WAYPOINT GENERATION BASED ON SENSOR AIMPOINT

THESIS

Shannon M. Farrell, Captain, USAF

AFIT/GAE/ENV/09-M01

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GAE/ENV/09-M01

WAYPOINT GENERATION BASED ON SENSOR AIMPOINT

THESIS

Presented to the Faculty

Department of Aeronautical Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Aeronautical Engineering

Shannon M. Farrell, BS

Captain, USAF

March 2009

AFIT/GAE/ENV/09-M01

WAYPOINT GENERATION BASED ON SENSOR AIMPOINT

Shannon M. Farrell, BS
Captain, USAF

Approved:

//signed//	10 Mar 2009
David R. Jacques (Chairman)	Date
//signed//	10 Mar 2009
Richard G. Cobb (Member)	Date
//signed//	10 Mar 2009
Lt Col Frederick G. Harmon (Member)	Date

Abstract

Secretary of Defense Robert M. Gates has emphasized a need for a greater number of intelligence, surveillance, and reconnaissance (ISR) assets to support combatant commanders and military operations globally. Unmanned systems, especially MAVs, used as ISR platforms provide the ability to maintain covertness during missions and help reduce the risk to human life. This research develops waypoint generation algorithms required to keep a point of interest (POI) in the field of view (FOV) of a fixed sensor on a micro air vehicle (MAV) in the presence of a constant wind.

Fixed sensors, while cheaper and less prone to mechanical failure than gimbaled sensors, provide challenges to keeping a POI in the FOV in the presence of wind since the MAV must adjust its attitude to maintain a desired flight path. As the vehicle adjusts its attitude, the POI may shift outside of the FOV. Wind affects MAVs more than other air vehicles because wind speeds can quickly approach the slow cruise speed of the vehicle.

This research builds upon previous work of maintaining a sensor aimpoint at a POI in the presence of wind. The algorithms establish waypoints at which a fixed sensor on a MAV at that waypoint points directly at the POI. As wind varies and the MAV does not perfectly reach each waypoint and attitude condition, the POI needs to remain in the FOV, ideally near the sensor aimpoint. Two scenarios are explored: one where the MAV orbits the POI, and the second where the MAV flies to align the sensor at the POI along a preferred look angle. The first scenario relies on a sensor aimed out the side of the MAV, while the second considers both a side-viewing sensor and a forward-viewing sensor. The research explored each scenario through hardware-in-the-loop testing as well as flight testing. The results indicate the algorithms keep the POI in the FOV at least 66% of the time as long as the wind speed was less than 25% of the cruise speed; at higher winds, the MAV has a difficult time tracking the flight path generated by the sensor aimpoint algorithm.

Acknowledgements

Thank you to the people who helped get me here today, particularly Col Robert Fredell, Col Jerry Couick, and Lt Col Dale Wright. Your mentorship, belief in me, and encouragement throughout all of my endeavors, especially attending AFIT, have provided no small amount of support as I encountered new challenges and opportunities.

Many thanks to Dr. David Jacques for his guidance throughout this research and all of my time at AFIT. Your insights, hard-learned lessons, and stories have helped me in more ways than simply completing this academic achievement. Your understanding, flexibility, and support through difficult times were greatly appreciated. You knew when to tell me to back off and focus on more important things and you knew when I needed to hear that. You helped keep me tracking the important goals and eliminate the chaff.

Thanks to Lt Col(s) Garry "Fozzy" Haase, Maj Steve Behm, Maj Adam Lenfesty, and Maj Brad "Buster" Pitzer for the many great Tuesday afternoon mentorship sessions. Those chances to get away and complete a reality check once a week were crucial to keeping sane while at AFIT.

The support of LCDR Ted Diamond, Maj Adam Rutherford, Maj Brett "Beaver" Taylor, and Capt Chris Booth has been instrumental in both the development and testing of this research as well as keeping a fun attitude in the ANT Lab. You were a great team that made hard work fun, especially the long drives to Camp Atterbury.

Lastly, thank you to my wonderful wife and our fantastic baby boy. Thank you to my son, who helped give me a better reason for working so hard here to finish this work. My greatest thanks, appreciation, and love goes to my wife. Without you, I could not have done this. From your taking care of big and little things to allow me to focus on my research and writing, to the copy editing and sanity checks you provided for that writing, I cannot thank you enough. Without your love and support, this would not have been possible, nor would I have ever wanted to attempt it. I love you.

Shannon M. Farrell

Table of Contents

Page

Page

Page

List of Figures

List of Tables

List of Symbols

List of Abbreviations

WAYPOINT GENERATION BASED ON SENSOR AIMPOINT

I. Introduction

1.1 Motivation

The prevalence of using unmanned systems in recent years has risen due to their low cost of acquisition, low cost of operation, and the ability to keep operators out of harms way. In a speech in 2008, Secretary of Defense Robert M. Gates cited the need for more use of unmanned systems in the Department of Defense (DoD). Specifically, he stated a need for additional intelligence, surveillance, and reconnaissance (ISR) assets to support combatant commanders and military members deployed around the world. Secretary Gates further called for the military to do "more to meet the needs of men and women fighting in the current conflicts while their outcome may still be in doubt."

The DoD continues to show strong emphasis in the development and operation of unmanned systems. The *Unmanned Systems Roadmap 2007-2032* (Office of the Secretary of Defense, 2007) noted that combatant commanders "highly desired" unmanned systems to perform a variety of battlefield roles, with the highest priority given to reconnaissance and surveillance. The ability to maintain covertness and reduce the risk of human life helped motivate the need for unmanned ISR systems. The second priority was target identification and designation. The report noted that this mission was a shortfall of current DoD unmanned aerial systems (UAS). The third priority was counter-mine warfare, to include the ability to defeat improvised explosive devices, a major threat to currently deployed forces.

The DoD report broke down classifications of UASs into four categories, based on gross takeoff weight. A small UAS, also commonly referenced as a micro aerial vehicle (MAV), was less than 55lbs and a tactical UAS ranged from 55lbs to 1320lbs. Previous research at AFIT along with on-going efforts (Diamond *et al.*, 2009) are exploring a general solution — that is not a point-solution design — for a tactical, cooperative, small, ISR UAS. The proposed system will provide a scalable ISR asset or system of

assets to tactical units from a single operator to company-sized military units. This research provides a component of these broader efforts.[1]

While using a tactical ISR MAV, the operator will more than likely identify a point of interest (POI) about which the operator would like to get more information. The operator may want to view the object from multiple angles or overfly the object. This research aims to develop a simple human-machine interface, another area cited by the DoD report as an area for future enhancements, to enable the rapid switching from flying pre-planned mission routes to surveilling a point of interest for an operator who controls one or more tactical ISR MAVs.

1.2 Enabling CONOPS

This research extends the general concept of operations developed by Diamond *et al.* (2009) by adding the capability to focus on a single point of interest during the tactical ISR patrol. The research simplifies the operator interface to control a particular MAV, allowing the operator to remove it from the patrol route, orbit or overfly a point of interest, and then resume the planned route.

1.2.1 Employment Scenarios. To help understand the motivation for this research, three scenarios have been developed and written. These scenarios describe situations in which a tactical combat unit would use the MAV system to gather real-time ISR data. These scenarios will provide a foundation for the more generic use cases developed later in section 3.2.

1.2.1.1 Scenario 1. A soldier in a small fire-team uses the MAV for real-time tactical ISR. While flying the MAV in an ISR pattern along the fire-team's projected patrol route, the soldier sees an object of interest in the sensor display that warrants additional surveillance. The soldier instructs the MAV to orbit the POI. The MAV sets up an orbit to keep the POI centered in the sensor display. During this time,

[1]Tactical, as used here, refers to the unit being supported and the ISR application, not the classification of the UAS.

the soldier views the POI from a variety of aspect angles. After deciding the POI is a non-factor, the soldier instructs the MAV to resume the previously programmed route.

1.2.1.2 Scenario 2. An operator uses the MAV to provide tactical ISR regarding an enemy encampment. The operator identifies a POI and wants a second look at the object. Not wanting to attract attention by orbiting the target, the operator instructs the MAV to overfly the POI with a single pass. Having positively identified the POI, the operator instructs the MAV to return to base.

1.2.1.3 Scenario 3. An operator, using the MAV for real-time tactical ISR during a patrol of a small town, sees a POI down a side-street through the side-view sensor. The operator instructs the MAV to overfly the POI, defining a set approach angle and altitude, to get a second view of the POI. The second look at the object indicates it is not of interest, so the operator tells the MAV to resume the pre-planned patrol route.

1.3 Research Objectives

This research aims to develop waypoint generation algorithms to keep a sensor aimed at a POI. These algorithms are developed for a tactical ISR MAV system in support of the route surveillance research currently being conducted by Diamond *et al.* (2009). This algorithm will be implemented as a software module that will work with the MAV control system used by that group. The algorithm will allow an operator to easily orbit or overfly a point of interest using a preferred sensor orientation while the MAV is flying a pre-planned route. The software should allow this operation with minimal effort or learning curve.

1.4 Assumptions

The author made several assumptions for this research effort, based on limitations of the hardware used as well as self-imposed limitations to scope the research effort. Many of these assumptions will provide a basis for future research efforts, as discussed in section 5.2.

1.4.1 Multiple MAVs or Cooperative Control. A single MAV served as the research case. Cooperative control, multiple MAV dynamic tasking, and human systems integration aspects remained outside the scope of this research, but are being addressed in other related research currently ongoing at AFIT.

1.4.2 Sensors. The sensors used had a fixed field of view and field of regard. The sensors lacked a zoom capability, either digital or optical. The MAV contained the sensors at a fixed angle of declination.

1.4.3 Target. The target was always fixed; therefore, the software did not require target-cueing capabilities or moving target algorithms. Instead of orbiting or overflying a given object of interest, the software identified the object's geographic coordinates and created a POI.

1.4.4 Obstructions. The POI was always in an open area without terrain obstructions. This allowed the MAV to not require terrain knowledge or the ability to avoid object collisions.

1.5 Thesis Outline

In this chapter, the motivation for the research was given, along with the research objectives and assumptions. Chapter II reviews what previous research has discovered on the subjects of the general systems engineering architectures for the route surveillance problem, path planning for MAVs, and algorithms for aiming the aircraft sensor at a specific target. The methodology of this research is described in Chapter III, to include concept refinement, development of use cases, the algorithms, and simulation and testing. Chapter IV analyzes the results from the simulations and tests. Finally, Chapter V summarizes this research and provides conclusions and suggestions for future work.

II. Background

This chapter discusses reports and papers of previous and one ongoing research efforts that compliment this research in sensor aimpoint flight planning. The research includes previous and current systems engineering research into applications of using MAVs in a tactical ISR role, as well as previous research into sensor resolution trade space, path planning, sensor aimpoint, and wind correction topics.

