3 (from Eqn 3.4). For pitch, Figure 4.2 yielded a command period of 10s and a lag time between command and response of

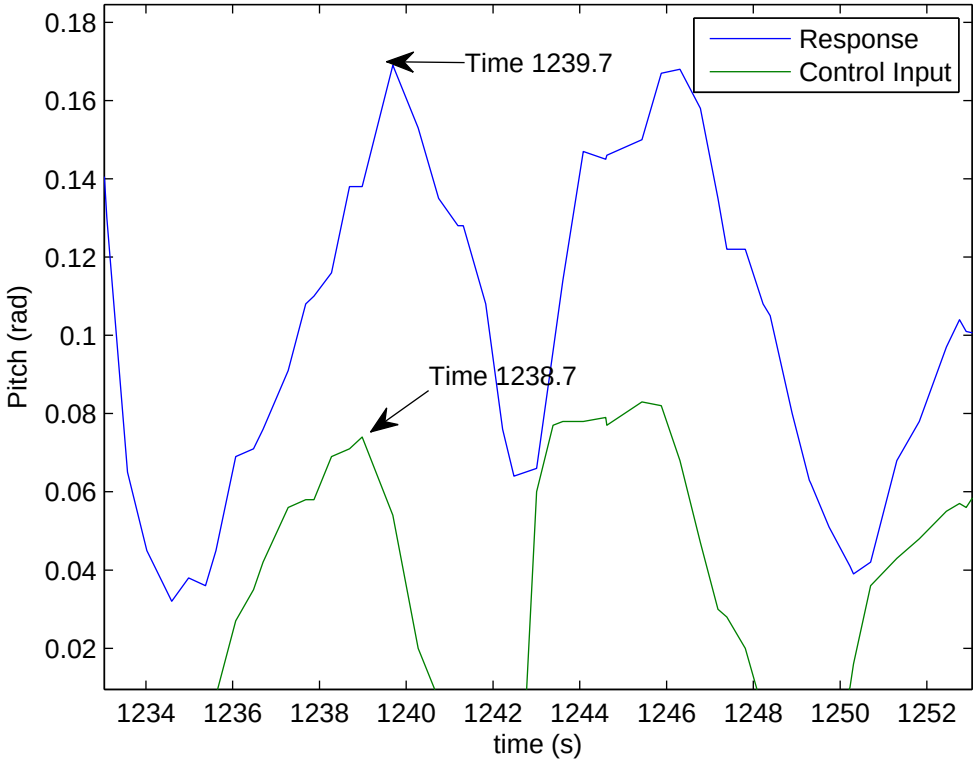

Figure 4.2: Typical Pitch Response to Command

1s, yielding T=1.15 and k_θ=0.86. Airspeed commands were step functions, so finding k_V was more straightforward(Figure 4.3). Using the fact that reaching 95% of a step function takes 3T for a first-order system, the typical rise time to the 95% level took 2.3 seconds yielding k_V=1.3. This was true for both large (as shown) and small steps in airspeed command.

This approach has an inherent error due to the first order assumption. Examining Figure 4.3, the response appears more second order than first order. The largest consequence is that the model will not capture the overshoot, but will reach the commanded airspeed similarly to the actual BATCAM/Kestrel autopilot system.

The model also contains constants (Section 3.4.2) that cannot be extracted from flight data, but are either chosen by the user or are determined through a more subjective tuning process. Maximums and minimums on commanded roll and pitch

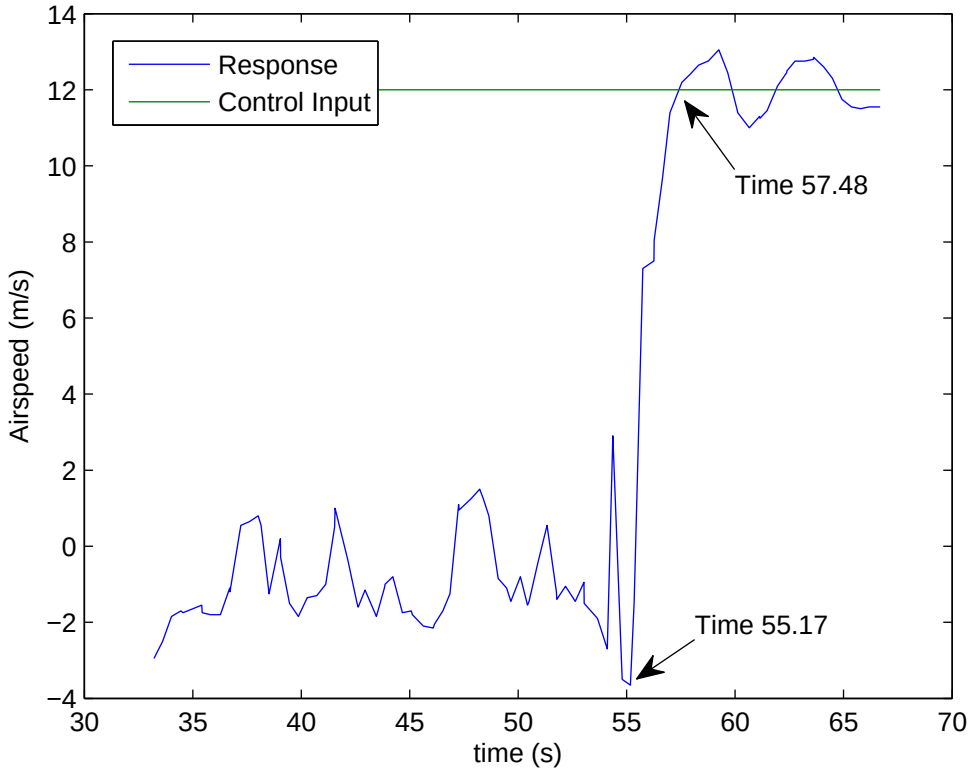

Figure 4.3: Typical Airspeed Response to Command

are set by the user and usually based on common practice. Behavior dealing with tracking the intended heading or altitude are based on user preferences. The following values were chosen based on creating "reasonable" behavior in the model.

The roll and pitch maximums (Eqns 3.7 and 3.8) are set to ±30 degrees. These were the limits used during flight test and were intended to prevent the airframe from reaching an uncontrollable state. One very practical reason for preventing an inversion was the autopilot. At large angles, the gyros in the Kestrel no longer yielded correct attitude measurements, and usually caused the autopilot to act erratically. When angles exceeded 45 degrees, the attitude changes very quickly and verged on instability. 30 degrees seemed a reasonable bound.

An upper limit to airspeed is necessary to create a realistic model. If you examine Eqn 3.4, without an upper limit, the UAV would have infinite control authority

with respect to airspeed. After a survey of flight data, V_{amax}=21.75 m/s appeared to be the maximum value with full throttle without being in a dive condition.

The next values ($k_\chi, k_h, a, \chi_{icpt}$) were determined subjectively. The values were based on desired behavior. For k_χ (see Eqn 3.7), the desired behavior would be to command a bank angle that would not saturate until there was a difference of about 30-40 degrees between commanded heading and actual heading. For this, k_χ=1 accomplished the desire. For k_h (see Eqn 3.8), the desired behavior would be not to saturate the commanded pitch until there was about 10 meters difference between commanded altitude and actual altitude. This led to a value of k_h=0.05. The cross distance parameters were also based on the 10 meter criteria (see Eqns 3.7, 2.3). By letting χ_{icpt} be $\pi/4$, a value of a=0.5 made σ approach $\pi/4$ at y_s=10 meters. 10 meters seemed reasonable because the 30 foot deviations allowed some margin before maximum control effort was applied.

This first-order approximation of the closed-loop BATCAM/autopilot system was chosen for a number of reasons. A good aerodynamic model does not exist for this airframe and the Kestrel autopilot is not fully documented. The BATCAM has some instabilities, and developing a controller from the ground up that both mimicked the autopilot would take some time. With two major missing portions of the closed-loop model (the plant and controller), it seemed reasonable to develop a model based on observed data.

Table 4.1 summarizes the model parameters.

Table 4.1: BATCAM Model Parameters

Parameter	Value	Equation
k_ϕ	2.3	3.4
k_θ	0.865	3.4
k_V	1.3	3.4
ϕ_{max}	$\pi/6$	3.7
θ_{max}	$\pi/6$	3.8
V_{amax}	21.75 m/s	-
k_χ	1	3.7
k_h	0.05	3.8
a	0.5	2.3
χ_{icpt}	$\pi/4$	2.3
k_1	0.12	3.24
k_2	8	3.27

4.3 Algorithm

The two modes of the algorithm (approaching the orbit-Mode 2, maintaining the orbit-Mode 3) have two different performance criteria. The first is measured with cross distance and arrival time, the second is measured with cross distance and angular position. The target visibility Section (4.4) will analyze target visibility based on the UAV's position in the orbit. Simulation and Flight Test results are analyzed using this criteria. In simulation, four wind cases are tested: no wind, 10% of nominal airspeed, 25% of nominal, and 50% of nominal. While orbiting, the algorithm is also tested by removing a UAV, and adding a UAV to the formation. The criteria for this will be time to reconfigure.

4.3.1 Simulation Performance. All simulations utilized Mathwork's Simulink software to propagate the state based on the model in the previous section. To create simulations for each mode, two Simulink models were created. The control algorithms in the first include velocity feedback to aid simultaneous arrival into the orbit. The second model provides airspeed feedback based on angular position in the orbit. Figure 4.4 displays the overall model as it appears in Simulink, and Figure 4.5 shows the individual BATCAM model. Associated MATLAB Code is in Appendix C.

Blocks labelled BC are the model with a controller for roll, pitch, airspeed, desired heading, and waypoint location. The cooperative control controller (CC Controller) takes UAV position, yaw, and wind information then feeds the individual BATCAM models waypoint location, commanded heading, commanded airspeed, and commanded altitude. Most of Matlab function blocks in the simulation are nothing more than taking the data vector and extracting the desired elements.

The settings for the Dryden wind model varied slightly for each wind case. Within the block parameters menu inside Simulink the wingspan was set to the BATCAM wingspan of 21 inches (adjusted to meters). The turbulence probability was set to "light" for the first three wind cases (0, 10, 25%), and set to "moderate" for the 50% case. All other settings were the default settings.

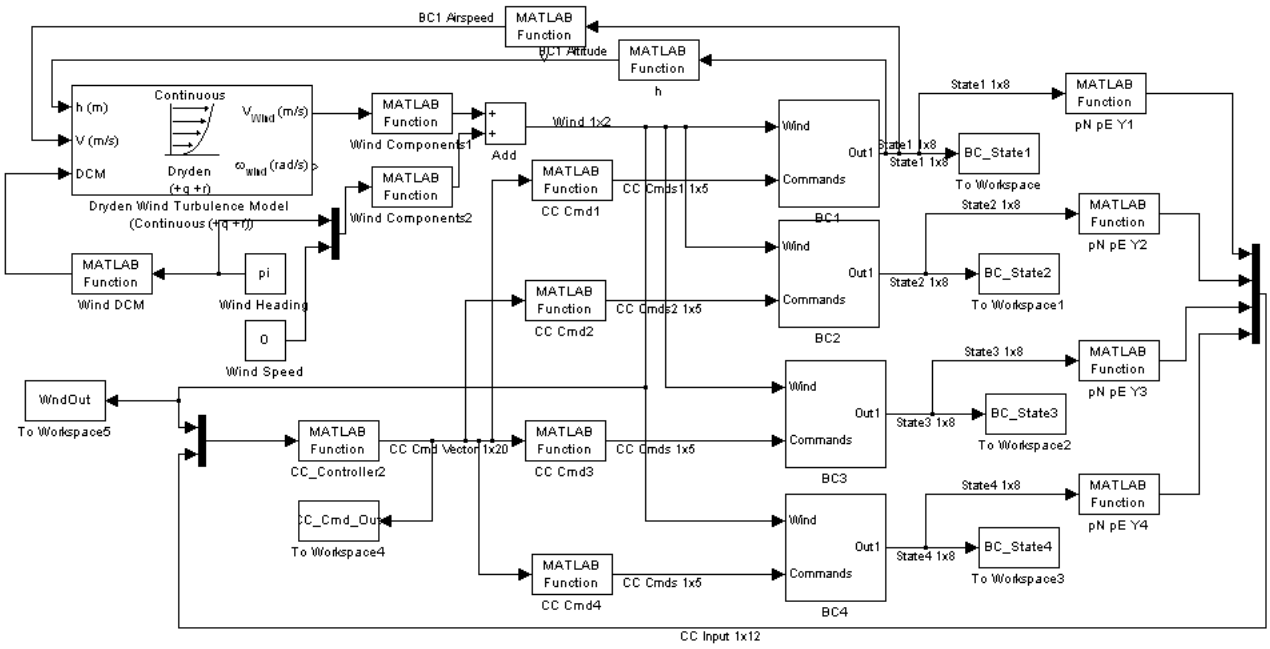

Figure 4.4: Overall Simulink Model

The first simulation takes all 4 UAVs and tests the ability to arrive at the same point (Mode 2) under the 4 wind conditions (Wind Heading 180 deg, Wind Speeds of 0, 1,17, 2.94, 5.88 m/s). These wind speeds are 10%, 25%, and 50% of the cruise speed (11.75 m/s). The Dryden wind model will induce randomness into the wind speed and direction. Table 4.2 displays the initial conditions. The intended 4 paths are straight lines starting at (p_N,p_E)=(300,0)(250,0)(200,0)(0,50) and ending at the target location. All UAVs are placed slightly off their intended paths, and will need to drop to the commanded altitude (see Table 4.2). The simulation ran for 25 seconds to provide enough time to reach the target. The scenario is set up to test the algorithm's ability to converge and stay on the intended path, and also arrive at the target at the same time.

Figures 4.6 and 4.7 present the position of all 4 UAVs under minimum and maximum winds for a simulation run of 25 seconds. Oscillations along the intended path increase, but all UAVs have enough control authority to maintain the intended path. Figures 4.8 and 4.9 plot airspeed vs time for wind speeds of 0 and 5.88 m/s,

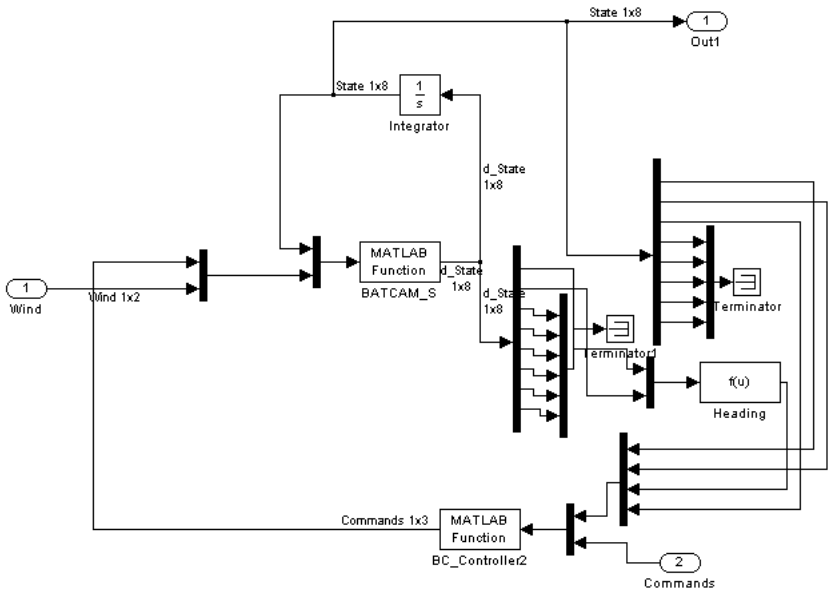

Figure 4.5: Individual BATCAM Simulink Model

Table 4.2: Mode 2 Simulation Initial Conditions

Item	Location (p_N, p_E)	Heading	Initial V_a	Initial Alt
Target	(200,200)	n/a	n/a	n/a
UAV1	(290,-10)	0	11.54	60
UAV2	(220,10)	0	11.54	60
UAV3	(200,0)	0	11.54	60
UAV4	(20,20)	0	11.54	60
Commanded Values	–	Varies	11.75	50

respectively. A quick comparison of the plots illustrates that maintaining the path requires much more control effort than the "no wind" case. Maximum required airspeeds differed by more than 33% with respect to the no wind case. Table 4.3 presents the performance of Mode 2 (Arrive Simultaneously) of the algorithm.

Arrival times for all wind cases did not differ by more than 2 seconds. In the worst case (wind=5.88 m/s), the time difference between first and last was 1.7 seconds. The increasing control effort displayed in the airspeed figures resulted in sooner arrival times. Average cross distances seemed largely dependent on initial conditions, but did slightly increase with increasing wind velocities. The minimum distances to the

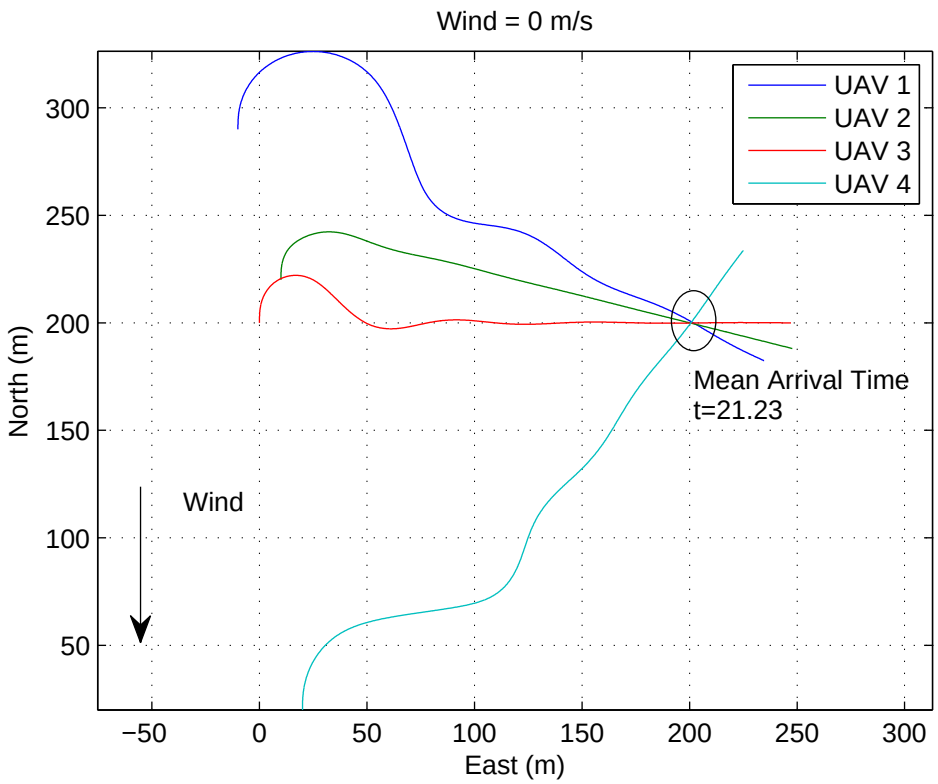

Figure 4.6: Simulation-Mode 2-Wind 0 m/s-UAV position

target d_{min} increased as much as 400% between the 0 wind case and the worst wind case. Depending on the UAV sensor's FOV, this miss distance could affect target visibility.

Overall, given the extracted capabilities of the BATCAM, the simulation predicts reasonable performance of the Mode 2 algorithm. Both arrival times and cross track distances appear favorable to positioning the UAV to view the target. The impact of UAV attitude on target visibility will be addressed in the Sensor Footprint results. It's important to note that this model is reducing a complex device like an autopilot down to a handful of equations. Depending on the actual implementation of the Kestrel control loops present at any given time, behavior could vary greatly. This will be investigated during flight tests.

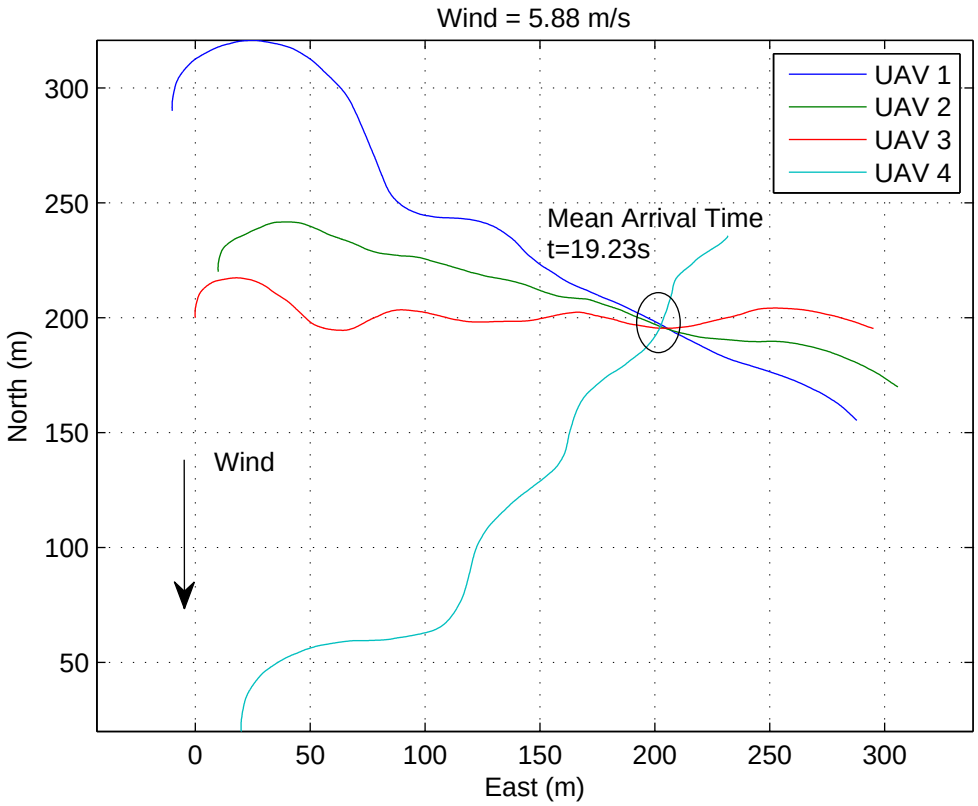

Figure 4.7: Simulation-Mode 2-Wind 5.88 m/s-UAV position

Table 4.3: Velocity Feedback Simulation Results

Item	UAV 1	UAV 2	UAV 3	UAV 4
t_{arrive} W=0m/s	21.60	20.90	21.03	21.37
t_{arrive} W=1.17m/s	21.34	20.70	20.87	21.37
t_{arrive} W=2.94m/s	20.75	20.08	20.30	21.05
t_{arrive} W=5.88m/s	19.45	18.53	18.72	20.24
mean y_s W=0m/s	10.57	2.05	4.54	11.60
mean y_s W=1.17m/s	10.31	2.48	4.59	11.82
mean y_s W=2.94m/s	9.81	2.98	4.55	12.03
mean y_s W=5.88m/s	9.30	3.75	4.82	12.18
d_{min} W=0m/s	0.84	0.11	0.11	0.51
d_{min} W=1.17m/s	0.17	1.05	0.89	0.09
d_{min} W=2.94m/s	0.75	2.11	1.96	0.43
d_{min} W=5.88m/s	1.46	2.64	4.35	3.65

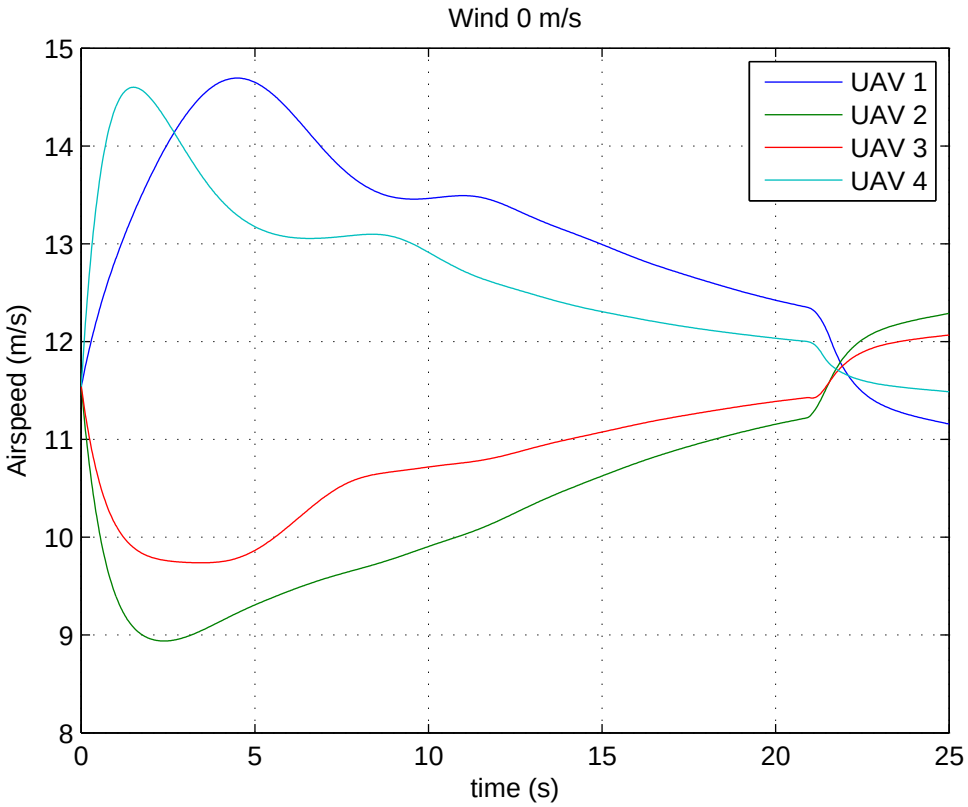

Figure 4.8: Simulation-Mode 2-Wind 0 m/s-UAV Airspeed vs Time

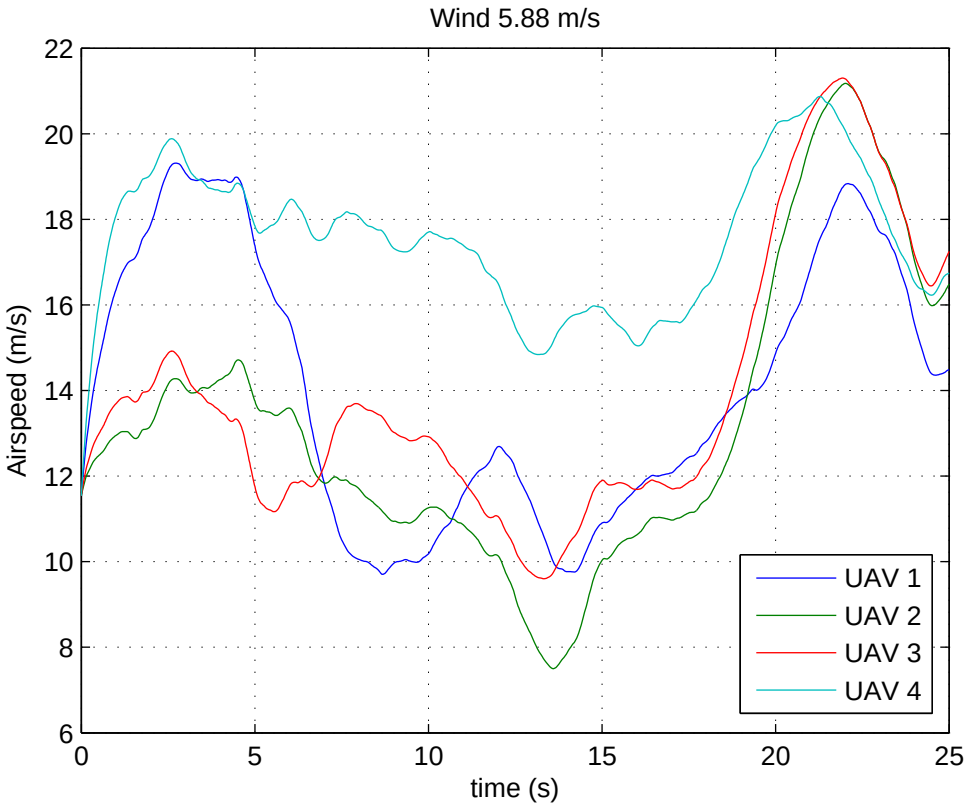

Figure 4.9: Simulation-Mode 2-Wind 5.88 m/s-UAV Airspeed vs Time

Table 4.4: Mode 3 Simulation Initial Conditions

Item	Location (pN, pE)	Heading	Initial V_a	Initial Alt
Target	(0,0)	n/a	n/a	n/a
UAV1	(-10, 65)	10°	11.54	50
UAV2	(70,0)	281°	11.54	50
UAV3	(0,-45)	191°	11.54	50
UAV4	(-90,10)	101°	11.54	50
Commanded Values	–	Varies	11.75	50

The Mode 3 (maintaining orbit) simulation will begin with UAV positions shown in Table 4.4 and again with a wind heading of 180°. The two measures of performance will be cross track distance and angular error. The target will be located at the origin, and the nominal orbit radius of 63 m gives an altitude of 50m (from Eqn 3.35). The UAV at the 6 o'clock position is given a large cross track error outside the desired orbit, and the UAV at the 9 o'clock position is given a cross track error inside the orbit. The simulation is run for 40 seconds so that all UAVs complete one orbit around the target. The remaining portion of this section presents the angular errors introduced from the addition and deletion of a UAV. The Sensor Footprint Section, which follows, presents the target visibility metric derived from the data.

The impact of wind starts to reveal itself during orbiting. Figures 4.10, 4.11, and 4.12 display the UAV positions over the 40 second simulation under 0%, 25%, and 50% of nominal airspeed. The 0 wind plot shows that all UAVs are able to converge to the desired orbit and remain there with little error in crosstrack (y_s). At 25% winds, the crosstrack error grows but all UAVs are able to remain on orbit. At 50% winds, instabilities start to occur. UAV 4 is never able to recover from the initial large initial crosstrack error (UAV 4 starts at the 6 o'clock position). On the upwind side of the orbit (right side) y_s decreases, but on the downwind side the UAV is unable to stay on orbit. Instabilities also occur with UAV 3 (starting at 9 o'clock pos) which also began with a large crosstrack error. Table 4.5 presents the mean angular errors (AE), mean crosstrack errors ($\bar{y_s}$), and settling time (t_s) for all Mode 3 simulation runs.

Settling time is defined as the time required to converge to, and remain at less than ± 5 meters y_s and less than 10° AE. From Table 4.5, none of the UAVs were able to settle at 50% winds. Remember that the desired angular position is based on UAV 1, so there is no angular error for this vehicle.

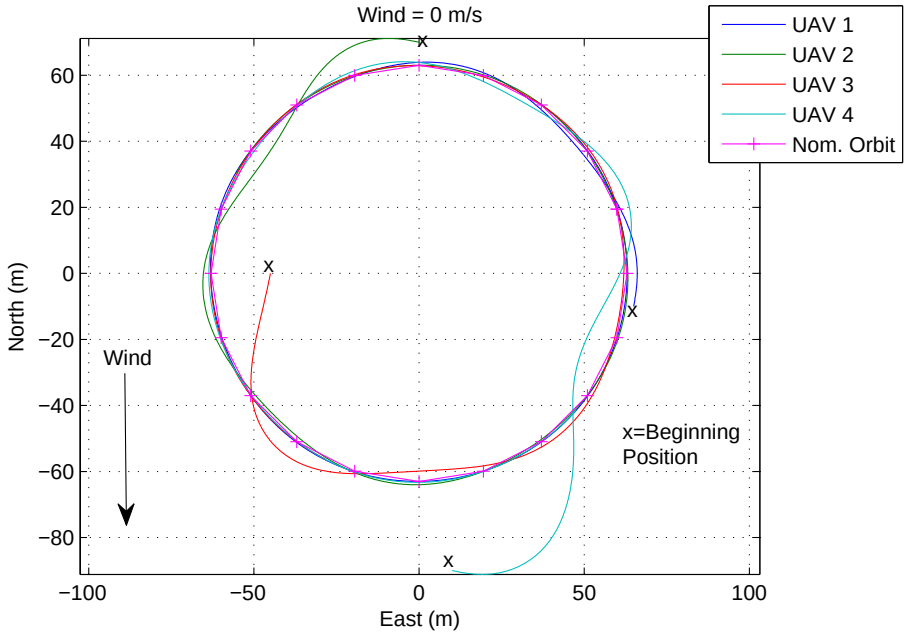

Figure 4.10: Simulation-Mode 3-Wind 0 m/s-UAV position

Figures 4.13, 4.14, 4.15, and 4.16 illustrate the crosstrack and angular errors as functions of time. Much like the data illustrated in the positional plots, the crosstrack plots illustrate the effect of wind on the ability to stay on orbit. UAV 4 error increases dramatically on the downwind side nearing 40 meters error. During that same section the angular error was negative, forcing a command to increase throttle to get back into position. This is a situation where the algorithm can have difficulties. The downwind side of the orbit increases the airspeed, the large negative angular error will also force a further increase in airspeed, resulting in an overshoot that may be unrecoverable. The combination of high winds (50% of V_{nom}) and large crosstrack errors causes instability in this algorithm. Winds up to 25% of V_{nom} appear to be within the capabilities of the BATCAM and the algorithm. If the UAV starts close

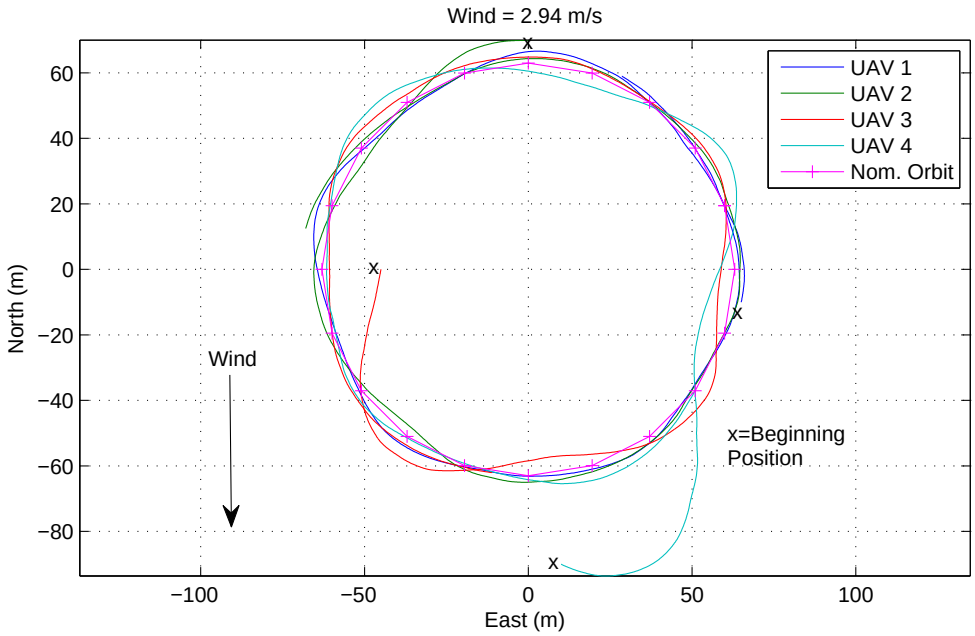

Figure 4.11: Simulation-Mode 3-Wind 2.94 m/s-UAV position

to the desired position (like UAV 1), then the algorithm can keep position within the box defined by t_s most of the orbit at 50% winds. Combining the results of Figures 4.12, and 4.16, the onset of instability appears to occur with AE greater than 25° under these wind conditions.

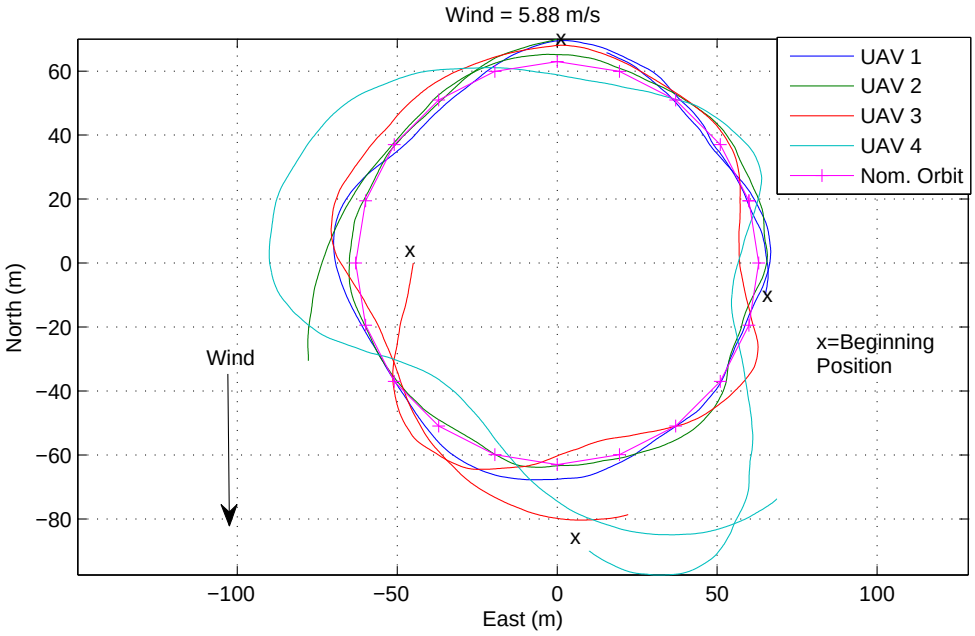

Figure 4.12: Simulation-Mode 3-Wind 5.88 m/s-UAV position

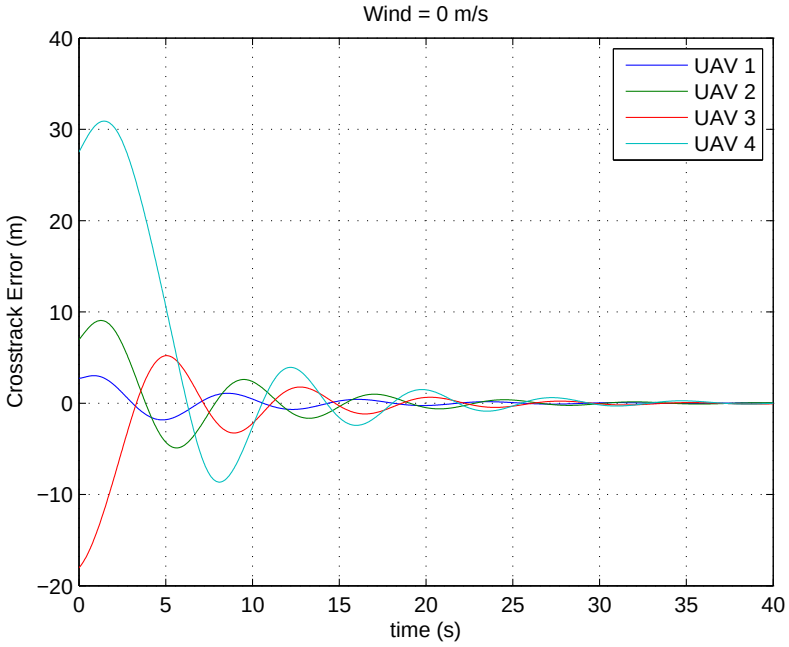

Figure 4.13: Simulation-Mode 3-Wind 0 m/s-Crosstrack Error vs Time

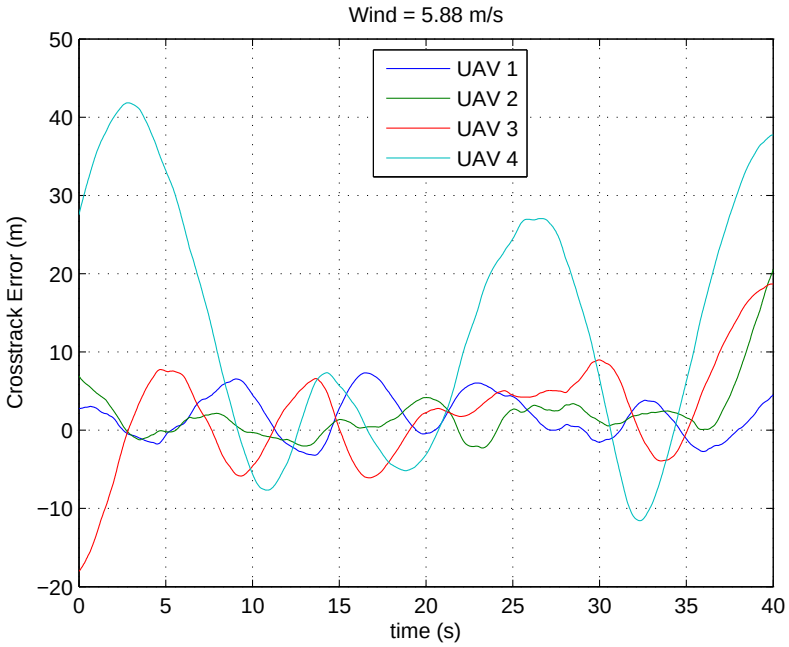

Figure 4.14: Simulation-Mode 3-Wind 5.88 m/s-Crosstrack Error vs Time

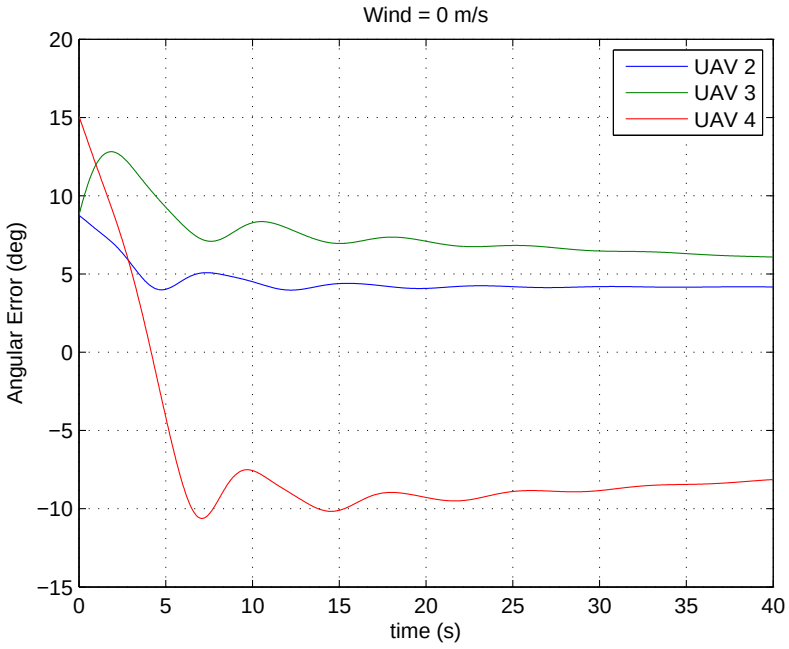

Figure 4.15: Simulation-Mode 3-Wind 0 m/s-Angular Error vs Time

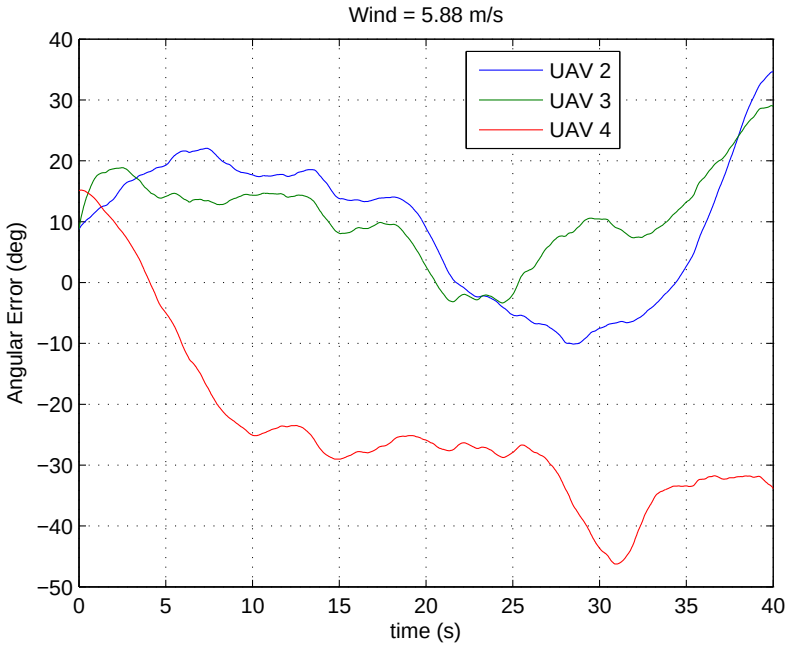

Figure 4.16: Simulation-Mode 3-Wind 5.88 m/s-Angular Error vs Time

Table 4.5: Mode 3 Simulation Results

Item	UAV 1	UAV 2	UAV 3	UAV 4
$\bar{AE}$ W=0m/s	-	4.5°	7.6°	8.7°
$\bar{AE}$ W=1.17m/s	-	4.8°	7.9°	8.8°
$\bar{AE}$ W=2.94m/s	-	6.2°	8.4°	12.9°
$\bar{AE}$ W=5.88m/s	-	12.8°	14.1°	26.0°
mean $\bar{y_s}$(m) W=0m/s	0.47	1.42	1.78	4.86
mean $\bar{y_s}$(m) W=1.17m/s	1.00	1.74	2.43	5.48
mean $\bar{y_s}$(m) W=2.94m/s	1.36	1.81	3.09	7.11
mean $\bar{y_s}$(m) W=5.88m/s	2.49	2.61	5.82	17.30
t_s(s) W=0m/s	0	2.85	5.40	15.36
t_s(s) W=1.17m/s	0	2.52	5.80	21.00
t_s(s) W=2.94m/s	0	8.1	9.6	-
t_s(s) W=5.88m/s	-	-	-	-

Table 4.6: Mode 3 Simulation Initial Conditions-Remove UAV

Item	Location (pN, pE)	Heading	Initial V_a	Initial Alt
Target	(0,0)	n/a	n/a	n/a
UAV1	(-10, 65)	10°	11.54	50
UAV2	(70,0)	281°	11.54	50
UAV3	(0,-53)	191°	11.54	45
Commanded Values	–	Varies	11.75	50

The next item to test in Mode 3 is the removal of a UAV from the formation. For this simulation, the scenario will reduce a 4 UAV orbit to 3 UAVs. The initial conditions are given in Table 4.6, the wind speeds are the same with the same heading of 180°. The same Mode 3 measures of performance are used: crosstrack distance (y_s), angular error (AE), and settling time (adjusting time). The same criteria for t_s will be used as before ($y_s < \pm 5$m, $AE < \pm 10°$). The UAVs are given smaller crosstrack errors, are placed at the 3 o'clock, 12 o'clock, and 9 o'clock positions and will need to re-adjust to 120° apart.

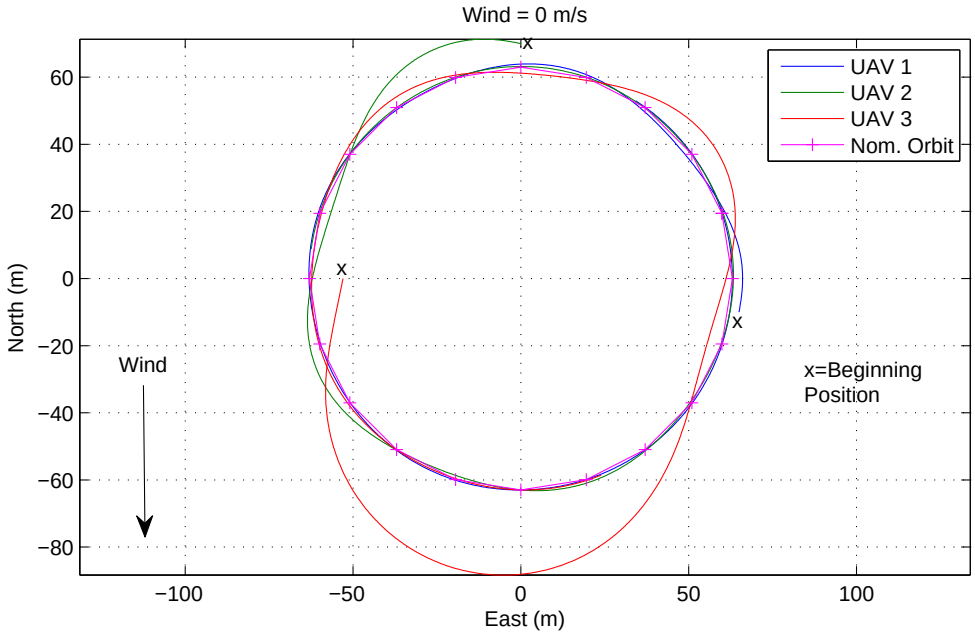

Figure 4.17: Simulation-Mode 3/Remove-Wind 0 m/s-UAV position

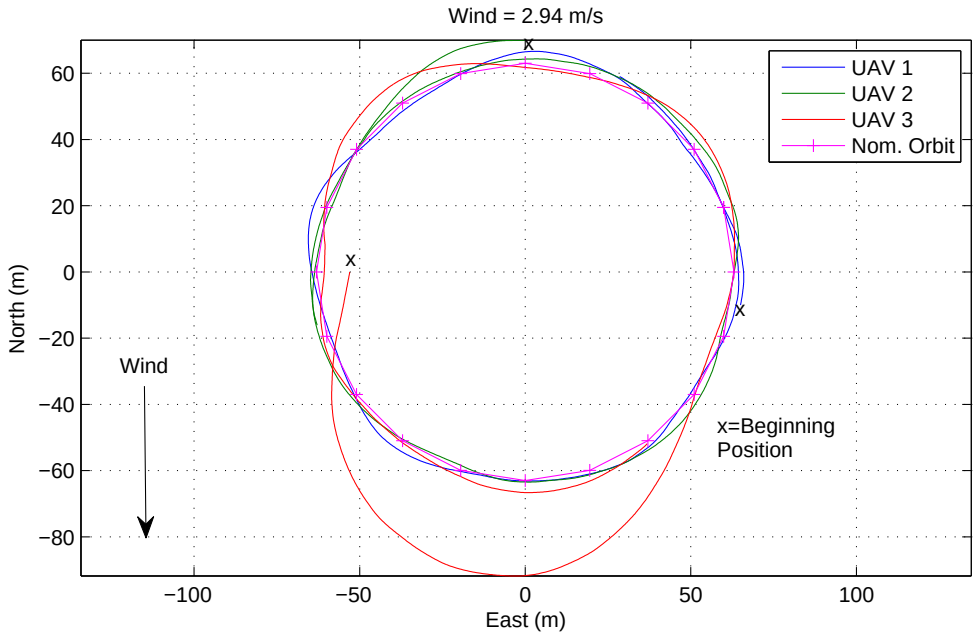

Figure 4.18: Simulation-Mode 3/Remove-Wind 2.94 m/s-UAV position

Figures 4.17, 4.18, and 4.19 displayed much the same results as the previous Mode 3 data: the algorithm can converge at wind speeds up to 25% of V_{nom}. The largest difference is the large overshoot of UAV 3. This is due to the large angular error initially, driving a large increase in airspeed to correct the error. Figure 4.22 provides a clearer picture of the angular error, where the amount of necessary correction at t=0 is slightly over 50 degrees. Also like the previous data set, instabilities creep in at 5.88 m/s (50% of V_{nom})(Fig 4.19) displayed by large crosstrack errors. Again, the downwind side of the orbit exacerbates overcorrections.