2.1 Fleeting Target Demonstrator

In 2006, the Air Force Special Operations Command and Air Force Research Lab produced an urgent need initiative to provide the ability to engage high-value, time-sensitive targets in near line-of-sight (500 m to 5 km) ranges. Abbott *et al.* (2007) developed a systems engineering plan to transition the initiative, known as the Fleeting Target Technology Demonstrator, to a traditional acquisition process. This systems engineering plan translated an operational need into an actionable systems architecture with a recommendation to use the Battlefield Air Targeting Camera Autonomous Micro-Air Vehicle (BATCAM).

As part of their systems engineering analysis, Abbott *et al.* conducted a risk assessment of how the BATCAM could meet the system requirements. The most challenging technical risk was the ability of an operator to detect and identify a given target using the BATCAM video sensors and computer interface. Abbott *et al.* conducted a sensor analysis of the BATCAM's fixed forward-viewing and side-viewing cameras to determine at what slant distance an operator could both detect and identify a target. They concluded that an operator would have to fly the BATCAM below 100m in altitude and would still have to orbit using the side camera to allow sufficient time to positively identify a target. Alternatively, they found that a higher resolution sensor could be used and would allow the operator to detect and identify targets at greater ranges.

Sakryd and Ericson (2008) continued researching the fleeting target problem and so extended the research of Abbott *et al.*. One emphasis of the later group's efforts was to focus on the software engineering aspects of the Fleeting Target problem and to decompose the system through use cases. As part of the research, Sakryd and Eric-

son developed a demonstrator for use by the AFIT Advanced Navigation Technologies (ANT) center based on the Fleeting Target architecture developed by Abbott *et al.*. The demonstrator system incorporated several recommendations from Abbott *et al.*, including a higher resolution sensor. Additional recommendations were either researched by others, such as Terning (2008) and Vantrease (2008), and incorporated into the final solution, or were developed by Sakryd and Ericson as part of their efforts.

The system demonstrator did not use the BATCAM, but instead modified a SIG Rascal 110 airframe, originally characterized by Jodeh (2006) for the ANT center. One of the modifications to the Rascal was to use the Procerus Kestrel (Procerus Technologies, 2008) autopilot instead of a CloudCap Piccolo autopilot. The primary reason for this change was that the BATCAM already used the Kestrel. The testbed Rascal was further modified with Black Widow KX141 analog video cameras mounted on uniaxial gimbals. Lastly, the demonstrator included a ground station unit consisting of a laptop and communications support devices.

One of the great advantages of the demonstrator system was the ability to write software algorithms in MATLAB, a standard tool in engineering programming, and have them interface to the Procerus Virtual Cockpit, the command and control software for the Kestrel autopilot. Terning and Vantrease, discussed below, both took advantage of this capability when writing their algorithms. The demonstrator was tested using hardware-in-the-loop (HIL) simulation as well as flight testing.

2.2 Route Surveillance

Diamond *et al.* (2009) sought to expand the previous research done by Abbott *et al.* (2007) and Sakryd and Ericson (2008) on the Fleeting Target Demonstrator by expanding the application of MAVs for a wide range of tactical ISR tasks. Their efforts were ongoing at the same time as the research in this report, and the author jointly worked with the group. This research supported the route surveillance problem discussed further in section 3.2.

The new research considered the problem of using one or more MAVs in cooperation to patrol a route which could be defined as a road, a perimeter, or an area. These MAVs

would transmit real-time sensor data to one or more base stations which would have either aircraft operators, sensor operators, or an individual trained to do both tasks. The fleeting target problem was a subset of the route surveillance problem, focused on the intercept of an identified and located moving target.

2.3 Sensor Resolution Trade Space

Abbott *et al.* (2007) investigated requirements for detection and identification of targets using visual sensors. They used the Civil National Imagery Interpretability Rating Scale (NIIRS) Guide for definitions of both detection and identification. The basic difference was that detection meant "the capability to find or discover the presence or existence of an" object, while identification meant "the capability to name an object by type or class." Equations were presented to determine the maximum range at which a target could be detected or identified based on the density of pixels on the target. The sensor resolution was based on a sensor that was currently in use by researchers at the Air Force Academy and set to 330 lines. Basing values on previous research by the Air Force Research Labortory for the Low Cost Autonomous Attack System, Abbott *et al.* determined it would take a minimum of 2-3 pixels per square meter for object detection and 10-15 pixels per square meter for identification. Abbott *et al.* evaluated the maximum detection and identification slant range for sensors with the same resolution, but where one had a 30° FOV and the other had a 60° FOV. As the FOV increased, the pixel density decreased, since the same number of pixels were spread out across a larger area. The results for maximum slant range to detect and identify a target are given in Table 2.1. The spread of the ranges resulted from the spread in required pixels for both detection and identification of a target and are based on the specific camera used for this research, as discussed above.

Table 2.1: Maximum Range to Detect and Identify

FOV	**Detect**	**Identify**
30°	367 m-449 m	164m-202 m
60°	182 m-224 m	82m-100 m

Abbott *et al.* evaluated how the change in altitude affected the ability of an operator to detect and identify a target. They assumed no depression angle on the sensor so at a higher altitude, the MAV could detect but not identify a target. As the MAV climbed in altitude, the sensor pixel resolution decreased. In some cases, however, a specific flight plan may not be accomplished except above a threshold altitude. In these instances, the operator must choose between putting a sensor on the point of interest with decreased pixel density and image recognition, or not being able to fly a path to see the point of interest at all.

If the MAV had a depressed sensor, the detection may not occur until the target was closer to the MAV but the target would be easier to identify. Abbott *et al.* discussed depressing the sensor to improve detection and identification and commented that the offset was a potential decreased ability to fly the MAV, since they assumed the operator would fly the MAV using visual cues from the sensor. They discovered another way to improve the sensor resolution would be to put a better sensor on the MAV. Changing from a sensor with a resolution of 330x330 to 768x494 improved the detection range by 50% for both the 30° and 60° FOV situations.

Without the consideration of a better camera and assuming a depressed sensor, the higher a MAV flies, the worse the sensor resolution, but the longer a point of interest will remain in the sensor FOV. This may be offset, to a degree, with a sensor which provided better resolution. The end result, however, was a trade space of altitude versus sensor resolution and time to make a discrimination decision.

2.4 Path Planning

2.4.1 Targets within Aircraft Turn Radius. Many path planning algorithms, such as the one developed by Burns (2007), assume targets lie outside of the minimum turning radius of the aircraft. Caveney and Hedrick (2005) explored the problem of path planning for targets within the aircraft turn radius. They used a geometric approach to fly the aircraft over a collection of targets in close proximity. Their report made no mention of including the effects of wind on their solution. Instead of developing a continuous flight path between all target locations, they developed a solution that

grouped the locations into subsets to visit with each flyover, while constraining the flyover such that the aircraft must actually fly over the destination point, not just have the sensor footprint over the point.

Caveney and Hedrick reduced the number of locations to reduce the flight time the aircraft spent while making turns back to the next subset of targets. They compared their geometric solution to a solution found by a greedy approach that did an exhaustive search for the largest groupings of targets, which they called the optimal solution. The solution was optimal because it grouped a set of targets together and then removed those targets from the remaining list of other targets, repeating this process until all targets were grouped, but without seeking the global minimum number of groups. Their geometric solution was 80% faster than the optimal solution on average. They demonstrated the results in both 10 point and 30 point scenarios, with the algorithm taking 2.44 s during the 30 point scenario compared to the 46.55 s for the optimal solution. The speed of this solution would allow the algorithm to be used in flight instead of only during mission planning.

2.4.2 Waypoint Guidance. To account for wind affects on MAV path planning, Osborne and Rysdyk (2005) used a lookup table in the MAV guidance system to determine the appropriate time to transition from one flight path segment to the next. This allowed the MAV to avoid reaching the waypoint before transitioning to the next flight segment, and causing the MAV to overshoot the waypoint. Osborne and Rysdyk did not consider a MAV with a sensor, so the sensor footprint was not part of their solution. Their lookup table took as inputs the MAV's course, the necessary course change to transition to the next segment, wind speed and direction, and the MAV's airspeed. The lookup table algorithm would output a proximity distance from the MAV to the waypoint at which to initiate the turn to avoid overshooting the waypoint.

In addition to simulating the MAV flying a course to demonstrate the lookup table's use, Osborne and Rysdyk demonstrated the lookup table would work for the MAV to circle a target. The circle flown was a many-sided polygon made up of individual way-

points. In the presence of wind, the ground speed was not the same, but by anticipating a turn before each segment, the aircraft circled the target.

2.4.3 Robust Wind Correction Algorithms. Many research efforts involving autopilots and flight planning algorithms did not account for wind, stating it was an "easily correctable issue through basic math" according to Robinson (2006). Therefore, he sought to develop algorithms which allowed for real-time updates of the winds and could be incorporated into commercial off-the-shelf (COTS) autopilots. To account for a sensor footprint, Robinson continuously updated the MAV's flight path to point the sensor at the target and not to place the aircraft above the target. He accounted for the difference in MAV location and desired sensor aimpoint location and offset the MAV's waypoint to point the sensor at the appropriate POI instead of placing the MAV above the POI.

2.4.4 Optimal Wind Corrected Flight Path Planning. Zollars (2007) evaluated dynamic flight path optimization for a MAV in a constant wind environment. His research sought the minimum-time flight path for a MAV with fixed sensors mounted in both a forward-looking and side-view configurations. He also evaluated the situation where the flight path was restricted, such as an urban environment, as well as a situation where both sensors would be used to track the target. In this last situation, he assumed the system would automatically transition between sensors as needed to track the target, eventually going into an orbit around the target for continuous surveillance.

Zollars' work differed from previous research because most previous research, according to Zollars, allowed for gimbaled sensors or focused on aircraft large enough to not be significantly affected by winds. Previous research, including Osborne and Rysdyk (2005) and Robinson (2006), accounted for wind correction by placing the aircraft directly above the target instead of putting the target in the sensor footprint. One of his assumptions was that the aircraft was a point mass. This made the analysis easier, but did not account for the aircraft banking and what that did to the sensor footprint during a turn. Without accounting for bank angle in an orbit, the sensor would aim at the incorrect location instead of the target.

Using MATLAB to conduct the discrete dynamic optimization, Zollars was able to identify which of six potential Dubins paths was the optimal solution. His solution was not perfect, however, since one of his assumptions was constant wind speed and direction for any given MAV flight. This was an area that would be further explored by Terning (2008) in subsequent research on path planning, as discussed in section 2.4.5.