The time needed to get the UAV into the correct position is shown in Table 4.7. The zero wind case sets the bench mark where UAVs 2 and 3 are in position by t=8.6 and t=22.1, respectively. For UAV 2, the time to get into position actually decreases with increasing wind. The geometry of the problem aids this. With UAV 2 starting at the 12 o'clock position, it immediately starts turning downwind, so the wind actually helps the UAV get into the desired position. From Figure 4.23, it actually is in the desired position in 2.9 seconds and remains in position until about

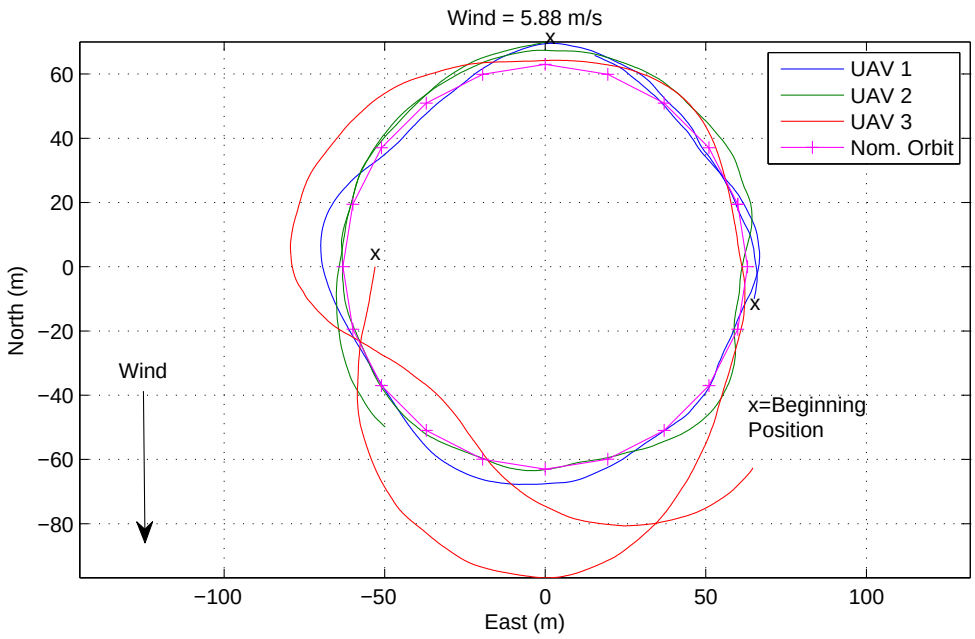

Figure 4.19: Simulation-Mode 3/Remove-Wind 5.88 m/s-UAV position

t=19s when UAV 1 (the reference) starts down the downwind side of the orbit. From an angular perspective, UAV 3 does not perform too bad (Fig 4.23). It reaches the desired position at t=25.5s and remains within 10 degrees of desired through the end of the simulation. Unfortunately at this time, the crosstrack error starts to exceed 25m (Figure 4.21). Both this data set and the previous display that the BATCAM does not have enough control authority to remain in position when winds are at the 5.88 m/s using the Dryden wind model.

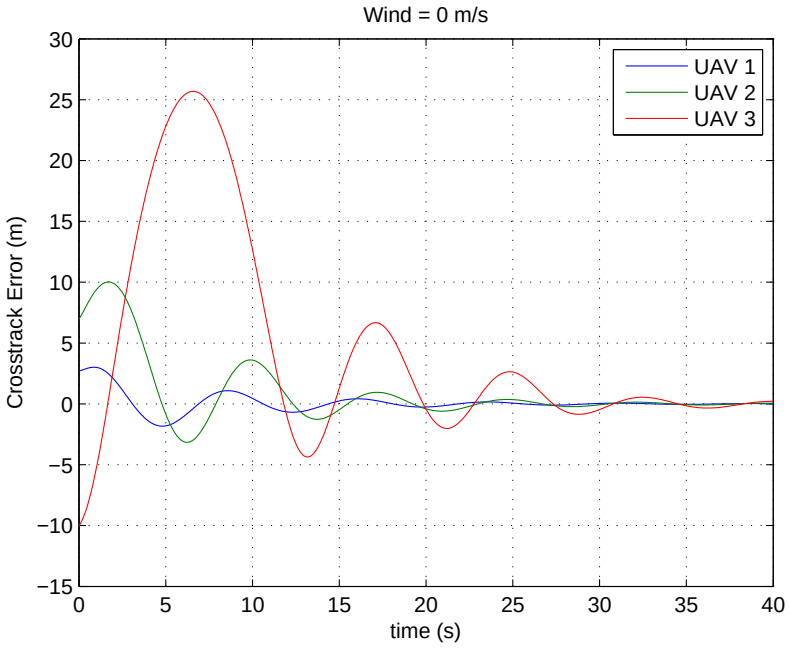

Figure 4.20: Simulation-Mode 3/Remove-Wind 0 m/s-Crosstrack Error vs Time

Table 4.7: Mode 3 Simulation Results-Remove UAV

Item	UAV 1	UAV 2	UAV 3
mean $\bar{y_s}$(m) W=0m/s	0.47	1.57	5.66
mean $\bar{y_s}$(m) W=1.17m/s	1.01	1.75	6.03
mean $\bar{y_s}$(m) W=2.94m/s	1.36	1.70	6.70
mean $\bar{y_s}$(m) W=5.88m/s	2.49	2.60	13.25
t_s(s) W=0m/s	-	8.6	22.1
t_s(s) W=1.17m/s	-	7.9	21.6
t_s(s) W=2.94m/s	-	6.0	25.5
t_s(s) W=5.88m/s	-	-	-

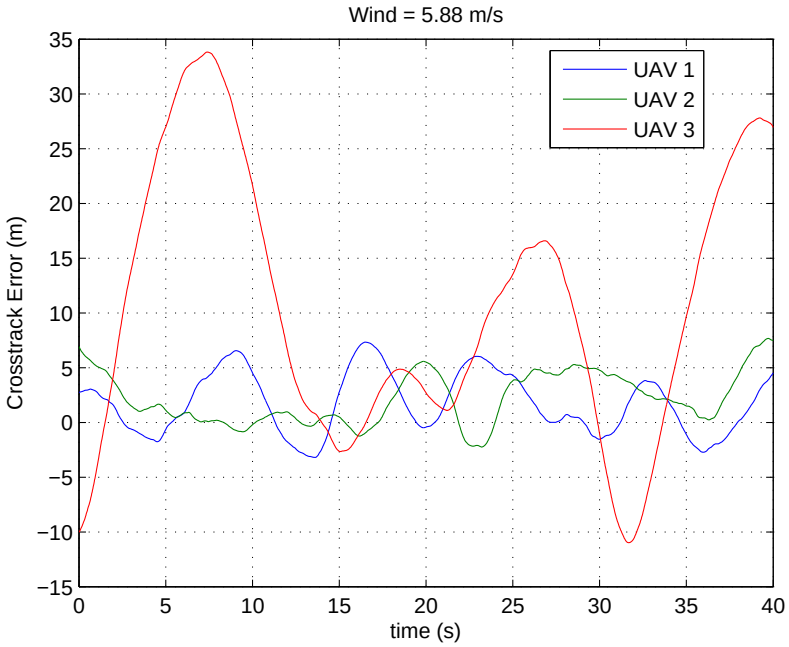

Figure 4.21: Simulation-Mode 3/Remove-Wind 5.88 m/s-Crosstrack Error vs Time

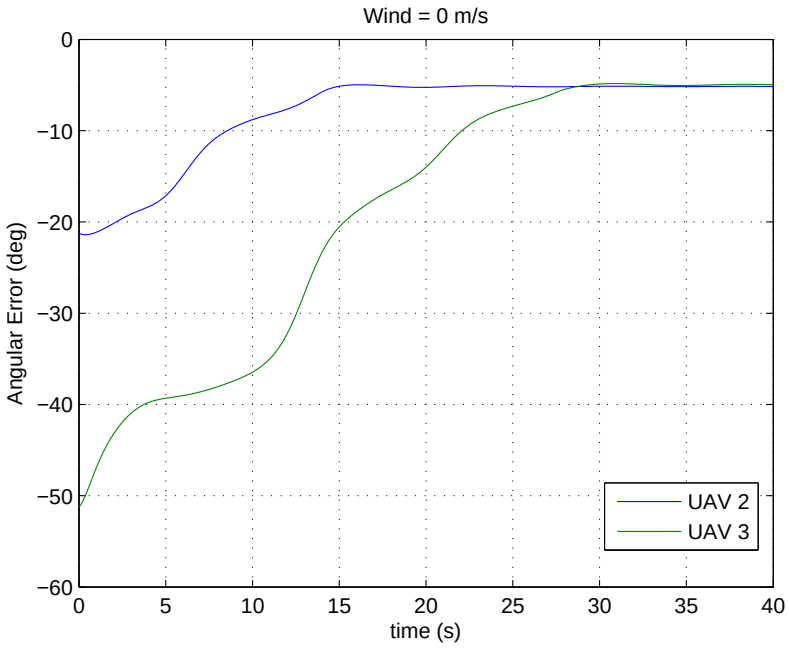

Figure 4.22: Simulation-Mode 3/Remove-Wind 0 m/s-Angular Error vs Time

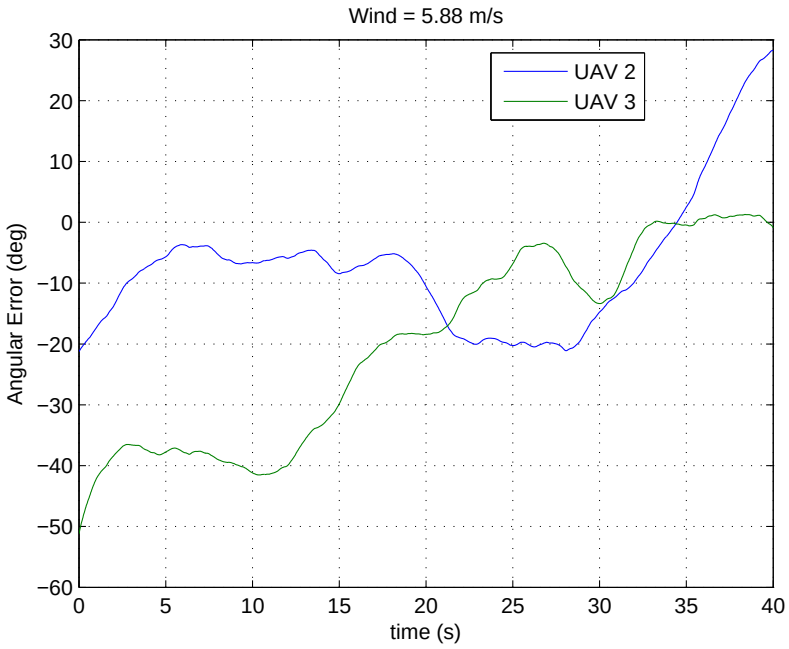

Figure 4.23: Simulation-Mode 3/Remove-Wind 5.88 m/s-Angular Error vs Time

Table 4.8: Mode 3 Simulation Initial Conditions-Add UAV

Item	Location (pN, pE)	Heading	Initial V_a	Initial Alt
Target	(0,0)	n/a	n/a	n/a
UAV1	(-10, 65)	10°	11.54	50
UAV2	(57,-32)	281°	11.54	50
UAV3	(-51,-32)	191°	11.54	45
UAV4	(-60,-20)	135°	11.54	50
Commanded Values	–	Varies	11.75	50

The final simulation for this section involves inserting 1 UAV into a 3 UAV formation and observing the time to adjust into an equally spaced 4 UAV orbit. Initial conditions are shown in Table 4.8, UAVs are placed at 3, 11, and two at 7 o'clock positions. This configuration forces two of the UAVs to slow down to approach the 9 and 12 o'clock positions(with respect to UAV1 at 3 o'clock), and the final UAV to speed up to reach the 6 o'clock position. The wind conditions will be the same as before, and small errors in both cross track and heading are built in. Measures of performance will again be mean crosstrack error y_s, and time to adjust t_s, with the same settling time criteria ($y_s < \pm 5$m, $AE < \pm 10°$). Angular error of UAVs 2, 3, and 4 will be presented as a function of time. The simulation will run for 40 seconds.

Figures 4.24, 4.25, and 4.26 display the UAV positions for the simulation. Typical for past simulations, large overshoots of the orbit do not begin until the last wind case ($|W|$=5.88 m/s). UAV 4 is commanded to increase airspeed to get to the desired position, consequently creating an overshoot at the bottom of the orbit in all three Figures. According to Table 4.9 this UAV does reach and hold the desired location for the first two wind cases, but is unable for the last. In the worst case, large corrections in heading due to large crosstrack errors again lead to unstable behavior. For the other three UAVs, crosstrack error remains fairly tight for all wind cases, with mean values under 6m.

From Table 4.9, the adjustment times remained very close to each other for UAVs 2 and 3 for the first three wind cases, but UAV 4 displayed increasing t_s with

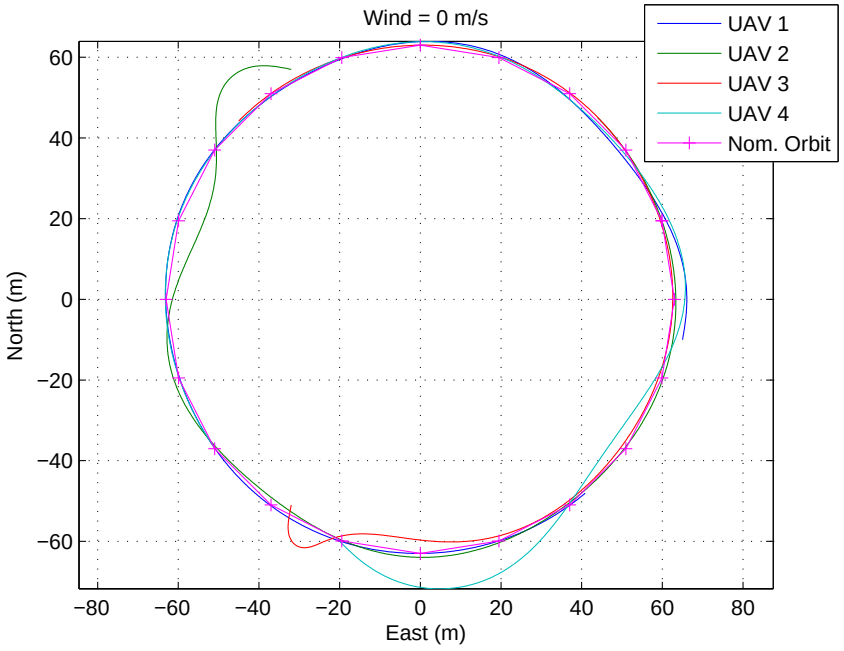

Figure 4.24: Simulation-Mode 3/Add-Wind 0 m/s-UAV position

increasing wind. This case had the largest initial difference in angular position with UAV 3 approximately 66° ahead of the desired position. This large difference did not lead to increasing adjustment times with worsening winds (except for the worst case). It appears that being ahead of the desired position is more favorable than being behind. Increasing airspeed tends to lead to larger overshoots and instability. None of the UAVs were able to hold their position within the designated box in the worst wind case.

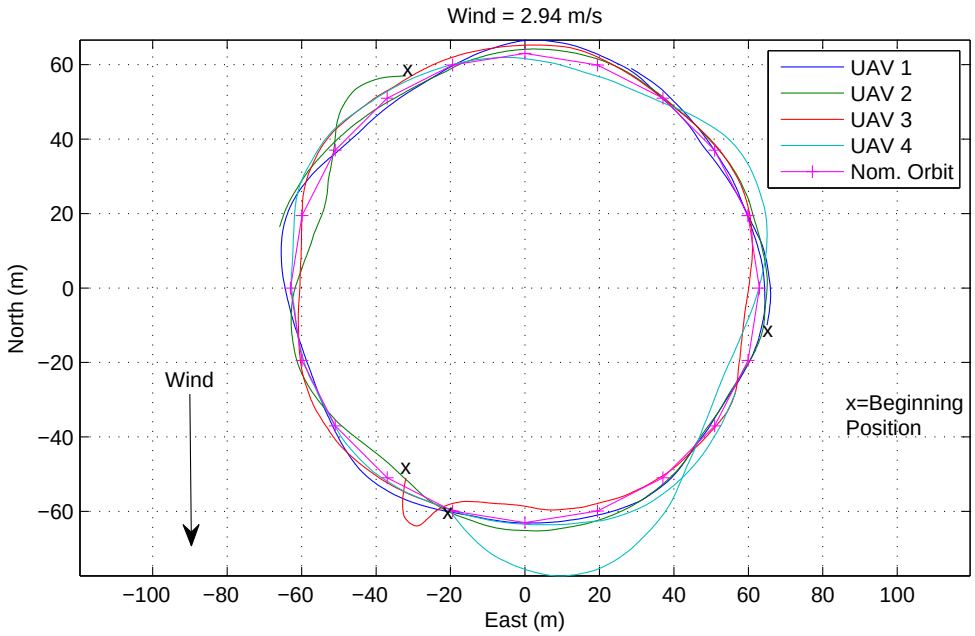

Figure 4.25: Simulation-Mode 3/Add-Wind 2.94 m/s-UAV position

Table 4.9: Mode 3 Simulation Results-Add UAV

Item	UAV 1	UAV 2	UAV 3	UAV 4
mean $\bar{y_s}$(m) W=0m/s	0.61	1.89	1.27	1.82
mean $\bar{y_s}$(m) W=1.17m/s	1.18	2.24	1.72	2.41
mean $\bar{y_s}$(m) W=2.94m/s	1.56	2.17	2.31	3.91
mean $\bar{y_s}$(m) W=5.88m/s	2.70	2.52	5.75	13.69
t_s(s) W=0m/s	-	9.09	14.16	5.96
t_s(s) W=1.17m/s	-	9.22	14.38	7.18
t_s(s) W=2.94m/s	-	9.80	14.29	13.13
t_s(s) W=5.88m/s	-	-	-	-

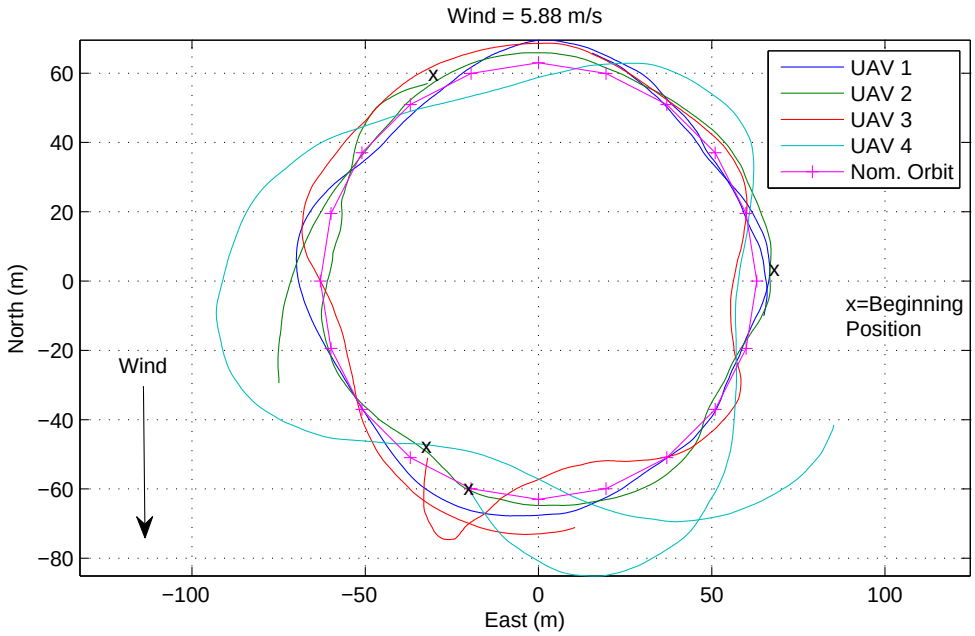

Figure 4.26: Simulation-Mode 3/Add-Wind 5.88 m/s-UAV position

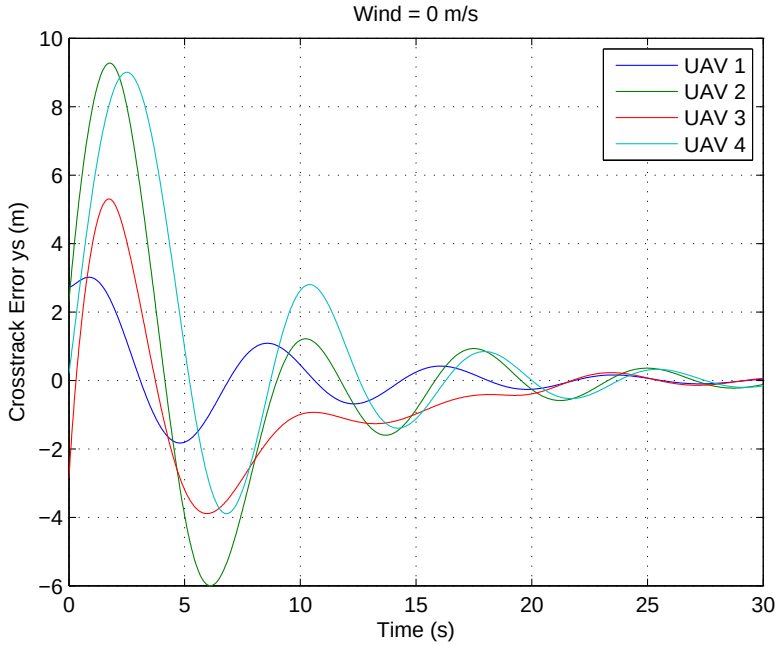

Figure 4.27: Simulation-Mode 3/Add-Wind 0 m/s-Crosstrack Error vs Time

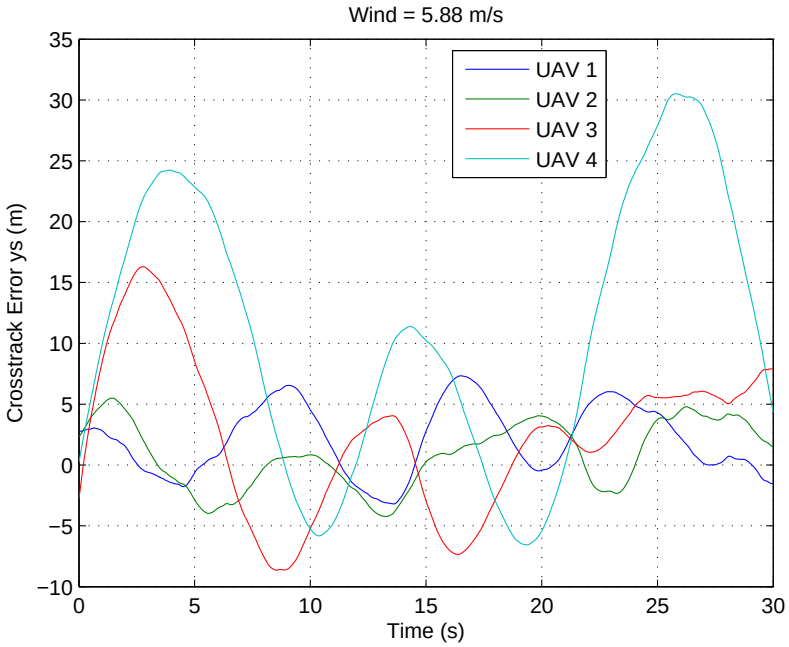

Figure 4.28: Simulation-Mode 3/Add-Wind 5.88 m/s-Crosstrack Error vs Time

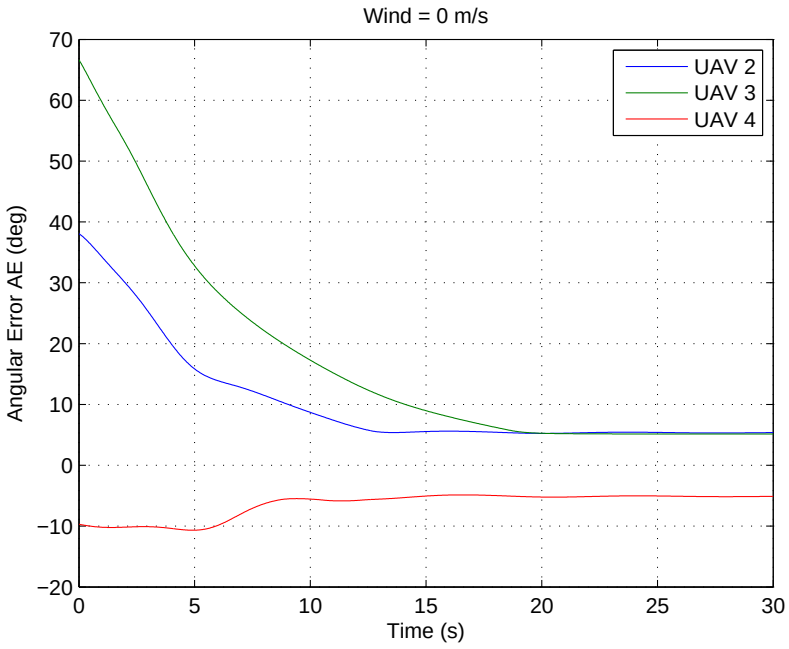

Figure 4.29: Simulation-Mode 3/Add-Wind 0 m/s-Angular Error vs Time

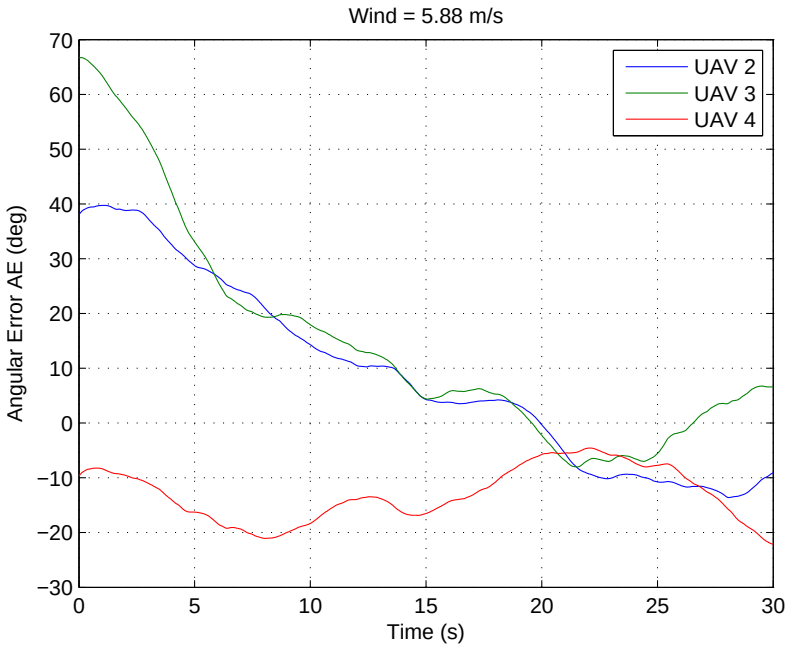

Figure 4.30: Simulation-Mode 3/Add-Wind 5.88 m/s-Angular Error vs Time

For all simulation results, UAVs were able to approach the orbit for all wind cases, but were unable to maintain orbit when winds were 50% of the nominal airspeed. While orbiting, commanding large airspeeds under windy conditions led to larger crosstrack errors and instability. The largest crosstrack errors occurred on the downwind side of the orbit and led to instability if the UAV had large initial crosstrack position. The algorithm is stable and can maintain orbits when winds are only 25% of V_{nom}, even with large initial errors. Adjustment times for a 63m radius orbit when adding and deleting UAVs was on the order of 14-21 seconds with initial angular errors of up to 66 degrees.