To account for the constant wind, Zollars created a virtual target which moved at a speed equal to the wind speed but in an opposite direction. This allowed his algorithm to create a path that would put the sensor footprint over the actual target. The algorithm used the Dubins path model as the initial estimate for the final flight path, then ran the various paths through a MATLAB optimization routine to produce the solution. His optimal solution relied on a first-order Euler discretization of the equations of motion to develop a series of state equations which could be optimized. Zollars optimized solutions for all three of his situations.

Zollars concluded that his work could serve as a benchmark for future path planning algorithms designed for MAVs to account for wind. A recommendation for future analysis was to incorporate variable winds, use higher-order Euler discretizations, and not to assume the aircraft was a point mass. Much of his work and future recommendations would form the basis for Terning's research as discussed below.

2.4.5 Pathmaker Algorithm. In conjunction with the research conducted by Sakryd and Ericson (2008), Terning (2008) developed a Pathmaker algorithm that determined the optimal flight path from a MAV to a target, given the target's current location along with direction and speed of travel. This built on the work of Zollars (2007), but instead of finding the time-intensive optimal solution, found a solution using a heuristic, iterative algorithm.

Terning was able to leverage the work of Zollars and use it as a baseline with which to compare the Pathmaker algorithm. Additionally, the Pathmaker algorithm always converged on a solution, unlike the methods used by Zollars. Terning demonstrated the Pathmaker solution was as accurate as the optimal solution given a stationary target and actually more accurate than the single optimal solution given a moving target. The

main difference was that the optimal solution assumed the MAV would fly in a straight line between solution points, but since the aircraft is banking as it turns, this was not the case. Furthermore, the Pathmaker solution was much faster than the optimal solution. In one example, Terning illustrated that the Pathmaker algorithm found a solution 38 times faster than the optimal method, and produced a solution that had the MAV sensor directed at the target's final position.

Lastly, Terning evaluated the use of the Pathmaker algorithm for repeated updates as the MAV closed in on the target. Terning proposed a solution for a target with a variable speed or course, or in the case of variable winds. This solution would dynamically call the Pathmaker algorithm a second time to create a new path for the MAV. It would repeat at some interval for every new wind or change in target speed or course. Communication latency between the ground station and the MAV limited this application of Pathmaker. By the time the algorithm ran and passed the new flight path back to the MAV, the situation which caused the algorithm to rerun likely changed, making the new solution inaccurate as well. This latency was one of the identified limitations of the Pathmaker solution, because the use of the system assumed the algorithm would run on the Fleeting Target base station and not onboard the MAV. The Pathmaker solution, however, was not designed to provide continuous target tracking. The Fleeting Target solution (Sakryd and Ericson, 2008), discussed in section 2.1, used the Pathmaker algorithm to get the target into the field of view (FOV) for the MAV, and then tracking would be conducted using a Cursor-on-Target algorithm developed by Vantrease (2008), discussed in section 2.5.2. Terning further noted that the Pathmaker algorithm only worked to put the target in the center of the FOV, not to any location in the sensor display. He assumed that once an operator visually detected the target in the sensor display, the operator would assume control using the Cursor-on-Target algorithm.

The Pathmaker algorithm relied heavily on inputs from the Procerus Kestrel autopilot, the autopilot chosen by the Fleeting Target research team (Sakryd and Ericson, 2008) for their test vehicle. This meant the Pathmaker algorithm relied on wind data from the Kestrel autopilot and did not have an external sensor to measure wind velocities.

Citing a lack of time and inclement weather as the limiting factors, Terning only conducted limited flight tests of the Pathmaker algorithm. Fortunately, Vantrease (2008) was able to test and validate the Pathmaker algorithm during a flight test in May 2008. While the testing was still limited to a single test point, it further indicated the Pathmaker algorithm worked properly as intended.

Terning had two future research recommendations of note. The first was the question of how the Kestrel autopilot derived the ground track heading and bearing data. His second recommendation was to implement the Pathmaker algorithm in an automatic loop to make MAV heading corrections based on wind changes in an effort to improve flight path optimization over long distances. In essence, this would monitor wind conditions and re-execute the Pathmaker algorithm if the winds exceeded some pre-determined thresholds.

2.4.6 Constant Wind with a Bounded Turning Rate. Having found their test MAV, a SIG Rascal 110, had flight speeds of 20 m/s and could encounter winds of 5 m/s, McGee *et al.* (2005) researched how to apply constant winds and a constrained turning rate of an aircraft to the Dubins path solution. They noted that in the presence of wind, an aircraft's ground path was not a straight line. They had to iteratively solve the Dubins path problem with a virtually moving target. As summarized by Burns (2007), McGee *et al.* made "the assumption that flying in a constant wind towards a target fixed to the ground is identical to flying towards a moving target without winds. The fixed target is replaced by a virtual target moving at the velocity of the wind, but in the opposite direction."

2.4.7 Geometric Waypoint Estimator. Working closely with Zollars (2007) and building upon research by McGee *et al.* (2005), Burns (2007) studied the problem of having an unmanned aerial vehicle (UAV) autonomously rendezvous with an airborne tanker aircraft. His methods incorporated Dubins paths to get from one point to the other and modified the work of McGee *et al.* to have a moving destination target instead of a fixed target.

Burns noted the approach by McGee *et al.* provided an optimal path for rendezvous but without providing an implementable algorithm. Burns developed an algorithm that was solved iteratively to produce a Dubins path from the UAV to rendezvous with the constantly moving tanker. One limitation of the algorithm was the assumption that both tanker and UAV experienced the same winds. A second was the slow computational speed of the algorithm. The major benefit was the development of a robust algorithm to rendezvous two moving vehicles.

2.4.8 Trajectory Generation for Effective Sensing. Developing an alternative method of trajectory generation or path planning, Kehoe *et al.* (2007) studied the differences in mission planning for an ISR sensorcraft above 10,000 ft and one at 200 ft. They explored the flight path planning to include obstacle avoidance and developed an iterative solution to find the optimal path through an obstacle-laden environment. They further explored the effectiveness of trying to sense their target when it would be obstructed by other objects. By assuming a fixed sensor, their research problem was more constrained and the aircraft would have to point at the target at any given time that sensor data would be recorded from the target. They restricted the aircraft to a constant altitude, but allowed targets to be placed on three-dimensional surfaces with various occlusion regions.

Kehoe *et al.* did not account for wind in their trajectory planning model, but they covered areas such as assuming a fixed sensor and target occlusion which most other researchers have avoided. They also used the entire sensor FOV instead of relying solely on the central sensor aimpoint. This allowed the sensor to merely pass over the target objective instead of having to aim directly at it.

2.4.9 Path Planning for Skid-to-Turn UAVs. Much of the path planning research focuses on aircraft that can conduct coordinated turns. Yokoyama and Ochi (2008) researched the path planning for aircraft that cannot or do not perform coordinated turns, but instead skid-to-turn. In general, most manned aircraft use coordinated turns instead of skid-to-turn maneuvers because the centripetal forces experienced in a skid-to-turn maneuver can impart a queasiness or air sickness feelings among the pilots

and passengers of the aircraft. UAVs do not need to worry about this; therefore they may fly with a skid-to-turn maneuver.

Yokoyama and Ochi also stated that by using a skid-to-turn maneuver, the aircraft wings do not need to bank which both aids in signal transmission because the wings shadow the antennas less often. Furthermore, by keeping the wings level, a sensor aimpoint should vary significantly less during an aircraft turn and simplify the camera control logic. They noted that their work was significant because the aircraft kinematics are not the same between an aircraft that uses bank-to-turn and an aircraft that uses skid-to-turn.

2.5 Sensor Aimpoint

For any ISR mission that is concerned with gathering sensor data about specific POIs, limited concern is with the physical location of the MAV, but the concern is rather with the location of the sensor footprint or sensor aimpoint. As discussed above, various researchers evaluated MAV path planning such that the MAV would have a sensor aimed at a specific location. Research, discussed below, focused on sensor aiming more than path planning.

2.5.1 "Good Helmsman". Rysdyk (2003, 2006) explored flight path planning for MAVs that kept the sensor aimed at a point of interest while the aircraft maneuvered. The Aerosonde MAV used in the research had the capability to perform coordinated turns which is not true of all MAVs. Additionally, the aircraft had a nose-mounted, two-axis gimbaled camera.

Rysdyk sought a trajectory which would maintain constant geometry between MAV and target. He explored two situations, both assuming coordinated flight. The first fixed the camera pan angle and the second fixed both angles, effectively making the sensor fixed in the aircraft.

The algorithms developed allowed for wind corrections if the wind data was available. Since the sensor had a field of view larger than the aimpoint, it did not have to be directly pointed at the target to have the target visible in the sensor display. As Rysdyk

pointed out, this could lead to the target wandering in the sensor display. Implementing wind corrections could keep the target much closer to center in the display. Iterating the algorithms allowed the aircraft to adjust to slowly varying winds. By implementing the algorithms for flight planning and wind correction, Rysdyk found the camera angles could be maintained nearly constant. This allowed the MAV to fly in wind speeds that approached its flight speed and still produce usable targeting imagery.

Rysdyk (2007) explored orbit path planning in wind speeds which exceeded the cruise speed of the MAV. He found that the orbit path would be described as a trochoidal path. This allowed the orbit path to be developed when aircraft orientation was of concern, such as in a sensor aimpoint situation.

2.5.2 Cursor-on-Target. Also working with Sakryd and Ericson (2008) and Terning (2008), Vantrease (2008) developed software to simplify the operator control of a MAV conducting the Fleeting Target mission. Vantrease provided a point-and-click user interface based on video images returned from a MAV sensor, which would direct the MAV to head directly towards the POI under constant user command. This research extended the previous efforts of Crouse (2007) which developed and demonstrated a preliminary Cursor-on-Target system. The focus was to point-and-click at a desired target. The MAV would fly to a commanded sensor aimpoint using this software controller.

Vantrease leveraged concurrent research of Sakryd and Ericson and Terning, as well as the inner control loops of the Kestrel autopilot. He further demonstrated the Cursor-on-Target algorithm through computer simulation, HIL simulation, as well as a flight test.[1]

Vantrease also conducted testing of the user interface, using different gains for the software as well as testing how a communications delay would affect the use of the software. He had several test subjects[2], some of whom were aviators and others who were not, use the Cursor-on-Target software in a software simulation to qualitatively evaluate its usability. While the pitch controller was overly sensitive to inputs during

[1]In full disclosure, the author of this research paper participated in the flight test of Vantrease's Cursor-on-Target software.

[2]This author also served as one of five test subjects for these tests

the flight testing of the software, the Cursor-on-Target controller performed very well. The research demonstrated that having a point-and-click operation of an aircraft would work.

2.5.3 Sensing with Full Field of View. Researchers (Geiger *et al.*, 2006) at Pennsylvania State University investigated path planning using direct collocation with nonlinear programming. They used a fixed camera in a downward position and planned the path to maximize the time a target was in the FOV for one or more sensors. Their research included cooperation between two UAVs to increase the sensor coverage area.