4.3.2 Flight Test Performance. This section presents the position accuracy of the algorithm under flight test conditions. As in the previous section, the measures of performance are crosstrack error (Mode 2 and Mode 3), arrival time (Mode 2), and angular error (Mode 3). The algorithm was tested two times in flight test. The first was unsuccessful because airspeed commands from the algorithm conflicted with the altitude control loop in the autopilot. Consequently, algorithm airspeed commands were ignored due to the tight altitude hold parameters. The second flight test successfully tested two BATCAMs with the algorithm. The data presented in this section is from the second flight.

Conditions were moderate during this test. Winds were out of the East South East (Heading of 300°) with speeds of 3.0 to 3.6 m/s. These winds were 30.6% of the steady level flight speed of the BATCAM. Temperature varied from 40 to 50 degrees F, skies were overcast, and visibility was approximately 1 mile due to haze.

Many measures were taken to ensure safety and positive control of all vehicles. UAVs were altitude separated by 100 feet (30m) as a collision avoidance measure. Also the CC algorithm software was outfitted with a safety measure that verified that Latitude/Longitude coordinates for each WP did not exceed the immediate area. Failsafe rally points were established so that the BATCAM would loiter a predefined point if

communication with the ground station was lost. The UAVs were also commanded to begin loitering if the GPS solution was lost.

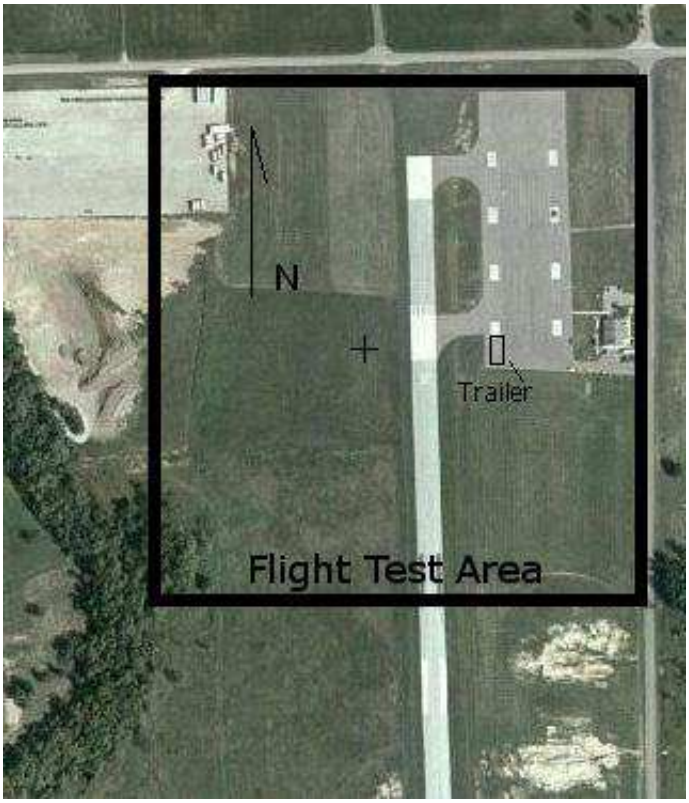

Figure 4.31: Flight Test Location at Camp Atterbury

UAVs were flown above the North end of the runway keeping all UAVs within vision (See Figure 4.31). The ground station/trailer was parked at the Southwest corner of the pad adjacent to the airfield operations building. The distance from ground station to the UAV rarely exceeded 1/4 mile. The GPS limits prevented overflight of the road to the north and east, the treeline to the west, and the line approximately 1/4 mile south of the trailer.

Six separate tests of the algorithm were conducted. Two tests apiece for three different airspeeds: 10 m/s, 11.75 m/s, and 14 m/s. The first flight test only used 1 nominal airspeed to center commands around. Analysis from this first flight test day implied that the "slow down" commands pushed the limits of the BATCAM's ability

to maintain lift. The three separate airspeeds for the second flight test day tried to quantify if algorithm performance was affected by nominal airspeed.

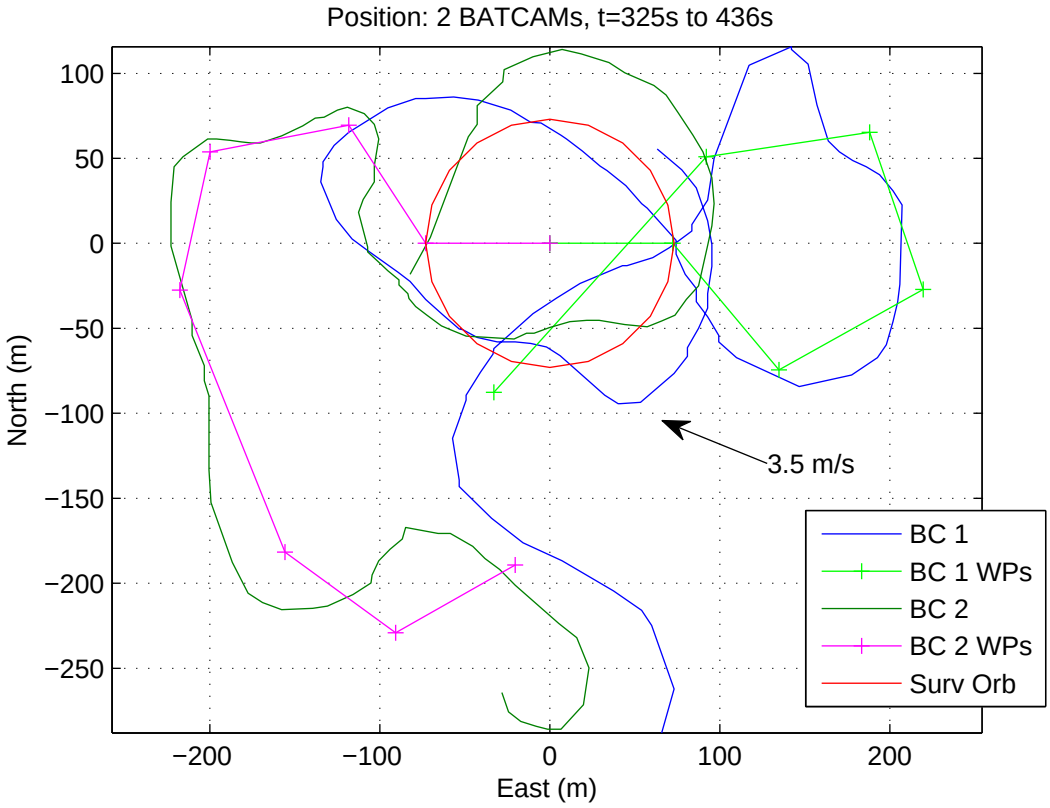

Figure 4.32: Flight Testing - Two BATCAM Test 1 - Position

Figures 4.32 and 4.33 present the position plots for the first two tests with two BATCAMs (BCs). Figures 4.34 and 4.40 illustrate typical orbit position plots during flight test. Table 4.10 presents the average measures of performance for Modes 2 and 3. The six tests were averaged into three sets of results based upon nominal airspeed. As a comparison, Table 4.11 displays the results from the first flight test.

As stated in Chapter III, the crosstrack error control loop (commanded heading) resides on the Kestrel autopilot, not in the CC algorithm. Once the algorithm provides the autopilot with a set of waypoints for all UAVs, the only commands the algorithm issues are airspeed commands. Internal to the autopilot is the method to keep the UAV

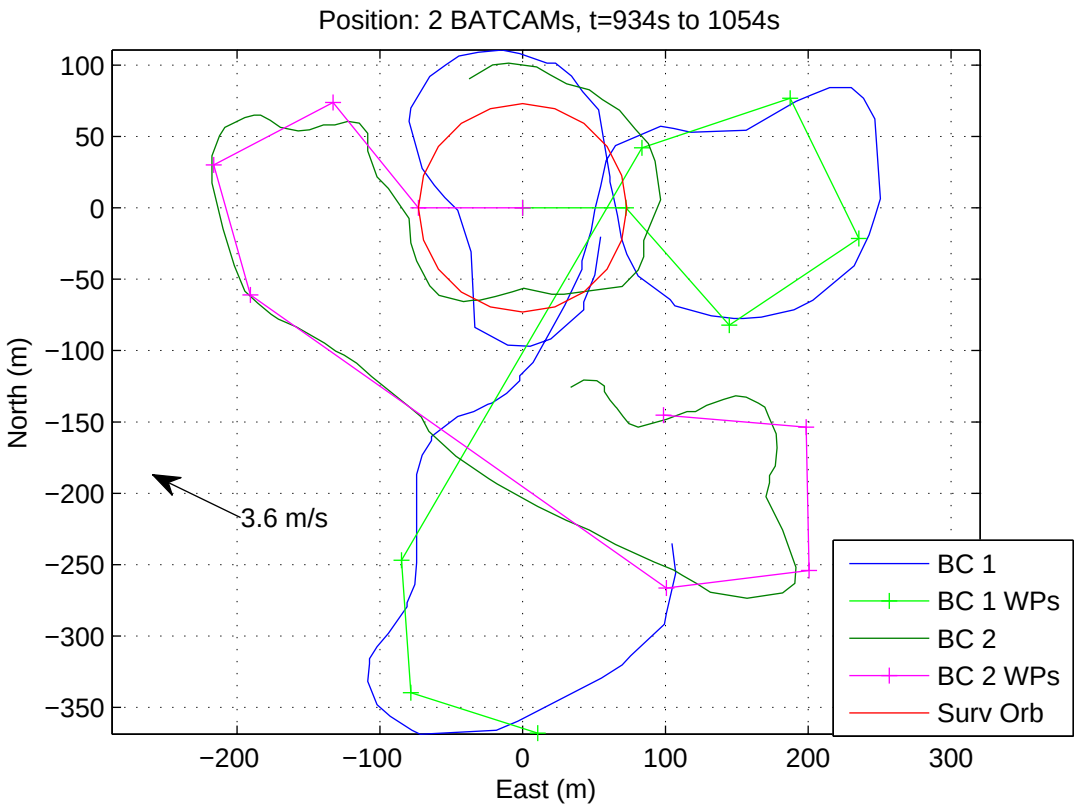

Figure 4.33: Flight Testing - Two BATCAM Test 2 (Mode 2 and 3) - Position

close to the intended path. Unfortunately a detailed description of the commanded heading method was not provided by Procerus for the Kestrel.

Due to the way the Kestrel Autopilot communicates with the ground station, it took approximately 7 to 9 seconds for the VC software to upload the waypoints to the BCs. Consequently, the first waypoint often appears "misplaced" since the CC algorithm creates a path off the most recent received position. During the delay, the UAV can travel as far as 100 meters. This is due to the wireless TCP/IP communication scheme. Each WP is a packet, so if the algorithm issues 7 WPs for each UAV, the communication must receive an "acknowledge" back from the autopilot before sending the next WP. Also the autopilot will not start executing the flight plan until the complete plan is received.

The Mode 2 results for the two BATCAM set differed from simulation by 200 to 300%. In the simulation, wind results for 25% of V_{nom} as a comparison, arrival times for all UAVs only differed by 1.6 seconds (Table 4.3). In flight test, arrival times differed by as much as 6 seconds. Comparing the arrival time differences for the three V_{nom}'s, the ability of the algorithm to get UAVs to arrive simultaneously degrades with increasing V_{nom}. This trend also exists in the crosstrack error results. Just like linear control theory would predict, increased velocity results in larger overshoots when using proportional feedback. Whether the desired path is parallel or perpendicular to the wind also affects crosstrack error. From Figure 4.33, the long straight section of BC 2's path is nearly downwind, leading to minimal crosstrack error, but BC 1's straight section is nearly crosswind, leading to larger errors.

Mode 3 results showed similar results to Mode 2: increasing V_{nom} increases positional errors. Angular error ranged from 30° to 40°, and crosstrack error ranged from 19 to 28 meters. The increased operating airspeed tends to increase the overshoot distances. The shape for all orbits took on a distinct look.

All orbit tracks (see Figures 4.40, 4.42, 4.44) displayed the "egg shaped" orbit with the large lobe at the north end of the orbit. Figure 4.34 contains the position of both BCs for 1 orbit and a plot of the angular error as a function of BC 2's angular position with respect to the target. Both BCs are orbiting CCW. BC 2 begins the plot with a headwind at the 7 o'clock position, with BC 1 at the 11 o'clock position. The angular error grows for the first half of the orbit because BC 1 is gaining on BC 2. As BC 2 reaches the downwind portion of the orbit, it quickly corrects much of the angular error (AE) with the superposition of increasing airspeed commands to correct the AE plus the wind boost. This chain of events appears to create the "egg." The target visibility section (Section 4.4) will address how this affects target visibility.

In general, the algorithm produced much larger crosstrack errors, angular errors, and arrival time differentials than simulation. The crosstrack error discrepancy is likely due to the different way the Kestrel autopilot corrects crosstrack. The method to

Table 4.10: Flight Test 2 Results-2 BATCAM Test

V_{nom}=11.75 m/s		
Item	BC 1	BC 2
mean y_s(m) mode 2	29.15	20.03
$\|\Delta t_{arrive}\|$(s) mode 2	–	5.05
mean AE(deg) mode 3	–	32.33°
mean y_s(m) mode 3	23.22	18.83
V_{nom}=10 m/s		
Item	BC 1	BC 2
mean y_s(m) mode 2	23.35	17.71
$\|\Delta t_{arrive}\|$(s) mode 2	–	1.94
mean AE(deg) mode 3	–	30.7°
mean y_s(m) mode 3	20.84	18.82
V_{nom}=14 m/s		
Item	BC 1	BC 2
mean y_s(m) mode 2	24.57	22.21
$\|\Delta t_{arrive}\|$(s) mode 2	–	6.25
mean AE(deg) mode 3	–	40.95°
mean y_s(m) mode 3	28.40	23.20

correct crosstrack (Eqn 2.2) in the simulation probably differs from Procerus's method programmed in the Kestrel autopilot. Also parameter settings can affect performance. Angular errors for all three nominal airspeeds were less than 41°. All arrival times were within 6.25 seconds of each other with the least difference at V_{nom}=10m/s.

Table 4.11: Flight Test 1 Results-2 BATCAM Test

Item	BC 2	BC 3
mean y_s(m) mode 2	10.45	32.11
$\|\Delta t_{arrive}\|$(s) mode 2	–	52.6
mean AE(deg) mode 3	–	159.8
mean y_s(m) mode 3	32.11	23.64

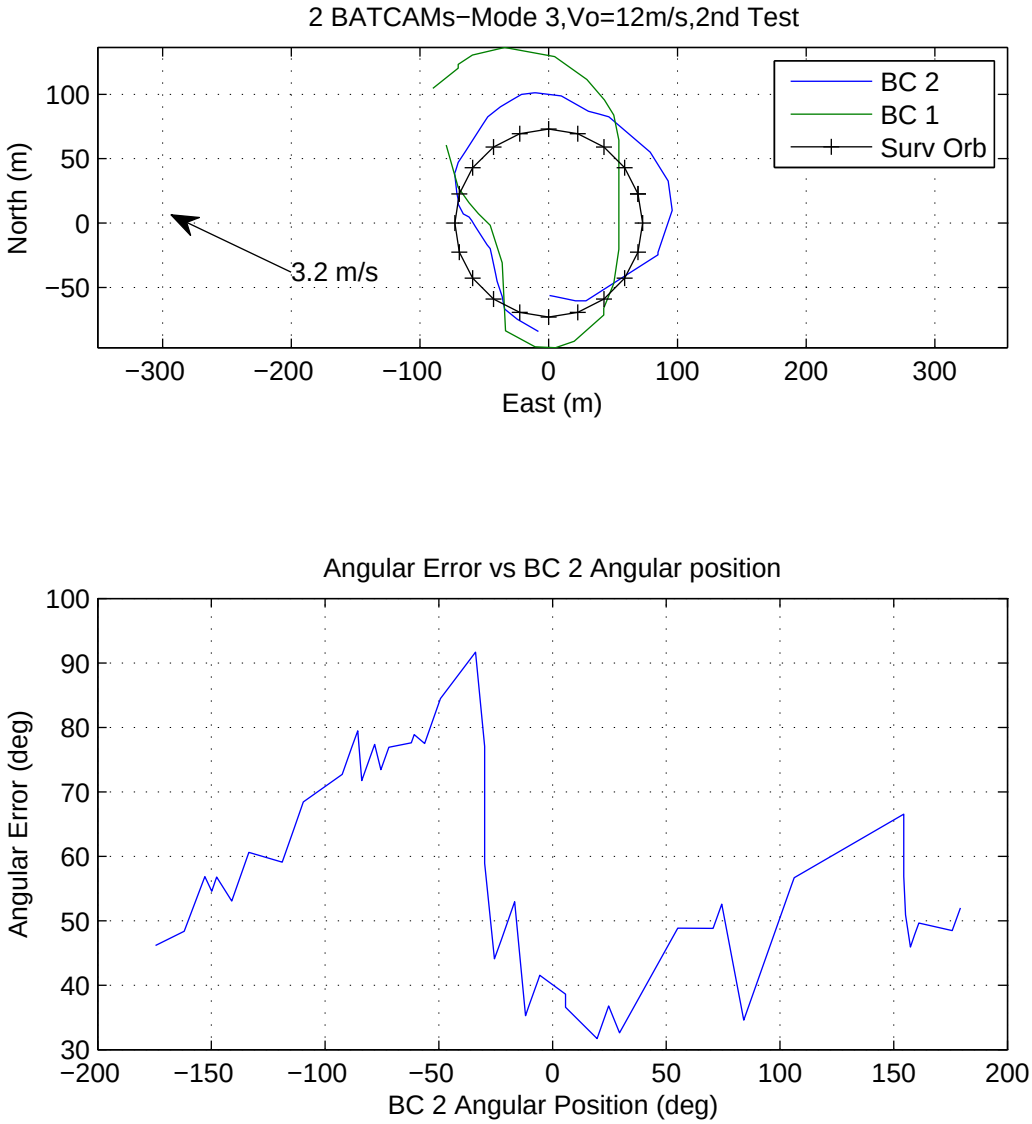

Figure 4.34: Flight Testing - Two BATCAM (Mode 3)-Typical Orbit - Position

4.4 Target Visibility

The final section of this chapter presents the performance of the algorithm in terms of visibility of the target. In addition to whether the target is in the FOV, the target will also only be visible from certain directions. For this analysis, the target will always be located at the origin and only be visible for headings $-\pi/2$ to $\pi/2$ (from the north). The measure of performance will be the time these conditions are met. For simulation, coverage for 2, 3 and 4 UAV formations for all four wind conditions will be presented. As before, errors in crosstrack and heading are introduced. For flight test 2 and 3 UAV formations were used.

4.4.1 Simulation Results. Two approaches will be presented to determine target visibility. The first will be a more restrictive approach that determines the time the target is visible in the FOV while in "the box." Figure 4.35 defines the box as the angle B as the angle from the vertical where 2B=360/(Num of UAVs). This is the angle that divides the surveillance orbit equally based on UAV quantity and decreases with increasing numbers of UAVs. Three box angles will be used: B=90°, B=60°, and B=45°. Ideally, the target will be visible the whole time when the UAV is in the box. Two metrics will be measured, time in the box, and the sum of all times for each UAV. When compared to the simulation time, this will provide a measure of effectiveness of the algorithm compared to the ideal case. The second approach will provide the total time for any UAV to view the target from the north (B=90°), regardless of the number of UAVs. This second approach (Overall Total time) accounts for overlap time between any two UAVs. From a human factors viewpoint, the first approach keeps the ground station personnel from having to adjust to the new perspective by keeping 1 UAV as the primary while in the box. The second approach will yield higher in-view times, but may disorient the operator with camera switching.

Illustrating the motion of the FOV in time is the best way to understand the processes that affect target visibility. Changes in velocity, attitude, altitude, and heading distort, rotate, and move the FOV leaving the in-view time metric somewhat

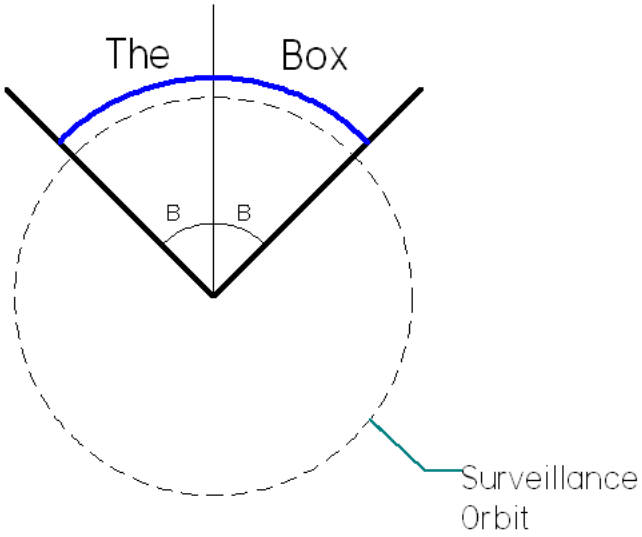

Figure 4.35: The Box Angle (B)

lacking. Unfortunately, a paper document is somewhat limited in its capabilities to present this. To compensate for this, X-Y overhead plots that show a sequence of FOV position and UAV position/heading will be used. These snapshot sequences will help explain the phenomena affecting target visibility.

The UAV initial conditions for each simulation are the same as used in Section 4.3. The two BATCAM simulation uses the same initial positions as UAV 1 and UAV 3 in Table 4.4. The three BATCAM simulation initial conditions are shown in Table 4.12. The four BATCAM case also uses Table 4.4 for beginning values. The two and four vehicle cases will follow the same trajectories as shown in Figures 4.10, 4.11, and 4.12, but the three BATCAM scenario used a new simulation. The induced errors in the initial conditions give a comparison of how the crosstrack and angular position control loops affect target visibility.

Tables 4.13, 4.14, and 4.15 present the in-view times and percentages for the two approaches. Increasing winds dramatically decreases the effectiveness of the UAV formation on target visibility. For all UAV formations, the target was visible less than

Table 4.12: Sensor Analysis-3 BATCAM Initial Conditions

Item	Location (pN, pE)	Heading	Initial V_a	Initial Alt
Target	(0,0)	n/a	n/a	n/a
UAV1	(-10, 65)	10°	11.54	50
UAV2	(57,-32)	281°	11.54	50
UAV3	(-60,-20)	135°	11.54	45
Commanded Values	–	Varies	11.75	50

50% of the time in the "box." Increasing the number of UAVs appears to make the surveillance more robust to increasing winds.

The overall times illustrate some interesting trends. Increasing the number of UAVs does improve the total target visibility times. For all wind cases, the highest percentages lie in the 4 UAV simulation. The increased redundancy aids the percentages, but not by large amounts. Looking at each wind case, the percentage spread between the 2 UAV case and 4 UAV case did not differ more than 10%. It appears that the extra redundancy aids the numbers with increasing wind. In the worst wind case, the 4 UAV formation had 10% more target visibility than the 2 UAV formation.

Initial conditions plays a large part on the performance of the in-view time. Ideally, the zero wind cases should all have 100% visibility for all 40 seconds of simulation. In the two BATCAM 0 Wind case, initial conditions caused the target to not be in the FOV for 2.34 seconds. For the first 0.87 seconds, UAV 1 was not in the upper half of the circle, and UAV 2 was proceeding south. As UAV 1 entered the upper half, from snapshot Fig 4.36 the FOV did not encompass the target until t=1.74 seconds. This is because the initial conditions that put the UAV outside the orbit and roll angle of zero caused the control loops to command a turn rate to get back on course. Consequently the turn rate increased the bank angle such that the FOV went too far East to see the target. The remaining 0.6 seconds of non-visibility comes from a period of between t=34.06 and t=34.66 where UAV 3 left the "box" before UAV 1 entered the "box."

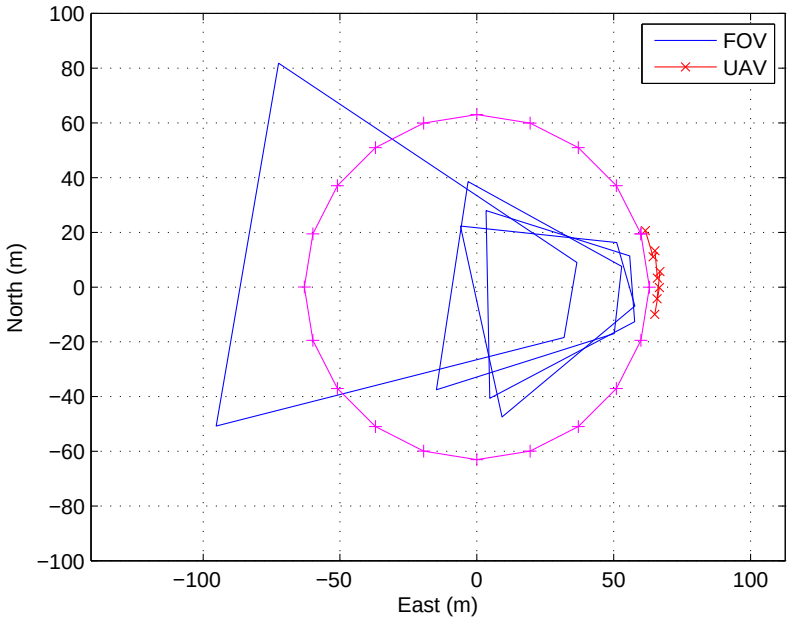

Figure 4.36: Sensor Analysis-Two BC Orbit-BC1 Position and FOV

Like the above example, crosstrack error has a dual effect on target visibility. When the UAV is not on the orbit, the FOV is also displaced. The command to correct the crosstrack error makes the UAV roll, moving the FOV again. A robust control loop for controlling crosstrack error has a negative impact on target visibility. The right balance seems to keep the target visible for small errors, and gradual corrections to keep positional requirements.

BC 3 for the 3 UAV formation illustrates the affect of increasing wind on the FOV and target visibility. Figures 4.37, 4.38, and 4.39 illustrate the position of BC 3 under 0, 2.94, and 5.88 m/s winds. As the wind produces crosstrack errors, the frequency of correction commands increases. These manifest as roll commands that move the FOV radially outward for crosstrack errors outside the orbit, and radially inward for crosstrack errors inside the orbit. For a radially outward movement, the FOV decreases in size, and conversely a radially inward movement increases its size. Crosstrack errors outside the orbit are more likely to move the FOV so that the target

is not visible. As the figures illustrate, increasing winds tend to cause outward errors decreasing target visibility.

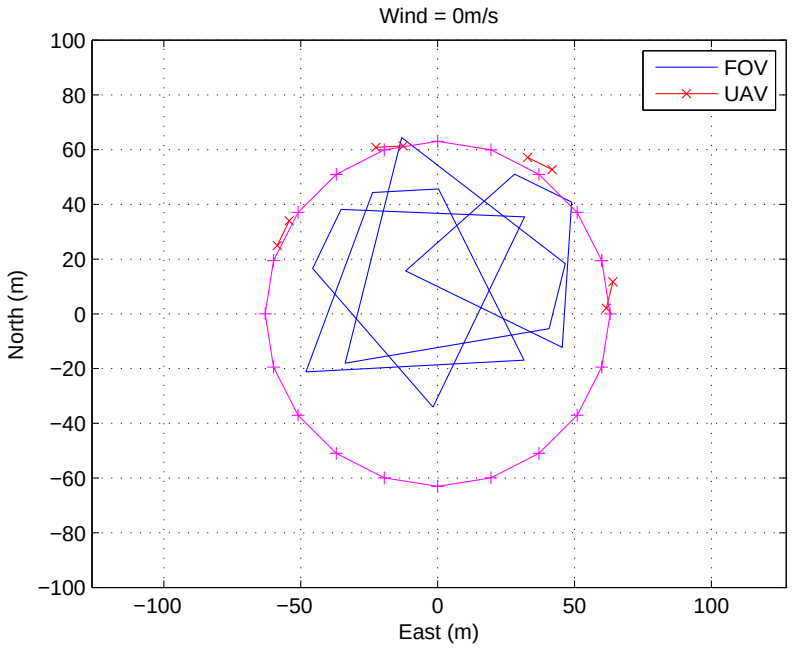

Figure 4.37: Sensor Analysis-BC3 and FOV, Wind 0 m/s

The wind direction and viewing angle were chosen to accentuate the effect of crab angle on target visibility. The largest effect appears in quadrant II, where the UAV first experiences the downwind side of the orbit. One way to alleviate the effects of crab angle is to change the shape of the orbit from circular to elliptical much like Rysdyk [16] and Farrell [7]. Rysdyk varied the radius as a function of heading, altering altitude. Farrell maintained constant altitude. Both approaches came up with ellipse-like solutions and would improve target visibility, but do not address an approach to handle crosstrack errors. The circular approach was chosen due to the Kestrel autopilot. The command set in the autopilot had a loiter command, and to create an ellipse would require construction with many waypoints. The delays already experienced with the Dubins path would be compounded with 10+ more waypoints to upload. It seems feasible to merge the ideas, but time did not allow during this research.

Table 4.13: Simulation Results-Target Visibility Time with 2 BATCAMs

Wind Condition	BC 1(%)	BC 3(%)	Box Total(%)	Overall Total(%)
0 m/s	21.39(53.5%)	16.93(42.3%)	38.32(95.8%)	37.66(94.1%)
1.17 m/s	18.27(45.7%)	16.17(40.4%)	34.44(86.1%)	33.75(84.4%)
2.94 m/s	17.66(44.2%)	12.70(31.8%)	30.36(75.9%)	29.82(74.6%)
5.88 m/s	12.23(30.6%)	5.12(12.8%)	17.35(43.4%)	17.35(43.4%)
all results are in seconds, percentages (%) are with respect to total simulation time 40s				

Table 4.14: Simulation Results-Target Visibility Time with 3 BATCAMs

Wind Condition	BC 1	BC 2	BC 3	Box Total(%)	Overall Total(%)
0 m/s	13.72	11.24	11.23	36.19(90.5%)	38.25(95.6%)
1.17 m/s	11.83	10.71	11.03	33.57(83.9%)	36.82(92.1%)
2.94 m/s	11.21	9.22	8.84	29.27(73.1%)	34.63(86.6%)
5.88 m/s	7.67	3.41	2.48	13.56(33.9%)	18.82(47.1%)
all results are in seconds, percentages (%) are with respect to total simulation time 40s					

Table 4.15: Simulation Results-Target Visibility Time with 4 BATCAMs

Wind Cond.	BC 1	BC 2	BC 3	BC 4	Box Tot.	Overall Tot.(%)
0 m/s	9.54	10.76	8.47	8.36	37.14(92.9%)	38.63(96.6%)
1.17 m/s	7.71	10.42	8.01	8.00	34.15(85.4%)	37.58(94.0%)
2.94 m/s	7.09	9.11	7.12	3.94	27.26(68.2%)	36.23(90.6%)
5.88 m/s	4.52	9.18	1.39	2.67	17.76(44.4%)	26.01(65.0%)
all results are in seconds, percentages (%) are with respect to total simulation time 40s						

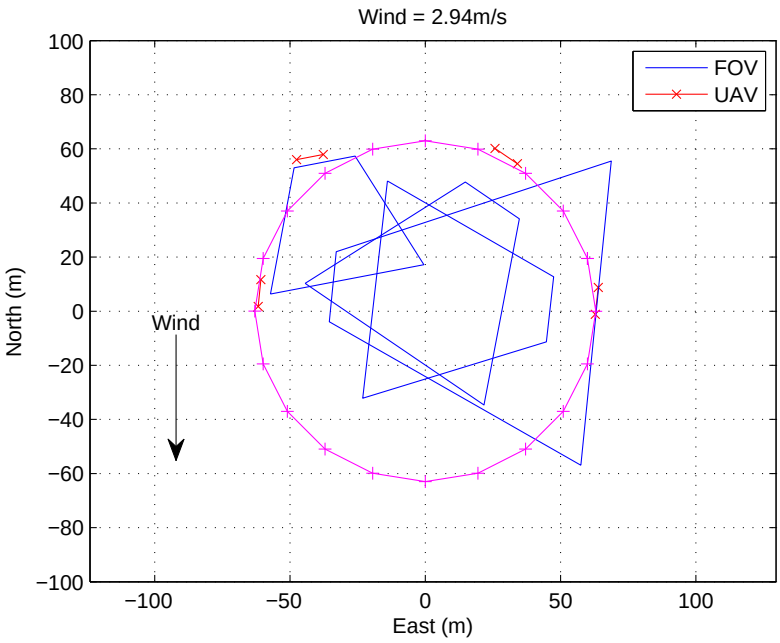

Figure 4.38: Sensor Analysis-BC3 and FOV, Wind 2.94 m/s

Overall, roll angle has a large effect on target visibility with a camera fixed to the fuselage. Even in no wind cases, initial conditions caused the target to leave the FOV due to changes in roll. The largest impact came from crosstrack errors, where the control loops altered roll angle to correct positional error. As wind increased, crab angle started to play a role in decreasing visibility in the portions of the orbit where a cross wind was prevalent.

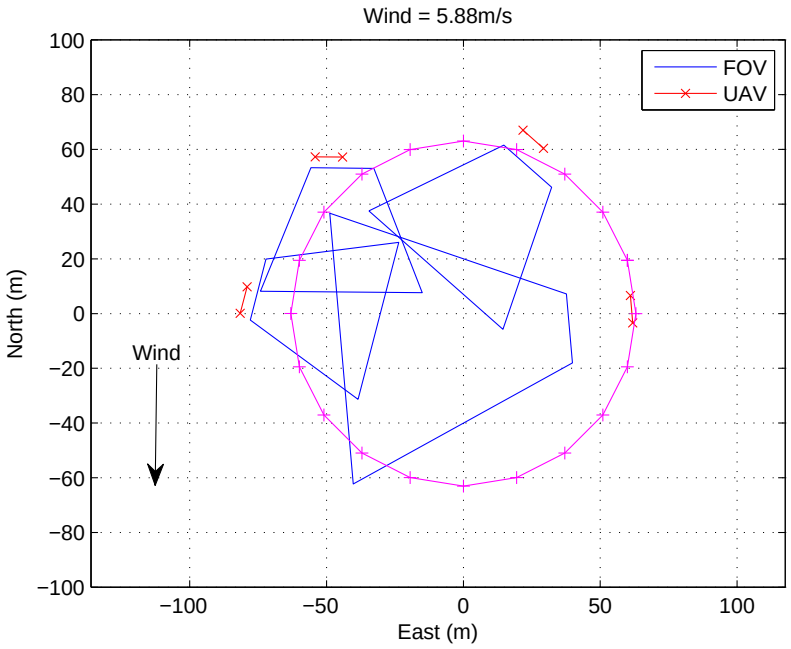

Figure 4.39: Sensor Analysis-BC3 and FOV, Wind 5.88 m/s

Table 4.16: Flight Test 1 Results-Target Visibility Time

Test	BC 1	BC 2	BC 3	Box Total	Overall Total(%)	Tot Orbit Time
2 BC Test 1	–	15.57	10.98	26.55(25%)	24.10(22.7%)	106.2
2 BC Test 2	–	13.03	16.42	29.45(25.6%)	28.00(24.3%)	115.2
3 BC Test 1	10.01	11.03	13.65	29.57(25.5%)	35.64(30.8%)	115.8
all results are in seconds, percentages (%) are with respect to total orbit time						

4.4.2 Flight Test Results. This section takes flight test data and determines the amount of time the target was visible. Like the simulation, the target will only be visible from the north, so when the target to UAV heading is between 90° and 270° the observed time is not counted. The same measures of performance will be used: in-view time in the "box", total in-view time in the box, and overall time. The "box" will be defined as B=90° for 2 BC tests and B=60° for 3 BC tests. Overall time is for any BC where B=90°.

Like Section 4.3.2, most of the results come from the second test flight, but as a comparison, Table 4.16 displays results from the first flight test. As stated previously, the airspeed feedback failed to work on the first flight test. Another big difference was the proportional gain was increased from 2 to 8 to match simulation parameters. Keep in mind that the CC algorithm does not provide heading commands, only airspeed commands to the autopilot. The Kestrel manages the crosstrack error correction (commanded heading).

Table 4.17 presents the average visibility times and percentages for the three V_{nom} cases, Figures 4.40 through 4.45 present the orbit tracks and FOV snapshot plots. All of the orbit tracks displayed the "egg" with the large lobe primarily on the north end of the orbit. Figures 4.40, 4.42, and 4.40 distinctly show a growing large lobe with increasing V_{nom}. With the "box" placed on the north end of the orbit, this large lobe caused decreased target visibility. The 14m/s plots did display a rotation of the large lobe in the direction of the wind, which increased target visibility for that

run since it brought the trajectory closer to the nominal orbit. In general, when the lobe was oriented north, poor visibility times resulted.

Just like the simulations, the roll commands necessary to correct crosstrack errors wreak havoc on the target visibility times (See FOV snapshot Figures 4.41, 4.43, and 4.45). Large crosstrack errors on the large side of the "egg" create commands that bring the FOV toward the UAV position. Consequently, the target quickly leaves the FOV until the roll angle decreases. Also the climbs to correct altitude also affect the FOV. Whenever the FOV moves in the direction of travel and appears ahead of the UAV position, the UAV is pitching up. Pitching up seems most prevalent in the transition from crosswind to headwind (and vice versa) conditions. Sometimes this helps target visibility, sometimes it hurts visibility.

In its attempt to keep angular position while orbiting, the algorithm negatively impacted target visibility. Comparing the first flight test data (Table 4.16) to the second flight test data (Table 4.17), very poor angular positioning still yielded slightly better target visibility. One possible reason is that the proportional gain for the second flight test may be too high. Crosstrack overshoots due to increasing airspeeds due to angular error caused rolling of the airframe, impacting visibility. As is, the proportional gain used in Equation 3.27 that worked well in simulation did not work well in flight testing. Unfortunately, the simulation does not closely represent the behavior of the Kestrel in crosstrack correction, so either the Kestrel needed further tuning to minimize crosstrack error or more testing is needed to find the gain that better balances positional feedback with target visibility.

Despite the non-functioning airspeed feedback, the first flight test did display some results that correlated with simulations (Table 4.16). Since the first test flew both 2 BCs and 3 BCs, the data displayed that target visibility increased with increasing number of UAVs. 2 BC formations had target visibility times around 23% and 3 BC formations increased the percentage to 30%. Due to the large angular errors, "box" totals saw no net benefit.

Overall the multi-UAV system had poor results. For the second flight test, box totals never exceeded 25% of the orbit time, and overall totals never exceeded 23% (Table 4.17). Without correct positional placement and attitude, the FOV cannot see the target. Crosstrack errors have the largest impact on target visibility. Not only do they move the FOV radially, but force the autopilot to roll the fuselage to correct for the error. With a body fixed camera, roll commands to correct crosstrack errors decreases in-view times. As seen in the first flight test, increasing numbers of UAVs does provide a certain amount of robustness and increases target visibility.

Table 4.17: Flight Test 2 Results-Target Visibility Time

Test	BC 1	BC 2	Box Total	Overall Total(%)	Tot Orbit Time
Test 1 V_{nom}=11.75m/s	14.42	20.54	34.96(21.9%)	32.44(22.7%)	159.5
Test 2 V_{nom}=11.75m/s	10.30	10.81	21.11(21.8%)	20.54(21.2%)	96.7
Test 3 V_{nom}=10m/s	8.44	11.03	19.47(18.5%)	19.61(18.6%)	105.2
Test 4 V_{nom}=10m/s	12.81	7.10	19.90(20.8%)	18.38(19.2%)	95.6
Test 5 V_{nom}=14m/s	7.25	10.61	17.86(15.2%)	16.37(13.9%)	117.4
Test 6 V_{nom}=14m/s	12.38	19.50	31.88(25.7%)	27.71(22.3%)	123.85
all results are in seconds, percentages (%) are with respect to total orbit time					

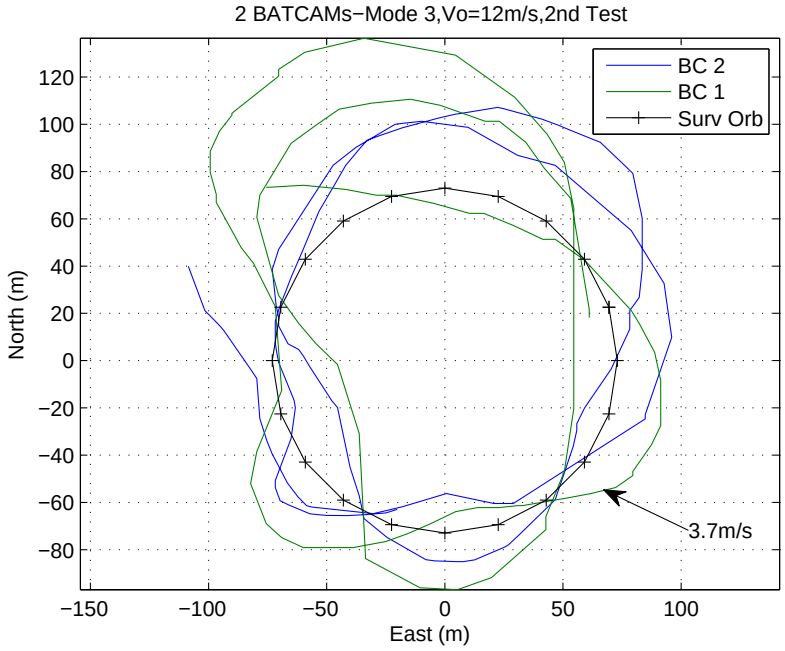

Figure 4.40: Flight Testing - Orbit Traces for V_{nom}=11.75m/s (Mode 3)

4.4.2.1 Flight Test 2.

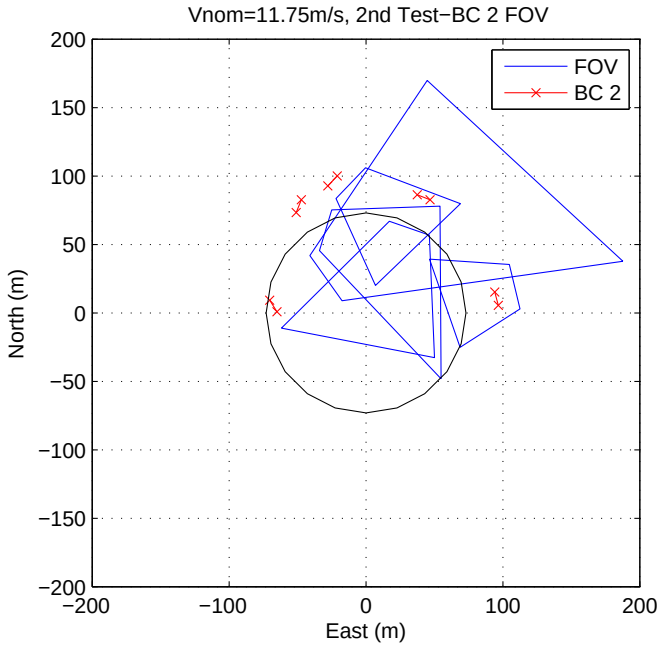

Figure 4.41: Flight Testing - FOV Snapshot for V_{nom}=11.75m/s

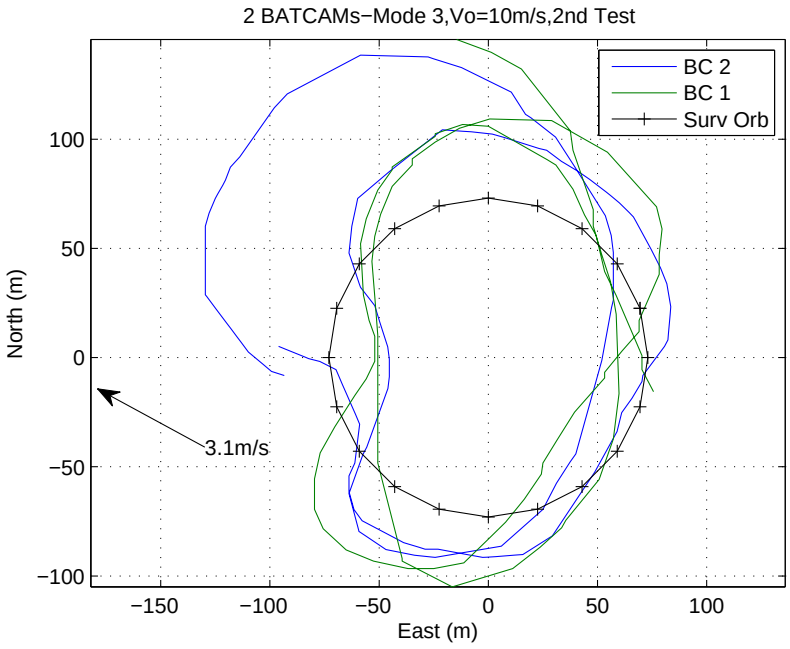

Figure 4.42: Flight Testing - Orbit Traces for for V_{nom}=10m/s

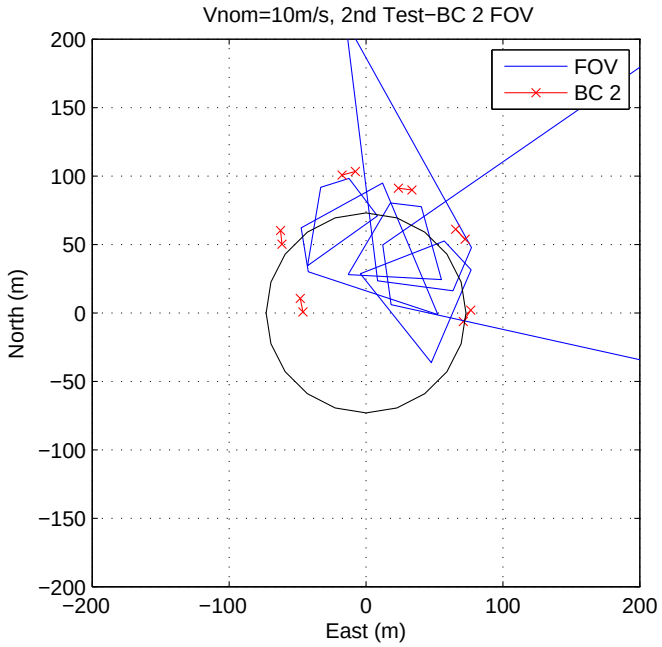

Figure 4.43: Flight Testing - FOV Snapshot for V_{nom}=10m/s

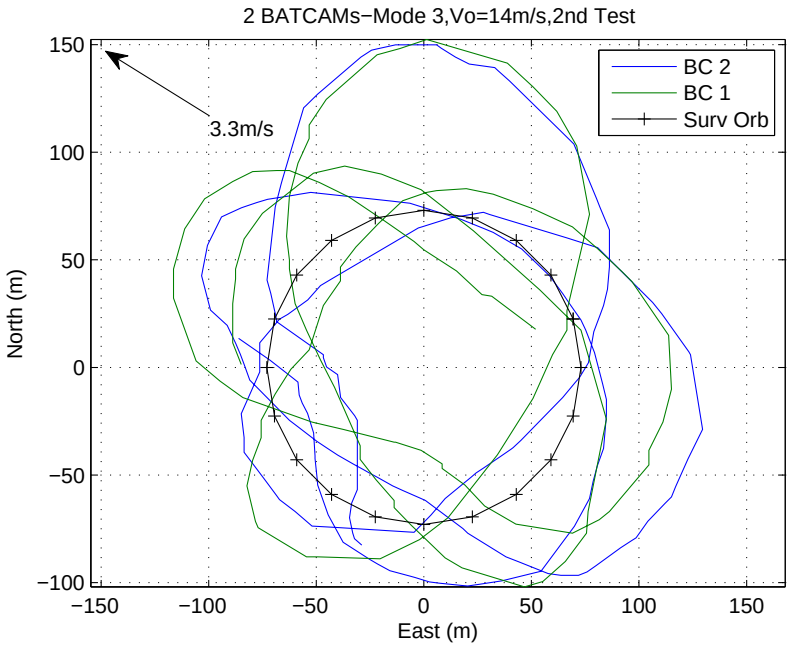

Figure 4.44: Flight Testing - Orbit Traces for V_{nom}=14m/s

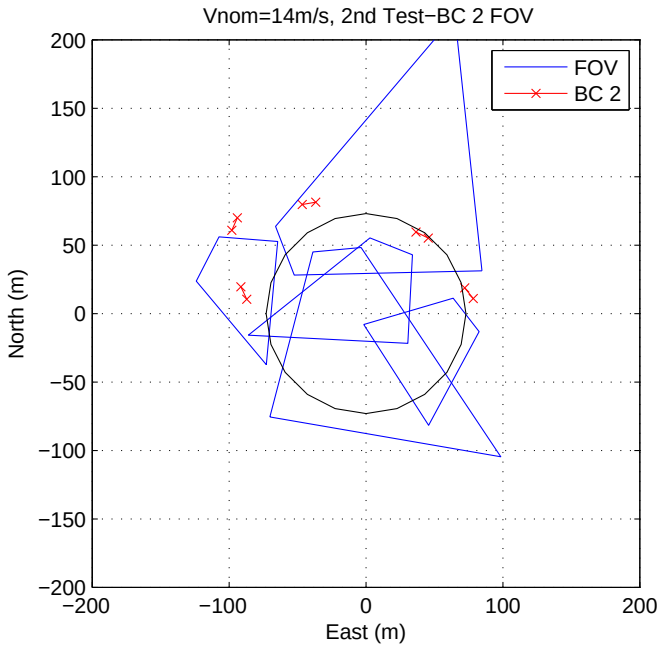

Figure 4.45: Flight Testing - FOV Snapshot for V_{nom}=14m/s

4.5 Summary

This chapter presented the results of the first order model of the BATCAM, chosen parameters for the CC algorithm, performance of the CC algorithm in simulation, and also flight test performance. Performance was quantified from both a positional and target visibility perspective.