The main thrust of the research of Geiger *et al.* was to maximize the area viewed by the sensors using a cost objective function. Their algorithms would then provide path planning back to the UAV. They used this algorithm for both moving and stationary targets. The fixed camera meant that their sensor FOV occasionally pointed at an undesirable location, particularly when the UAV banked to turn. They found that a cloverleaf pattern provided the maximum target coverage, using the downward pointing camera. Additionally, they investigated the problem with a single UAV against a stationary target, but while the UAV experienced constant wind. Their conclusion was to fly a figure eight around the target, with the path essentially running perpendicular to the wind, avoiding any headwind or tailwind problems.

Geiger *et al.* concluded that their algorithm solved a complex objective function to build a path to keep the target in the FOV the maximum amount of time. Unfortunately, they found that the method was too slow to be run as a real-time operation through an onboard computer in the UAV.

2.6 Wind Correction

While some research discussed above accounted for wind corrections during path planning, Pachter *et al.* (2008) considered the problem specifically for MAVs. MAVs may have increased problems in estimating aircraft heading and wind velocity due to the low accuracy of the small navigation sensors used on a MAV.

Key points made in Pachter *et al.*'s research include the inaccuracy of a MAV's heading angle due to the low accuracy of micro electro-mechanical systems (MEMS) rate gyros and the compasses used in MAV autopilots. Even when using global positioning system (GPS) measurements to capture ground speed, Pachter *et al.* noted that the heading and GPS ground track are not equal in the presence of wind. Knowing the effects of wind are important, particularly when a MAV has a fixed sensor that needs to be pointed at a target, they developed algorithms that provide real-time estimates of aircraft heading and wind velocity using GPS, MEMS, and air data measurements. They tested their algorithms and Unscented Kalman Filter solution using a Nighthawk MAV, a successor to the BATCAM. They conclude that their method can estimate the MAV's heading with an average root mean square error between 2 and 8 degrees, increasing as the wind increases.

2.7 Conclusion

This chapter reviewed literature relevant to this research topic. While some previous research considered navigation based on keeping a sensor aimed at a POI, few of them have developed usable algorithms, particularly for use with a fixed sensor. Rysdyk (2003, 2006) provided the best information on sensor aimpoint path planning. Additionally, Terning (2008) developed initial research to intercept a POI using Dubins paths along with aligning the sensor to the target at intercept. These authors provided a strong basis for the development of sensor aimpoint waypoint generation algorithms.

III. Method

This chapter outlines the methods and development of the algorithms used to generate waypoint solutions for both the orbit and overflight cases. First, refinement of the research concept is presented, followed by elaboration of the scenarios presented in section 1.2.1 into developed use cases. From the use cases, an object-oriented software class diagram and a sequence diagram are developed. These diagrams lead into the development and discussion of the sensor aimpoint algorithms. Finally, a user interface concept is discussed.

3.1 Concept Refinement

The Fleeting Target system as developed by Sakryd and Ericson (2008) was incorporated as a subset of the applications for route surveillance as developed by Diamond *et al.* (2009). Under the Fleeting Target concept, an operator would either have the MAV on orbit, or launch the MAV in response to a need for tactical ISR. The operator would know *a priori* the location, course, and speed of a target. The operator would then input the target's information into the Pathmaker algorithm, which would devise an optimal flight path from the MAV to the target and direct the MAV to fly that path. Once the target entered the MAV's sensor field of view, the operator would take over manual control of keeping the target in the sensor display using the Cursor-on-Target algorithm.

When considered in the wider context of the route surveillance mission, the operator may not have a known target. The MAV may be flying an assigned route when the operator identifies a point of interest in the sensor display. The Fleeting Target system has no way for the operator to easily keep that point of interest in the sensor display, except to engage the Cursor-on-Target algorithm and manually control the MAV. This research develops algorithms to generate waypoints which will keep the MAV's sensor aimed at a chosen POI.

3.2 Use Cases

The route surveillance problem explored by Diamond *et al.* (2009) developed multiple use cases to define the functionality of the system. Use cases are useful for systems engineering analysis to translate system required capabilities to functions that can be coded in software or built into a system. Several of the use cases for the route surveillance problem apply for this research. Additionally, two use cases, Orbit POI and Overfly POI, are specific to this research and are given as fully dressed use cases in Appendix B.

Figure 3.1 shows the relationships among the use cases for this research. The use cases are organized by the software system that manages the function performed by each use case. For the route surveillance research, the route surveillance research team created the command and control (C2) user interface software and used a commercial-off-the-shelf (COTS) flight management system. The orbit and overflight functions were developed as part of the sensor aimpoint research and provided to the route surveillance research team for the C2 user interface.

As described in the use cases below, the operator is a primary actor for the system. The operator supplies the stimulus for the system to act by designating a POI, establishing flight parameters such as which sensor to use, desired altitude, desired airspeed, preferred view orientation, number of waypoints, and initial ground track. The operator also commands either an orbit or an overflight. Support roles are filled by the flight management system and the command and control system. The operator interacts with the support systems, which in turn call the waypoint generator. The sensor aimpoint waypoint generator creates waypoints and passes them back to the flight management system which uploads the waypoints to the MAV and commands the MAV to fly that collection of waypoints. The operator then monitors the sensor feed and makes corrections as desired.

3.2.1 Orbit POI. The MAV starts on a pre-determined route surveillance mission. The Operator sees a POI in the user interface sensor display and directs the MAV to orbit the POI. The MAV exits the planned route, flies to the orbit, and orbits the POI, keeping the sensor aimpoint centered on the POI. The MAV may transition between

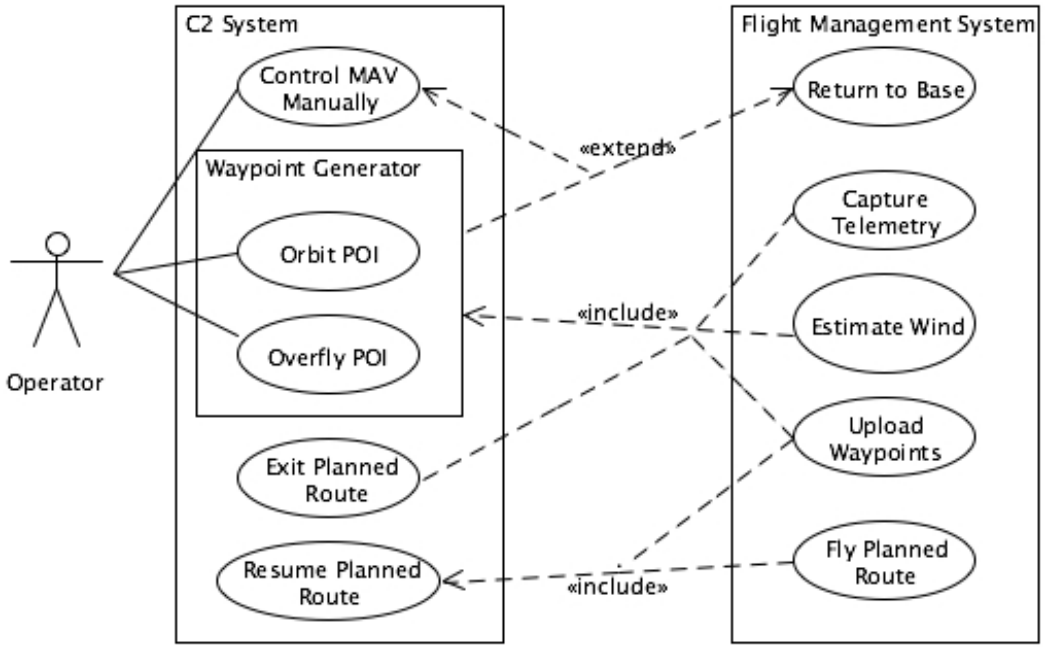


Figure 3.1: The use case diagram shows the relations between each of the use cases.

sensors during this procedure to provide the best image to the Operator. After the Operator determines the MAV has orbited the POI sufficiently, the Operator commands the MAV to resume the pre-determined route.

3.2.2 Overfly POI. The MAV starts on a pre-determined route surveillance mission. The Operator sees a POI in the user interface sensor display and directs the MAV to overfly the POI. The MAV exits the planned route, flies to the POI overflight path, and flies over the POI using a specified sensor view angle. The MAV may transition between sensors during this procedure to provide the best image to the Operator. Upon completion of the overflight, unless otherwise directed by the Operator, the MAV will resume the planned route.

3.2.3 Use Cases Referenced from Route Surveillance Problem. These use cases are only presented briefly to give a better understanding of the system goals outside the scope of, but related to, this research.

3.2.3.1 Fly Planned Route. Using the flight management system, the Operator designates a route for the MAV to fly. The MAV flies along those waypoints, repeating the waypoints sequentially (i.e., given waypoints 1 through 6, the MAV goes from 4, to 5, to 6, to 1...) until the Operator gives some other order or the MAV must *Return to Base*.

3.2.3.2 Estimate Wind. The flight management system estimates the wind field in which the MAV is currently flying. This estimated wind is then available as an input into other use cases.

3.2.3.3 Capture Telemetry. The flight management system captures real-time flight telemetry from the MAV. Among other data, this telemetry includes MAV position, MAV airspeed, MAV orientation, MAV ground speed and ground track, and the currently selected "goto" waypoint.

3.2.3.4 Return to Base. Either autonomously or upon command from the Operator through the flight management system, the MAV flies back to the recovery point to refuel, for maintenance, or upon completion of a mission.

3.2.3.5 Upload Waypoints. Either autonomously or upon command from the Operator, the flight management system uploads waypoints to the MAV.

3.2.3.6 Control MAV Manually. The Operator directs the MAV manually, inputting commands through the sensor display. This serves as a point and click flight control method. This use case directly relates to the research accomplished by Vantrease (2008), discussed in section 2.5.2.

3.2.3.7 Exit Planned Route. Prior to initiating either *Orbit POI* or *Overfly POI*, the MAV exits the planned route. The system should have memory to store the planned route such that the MAV can resume the planned route later, through the *Resume Planned Route* use case.

3.2.3.8 Resume Planned Route. Upon the conclusion of either *Orbit POI* or *Overfly POI*, the MAV resumes the previous planned route, by uploading the waypoints of the previously planned route through the *Upload Waypoints* use case. The system should recall the previously planned route so that it may resume the route without any action from the Operator.

3.3 Domain Diagrams

Two Unified Modeling Language domain diagrams better explain how the systems and software interact. The class diagram shows attributes and methods or functions for each object in the system, along with relationships between the classes. The sequence diagram follows the message traffic sent between algorithms or pieces of software, typically between classes, as they interact to accomplish the desired goal, or use case.

3.3.1 Class Diagram. The sensor aimpoint software class diagram, as shown in Figure 3.2, depicts a subset of classes and functions from a collection of all possible classes, methods, and attributes. The flight management and command and control systems have many classes, attributes, and methods. Only those classes, attributes, and methods required for interaction with the sensor aimpoint waypoint generator class are included in the class diagram. They are displayed as supporting classes to the main sensor aimpoint algorithm classes.

Figure 3.2 is organized by packages. First is a command and control package that includes a view class, a controller class, and the model class. The view class represents the user interface, while the controller translates commands entered through the user interface into functions that can be accomplished by the model. The model includes the waypoint generator algorithms as well as supporting classes.

The `Parameters` supporting class has multiple attributes, not all of which are set at any given time. The two waypoint algorithms require both `desiredAltitude` and `desiredAirspeed`, while only the overflight algorithm requires the preferred viewing angle, `viewOrientation`. The orbit algorithm requires an input number of waypoints,

`numWaypoints`, and `initialGroundTrack`, the initial ground track for the MAV to intersect the orbit.

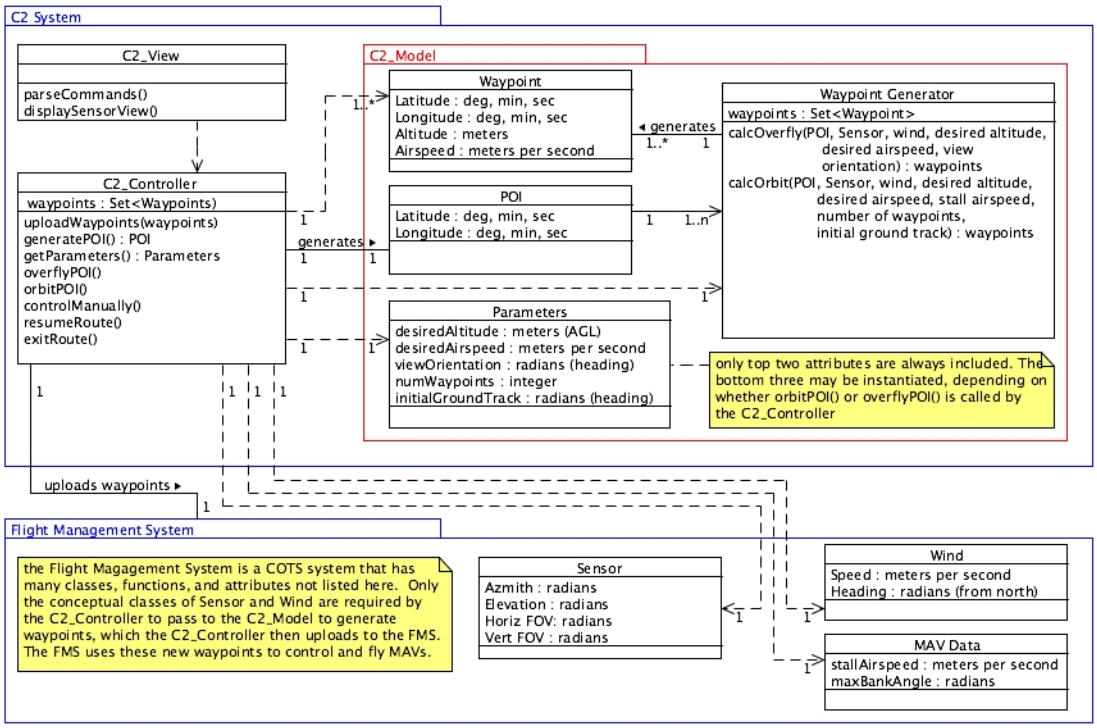


Figure 3.2: The class diagram shows attributes and methods or functions for each object in the system, along with relationships between the classes.

3.3.2 Sequence Diagram. Sequence diagrams show how objects, the instantiations of classes, interact dynamically. They provide a visual representation of how each object will call methods or functions in another object to accomplish a goal. The two sequence diagrams in Figures 3.3 and 3.4 show the message traffic for the orbit and overflight algorithms. Each of the sequence diagrams starts when the Operator, using the user iterface, directs the MAV to orbit or overfly a POI.

For the orbit case, the Operator commands the MAV to orbit a POI; this is the message from the Operator to the :C2_View class. :C2_View sends the message `orbitPOI` to the :C2_Controller which gets some of the necessary inputs into the orbit algorithm. :C2_Controller gets these parameters from :Parameters with the `getParameters()` call. These parameters are stored in the user interface as inputs from the Operator that have been set prior to the command to orbit the POI. :C2_Controller stores the parameters

and generates the location of the POI in :POI with the `generatePOI()` message. The generation of the POI is accomplished either by prompting the Operator for a location in latitude, longitude, and MSL altitude, or by computing these values after the Operator selected the POI through a graphical user interface. The POI location is returned to :C2_Controller as another input to the orbit algorithm. :C2_Controller also gets necessary telemetry from the flight management system, :FMS, which are the last variables needed to run the orbit algorithm. :C2_Controller sends the `calcOrbit()` command to :Waypoint_Generator which runs the orbit algorithm and returns a set of orbit waypoints to :C2_Controller. :C2_Controller uploads the waypoints to :FMS by recursively calling its own `uploadWaypoints(waypoints)` command. At this point, the waypoints are loaded in the flight management system. Either autonomously or upon command by the Operator, the waypoints are uploaded to the MAV and the MAV is commanded to fly those waypoints and return the appropriate sensor data back to the ground station. The MAV continues flying the orbit until the Operator commands it back to the previously planned route. This last step is not part of the sequence diagram in Figure 3.3 as that is not part of the sensor aimpoint research.

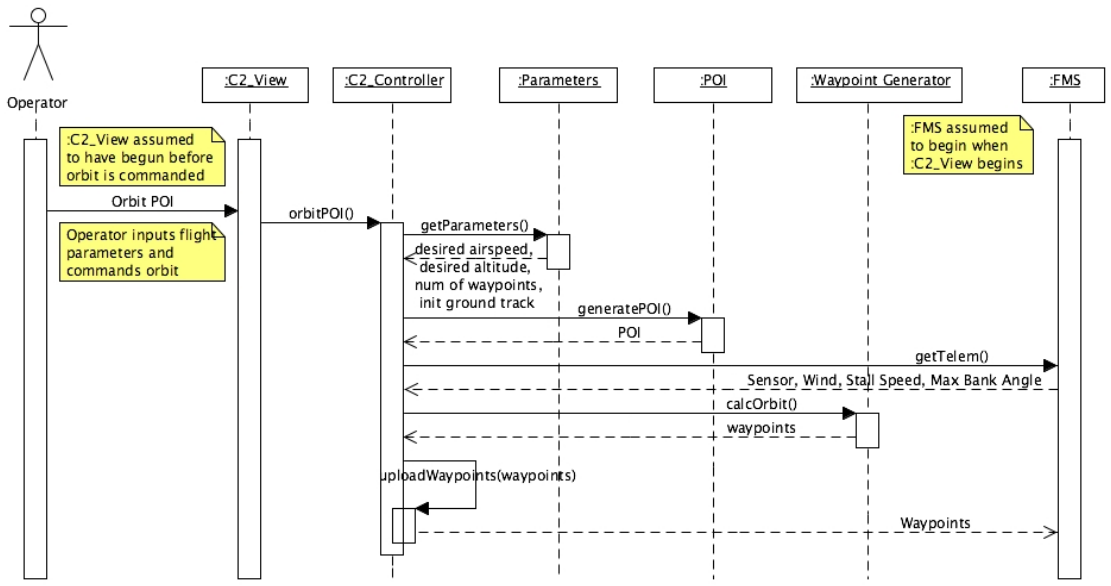


Figure 3.3: The message traffic sent between classes as a set of orbit waypoints is generated.

The message traffic for the overflight case is very similar to the orbit case discussed above. The parameters required from :Parameters are different for the overflight algorithm than the orbit algorithm, so a different set of values are returned. Just as in the orbit case, these parameters would have been input into the user interface by the Operator prior to commanding the overflight. Likewise, the telemetry variables returned from :FMS are different for the overflight case, as noted in Figure 3.4. The remainder of the message traffic between software classes is the same for the overflight case as for the orbit case.

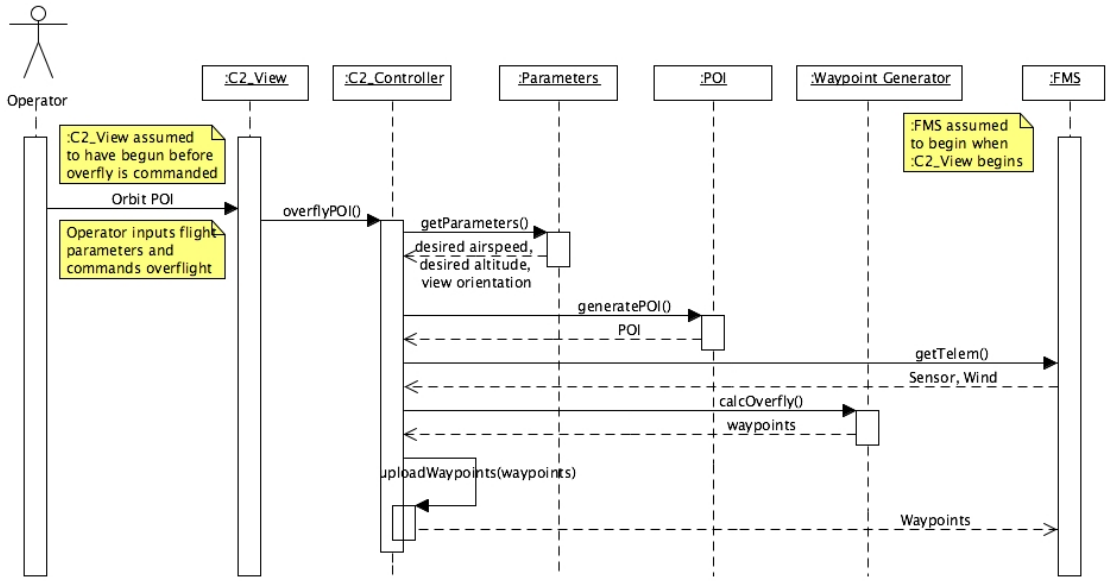


Figure 3.4: The message traffic sent between classes as a set of overflight waypoints is generated.

3.4 Definitions

During the course of the research, certain areas and topics required clarification to avoid potential confusion. Additionally, this set standards for the rest of the research with regard to these areas. These definitions directly supported the algorithm development and form a foundation for understanding some of the topics in the next section.

3.4.1 Coordinate Systems. Throughout the research, a north-east-down reference system is maintained. This means the x-axis points out the nose of the MAV, the

y-axis points out the right wing, and the z-axis points towards the ground. All altitudes are technically in a negative z-direction, although any time altitude is input into an algorithm, it is input as a positive value. If necessary, the equations add a negative sign to account for the altitude along the z-axis.