Using basic control theory, flight test data yielded first order constants for airspeed, roll, and pitch. Other parameters that closed the loop and mimicked the Kestrel autopilot behavior were derived from actual limits set during flight testing, and reasonable desired behavior.

The first round of simulation analyzed the ability of the BATCAM model to correct and maintain desired position under wind conditions of 0%, 10%, 25% and 50% of nominal airspeed. Position accuracy was based on arrival time and mean crosstrack error for Mode 2 (simultaneous orbit approach), and based on settling time, mean angular error, and mean crosstrack error for Mode 3 (maintaining orbit). For Mode 2 simulation, UAV arrival times differed by less than 0.7 seconds for the no wind case, and less than 2 seconds for the 50% wind case. Crosstrack error steadily increased as wind increased, but was largely dependent on initial conditions. For Mode 3 simulation, the CC algorithm was able to maintain orbit for the first three wind cases, but instabilities appeared for the $50\%V_{nom}$ case. Again, mean crosstrack error and mean angular error showed a strong dependence on initial conditions. Mean angular error for all UAVs rose from 6.9° for 0 wind to 17.6° for 50% wind, mean crosstrack error rose from 2.11 m for 0 wind to 7.05 m for 50% wind. Settling time for the 4 UAV formation was 15.36 seconds for o wind, and never settled into the proper positions for the worst wind case due to large crosstrack errors. These instabilities begin with the combination of moderate crosstrack error on the downwind side of the orbit.

Performance of the CC algorithm was flight tested under three nominal airspeeds: 10 m/s, 11.75 m/s, and 14 m/s. Performance in both Mode 2 and Mode 3

showed increasing error with increasing V_{nom} where average arrival time differentials rose from 1.94 to 6.25 s, average angular error rose from 30.7° to 40.9°, average Mode 2 crosstrack errors rose from 17 to 22 m, and average Mode 3 crosstrack errors rose from 18 to 23 m. Surveillance orbits took on an "egg" shaped appearance due to cycle of increasing angular error followed by an angular correction when wind aided the increased commanded airspeed.

Target visibility simulations analyzed 2, 3, and 4 UAV formations under all 4 wind conditions and quantified time when the target was in the FOV. Two approaches were taken: one measuring the time the UAV was in "the box", the other measuring the time any UAV could see the target from the north side of the orbit. Overall target in-view times ranged from 96.6% for the 4 BC zero wind case to 43.4% for the 2 BC 50% wind case. Percentages not at 100% for the 0 wind case was due to initial conditions. Increasing the numbers of UAVs improved in-view times as wind increased. For 50% winds, 4 UAV formations kept the target visible 65% of the time verse only 43% for the 2 UAV formation. Crosstrack error had a dual effect on target visibility. Large crosstrack errors outside the orbit both translated the FOV, decreased its size, and caused radial motion away from the target. Roll commands to correct the crosstrack error pointed the FOV more directly downward shrinking size and moving it radially. So as crosstrack error increased, target visibility decreased.

Large crosstrack errors due to the "egg shaped" orbits during flight testing resulted in poor target visibility. Much like simulations, the necessary roll to correct crosstrack errors moves the FOV away from the target. Since the large lobe of the "egg" resided primarily on the north side of the orbit, target visibility times ranged from 14 to 23% of the total orbit time. The first flight test did display increasing target visibility with increased number of UAVs, much like simulations.

V. Conclusion

This research accomplished three major tasks: developed a closed loop model for the BATCAM, developed a cooperative control algorithm to control multiple BATCAMs to accomplish surveillance, and analyzed the performance of the algorithm in both simulation and flight test environments.

5.1 Conclusions

The algorithm met its overall goal of commanding multiple UAVs in real-time to approach and maintain an orbit about a fixed target. During flight testing UAV arrival times at the surveillance orbit differed by an average value of 4.4 seconds and maintained angular position while orbiting within an average value of 34°. Wind conditions averaged 30% of V_{nom} for these results.

Simulation data showed mixed results in predicting flight test phenomena. Simulation did predict the increasing orbit overshoots with increasing wind, and also predicted decreased target in-view time with these overshoots. Simulation did not predict the extent of crosstrack error that existed in flight test data.

Wind conditions had a large impact on the positional performance of the algorithm. In simulation, both mode 2 and mode 3 showed increasing errors in position and target visibility. In mode 2 mean crosstrack errors displayed only small increases (4.4%) from 0 wind to 50% wind, but mode 3 mean crosstrack errors jumped 331% from 2.13m to 7.05m. For the worst wind case, mode 3 never converged to maintain position in orbit, and was unable to re-adjust the UAVs in the add or remove UAV simulations. The algorithm was able to maintain position and adjust at 25% winds.

The simulation wind cases pointed out that the useable wind speed range for the algorithm lies between zero and at least 25% of V_{nom}. Somewhere between 25% and 50% of V_{nom} instabilities begin especially in mode 3 (maintain orbit). If the crosstrack errors exceed a certain amount, then the algorithm overshoots the intended path on the downwind side of the surveillance orbit and is unable to maintain the desired

orbit. Flight test data also displayed these overshoots when wind averaged 37% of V_{nom}.

Both simulation and flight test data show that increasing numbers of UAVs can improve target in-view percentages with increasing winds. Overall percentages in the worst wind case between the 2 BC simulation and the 4 BC simulation differed by 21.6%. Also increasing winds had less of an effect with more UAVs. When winds increased from 0 to 50% of V_{nom}, the four UAV formation only lost 31% visibility time where the 2 UAV formation lost 51% of target visibility time. The "box" approach showed mixed results where regardless of the number of UAVs, in-view percentages remained similar for all wind cases.

Crosstrack error had the largest impact on target visibility due to its dual effect on the FOV. The error itself translates the FOV away from the desired point and the roll command to the error induces further FOV movement. For the purposes of keeping the target in-view, the crosstrack feedback loop must be designed to balance the need to correct crosstrack error and roll commands that cause target visibility loss.

Roll has the largest affect on the FOV for a body fixed side camera. Since roll not only translates the FOV but also alters the size of the FOV, maintaining roll and minimizing changes in roll becomes very important. Due to the size and mass of the BATCAM, this can be challenging. As both simulation and flight test shows, roll caused the target to leave the FOV, especially with increasing winds.

5.2 Recommendations for Future

Expanding on this work could go in many directions. One direction refines this work, the next creates a new branch of research with the same basic elements. The specialized nature of Cooperative Control means many branches of research can stem from the same idea by just altering the scenario, operating environment, or even the assumptions.

This research left some unfinished work. A full 6 DOF model was started and lies in Appendix A, but work remains in closing the loop and finding a stable controller that mimics the Kestrel. The approach took DeLuca's [5] aerodynamic work and linearized it to generate the aerodynamic forces and moments. Look up tables is another approach for these forces and moments. A controller remains undesigned.

Due to the limits of flight testing at Camp Atterbury, the "magic combination" of control loop settings was never found that maximized target in-view time using the built in loiter command. Since the graduates of the March 2009 group were the first to fly the BATCAM, more than half the flight testing entailed creating a stable set of gains for the autopilot. Work remains to hone these gains for surveillance purposes.

Mode 2 is based on the arbitrary constraint of simultaneous arrival. This approach simplifies certain aspects of the problem, but is not the only way to approaching the surveillance orbit. One possible modification of the algorithm is to change Mode 2 such that the UAVs arrive asynchronously. Both collision avoidance and readjustment become major components of the problem, but may be a more flexible approach.

An optimal solution to this problem will provide a gauge as to the quality of this heuristic approach. This comparison can quantify the upper limit to this concept in terms of performance. As shown in Section 2.3.2 [15], finding the right simplifications to the optimal problem can possibly yield an approach worthy of real-time use.

This research illustrated that there is a trade space to explore in minimizing crosstrack error and maximizing target in-view time. The current algorithm only strives for positional accuracy, and does not account for its affect on FOV movement. Finding this balance between positional accuracy and keeping the FOV on the target given the constraints of the hardware (BATCAM/Kestrel) will take some work both in simulation and flight testing.

This vein of research is highly specialized and requires pulling together knowledge from many disciplines. Hands on work in Cooperative Control can be quite

challenging, yet rewarding. No textbook or simulation can provide the experience of integrating software with hardware.

Appendix A. Unfinished BATCAM Analysis

A.1 Mass Properties

Mass properties include the mass, the center of gravity location, and the three moments of inertia. The first two properties are measured using direct measurements: one from a scale, and the second from a simple balance test. The moments of inertia are derived from angular velocity measurements.

Figure A.1 illustrates the equipment used to measure angular velocity. This rotational table made by PASCO comes with DataStudio software and a rotary motion sensor that can measure the angular position, velocity, and acceleration. Measurements are taken at 0.1 second intervals. The table is accelerated by weights attached to string that rest on the outer pulley and wind around the main rotational axis. Three different weights are used to accelerate the table: 500 grams, 200 grams, and 50 grams.

Figure A.1: Moment of Inertia Measurement Equipment

The procedure is as follows. Once the BATCAM is mounted on the table in the desired orientation, the software is set to start recording angular velocity. The weight

is attached to the twine and the table is allowed to freely rotate for a few seconds. The software graphs the data and displays the angular velocity as a function of time. Since air drag begins to affect measurements at high speeds, only the data in the linear portion of the graph is used. The software calculates a linear fit on the selected data and the slope (angular acceleration) is recorded. The BATCAM is removed and the steps are repeated to find the inertia of the table itself. Three runs are conducted for each weight for a total of 9 runs per axis. The final angular acceleration ($\dot{\omega}_{500,200,50}$) is the mean of all three runs for each weight. The tension(T) in the twine is

$$T_{500,200,50} = (m_w - m_f)(g - \frac{d\dot{\omega}_{500,200,50}}{2}) \tag{A.1}$$

where m_w is the mass of the weight, m_f is an equivalent mass to account for friction, g is the acceleration of gravity, and d is the diameter of the spindle at the base of the table. The moment of inertia (I) becomes

$$I_{500,200,50} = \frac{dT_{500,200,50}}{2\dot{\omega}_{500,200,50}} \tag{A.2}$$

The final moment of inertia of the BATCAM for each axis averages the individual MOIs from each weight less the MOI of the table.

$$I_{xx,yy,zz} = \frac{I_{500} + I_{200} + I_{50}}{3} - I_{table} \tag{A.3}$$

Realize that I_{table} is calculated in the same manner as other MOIs, just without the BATCAM on the table. The results for all needed dimensions and mass properties are shown in the table below.

A.2 Aircraft Modelling-Deluca

Anthony DeLuca [5] completed a thorough aerodynamic analysis of the BATCAM airframe. In addition to the Lift and Drag data for various airspeeds and angles of attack, the research also included are the control surface moments for dif-

Table A.1: Dimensions and Mass Properties

Item	Value
Wingspan (b)	21 in
Mean Geometric Chord	5.94 in
Wing Reference Area	103.7 in^2
Mass	0.425 kg
I_{xx}	0.0028 kg m^2
I_{yy}	0.0088 kg m^2
I_{zz}	0.0091 kg m^2

Table A.2: Summary of Stall Angles

	10 mph α_{stall}	20 mph α_{stall}	30 mph α_{stall}	50 mph α_{stall}
Flex Wing	8.7°	12.7°	14.8°	7.5°
Rigid Wing	12.8°	12.7°	8.5°	–

ferent elevon deflections, and thrust coefficients for the propeller/motor. Since the BATCAM wing is flexible, much of the analysis focuses on the differences between this wing and a rigid wing.

Comparisons between rigid and flexible wings revealed that flexible wings delay stall conditions for 3 out of the 4 airspeeds (20, 30, 50 mph). When aerodynamic forces deflect the trailing edge upwards, the relative angle of attack (AoA) decreases. The exception to this was at 10 mph, where stall occurred at 8.7° verse 12.8° AoA for a rigid wing. Apparently a laminar separation bubble manifests itself as slight undulations in the lift line, degrading aerodynamic efficiency. Table A.2 summarizes the AoA's where stall conditions occurred.

Figure A.2 illustrates the lift and drag plots for the BATCAM at the above airspeeds at various angles of attack(α). The 50 mph data was incomplete due to wing damage at the last data point. Note the increased drag at 10 mph due to the same degraded aerodynamic efficiencies as stall.

The BATCAM's V-tail introduces some coupling of control moments compared to the traditional aileron, elevator, and rudder. Three [5] cases were examined: de-

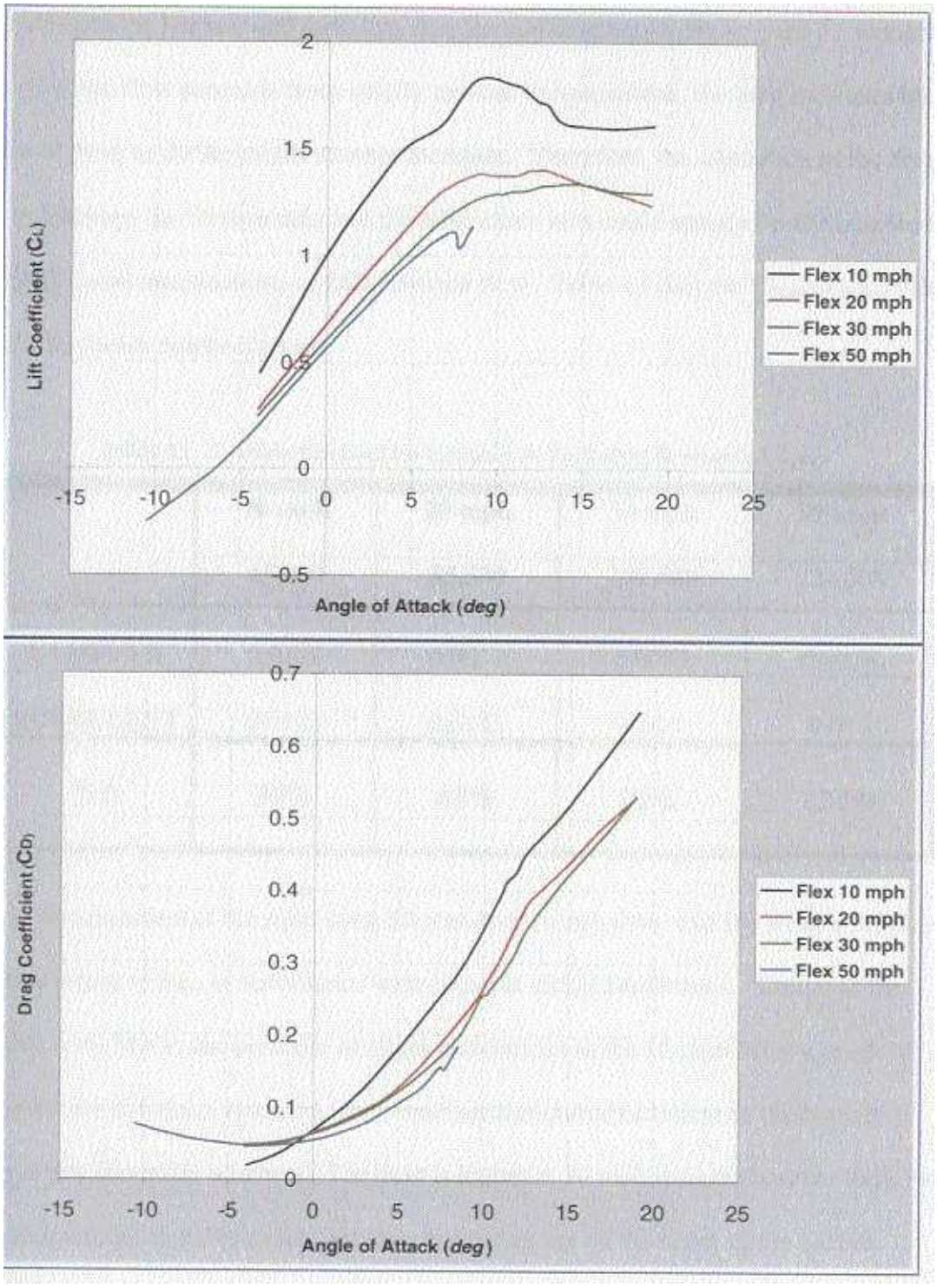

Figure A.2: BATCAM Flexible Wing C_L vs α and C_D vs α [5]

Table A.3: Average Slopes of Elevon Runs

30 mph	Single	Tandem	Opposed
$\partial C_m/\partial \delta_e$	0.0087	0.0133	0.0020
$\partial C_l/\partial \delta_e$	0.0012	0.0005	0.0021
$\partial C_n/\partial \delta_e$	-0.0043	-0.0015	-0.0072

flection of a single elevon, deflection of both elevons in the same direction (tandem), and deflection of both elevons in opposing directions. Ideally, opposing deflections create rolling and yawing moments with negligible pitching for heading and direction changes. Tandem deflections create pitching with minimal rolling and yawing for attitude and pitch control.

For the single deflection, all three moment directions behaved nearly linearly up to 5°. In both tandem and opposing cases, linear behavior held up to 8 °. Table A.3 presents the average slope per degree of deflection (δ_e) for the non dimensionalized moment coefficients of roll(C_l), pitch (C_m), and yaw (C_n).

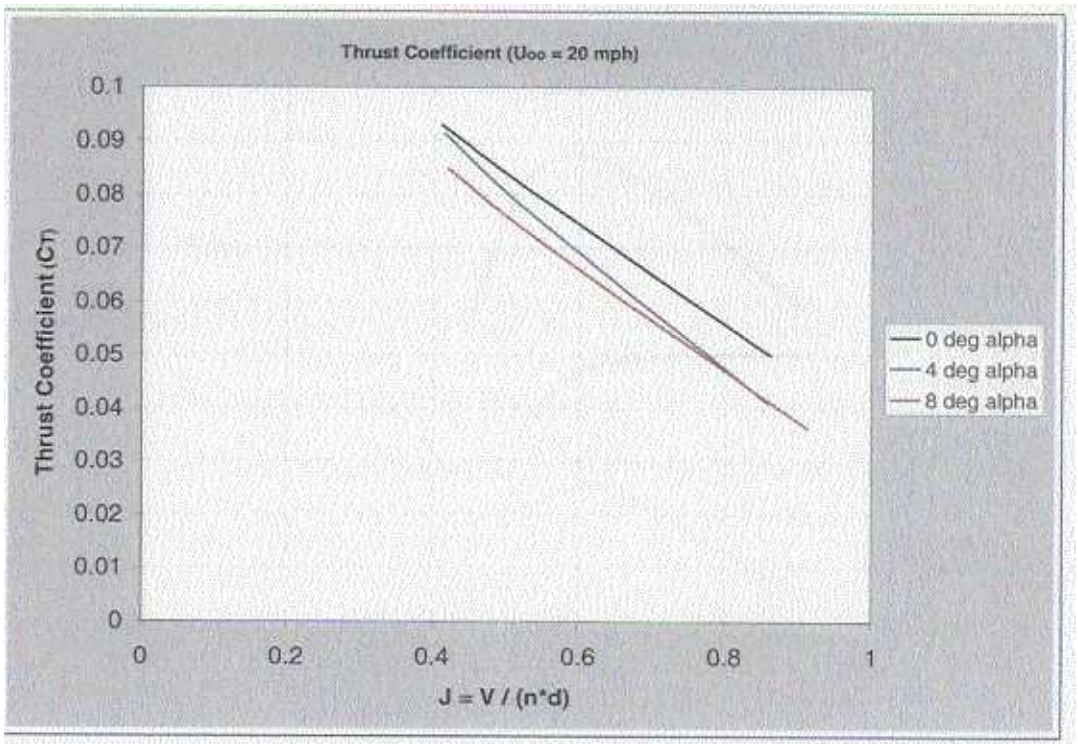

Figure A.3: BATCAM Thrust Coefficient vs Advance Ratio [5]

The thrust coefficients (C_T) were very consistent, linear, and well behaved. Figure A.3 presents C_T at 20 mph. The power coefficients are plotted as a function of the advance ratio Ja, which is defined as:

$$Ja = \frac{V_a}{\eta d} \tag{A.4}$$

where V_a is the airspeed, η is the prop speed in revolutions per second, and d is the diameter of the prop. At $\alpha = 0$ the BATCAM would overcome a 20 mph headwind at approximately 1/2 throttle (8000 RPM). Axial force measurement showed no dependence on angle of attack.

DeLuca presented a very solid aerodynamic analysis for use in model building. The above will be used to create linear relationships of aerodynamic coefficients centered around level flight conditions. From these relationships, an estimate for aerodynamic forces (Lift, Drag, Thrust, etc) will be incorporated into the model.

A.3 Linearized Aerodynamic Coefficients

This section pulls data from DeLuca [5] presented in Chapter II and creates linear equations for aerodynamic coefficients around level steady flight. These non dimensional coefficients provide the means to find the forces and moments on the airframe. The major aerodynamic forces and moments on an airframe are : Lift(L), Drag(D), Side Force (Y), Moments due to control surface deflections($\bar{L}$, M,N), and

Thrust(Th). These quantities are defined [18] [5] as:

$$\begin{aligned} L &= \bar{q}SC_L \\ D &= \bar{q}SC_D \\ Y &= \bar{q}SC_Y \\ \bar{L} &= \bar{q}SbC_l \\ M &= \bar{q}S\bar{c}C_M \\ N &= \bar{q}SbC_N \\ Th &= C_T\rho\eta^2 d^4 \end{aligned} \tag{A.5}$$

where $\bar{q}$ is the free stream dynamic pressure($=\rho V_a^2/2$), S is the wing reference area, b is the Wingspan, $\bar{c}$ is the mean geometric chord, ρ is the air density, η is the propeller speed in revolutions per second, and d is the propeller diameter.

The coefficients will be linearized with their respective independent variable (either AoA, Side slip angle, Advance Ratio, or elevon deflection). For the BATCAM, level steady flight occurs at

$$\begin{aligned} V_{ao} &= 11.75m/s \\ Throttle &= 50\% \end{aligned} \tag{A.6}$$

Using this data, the nearest appropriate data from DeLuca is chosen. Since 11.5 is approximately 25 mph, either the 20mph or the 30 mph can be used. For this research, all linearized coefficient equations will use 0 degrees AoA and 20 mph curves from Deluca's data (Figure A.2). Drawing a straight line from the points at 0 deg AoA to 5 deg AoA, the linearized C_L and C_D as functions of α (in radians) are

$$\begin{aligned} C_L(\alpha) &= 5.730\alpha + 0.7 \\ C_D(\alpha) &= 1.203\alpha + 0.075 \end{aligned} \tag{A.7}$$

From Deluca [5], the side force coefficient (C_Y) remained constant in the AoA range of 0 to 5 degrees. Also for side slip angles (β) up through 12 degrees, C_Y increased

linearly. The relationship between C_Y and β(in radians) was

$$C_Y = -0.573\beta \tag{A.8}$$

For the thrust coefficient, from Figure A.3 the linearized equation as a function of Ja becomes

$$C_T(Ja) = -0.1Ja + 0.135 \tag{A.9}$$

Next is to gain equations for C_L, C_M, and C_N as a function of elevon deflection. Using the derivatives shown in Table A.3 the coefficient equations become

$$\begin{aligned} C_L &= \frac{\partial C_l}{\partial \delta_e}\delta_e \\ C_M &= \frac{\partial C_M}{\partial \delta_e}\delta_e \\ C_N &= \frac{\partial C_N}{\partial \delta_e}\delta_e \end{aligned} \tag{A.10}$$

For this analysis, the elevons will only be used in some linear combination of tandem and opposing configurations. Realize that the partials for the tandem set and the opposing set must be kept together in order to describe the coupling effect of the V-tail. In theory [5], a tandem command should only create a pitching moment. DeLuca's data showed a light rolling and a yaw. For the sake of model building $\partial C_L/\partial \delta_e$ and $\partial C_N/\partial \delta_e$ will be zero to eliminate any asymmetries that may have existed in DeLuca's UAV. Also in theory, opposing elevons should only create roll and yaw. Once again for opposing elevon commands, $\partial C_M/\partial \delta_e$ will be zero in the model. So for tandem elevon deflections

$$\begin{aligned} C_L &= 0\delta_e \\ C_M &= 0.0133\delta_e \\ C_N &= 0\delta_e \end{aligned} \tag{A.11}$$

and for opposing deflections

$$\begin{aligned} C_L &= 0.002\delta_e \\ C_M &= 0\delta_e \\ C_N &= 0.0072\delta_e \end{aligned} \tag{A.12}$$

Typically any small eccentricities in control surfaces or the airframe are trimmed out to get the UAV to fly straight.