The right hand rule applies so that MAV yaw, ψ_a, goes from zero if the MAV's nose points due north, positive in a clockwise fashion; therefore, $\psi_a = 90°$ has the MAV pointed due east and $\psi_a = -90° = 270°$ has the MAV pointed due west. Pitch, θ_a, becomes positive as the MAV's nose goes away from the ground, keeping the conventional concept of positive pitch. Roll, ϕ_a, is positive as the left wing rises. Figure 3.5 shows the coordinate system used for the research.[1]

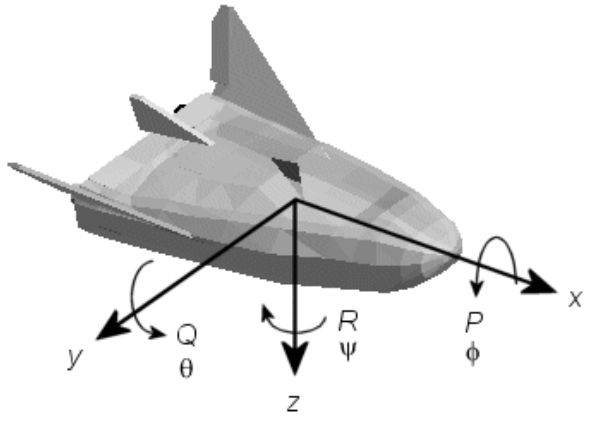


Figure 3.5: The coordinate system used for this research.

The sensor orientation is kept in the same fashion, relative to the MAV. Sensor azimuth or yaw, ψ_s, goes from zero if pointing out the nose and becomes positive in a clockwise fashion; therefore, $\psi_s = 90°$ is out the right wing and $\psi_s = -90°$ is out the left wing. Sensor elevation or pitch, θ_s, is negative as the sensor points away from the longitudinal axis of the MAV towards the ground; due to an assumption that the MAV would be used to surveil ground targets, positive sensor pitch is not considered. Sensor roll, ϕ_s, is always assumed to be zero.

The sensor is assumed to be located at the same Cartesian coordinates as the MAV. This is tantamount to assuming that the sensor is directly underneath the MAV's GPS

[1]Image from www.mathworks.com/access/helpdesk/help/toolbox/aeroblks/f3-22568.html

antenna. This is not always a valid assumption; however, the differences of centimeters from the sensor's actual location to the location of the GPS antenna are not noticable in the sensor aimpoint determination, especially when the MAV is in the presence of wind.

3.4.2 Waypoints. A waypoint is defined as a latitude, longitude, altitude above ground level, and airspeed. The algorithms, however, relate all coordinates to the MAV as relative distances in meters. The MAV is considered at $(0, 0)$ and all other distances were then at a $(\Delta x, \Delta y)$ to the MAV. Additional algorithms convert the output $(\Delta x, \Delta y)$ into latitude and longitude values which are then passed to the flight management system.

3.5 Algorithm Methodology

This research develops the algorithms to implement the *Orbit POI* and *Overfly POI* use cases along with supporting algorithms for estimating wind, for knowing the sensor aimpoint given a MAV position and attitude, for knowing the footprint of the sensor field of view on the ground, and for transition to and from the sensor aimpoint flight paths. The starting and ending points are a planned route for the MAV.

3.5.1 Wind Estimation. To calculate the aimpoint algorithms in the presence of wind, an estimation of the wind must be known. Methods such as those developed by Pachter *et al.* (2008) can provide wind estimates. Additionally, the autopilot used for the research, the Procerus Technologies Kestrel, estimates the winds and transmits the wind estimates to the flight management software, Virtual Cockpit. Virtual Cockpit, in turn, makes those values available as variables to other software as well as displaying wind information in the flight management display.

Discussions with a representative from Procerus Technologies revealed how the Kestrel calculated the wind speed and heading. The Kestrel divides all possible aircraft headings into different sections. Using the heading given by the GPS ground track, the Kestrel compares GPS ground speed to pitot static airspeed at each point in time. The difference between ground speed and airspeed is stored for that heading section. Each section is independently filtered before the Kestrel combines the information from all sections to provide a wind estimation. (Bosley, 2008) It is for the reason of combining

wind estimation from each heading section that the *Kestrel User Guide* comments: "The reported wind information is most accurate after the aircraft has flown at least one complete circle." (Procerus Technologies, 2008)

A detriment to the wind estimation method used by the Kestrel is that it slowly averages the winds. During hardware-in-the-loop (HIL) simulations, the abrupt changes in wind speed or heading were made between test points. The Kestrel had to fly two or three complete circles to calculate a new wind. It was unclear how much the Kestrel tempered the new wind information. Discussions with the representative from Procerus Technologies indicated the Kestrel did not have a limited time period in which to average the winds, but seeing as the reported wind eventually converged with the simulated wind, the Kestrel would seem to have a limited memory of recorded winds. Therefore, given enough time, the Kestrel would notice the new winds were prevalent, but the data could be tempered by the previously recorded winds. While not a perfect solution to identifying the winds present around a MAV, the Kestrel provided enough information for the algorithms to work.

On a final note, in aviation terms wind is usually given in a direction or heading it came out of and a speed. For example: "winds out of the southwest at 10 knots," or "winds out of 210 at 10 knots." In planning for the sensor aimpoint algorithms, especially in reading the manual for the Kestrel autopilot, the term "wind heading" was used. This led to the assumption that the Kestrel and Virtual Cockpit output the heading of the wind, instead of the direction from which the the wind came. The Kestrel, in fact, outputs wind direction in standard aviation terms. This was discovered after the algorithms had been coded. Instead of changing the algorithms, the wind direction inputs were converted before being input into the algorithms. The rest of the research assumes the wind heading was the direction toward which the wind went, just as an aircraft heading was the direction toward which the aircraft went.

3.5.2 Sensor Aimpoint. A basic body-to-reference, subscript r/b, coordinate transformation matrix is given in Equation 3.1, where c means cosine and s means sine. As the basic body-to-reference coordinate transformation matrix is applied to

the specific use, discussed below, the actual body and reference subscripts are used. The sensor aimpoint algorithm transforms a representational sensor vector, $\vec{s}$, to the sensor orientation, from that to the MAV orientation, and finally to the reference frame orientation, as indicated in Equation 3.6. The subscripts are as follows: sensor-to-airframe is given as a subscript s/a, and airframe-to-inertial is given as a subscript r/a.

$$C_{r/b} = \begin{bmatrix} \mathrm{c}\theta\mathrm{c}\psi & -\mathrm{c}\phi\mathrm{s}\psi + \mathrm{s}\phi\mathrm{s}\theta\mathrm{c}\psi & \mathrm{s}\phi\mathrm{s}\psi + \mathrm{c}\phi\mathrm{s}\theta\mathrm{c}\psi \\ \mathrm{c}\theta\mathrm{s}\psi & \mathrm{c}\phi\mathrm{c}\psi + \mathrm{s}\phi\mathrm{s}\theta\mathrm{s}\psi & -\mathrm{s}\phi\mathrm{c}\psi + \mathrm{c}\phi\mathrm{s}\theta\mathrm{s}\psi \\ -\mathrm{s}\theta & \mathrm{s}\phi\mathrm{c}\theta & \mathrm{c}\phi\mathrm{c}\theta \end{bmatrix} \tag{3.1}$$

$$\begin{aligned} \vec{\mathrm{aim}} &= C_{r/a}C_{a/s}\vec{s} \\ \vec{s} &= [1,0,0]^T \end{aligned} \tag{3.2}$$

Using the aim vector from Equation 3.2, the angle between the ground and the slant range line from MAV to aimpoint is found using Equation 3.3. The horizontal distance, or radial distance, from the MAV's position above the ground to the sensor aimpoint is likewise found using Equation 3.4, where acft_z is MAV altitude above ground level. The bearing from MAV to aimpoint is then calculated with Equation 3.5; note, the MATLAB function `atan2` is used for the arctangent in Equation 3.5 as it returned the four quadrant arctangent for the two input elements. Finally, the sensor aimpoint coordinates are found by summing the MAV position with the components of the horizontal vector from MAV to aimpoint and setting the altitude of the aimpoint to ground level, as indicated in Equation 3.6.

$$\tan\theta_{aim} = \frac{\mathrm{aim}_z}{\sqrt{\mathrm{aim}_x^2 + \mathrm{aim}_y^2}} \tag{3.3}$$

$$R_{aim} = \frac{\mathrm{acft}_z}{\tan\theta_{aim}} \tag{3.4}$$

$$\psi_{aim} = tan^{-1}\frac{\mathrm{aim}_x}{\mathrm{aim}_y} \tag{3.5}$$

$$\text{sensor aimpoint} = \left[\left(\text{acft}_x + R_{aim}\cos\psi_{aim}\right), \left(\text{acft}_y + R_{aim}\sin\psi_{aim}\right), 0\right] \tag{3.6}$$

The sensor aimpoint offset waypoint coordinates are found by taking the POI ground coordinates relative to the MAV, $(\text{POI}_x, \text{POI}_y)$ and subtracting the sensor aimpoint offsets shown in Equation 3.7. The waypoint altitude and airspeed remain the same as the MAV's altitude, h, and airspeed, V_a, at the time the algorithm is initiated. The resulting sensor-on-POI (SoP) waypoint coordinates are given by Equation 3.8.

$$(\Delta x, \Delta y) = \left[\left(\text{acft}_x + R_{aim}\cos\psi_{aim}\right), \left(\text{acft}_y + R_{aim}\sin\psi_{aim}\right)\right] \tag{3.7}$$

$$\text{SoP Waypoint} = \left[\text{POI}_x - \Delta x, \text{POI}_y - \Delta y, h, V_a\right] \tag{3.8}$$

3.5.3 Sensor Footprint. Typically the only concern for this research was the sensor aimpoint, and not the sensor footprint; however, certain occasions, such as a particular instance in the overflight algorithm discussed in section 3.5.5, relied on knowing the total sensor footprint. The sensor footprint is the horizontal plane or ground area captured by the sensor at a single instance in time. The sensor footprint relies upon MAV position, MAV attitude, sensor attitude, and sensor FOV.