This is a limited set of the many actual aerodynamic forces and moments, but does capture the major components. Forces and moments will use these coefficient linearizations and apply equation 3.5 for calculation.

A.4 6 DOF Model

The 6 DOF flat earth equations [18] are

$$\begin{aligned} \dot{v_B} &= -\Omega_B v_B + B_B g_o + \frac{F_B}{m} \\ \dot{\omega_B} &= -J^{-1}\Omega_B J\omega_B + J^{-1}T_B \\ \dot{Phi} &= E(\Phi)\omega_B \\ \dot{p}_{NED} &= B_B^T v_B \end{aligned} \tag{A.13}$$

where v_B, ω_B are the velocity and angular velocities in the body frame, Φ is the attitude vector (roll(ϕ), pitch(θ), yaw(ψ)), and p_{NED} is the position vector in the NED frame. F_B and T_B are the body forces and body torques, which are usually aerodynamic (F_B =(T-D Y -L)^T, T_B =($\bar{L}$ M N)^T). B_B is the transpose of C_b^n from

Eqn 3.11. The other associated matrices are (c=cos, s=sin, t=tan):

$$
\begin{aligned}
J &= \begin{bmatrix} I_{xx} & 0 & -I_{xz} \\ 0 & I_{yy} & 0 \\ -I_{xz} & 0 & I_{zz} \end{bmatrix} \\
\Omega_B &= \begin{bmatrix} 0 & -R & Q \\ R & 0 & -P \\ -Q & P & 0 \end{bmatrix} \\
E(\Phi) &= \begin{bmatrix} 1 & t\theta s\phi & t\theta c\phi \\ 0 & c\phi & -s\phi \\ 0 & \frac{s\phi}{c\theta} & \frac{c\phi}{c\theta} \end{bmatrix}
\end{aligned}
\tag{A.14}
$$

where $\omega_B = (\mathrm{P\ Q\ R})^T$ and the moments of inertia are from the mass properties section. I_{xz} was not measured, but is usually small compared to the other three moments and may be set to zero or a percentage of I_{xx}.

To properly calculate aerodynamic forces and moments use

$$
v_R = v_B - B_B \begin{bmatrix} W_N \\ W_E \\ W_D \end{bmatrix} \tag{A.15}
$$

relative velocity v_R.

This is the open loop model. The controller and closed loop model are unfinished. The MATLAB implementations of the aerodynamic forces and the flat earth 6 DOF equations are in the next section.

A.5 MATLAB files

Listing A.1: SourceCode/BATCAM_FE.m

```
function St_dot=BATCAM_FE(St)
%this calculates the first order derivatives for state propogation...
    given
%the state and inputs using 6 DOF flat earth equations (Lewis&
% Stevens p46) The state (St) is a 1x17 state:
%   [u v w]  velocities in the body frame (1st three elements)
%   [p q r] angular velocities in the body frame (2nd three ...
   elements)
%   [phi th psi] attitude (roll pitch yaw) (3rd three elements)
%   [pe pn pd] position in the NED frame (4th three elements)
%
%   [Th d_O d_T We Wn]
%   Th = throttle in percent e.g.(.5 = 50%)
%   d_O opposing elevon cmd in degrees
%   d_T tandem elevon cmd in degrees
%   We is east component of wind
%   Wn is north component of wind

%separate out State and Inputs
Inp=St(13:17);
vb=St(1:3);
wb=St(4:6);
att=St(7:9);
pNED=St(10:12);
Th=Inp(1);
d_O=Inp(2);
d_T=Inp(3);
We=Inp(4);
Wn=Inp(5);

%mass properties
m=.425;
```

```
g=9.81;
Ixx=0.0028;
Iyy=0.0088;
Izz=0.0091;
J=[Ixx 0 0;0 Iyy 0;0 0 Izz];

%calculate necessary Matrices
Om_b=[0 -wb(3) wb(2);wb(3) 0 -wb(1);-wb(2) wb(1) 0];
Bb=[cos(att(2))*cos(att(3)) cos(att(2))*sin(att(3)) -sin(att(2)); ...
    ...
     -cos(att(1))*sin(att(3))+sin(att(1))*sin(att(2))*cos(att(3)) ...
        cos(att(2))*cos(att(3))+sin(att(1))*sin(att(2))*sin(att(3))...
         sin(att(1))*cos(att(2)); ...
     sin(att(1))*sin(att(3))+cos(att(1))*sin(att(2))*cos(att(3)) -...
        sin(att(1))*cos(att(3))+cos(att(1))*sin(att(2))*sin(att(3))...
         cos(att(1))*cos(att(2))];
EP=[1 tan(att(2))*sin(att(1)) tan(att(2))*cos(att(1)); ...
    0 cos(att(1)) -sin(att(1)); ...
    0 sin(att(1))/cos(att(2)) cos(att(1))/cos(att(2))];

%calculate airspeed (assumes 0 down component of wind)
%vr = velocity relative to wind
vr=vb-Bb*[Wn; We; 0];
Va=norm(vr);
%calculate angle of attack, side slip angle
alph=atan(vr(3)/vr(1));
beta=asin(vr(2)/Va);

%call function that calculates aero forces
if Th < 0
    Th=0;
end
if Th > 1
    Th = 1;
end
```

```
61
   Fa=AeroForces(Va, alph, beta, Th, d_O, d_T);

   Fxyz=Fa(1:3)';
   LMN=Fa(4:6)';
66
   %Calculate time derivatives
   d_vb=-Om_b*vb+Bb*[0 0 g]'+Fxyz/m;
   d_wb=-J^-1*Om_b*J*wb+J^-1*LMN;
   d_att=EP*wb;
71 d_pNED=Bb'*vb;



   St_dot=[d_vb; d_wb; d_att; d_pNED];
```

Listing A.2: SourceCode/AeroForces.m

```
   function Fa=AeroForces(Va,alpha, beta, Th, d_O, d_T)
   %This function calculates the aerodynamic forces(Fxyz) and moments...
      (LMN on
   %the BATCAM airframe.  All units are in metric
   %Inputs
 5 %   Va=airspeed
   %   alpha= angle of attack
   %   beta= side slip angle
   %   Th = throttle percentage
   %   d_O = opposing elevon deflection command (in degrees)
10 %   d_T = tandem elevon deflection command (in degrees)
   %Outputs
   %   Fa = [Fx Fy Fz L M N]
   %   [Fx Fy Fz] are the forces in the xyz direction
   %   [L M N]  are moments about the x y z axes
15
   %BATCAM properties
   rho=1.204;  %air density in metric
```

```
S=8.73*5.94*2*2.54^2/100^2; %wing area [m^2]
b=21*2.54/100;
c_bar=5.94*2.54/100;  %mean geometric chord [m]
q_bar=rho/2*Va^2; %free stream dynamic pressure
d=8.5*2.54/100;  %prop diameter


%Calculate coefficients
CL=5.73*alpha + 0.75;
CD=1.203*alpha + 0.075;
CY=-0.573*beta;
Clmn_T=[0.000 .0167 -.00];
Clmn_O=[.0021 .00 -.0072];

%Calculate forces/moments
D=q_bar*S*CD; %Drag
L=q_bar*S*CL; %Lift
Y=q_bar*S*CY; %Side force
LMN_T=q_bar*S*[b c_bar b].*Clmn_T*d_T; %Tandem elevon moments
LMN_O=q_bar*S*[b c_bar b].*Clmn_O*d_O; %Opposing elevon moments

%Calculate Thrust
nu=117*Th;  %revolutions/s
J=Va/(nu*d); %advance ratio
Ct=-.1*J+.135; %coefficient of thrust
T=Ct*rho*nu^2*d^4; %Thrust

%output
Fa=[T-D Y -L LMN_T+LMN_O];
```

Appendix B. CC Algorithm Source Code

The C++ source code for the dialog box that implements the CC algorithm that interfaces with VC is on the accompanying CD. Only the Matlab® files that implement the algorithm are included. The C++ code utilizes the concepts in Procerus's DevDemo sample application which illustrates how to construct and transmit VC packets, and also the Matlab® capability that interfaces with C++ using the "mcc -B csharedlib:*** ***.m ..." command.

B.1 Matlab Files for CC Algorithm

Here are the associated Matlab® routines and dependencies implemented in the C++ source code. Refer to Section 3.6.4 for descriptions of the routines.

Listing B.1: SourceCode/EnterOrbit.m

```
function [LatLonHdg r]=EnterOrbit( LatLon_t, alt, th_d, Vg, numUAV, offst, ...
    GPS_ht)
%This Function provides a matrix (nx3) of the [lat lon hdg] for n UAV's to
%enter an orbit around a target positioned at LatLon_t=[lat lon].  The
%orbit is a circle divided into numUAV equal parts at altitude (alt) and
%with sensor depression angle th_d.  All vehicles enter orbit
%counterclockwise.
%Inputs:
%   LatLon_t = [lat lon] in degrees for the target position
%   alt = altitude of orbit (in meters above ground level)
%   th_d = sensor depression angle (radians).  Algorithm assumes the sensor
%   is side facing to view the target in the middle of the orbit
%   Vg =  the ground speed of the vehicle (user must find ground speed
%   given airspeed and wind conditions)
%   numUAV =  the number of vehicles to orbit the target
%   offst = any angular offset (in radians).  At offst=0, the first point
%   is at 0 rad  (at point directly east of target location)
%   GPS_ht =  height above mean sea level (in meters)
%Outputs:
%   LatLonHdg = nx3 matrix where each row denotes the [lat lon hdg] to
```

```
%   enter the orbit.  n equals numUAVs.  lat/lon are in decimal degrees,
%   heading is in radians.
%   r = orbit radius (in meters)
%
%   This function needs the Orbit_r function,Cart2GPS function, and
%   Zeroto2pi function.

[r phi]= Orbit_r(alt, th_d, Vg);
ii=1:numUAV;
theta=2*pi/numUAV*(ii-1);  %create angle increments
xy=r*[cos(theta+offst); sin(theta+offst)]; %create xy coordinates
[lat lon]=Cart2GPS(LatLon_t, xy(1,:)', xy(2,:)', GPS_ht);
hdg=Zeroto2pi(2*pi-(theta+offst))';
LatLonHdg=[lat lon hdg];
```

Listing B.2: SourceCode/CreateOrbit.m

```
function [Wp d_mtx  Hdg_mtx rf]=CreateOrbit( GPS, Hdg, r_min, GPS_ht)
%This function creates a series of waypoints based on initial and final GPS
%points and headings.  The lengths of all paths are the same within 5 meters.
%The waypoints are along a Dubins path.  A waypoint
%is provided at least every 90 degrees along curved portions of the path
%
%Inputs are:
%   GPS = nx4 matrix denotes the initial and final values of GPS
%   coordinates (lat, lon) for each vehicle (in degrees).  Each row is the
%   set for each vehicle [Lat_initial Lon_initial Lat_final Lon_final]
%   Hdg = nx2 matrix denoting the initial and final headings for each
%   vehicle (in radians).  Each row is for each vehicle [Hdg_i Hdg_f]
%   r = the minimum turn radius (scalar--in meters)
%Outputs are:
%   Wp = 2nxm matrix providing the waypoints(in degrees).  Rows 1 and 2 ...
   denote the set
%   of points for vehicle 1[Lat; Lon], Rows 3 and 4 denote the set for ...
   vehicle 2, and
```

```
%   so on.  Since the number of columns for each vehicle can vary, the
%   dimension m denotes the length of the longest set. The end of shorter ...
    sets are
%   padded with the number -100 since latitude cannot exceed +/-90.  The
%   user must account for the fact that if he/she sees a -100 in a latitude
%   measurement, the end of the list has been reached.
%   d = nxm matrix denoting the distance at each waypoint of the Dubins
%   path(in meters) of each vehicle-total dist is at last point.  this is
%   also padded with -100 for shorter vectors
%   Hdg_Mtx nxm matrix of heading at each waypoint
%   r =  1xn vector denoting the radius of the Dubins path.  Realize this ...
    algorithm assumes
%   that the radii for each end of the path are the same.


ro=r_min;
numUAV=size(GPS); numUAV=numUAV(1);
% i=3020:40:3160;
% GPSi=r_GPS(i,:);
% [lat_f lon_f]=Cart2GPS([39.3435 -86.0345],[r_orbit 0 -r_orbit 0]',...
%     [0 r_orbit 0 -r_orbit]',216);
% GPSf=[lat_f lon_f];
% Hi=att(i,3);
% Hf=[0 270 180 90]'*pi/180;
% GPS_ht=Alt_GPS(1);
% ro=20;
 rf=zeros(1,numUAV)+r_min;


%generate initial distances
for ii=1:numUAV
    [W d_temp Htmp]=Dubins([-1 -1;0 0],[GPS(ii,1:2);GPS(ii,3:4)],...
        [Hdg(ii,1) Hdg(ii,2)], ro, GPS_ht);
    d(ii)=d_temp(1);
```

```
        if ii > 1 && d(ii)>dmax(1)
            dmax=d_temp;
            Wmax=W;
            Hmax=Htmp;
    55      elseif ii == 1
            dmax=d_temp;
            Wmax=W;
            Hmax=Htmp;
        end
    60
    end
    %sort from largest to smallest
    [ds iis]=sort(d,'descend');
    lenWmax=size(Wmax); lenWmax=lenWmax(2);
    65 Wp=zeros(2*numUAV,lenWmax);  % create initial Wp matrix with zeros
    Wp((2*iis(1)-1):(2*iis(1)),:)=Wmax;
    d_mtx=zeros(numUAV, lenWmax); %create initial d_mtx w/zeros
    d_mtx(iis(1),:)=dmax; %place max dist row in right place according to sort
    Hdg_mtx=zeros(numUAV, lenWmax); %create initial Hdg_mtx w/zeros
    70 Hdg_mtx(iis(1),:)=Hmax; %place max dist row in right place according to sort

    Kp=.05;
    %adjust radii to get distances all the same
    for ii=2:numUAV
    75      ctr=1;
        dd=100;  %get things started
        dr=2;
        d_dr=2;
        extra_turn=0;
    80      extra_turn_flg=1;
        dd_sign=1;
        ctr_flg=0;
        while abs(dd) > 5
            rf(iis(ii))=rf(iis(1))+dr;
    85          %recalculate with new radius
```

```
        [W d_temp Hdg_tmp]=Dubins([-1 -1;extra_turn 0],[GPS(iis(ii),1:2);GPS(...
            iis(ii),3:4)],...
             [Hdg(iis(ii),1) Hdg(iis(ii),2)], rf(iis(ii)), GPS_ht);
        d(iis(ii))=d_temp(1); %grab path length
        if extra_turn && extra_turn_flg
 90          extra_turn_flg=0;
             if d(iis(ii)) > dmax
                 d_dr=-d_dr;
             end
        end
 95
        dd=d(iis(1))-d(iis(ii)); %find difference btwn longest and current

        ctr=ctr+1;
        dr=dr+d_dr;
100
        if (dd*dd_sign < 0)  % detects an overshoot
             dd_sign=-dd_sign;
             d_dr=-d_dr/2;  %halve increment of change
        end
105     if (ctr > 100)  %  if doesn't converge add extra turn and reset ...
            everything
             if ctr_flg  %if it doesn't converge after 200 just return
                 return;
             end
             extra_turn=1;
110          dr=0;
             d_dr=1;
             ctr=0;
             ctr_flg=1;
             dd_sign=1;
115     end

    end
    lenW=size(W); lenW=lenW(2);
```

```
        if lenW > lenWmax
120         Wp=[Wp (zeros(2*numUAV,(lenW-lenWmax))-100)];  %add length to Wp
            d_mtx=[d_mtx zeros(numUAV, lenW-lenWmax)-100]; %add length to d_mtx
            Hdg_mtx=[Hdg_mtx zeros(numUAV, lenW-lenWmax)-100]; %add length to ...
                Hdg_mtx
            lenWmax=lenW;
        elseif lenW < lenWmax
125         W=[W (zeros(2,(lenWmax-lenW))-100)]; %extend W
            d_temp=[d_temp zeros(1,lenWmax-lenW)-100]; %extend d_temp
            Hdg_tmp=[Hdg_tmp zeros(1,lenWmax-lenW)-100]; %extend Hdg_tmp
        end
        Wp((2*iis(ii)-1):(2*iis(ii)),:)=W; %insert W into Wp
130     d_mtx(iis(ii),:)=d_temp; %insert d_temp into d_mtx
        Hdg_mtx(iis(ii),:)=Hdg_tmp;


    end
```

Listing B.3: SourceCode/UpdateGrndSpd.m

```
    function Vd=UpdateGrndSpd(Cur_GPS, Cur_Wp, Wp_mtx, d_mtx, V_nom, GPS_ht)
    % This function calculates the necessary ground speed to complete the
    % remaining waypoints in the flightplan so that all vehicles arrive at the
  4 % last waypoint at the same time.
    %Inputs:
    %   Cur_GPS =  nx2 array of [lat lon] coordinates of each Vehicle.  n
    %   denotes the number of vehicles.
    %   Cur_Wp = 1xn vector of numbers.  Each number denotes the next waypoint
  9 %   in the flight plan.
    %   Wp_mtx = 2nxm matrix where every row pair are the waypoints for each
    %   vehicle.  See the CreateOrbit command for more details.
    %   d_mtx = nxm matrix of distances remaining in the flightplan. See the
    %   CreateOrbit command for more details
 14 %   SpdRnd = 1x2 vector that contains the min and max velocity[Vmin Vmax]
    %   GPS_ht = the reference GPS height (height above MSL)
    %Outputs:
```

```
%   Vd = 1xn vector of updated velocities [m/s]required for each vehicle to
%   arrive all at the same time.  The routine uses the Vmax value for the ...
    flight
%   plan with the greatest distance. If no solution exists within the given
%   velocity range, the function returns a Vd vector of zeros.

% Calculate distances from current position to next waypoint
numUAV=length(Cur_Wp);
numWp=size(Wp_mtx); numWp=numWp(2);
for ii=1:numUAV
    [d_tmp H]=DistFromGPS([Cur_GPS(ii,:); Wp_mtx((2*ii-1):(2*ii),Cur_Wp(ii))...
        '],GPS_ht);
    d_tot(ii)=d_tmp(2)+d_mtx(ii,Cur_Wp(ii));
end
d_mean=mean(d_tot);
t=d_mean/V_nom;  %find the time to complete using nominal V and mean dist.
Vd=d_tot/t;
```

Listing B.4: SourceCode/Airspeed.m

```
function Va=Airspeed(Vg, Vw, V_limits)
% This function returns the necessary airspeed, necessary heading, crab
% angle, and a flag stating whether the needed airspeed exceeds limits.
%Inputs:
%   Vg = 1x2 vector that contains desired [Ground_Speed Heading(rad)]
%   Vw = 1x2 vector that contains [Wind_Speed Wind_Heading(rad)]
%   V_limits = 1x2 vector with max/min values [V_max V_min]
%Outputs
%   Va = 1x4 vector that contains [Airspeed, Heading(rad), crab_angle(rad),
%   flag]
% All speeds must be in the same units, all angles must be in radians.
% Positive crab angles denote CCW angle from ground heading to airspeed
% heading.
th_g=pi/2-Vg(2);
th_w=pi/2-Vw(2);
Vg_vec=Vg(1)*[cos(th_g) sin(th_g)];
```

```
Vw_vec=Vw(1)*[cos(th_w) sin(th_w)];
Va_vec=Vg_vec-Vw_vec;
Va(1)=norm(Va_vec);  %airspeed
Va(2)=Zeroto2pi(pi/2-atan2(Va_vec(2), Va_vec(1)));
Va(3)=atan2(Va_vec(2), Va_vec(1))-th_g;
if (Va(1) > V_limits(1)) || (Va(1) < V_limits(2))
    Va(4)=1; %calculated airspeed exceeds limits
else
    Va(4)=0; %Calculated airspeed is good
end
```

Listing B.5: SourceCode/MaintainOrbit.m

```
function Vd=MaintainOrbit(GPS, GPS_tgt,R_Vo, Kp_Slop, GPS_ht)
%This function is used to maintain a CCW orbit around a specified target
%location.  The output is a set of ground speeds that will adjust the
%current positions of the vehicles to maintain an equally spaced orbit.
%Inputs:
%   GPS = nx2 array of current latitude/longitude positions of each
%   vehicle. Each row is [lat lon] and n is the number of vehicles.  All
%   latitude/longitude values are in decimal degrees.
%   GPS_tgt = 1x2 vector of the target location. [lat lon]
%   R_Vo = 1x2 vector of [Orbit_Radius Nominal_Speed] Radius is in meters,
%   Speed is in m/s.
%   Kp_Slop = 1x2 vector of [Proportional_Gain Angular_Slop].  Kp is a
%   multiplier on the angular error (e=Ang-Ang_des) Slop is the angular
%   window that is considered in the correct position. (e.g. 0.1745 rad (10
%   deg) is +/- 5 degrees is considered 'good')  This is in radians.
%Outputs:
%   Vd = 1xn vector of desired velocities (in m/s)

%Calc distances from center
numUAV=size(GPS);  numUAV=numUAV(1);
d=DistFromGPS([GPS_tgt; GPS], GPS_ht); %Calculate distances
d=d(2:numUAV);
%Calc angles from center
```

```
[pn pe]=GPS2Cart([GPS_tgt(1);GPS(:,1)]',[GPS_tgt(2);GPS(:,2)]',GPS_ht);

UAV_ang=atan2(pn,pe);%calc angular position for each UAV
UAV_ang=UAV_ang(2:(numUAV+1));
dUAV_ang=UAV_ang-UAV_ang(1); %adjust WRT first UAV os UAV1 is at 0 rad
dUAV_ang=Zeroto2pi(dUAV_ang);

[dUAV_ang_s iis]=sort(dUAV_ang, 'ascend');  % sort angle differences
ii=1:numUAV;
Ang_Des=(ii-1)*2*pi/numUAV;  %calculate desired angular position
Ang_Err=Ang_Des-dUAV_ang(iis);
Vd=zeros(1,numUAV)+R_Vo(2);
for ii=2:numUAV  %loop thru sorted list and
    if abs(Ang_Err(iis(ii))) < Kp_Slop(2)
        Vd(iis(ii))=R_Vo(2);
    else
        Vd(iis(ii))=R_Vo(2)+Kp_Slop(1)*Ang_Err(iis(ii));
    end
end
```