The basic sensor footprint is found as an array of points based upon the horizontal FOV, FOV_H, and the vertical FOV, FOV_V, assuming a sensor looking straight ahead of the MAV. The footprint is then transformed twice using the body-to-reference transformation matrix given in Equation 3.1. This process is identical to the double transformation of the sensor vector, $\vec{s}$, as discussed in section 3.5.2. The transformation is shown in Equation 3.10. Finally, by relating the footprint points to the MAV position,

as shown in Equation 3.11, through a parametric variable, t, footprint waypoints are found that indicate the corners of the sensor footprint on the ground.

$$\text{footprint}_0 = \begin{bmatrix} 1 & , & +\tan\frac{\text{FOV}_H}{2} & , & +\tan-\frac{\text{FOV}_V}{2} \\ 1 & , & -\tan\frac{\text{FOV}_H}{2} & , & +\tan-\frac{\text{FOV}_V}{2} \\ 1 & , & -\tan\frac{\text{FOV}_H}{2} & , & +\tan+\frac{\text{FOV}_V}{2} \\ 1 & , & +\tan\frac{\text{FOV}_H}{2} & , & +\tan+\frac{\text{FOV}_V}{2} \end{bmatrix} \tag{3.9}$$

$$\text{footprint} = C_{r/a}C_{a/s}\text{footprint}_0 \tag{3.10}$$

$$\begin{aligned} t &= \frac{\text{acft}_z}{|\text{footprint}_z|} \\ x &= \text{acft}_x + t \cdot \text{footprint}_x \\ y &= \text{acft}_y + t \cdot \text{footprint}_y \\ z &= 0 \\ \text{footprint waypoint} &= \begin{bmatrix} x & y & z \end{bmatrix}; \end{aligned} \tag{3.11}$$

3.5.4 Orbit Algorithm. The orbit algorithm relies solely on the side sensor while the MAV is in the orbit. An operator could use the front sensor to get the MAV into position, but only the side sensor would keep the POI in focus throughout the orbit. The side sensor is assumed to be situated with an azimuth of $\pm 90°$ so that it is directly perpendicular to the longitudinal axis of the MAV. If this assumption is not made, then the MAV will have a spiral flight path around the POI, spiraling towards the POI if the absolute value of the azimuth is less than 90°, and spiraling away from the POI if the absolute value of the azimuth is greater than 90°. The algorithm corrects the side sensor azimuth to $\pm 90°$, based on if the sensor azimuth is less than 90° or not.

Initially, attempts were made to iteratively solve for bank angle, ϕ_a, and airspeed, V_a. Two equations for the orbit radius were used, one given in Equation 3.12, which related radius to airspeed and bank angle. The second, given in Equation 3.13, related

radius to the horizontal sensor aiming distance using $(\Delta x, \Delta y)$ as the output from the sensor aimpoint algorithm.

$$R = \frac{V_a^2}{g \tan(\phi_a)} \tag{3.12}$$

$$R = \sqrt{\Delta x^2 + \Delta y^2} \tag{3.13}$$

The algorithm solved for each radius and compared the results. The algorithm varied the bank angle until the difference between the radii was less than some arbitrary small error distance, perhaps on the order of 2.5 to 5m. This process could find a bank angle that exceeded a maximum bank angle, either determined by MAV dynamics or by the limitation that the summation of the bank angle and the sensor depression angle had to be less than $\pm 90°$.

One problem with the results of the algorithm discussed above was that the two radii rarely converged perfectly. This meant that the sensor aimpoint would not always be perfectly centered for each waypoint. The results were usable, but less than optimal; furthermore, a simpler algorithm could be used to solve for the bank angle and airspeed. This second method, discussed below, produces better results than the former method and in a shorter time. For example, running both algorithms 100 times with the same inputs for POI, MAV altitude, sensor choice, wind, initial ground track, airspeed, and 12 waypoints to output, the former algorithm took 1.2834s on average while the next algorithm averaged 0.0260s.

As indicated above, the orbit radius depends on both the ground speed and bank angle to maintain an orbit as well as the MAV attitude and sensor attitude which define the sensor aimpoint. Ground speed is required instead of airspeed because ground speed accounts for wind, whereas airspeed remains constant to the commanded airspeed regardless of wind. Ground speed, V_g, and heading, ψ_a, at a particular waypoint in the orbit are found by relating the commanded airspeed, V_a, commanded ground track angle, χ_g, wind speed, V_w, and wind track angle, χ_w.

The heading is found by solving a heading and speed vector triangle, as shown in Figure 3.6, with trigonometric relations. The known values do not produce fully realized vectors for either the ground or the air information; in each case, only one of the two required values is known. Two cases exist, one for $\psi_a < \chi_g$ and the other for $\psi_a > \chi_g$. The solutions for ψ_a are shown in Equation 3.14.

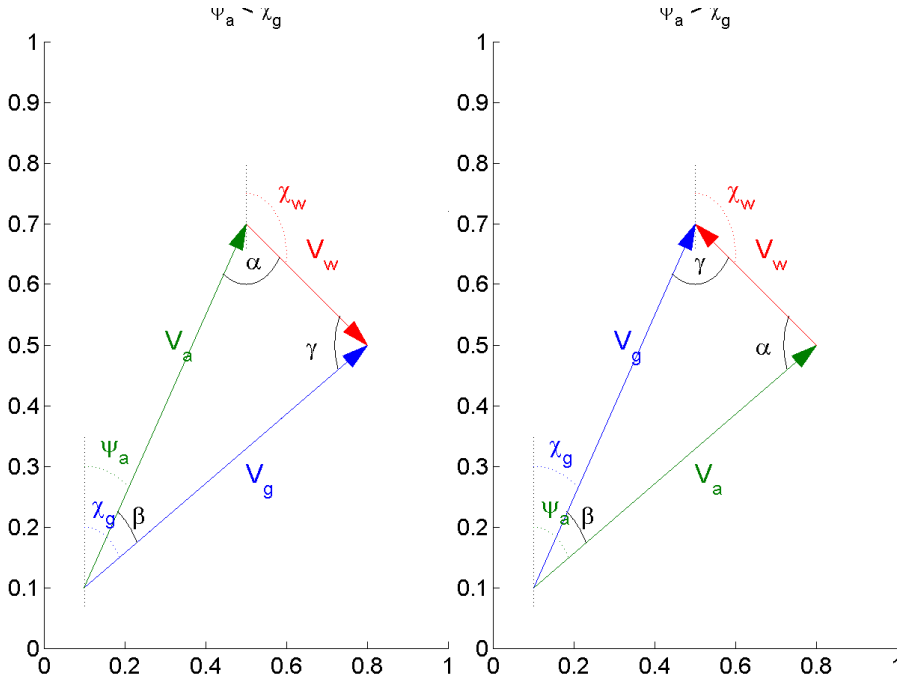


Figure 3.6: The wind vector triangle for both $\psi_a < \chi_g$ and $\psi_a > \chi_g$.

$$\psi_a = \begin{cases} \chi_g - 2\pi + \sin^{-1}\left(\frac{V_w}{V_a}\sin\left(\chi_g - \chi_w\right)\right) & \text{, for } \psi_a < \chi_g \\ \chi_g - \sin^{-1}\left(\frac{V_w}{V_a}\sin\left(\chi_w - \chi_g\right)\right) & \text{, for } \psi_a > \chi_g \end{cases} \tag{3.14}$$

Once the MAV heading is found, the ground speed is easily obtainable using vector components as shown in Equation 3.15 for the result shown in Equation 3.16.

$$\begin{aligned} V_{a_N} &= V_a \cos\psi_a & V_{a_E} &= V_a \sin\psi_a \\ V_{w_N} &= V_w \cos\chi_w & V_{w_E} &= V_w \sin\chi_w \\ V_{g_N} &= V_{a_N} + V_{w_N} & V_{g_E} &= V_{a_E} + V_{w_E} \end{aligned} \tag{3.15}$$

$$V_g = \sqrt{V_{g_N}^2 + V_{g_E}^2} \tag{3.16}$$

The equation for orbit radius as a function of ground speed, V_g, and bank angle, ϕ_a, is given in Equation 3.17. The equation for radius from the MAV to the POI as a function of sensor aimpoint, where h is altitude and θ_s is sensor depression, is given by Equation 3.18.

$$R = \frac{V_g^2}{g \tan(\phi_a)} \tag{3.17}$$

$$R = \frac{h}{\tan(\phi_a + \theta_s)} \tag{3.18}$$

By taking Equations 3.17 and 3.18 and setting them equal to each other, ϕ_a can be solved using the quadratic equation for the solution given in Equation 3.19. It is possible to obtain bank angles that have complex values based on the input ground speed and altitude. As the airspeed decreases, the bank angle required to maintain the same radius of an orbit also decreases; therefore, when this situation occurs, the algorithm iterates the process to solve for heading, ground speed, and bank angle by incrementally changing the commanded airspeed, V_a for the waypoint, in -0.25m/s steps.

$$\phi_a = \tan^{-1}\left(\frac{(-V_g^2 + hg) \pm \sqrt{(V_g^2 - hg)^2 - 4(hg\tan\theta_s)(V_g^2\tan\theta_s)}}{2(hg\tan\theta_s)}\right) \tag{3.19}$$

The solution to Equation 3.19 produces two results for ϕ_a, both of which are valid bank angles. The algorithm chooses the lesser of the two bank angles, or the bank angle closest to zero, for the desired bank at the waypoint. This is a practical decision which allows a better view of the side of a POI, but increases the orbit radius, reducing the sensor resolution for the POI, as discussed in section 2.3.

Given the necessary heading, ψ_a, and bank angle, ϕ_a, for the waypoint, the location of the waypoint can be found. The pitch, θ_a, of the MAV is assumed to be negligible in the turn, thus allowing a small angle approximation. This is not a strictly valid assumption, but the pitch angle is sufficiently small to not heavily impact the sensor

aimpoint. The MAV altitude, attitude, and sensor attitude are plugged into the sensor aimpoint algorithm, as discussed in section 3.5.2, to get sensor aimpoint offset values from the POI at that waypoint.

This process is repeated n-times to produce n-waypoints around the POI. The number of waypoints is passed as an input to the algorithm, so the algorithm is robust enough to handle any number of desired waypoints. As the process iterates to generate the waypoint list, the algorithm incrementes the commanded ground track, χ_g, as indicated in Equation 3.20. If the sensor is pointed out the right wing, χ_g increases so that the MAV will make right hand or clockwise turns about the POI; if the sensor is pointed out the left wing, χ_g decreases with each increment.

$$\chi_g|_{new} = \chi_g|_{old} \pm \frac{360^\circ}{n} \tag{3.20}$$

Four situations occurr which produce orbit waypoints that are unacceptable solutions. In high winds or high MAV stall speeds, the iterations on airspeed can produce a waypoint that requires the MAV to fly slower than the stall speed or a minimum airspeed. Since Equation 3.17 requires ground speed, this means the waypoint that requires the slowest airspeed occurrs when the MAV had a pure tailwind. The valid orbit solution in this case requires a higher orbit altitude. At a higher altitude, the orbit radius is greater, as indicated by Equation 3.18. Since the orbit radius is greater, the MAV can maintain an orbit at a higher ground speed, thus a higher minimum airspeed. This is implemented by iteratively solving for MAV heading, ground speed, and necessary bank angles, at a situation where the minimum airspeed is used for V_a and the wind heading is used for χ_g in Equations 3.14, 3.16, and 3.19. The iterations involve incrementing the MAV's altitude in 10m steps from the initially desired altitude until a valid solution is found. This altitude becomes the minimum orbit altitude for the entire orbit. One detriment to this solution is that the higher orbit altitude increases the slant range to the POI, reducing the sensor resolution for the POI, as discussed in section 2.3. In the implementation of this algorithm, the orbit waypoint solution should be generated, and the operator should be notified of the discrepancy in the desired altitude and the actual

orbit altitude. The operator can then have the opportunity to decline the new solution if having reduced sensor resolution is less preferable than actually keeping the POI in the sensor FOV.