Listing B.6: SourceCode/Dubins.m

```
%% Dubins path calculator
function [Wp d Hdg_W]=Dubins(turn, GPS, Hdg, r, GPS_ht)
% Dubins path calculator
%   This function calculates a set of GPS waypoints and total distance for
%   a turn-straight-turn path(Dubins path) given an initial position and
%   heading.
% Inputs:
%   turn is a 2x2 vector denoting direction of turns. The first row
%      designates direction of turns. first number of first row denotes
%       initial turn, second denotes final turn.
%     For the first row
%       [1 1] = CCW CCW turn scheme, [-1 -1] = CW CW turn scheme, no
%       extra turns
%       [-1 1] = CW CCW turns, [1 -1] = CCW, CW turns
```

```
%     The second row denotes whether to add an extra turn to either end
%       [0 0] denotes no extra turns
%       [1 0] denotes extra turn on initial circle
%       [0 1] denotes extra turn on final circle
%     total turn mtx for CCW CCW/extra no extra is [1 1;1 0]
%   GPS is a 2x2 matrix of GPS coordinates of initial and final
%   coordinates.
%       [Lat_initial Lon_initial]
%       [Lat_final   Lon_final  ]
%   Hdg is a 1x2 vector with initial and final headings in radians
%       [Heading_initial Heading_final]
%   r is the radius of the turns.  r and d are in meters
%   GPS_ht is the reference height
%
% Outputs:
%   Wp is a 2xn set of GPS waypoints.  One waypoint is provided for every
%       90 deg of circle sweeped, one at the final circle entry, one at the
%       final destination, and one past the destination in the direction of
%       the final heading?????
%   d 1xn vector that denotes the distance remaining at each waypoint.
%   d(1) is the total distance, d(n) is zero.
%   A is the total radians swept in the path.

% convert GPS to meters
[y x]=GPS2Cart(GPS(:,1), GPS(:,2), GPS_ht);

% convert Heading to NED coordinates (+x = East, +y = North) where zero
% angle is along x axis
NEDHdg=pi/2-Hdg;

% find turn circle centers
    %circle 1
    x_c1=x(1)+r*cos(NEDHdg(1)+turn(1,1)*pi/2);
    y_c1=y(1)+r*sin(NEDHdg(1)+turn(1,1)*pi/2);
    %circle 2
```

```
        x_c2=x(2)+r*cos(NEDHdg(2)+turn(1,2)*pi/2);
        y_c2=y(2)+r*sin(NEDHdg(2)+turn(1,2)*pi/2);
52
    % find points of tangency between two circles
    if turn(1,1)*turn(1,2) == 1 %for initial and final circles in the same dir
        if turn(1,1) == 1  %for ccw on circle 1 and circle 2
            Ai_c1=Zeroto2pi(NEDHdg(1)+3*pi/2);
57          Af_c1=Zeroto2pi(atan2(y_c2-y_c1, x_c2-x_c1)+3*pi/2);
            Ai_c2=Af_c1;
            Af_c2=Zeroto2pi(NEDHdg(2)+3*pi/2);
            A1=Zeroto2pi(Af_c1-Ai_c1)+turn(2,1)*2*pi;
            A2=Zeroto2pi(Af_c2-Ai_c2)+turn(2,2)*2*pi;
62      else    % cw for circle 1 and cw for circle 2
            Ai_c1=Zeroto2pi(NEDHdg(1)+pi/2);
            Af_c1=Zeroto2pi(atan2(y_c2-y_c1, x_c2-x_c1)+pi/2);
            Ai_c2=Af_c1;
            Af_c2=Zeroto2pi(NEDHdg(2)+pi/2);
67          A1=-m2piToZero(Af_c1-Ai_c1)+turn(2,1)*2*pi;
            A2=-m2piToZero(Af_c2-Ai_c2)+turn(2,2)*2*pi;
        end
        d_str=norm([x_c2-x_c1 y_c2-y_c1]);
    else  %for circles in opposite direction
72      %check if points are too close for this
        if norm([x_c2-x_c1 y_c2-y_c1]) < 2*r
            sprintf('\nPoints are too close for cw/ccw turn--abort.');
            return
        end
77      if turn(1,1) == 1 % for ccw for circle 1, cw for circle 2
            %compute angle btwn circle centers and tangent line
            th=atan(r/norm([x_c2-x_c1 y_c2-y_c1]));
            Ai_c1=Zeroto2pi(NEDHdg(1)+3*pi/2);
            Af_c1=Zeroto2pi(atan2(y_c2-y_c1, x_c2-x_c1)+3*pi/2+th);
82          Ai_c2=Zeroto2pi(atan2(y_c2-y_c1, x_c2-x_c1)+pi/2+th);
            Af_c2=Zeroto2pi(NEDHdg(2)+pi/2);
            A1=Zeroto2pi(Af_c1-Ai_c1)+turn(2,1)*2*pi;
```

```
            A2=-m2piToZero(Af_c2-Ai_c2)+turn(2,2)*2*pi;
        else  %find angles for cw circle 1 and ccw circle 2
            th=atan(r/norm([x_c2-x_c1 y_c2-y_c1]));
            Ai_c1=Zeroto2pi(NEDHdg(1)+pi/2);
            Af_c1=Zeroto2pi(atan2(y_c2-y_c1, x_c2-x_c1)+pi/2-th);
            Ai_c2=Zeroto2pi(atan2(y_c2-y_c1, x_c2-x_c1)+3*pi/2-th);
            Af_c2=Zeroto2pi(NEDHdg(2)+3*pi/2);
            A1=-m2piToZero(Af_c1-Ai_c1)+turn(2,1)*2*pi;
            A2=Zeroto2pi(Af_c2-Ai_c2)+turn(2,2)*2*pi;
        end
        d_str=norm([(x_c2+r*cos(Ai_c2))-(x_c1+r*cos(Af_c1)) ...
            (y_c2+r*sin(Ai_c2))-(y_c1+r*sin(Af_c1))]);

    end

    % Calculate A
    A=A1+A2;

    % Calculate waypoints
        % calculate points on circle 1
        A1div=ceil(A1/(pi/2));
        ii=1:(A1div+1);
        x_wp=x_c1+r*cos(Ai_c1+A1*turn(1,1)*(ii-1)/A1div);
        y_wp=y_c1+r*sin(Ai_c1+A1*turn(1,1)*(ii-1)/A1div);
        d=r*A1*(ii-1)/A1div;
        if turn(1,1)==1
            Hdg_W=Zeroto2pi(2*pi-(Ai_c1+A1*(ii-1)/A1div));
        else
            Hdg_W=Zeroto2pi(pi-(Ai_c1-A1*(ii-1)/A1div));
        end

        %calculate points for circle 2 (including endpoint
        dii=length(x_wp);
        A2div=ceil(A2/(pi/2));
        ii=1:(A2div+1);
```

```
        x_wp=[x_wp x_c2+r*cos(Ai_c2+A2*turn(1,2)*(ii-1)/A2div)];
        y_wp=[y_wp y_c2+r*sin(Ai_c2+A2*turn(1,2)*(ii-1)/A2div)];
122     d(ii+dii)=r*A1+d_str+r*A2*(ii-1)/A2div;
        if turn(1,2)==1
            Hdg_W(ii+dii)=Zeroto2pi(2*pi-(Ai_c2+A2*(ii-1)/A2div));
        else
            Hdg_W(ii+dii)=Zeroto2pi(pi-(Ai_c2-A2*(ii-1)/A2div));
127     end


        %convert to GPS coordinates
        d=d(length(d))-d;
132     [Wp(1,:) Wp(2,:)]=Cart2GPS([GPS(1,1) GPS(1,2)], x_wp, y_wp, GPS_ht);
```

Listing B.7: SourceCode/Orbit_r.m

```
    function [r phi]=Orbit_r(h, Theta_s, Vg)
 2  %This function solves for the orbit radius and bank angle of an orbiting
    %airplane/UAV given an altitude(h), a side sensor depression angle
    %(theta_s), and a ground speed(Vg).  This function assumes that the bank
    %angle is small and only returns the larger radius (the solution actually
    %has two solutions)  All answers are given in meters and radians.  Solution
 7  %also assumes that altitude is constant.
    %Inputs:
    %   h = altitude [meters]
    %   Theta_s = depression angle of side sensor
    %   Vg = ground speed
12  %Outputs:
    %   r = orbit radius
    %   phi = bank angle

    %solve the quadratic equation--give only larger solution
17  g=9.81;   % acceleration of gravity
    a=1;
    b=-(h*tan(Theta_s)+Vg^2/g*tan(Theta_s));
    c=-h*Vg^2/g;
```

```
r=(-b+sqrt(b^2-4*a*c))/(2*a);
phi=atan(Vg^2/(g*r));
```

Listing B.8: SourceCode/Cart2GPS.m

```
% Cartesian to GPS conversion
function [Lat Lon]=Cart2GPS(GPS_o, x, y, GPS_ht)
% This function generates GPS coordinates [Lat Lon] from a given GPS origin
% GPS_o [Lat Lon], x y coordinates (in meters), and a GPS reference height.
% The GPS origin must correlate with the xy origin.
% This function is only good for small distances (>1000km)

%set the origin
Lat_o=GPS_o(1)*pi/180;
Lon_o=GPS_o(2)*pi/180;

%Calculate Rm, Rp
a=6378135;  %equatorial radius in meters
e=0.0818191908426;
Rm=(a*(1-e^2))/(1-e^2*sin(Lat_o)^2).^1.5;
Rp=a/(1-e^2*sin(Lat_o)^2)^0.5;

Lat=Lat_o+y/(Rm+GPS_ht);
Lon=Lon_o+x/((Rp+GPS_ht)*cos(Lat_o));

% convert back to deg
Lat=Lat*180/pi;
Lon=Lon*180/pi;
```

Listing B.9: SourceCode/GPS2Cart.m

```
%% GPS2Cart

function [pn pe]=GPS2Cart(Lat, Lon, h_o)
% This function converts a set of latitude, longitude, height values to
% cartesian pn, pe values in meters.  The first point is set as the
% origin.  Equations come from slide 1-29 of  EENG 533 from Spring 2008.
```

```
% this assumes Lat/Lon are in degrees.  h_o is the GPS height of the first
% point. This function is only good for small distances (>1000km)

%Convert to radians
Lat=Lat*pi/180;
Lon=Lon*pi/180;

%set the origin
Lat_o=Lat(1);
Lon_o=Lon(1);


%Calculate Rm, Rp
a=6378135;  %equatorial radius in meters
e=0.0818191908426;
Rm=(a*(1-e^2))/(1-e^2*sin(Lat_o)^2).^1.5;
Rp=a/(1-e^2*sin(Lat_o)^2)^0.5;
pe=(Rp+h_o)*cos(Lat_o)*(Lon-Lon_o);
pn=(Rm+h_o)*(Lat-Lat_o);
```

Appendix C. Simulink Model Matlab® Code

This appendix contains the Matlab® code used for the Simulink model.

C.1 BATCAM Closed Loop Model

Listing C.1: SourceCode/BATCAM_S.m

```
function dy=BATCAM_S(y)
%Calculates the derivatives of the closed-loop BATCAM model state
%The state:
%   y(1)= pN  North position
%   y(2)= pE  East position
%   y(3)= h  Altitude
%   y(4)= h dot
%   y(5)= Va Airspeed
%   y(6)= R  Roll angle
%   y(7)= P  Pitch angle
%   y(8)= Y  Yaw Angle (clock angle)
%Inputs
%   y(9)= Vac Commanded airspeed
%   y(10)= Rc  Commanded roll angle
%   y(11)= Pc  Commanded Pitch angle
%   y(12)= wN North component of wind
%   y(13)= wE East component of wind
%Outputs
%   dy(1)= pN dot
%   dy(2)= pE dot
%   dy(3)= h dot
%   dy(4)= h dotdot
%   dy(5)= Va dot
%   dy(6)= R dot
%   dy(7)= P dot
%   dy(8)= Y dot


%First Order Constants
```

```
kV=1.3; % 95% of commanded in 2.3 seconds
kR=2.3;
kP=.865;

% roll wind disturbance
kwR=1;  % 1=1 degree of disturbance for every m/s of wind
wR=kwR*0.0174*norm([y(11) y(12)]);

%BATCAM properties
m=0.425;
g=9.81;
rho=1.204;  %air density in metric
S=8.73*5.94*2*2.54^2/100^2; %wing area [m^2]
%b=21*2.54/100;
%c_bar=5.94*2.54/100;  %mean geometric chord [m]
q_bar=rho/2*y(5)^2; %free stream dynamic pressure
%d=8.5*2.54/100;  %prop diameter

%Calculate lift
alpha=0;
CL=5.73*alpha + 0.7;
L=q_bar*S*CL; %Lift



dy(1)=y(5)*cos(y(8))*cos(y(7))+y(12);
dy(2)=y(5)*sin(y(8))*cos(y(7))+y(13);
dy(3)=y(5)*sin(y(7));
dy(4)=L*cos(y(6))*cos(y(7))/m - g;
dy(5)=kV*(y(9)-y(5));
dy(6)=kR*(y(10)-y(6))+wR;
dy(7)=kP*(y(11)-y(7));
dy(8)=g/y(5)*tan(y(6));
```

C.2 Mode 2 Controller

Listing C.2: SourceCode/CC_Controller1.m

```
function Cmd=CC_Controller1(yin)
%This function is used in conjunction with Simulink model BATCAM_1 to model
%the velocity feedback algorithm
%Inputs:
%   yin(1:2) = wind vector [wNorth wEast]
%   yin(3:5) = position of UAV1 [pN pE Yaw]
%   yin(6:8) = position of UAV2 [pN pE Yaw]
%   yin(9:11) = position of UAV3 [pN pE Yaw]
%   yin(12:14) = position of UAV4 [pN pE Yaw]
%Outputs
%   [pNd pEd] = desired location for the UAV
%   Chi_s = desired path heading
%   Vac = commanded airspeed
%   hc = commanded altitude
%   Cmd(1:5) = [pNd pEd Chi_s Vac hc] for UAV1
%   Cmd(6:10) = [pNd pEd Chi_s Vac hc] for UAV2
%   Cmd(11:15) = [pNd pEd Chi_s Vac hc] for UAV3
%   Cmd(16:20) = [pNd pEd Chi_s Vac hc] for UAV4

%Set up target and desired paths
pN_d=200;
pE_d=200;
pN_i=[300 250 200 0]';
pE_i=[0 0 0 50]';
dpN=pN_d-pN_i;
dpE=pE_d-pE_i;
Chi_s=atan2(dpE,dpN);
hc=50;

%Find commanded velocities
kV=.12;
V_nom=11.75;
dpN_u=[yin(3) yin(6) yin(9) yin(12)]'-pN_d;
dpE_u=[yin(4) yin(7) yin(10) yin(13)]'-pE_d;
```

```
d_i=[norm([dpE_u(1) dpN_u(1)]);
    norm([dpE_u(2) dpN_u(2)]);
    norm([dpE_u(3) dpN_u(3)]);
    norm([dpE_u(4) dpN_u(4)])];
d_mean=mean(d_i);
Vgc=V_nom-kV*(d_mean-d_i);
Y=[yin(5) yin(8) yin(11) yin(14)]';
Vw=[yin(1) yin(2);yin(1) yin(2);yin(1) yin(2);yin(1) yin(2)];
Vac_v=[Vgc.*cos(Y) Vgc.*sin(Y)]-Vw;
Vac=[norm(Vac_v(1,:));norm(Vac_v(2,:));norm(Vac_v(3,:));norm(Vac_v(4,:))];
for ii=1:4
    if Vac(ii)> 21.75
        Vac(ii)=21.75;
    end
end

%Create output
Cmd(1:5) = [pN_d pE_d Chi_s(1) Vac(1) hc];
Cmd(6:10) = [pN_d pE_d Chi_s(2) Vac(2) hc];
Cmd(11:15) = [pN_d pE_d Chi_s(3) Vac(3) hc];
Cmd(16:20) = [pN_d pE_d Chi_s(4) Vac(4) hc];
Cmd=Cmd';
```

C.3 Mode 3 Controller

Listing C.3: SourceCode/CC_Controller2.m

```
function Cmd=CC_Controller1(yin)
%This function is used in conjunction with Simulink model BATCAM_1 to model
%the velocity feedback algorithm
%Inputs:
%   yin(1:2) = wind vector [wNorth wEast]
%   yin(3:5) = position of UAV1 [pN pE Yaw]
%   yin(6:8) = position of UAV2 [pN pE Yaw]
%   yin(9:11) = position of UAV3 [pN pE Yaw]
%   yin(12:14) = position of UAV4 [pN pE Yaw]
```

```
%Outputs
%   [pNd pEd] = desired location for the UAV
%   Chi_s = desired path heading
%   Vac = commanded airspeed
%   hc = commanded altitude
%   Cmd(1:5) = [pNd pEd Chi_s Vac hc] for UAV1
%   Cmd(6:10) = [pNd pEd Chi_s Vac hc] for UAV2
%   Cmd(11:15) = [pNd pEd Chi_s Vac hc] for UAV3
%   Cmd(16:20) = [pNd pEd Chi_s Vac hc] for UAV4

%Set up target and desired paths
pN_d=0;
pE_d=0;
hc=50;

%Find commanded velocities
V_nom=11.75;
pN_u=[yin(3) yin(6) yin(9) yin(12)]';
pE_u=[yin(4) yin(7) yin(10) yin(13)]';
pNpE=[pN_u pE_u];
pNpE_tgt=[pN_d pE_d];
Vgc=MaintainOrbit2(pNpE, pNpE_tgt,V_nom,[.5 .1]);
Y=[yin(5) yin(8) yin(11) yin(14)]';
Vw=[yin(1) yin(2);yin(1) yin(2);yin(1) yin(2);yin(1) yin(2)];
Vac_v=[Vgc'.*cos(Y) Vgc'.*sin(Y)]-Vw;
Vac=[norm(Vac_v(1,:));norm(Vac_v(2,:));norm(Vac_v(3,:));norm(Vac_v(4,:))];
for ii=1:4
    if Vac(ii)> 21.75
        Vac(ii)=21.75;
    end
end
%Find Desired path heading Chi_s
ii=1:4;
Ang_d=-(ii-1)*pi/2;
Chi_s=atan2(pE_u,pN_u)-pi/2;
```

```
   %Create output
47 Cmd(1:5) = [pN_d pE_d Chi_s(1) Vac(1) hc];
   Cmd(6:10) = [pN_d pE_d Chi_s(2) Vac(2) hc];
   Cmd(11:15) = [pN_d pE_d Chi_s(3) Vac(3) hc];
   Cmd(16:20) = [pN_d pE_d Chi_s(4) Vac(4) hc];
   Cmd=Cmd';
```

Bibliography

1. *Kestrel Autopilot System - Kestrel User Guide, Kestrel Autopilot Virtual Cockpit ver 2.4.2.*
2. AFRL/RYAR. "OHARA Cooperative Control". *Unmanned Systems - AFRL UAV Workshop 2008.*
3. Associates, Applied Research. *Battlefield Air Targeting Camera Autonomous Micro Unmanned Aerial Vehicle (BATCAM UAV) Operators Manual.* Applied Research Associates, 2006.
4. Beard, Randal W. "Multiple UAV Cooperative Search under Collision Avoidance and Limited Range Communication Constraints". *2003 IEEE Conference on Decision and Control.*
5. DeLuca, Anthony M. *Experimental Investigation into the Aerodynamic Performance of Both Rigid and Flexible Wing Structured Micro-Air-Vehicles.* Master's thesis, Air Force Institute of Technology, 2004.
6. Dubins, L. E. "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents". *American Journal of Mathematics*, 1957.
7. Farrell, Shannon. *Flying Unmanned Aerial Vehicles based on Sensor Aimpoint.* Master's thesis, Air Force Institute of Technology, 2009.
8. Jodeh, Nidal M. Capt USAF. *Development of Autonomous Unmanned Aerial Vehicle Research Platform: Modelling, Simulating, and Flight Testing.* Master's thesis, AFIT, 2006.
9. Kingston, Randall Beard Ryan Holt, Derek. "Decentralized Perimeter Surveillance Using a Team of UAVs". *IEEE Transactions on Robotics*, 2005.
10. Mathworks, Inc. *Aerospace Blockset 3, User's Guide.* Mathworks, Inc, 2008.
11. Ogata, Katsuhiko. *Modern Control Engineering, 2nd Ed.* Prentice Hall, 1990.
12. Potter, Merle C and David C Wiggert. *Mechanics of Fluids.* Prentice Hall, 1991.
13. Raquet, John. "EENG 533: Navigation Using the GPS - Course Handouts, Spring 2008". EENG 533 Course, Air Force Institute of Technology.
14. Rasmussen, J.W. Mitchell P.R. Chandler C.J. Schumacher A.L. Smith, S.J. "Introduction to th eMultiUAV2 Simulation and Its Application to Cooperative Control Research". *2005 American Control Conference.*
15. Richards, John Bellingham Michael Tillerson Jonathan How, Arthur. *Coordination and Control of Multiple UAVs.* Technical report, Massachusetts Institute of Technology, 2002.
16. Rysdyk, Rolf. "UAV Path Following For Constant Line of Sight". *AIAA*, 2003.

17. Sakryd, Gregory A. LCDR USN and Doug Ericson Capt USAF. *System Engineering for the Fleeting Target Technology Demonstrator*. Master's thesis, AFIT, 2008.

18. Stevens, Brian L and Frank L Lewis. *Aircraft Simulation and Control*. John Wiley Sons, Inc., 1992.

19. Terning, Nate A. Capt USAF. *Real Time Path Optimization for Tracking Stop-and-Go Targets with Micro Air Vehicles*. Master's thesis, AFIT, 2008.

20. Vantrease, Troy H. *Development and Employment of a Semi Autonomous Cursor On Target Navigation System for Micro Air Vehicles*. Master's thesis, AFIT, 2008.

21. Zollars, Michael D. 1 Lt USAF. *Optimal Wind Corrected Flight Path Planning for Autonomous Micro Air Vehicles*. Master's thesis, AFIT, 2007.

Vita

Capt Chris Booth graduated from the University of Utah in 1993 with a Bachelor's in Mechanical Engineering with an emphasis in controls. In 2006, he completed a Master of Science in Aerospace Engineering at North Carolina State University with an emphasis on structures. Between 1993 and 2001 he worked in the wood truss industry as both a designer and design manager. He joined the Air Force in 2001.

His first assignment was at Robins AFB in Georgia as both manager of the Aircraft Battle Damage Repair (ABDR) program and a Structures Engineer in C-130 System Program Office. Duties included maintaining wartime readiness for the ABDR engineer group in addition to designing structural repairs for C-130s undergoing Depot level maintenance. In 2003, Capt Booth deployed in support of OPERATION IRAQI FREEDOM supporting the 410th Expeditionary Air Wing. Wartime duties engineer included ABDR and Depot liaison for HC-130s tasked to the Search and Rescue mission.

The second assignment in 2004 was to the 4th Space Launch Squadron supporting the new Evolved Expendable Launch Vehicle (EELV) Program at Vandenberg AFB. He served as the Range Support Engineer resolving telemetry, flight safety, system safety, communications, and environmental issues associated with standing up the new Atlas V and Delta IV launch complexes. He was involved in the successful launches of NROL-22 and DMSP-17 missions. Moving to the 30th Launch Support Squadron as the Launch Infrastructure Engineer, he coordinated with base agencies to resolve infrastructure issues to launch pads. This position supported multiple Minotaur and Missile Defense Agency launches.

In September 2007 he entered the School of Engineering and Management at the Air Force Institute of Technology.

Permanent address: 2950 Hobson Way
Air Force Institute of Technology
Wright-Patterson AFB, OH 45433

NONPRINT FORM

1. Type of Product:	2. Operating System/Version:	3. New Product or Replacement:	4. Type of File:
CD	Windows/MAC/Linux	New Product	many

5. Language/Utility Program:
N/A

6. # of Files/# of Products:	7. Character Set:	8. Disk Capacity:
		700 MB
	9. Compatibility: ISO 9660	10. Disk Size:

11. Title:
Surveillance Using Multiple Unmanned Aerial Vehicles

12. Performing Organization:	13. Performing Report #:	14. Contract #:
Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/ENY) 2950 Hobson Way WPAFB OH 45433-7765		15. Program Element #:

16. Sponsor/Monitor:	17. Sponsor/Monitor # Acronym:	19. Project #:
Left intentionally Blank	18. Sponsor/Monitor #:	20. Task #:
		21. Work Unit #:

22. Date:	23. Classification of Product:
26 March 2009	Unclassified
24. Security Classification Authority:	**25. Declassification/Downgrade Schedule:**

26. Distribution/Availability:

DTIC FORM 530 JUN 94 | Security Classification of this Page:

NONPRINT FORM

27. Abstract:

The CD contains MATLAB source code files, C++ source code files and executables associated with the AFIT Master's Thesis "Surveillance Using Unmanned Aerial Vehicles" written by Capt Christopher Booth in Mar 2009.

28. Classification of Abstract:	**29. Limitation of Abstract:**
Unclassified	
30. Subject Terms:	**30a. Classification of Subject Terms:**

31. Required Peripherals:

32. # of Physical Records:	**33. # of Logical Records:**	**34. # of Tracks:**
35. Record Type:	**36. Color:**	**37. Recording System:**
38. Recording Density:	**39. Parity:**	**40. Playtime:**

41. Playback Speed:	**42. Video:**	**43. Text:**	**44. Still Photos:**	**45. Audio:**

46. Other:

47. Documentation/Supplemental Information:

AFIT Master's Thesis "Surveillance Using Unmanned Aerial Vehicles", Mar 2009

48. Point of Contact and Telephone Number:

Richard G. Cobb
richard.cobb@afit.edu
937-255-3636x4559

DTIC FORM 530 JUN 94 | Security Classification of this Page:

REPORT DOCUMENTATION PAGE		Form Approved OMB No. 0704–0188

1. REPORT DATE (DD–MM–YYYY)	2. REPORT TYPE	3. DATES COVERED (From — To)
26-03-2009	Master's Thesis	Sep 07 – Mar 09

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Surveillance Using Multiple Unmanned Aerial Vehicles	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
Booth, Christopher E., Capt, USAF	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/ENY) 2950 Hobson Way WPAFB OH 45433-7765	8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/GSS/ENY/09-M02
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Intentionally left blank	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This study examines the performance and limitations of a heuristic cooperative control (CC) surveillance algorithm for multiple unmanned aerial vehicles (UAVs) under both simulation and demonstration. The algorithm generates Dubin's based paths and provides velocity feedback to accomplish simultaneous arrival onto a surveillance orbit around the target and maintains position while orbiting. TheCC algorithm has two modes: one that generates commands to multiple UAVs for simultaneous arrival to a surveillance orbit, and one that maintains equal angular spacing about the orbit. In addition to positional performance metrics, percentage of target in-view time was also measured based on the UAV's side camera field of view (FOV). Simulation tested both modes under wind conditions of 0%, 10%, 25%, and 50% of the nominal airspeed (V_{nom}). Results showed that the algorithm maintained UAV position with winds 25% of V_{nom}, but instabilities appeared at 50% where large overshoots appeared on the downwind side of the orbit. Target visibility was most impacted by crosstrack errors that steadily grew with increasing winds. Roll of the UAV showed the greatest impact on the FOV due to its coupling effect with crosstrack error. Overall target in-view time also improved with increasing numbers of UAVs for all wind conditions.

15. SUBJECT TERMS
UAV, MAV, cooperative control, surveillance, fixed target, algorithm, model

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE	UU	173	Richard G. Cobb
U	U	U			19b. TELEPHONE NUMBER (Include Area Code) Richard.cobb@afit.edu 937-255-3636x4559