A second orbit solution problem occurrs when wind speed is greater than or equal to desired airspeed. This produces a zero or negative ground speed in the orbit. As the MAV would be difficult if not impossible to fly in this situation, it is unlikely that this would occur; however, if it does, the algorithm produces an error stating the wind speed is too great for the MAV to maintain an orbit.

In a situation where a high airspeed and low orbit altitude are desired, an invalid orbit solution is generated. As these variables work together to produce an orbit solution, explicit values for the variables that cause the problem are not known. Given a high airspeed and low altitude, the mathematical solution for ϕ_a from Equation 3.19 can be unreasonable. For example, with a MAV that has a sensor azimuth of $-90°$ and given a desired airspeed of 100kts and a desired altitude of 50m, the resultant bank angle is $68.5°$. This will not point the sensor at the POI nor will it allow the MAV to execute a counter-clockwise orbit. By incrementing the altitude in 10m steps, a valid orbit point for the desired airspeed is found. This iteration can lead to a problem where the orbit waypoints are not all at the same altitude, which would have the MAV climb and descend throughout the orbit, which, in turn, would vary pitch greatly and invalidate the small angle approximation used for pitch in the orbit calculations. A final check is done before the orbit waypoints are output to verify that the first and the last orbit waypoints are at the same altitude. If they are, the algorithm outputs the waypoints. If the altitudes are different, the algorithm re-runs using the final altitude as the desired altitude for the entire orbit. This slows the algorithm slightly, but ensures an orbit with a consistent altitude. As discussed above, one detriment to this solution is that the higher orbit altitude increases the slant range to the POI, reducing the sensor resolution for the POI, as discussed in section 2.3.

A final unacceptable orbit solution can occurr when the solution for ϕ_a exceeds a maximum allowable bank angle for the MAV. This maximum bank angle, ϕ_{max}, may be determined by MAV dynamics or by flight control software. If ϕ_a is greater than ϕ_{max},

the orbit algorithm increments the flight plan altitude and adjusted so the entire orbit is maintained at the same altitude, as discussed above.

3.5.5 Overflight Algorithm. The overflight algorithm focuses on establishing waypoints for a preferred sensor orientation or look angle on the POI. It finds the necessary MAV heading, ψ_a, from the preferred look angle, χ_L, and the sensor azimuth, ψ_s. The relation is given in Equation 3.21. Neither the look angle nor the heading are dependent on wind in this case, since the ground track heading, χ_g, has not been established.

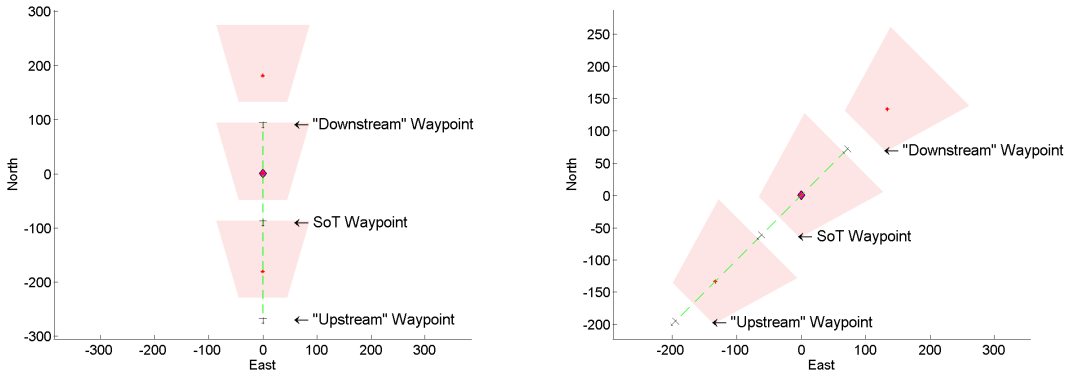
$$\chi_L = \psi_a + \psi_s \tag{3.21}$$

Given the MAV heading and altitude, the overflight algorithm calls the sensor aimpoint algorithm to get a necessary offset from the POI for the sensor-on-POI waypoint. The overflight algorithm assumes the MAV is in steady, level flight at this waypoint; therefore, the MAV has no pitch and no bank. While the actual orientation of the MAV at that waypoint may have pitch and bank, for planning purposes, pitch and bank are not necessary.

The sensor-on-POI waypoint coordinates are found using Equation 3.8, repeated as Equation 3.22, and as discussed in section 3.5.2. To increase the likelihood the MAV will have the sensor aimed at the POI at the sensor-on-POI waypoint, two additional waypoints are found, one "upstream" of or before the sensor-on-POI waypoint, and the other "downstream" or past the sensor-on-POI waypoint, as indicated in Figure 3.7. This creates a flight path that should keep the sensor aimed at the POI at the preferred sensor orientation by allowing the sensor to sweep over the POI as the MAV flies along the path.

$$\text{SoP Waypoint} = [\text{POI}_x - \Delta x, \text{POI}_y - \Delta y, h, V_a] \tag{3.22}$$

The additional waypoints need to be along a straight line ground track for the MAV to fly in the presence of wind. Using vector components, indicated in Equation



(a) Front Sensor, $\chi_L = 000°$ (b) Front Sensor, $\chi_L = 045°$

Figure 3.7: The waypoints from the overflight algorithm in two different look angles.

3.15, ground speed, V_g, is found using Equation 3.16, along with the ground track, χ_g, as shown in Equation 3.23 for the given airspeed, heading, and wind. The MATLAB command `atan2` is used for the arctangent in Equation 3.23 as it returned the four quadrant arctangent for the two input elements. This ensures χ_g outputs as a heading relative to a north reference.

$$\chi_g = \tan^{-1} \frac{V_{g_E}}{V_{g_N}} \tag{3.23}$$

The "upstream" and "downstream" waypoints are then found by taking offsets from the sensor-on-POI waypoint. As before, the altitude and airspeed remain the same as the MAV's altitude, h, and airspeed, V_a at the time the algorithm is initiated. A magnitude, q, is required to space the "upstream" and "downstream" waypoints from the sensor-on-POI waypoint. This magnitude can be any value as long as the additional waypoints are far enough from the sensor-on-POI waypoint so the MAV flies straight and level as it encounters the sensor-on-POI waypoint. The magnitude, q is set equal to the maximum distance the sensor footprint reached along the ground. In this manner, if the MAV flies in a no-wind situation while using the front sensor, the "upstream" waypoint is just before the POI entered the sensor field of view. The "downstream" waypoint is an equal distance away, but after the MAV's sensor aimpoint passed over the POI.

Equation 3.24 gives the solution for the "upstream" and "downstream" waypoints, with subtraction for the former and addition for the later.

$$\text{Waypoints} = [\text{SoP}_x \mp q\cos\chi_g, \text{SoP}_y \mp q\sin\chi_g, h, V_a] \tag{3.24}$$

The overflight algorithm outputs three waypoints: the sensor-on-POI, "upstream," and "downstream" waypoints. By combining Equations 3.22 and 3.24, the complete set of waypoints is shown in Equation 3.25.

$$\begin{bmatrix} \text{"Upstream"} \\ \text{SoP} \\ \text{"Downstream"} \end{bmatrix} = \begin{bmatrix} \text{SoP}_x - q\cos\chi_g, \text{SoP}_y - q\sin\chi_g, h, V_a \\ \text{POI}_x - \Delta x, \text{POI}_y - \Delta y, h, V_a \\ \text{SoP}_x + q\cos\chi_g, \text{SoP}_y + q\sin\chi_g, h, V_a \end{bmatrix} \tag{3.25}$$

3.5.6 Variable Winds Algorithms. The orbit and overflight algorithms generate waypoints in the case of a constant wind field. Aircraft rarely encounter a sustained constant wind field; instead, they must fly in winds which vary in magnitude, direction, or both. Constantly adjusting the waypoints to account for a wind that changes slightly in magnitude or direction could lead to a situation where the waypoints constantly change and update because the winds change. Gusts are changes in wind that occur in short periods of time, different from the steady state wind. This presents three wind conditions: steady state, varying, and gusting.

Since the algorithms already discussed handle steady state winds, the next condition to consider would be varying, but not gusting, winds. An additional algorithm handles this situation. This varying wind algorithm monitors the wind reported by the MAV. The algorithm takes as an initial value the state of the wind when the orbit or overflight algorithms are called. The varying wind algorithm then compares the current wind value to the initial value. When either the wind direction or magnitude exceeds user-defined limits, the varying wind algorithm calls the orbit or overflight algorithm again, using the new wind values with the previously determined values for desired al-

titude, desired airspeed, number of waypoints, and look angle or initial ground track angle, as appropriate.

By only comparing instantaneous new values of winds to the previous wind, gusts will initiate the varying wind algorithm to run either the orbit or overflight algorithms for an additional time. This results in constantly changing waypoints, which is undesirable. To avoid the gusting situation but handle the varying winds situation, the varying winds algorithm has to have knowledge of not just the initial and the instantaneous winds, but all wind values captured in a set time period. To avoid tracking time with telemetry data that may not transmit on a set time frequency, the algorithm simply track the last 120 wind data points. It then assumes any instantaneous wind speeds that exceed some threshold magnitude or change in direction are considered a gust and excludes those from the wind history used in the varying wind algorithm.

3.5.7 Transition from Route to Orbit or Overflight. The orbit and overflight algorithms set up the sensor to point at the POI while the MAV flies along the commanded path. The MAV needs to get from its position when the sensor aimpoint algorithm is commanded to the initial waypoint in the algorithm flight path. A flight management system can command the MAV to simply fly from the position to the initial waypoint. Less critical for the orbit algorithm, since the MAV will eventually become aligned to the orbit path and get the sensor aimpoint on the POI, relying on the flight management system to get the MAV lined up for the overflight situation could lead to situations where the MAV is not aligned in time. Specifically, unless the MAV happens to approach the overflight path already aligned, the MAV will likely overshoot or undershoot the first waypoint on the overflight path. Not being aligned by the first waypoint could likely mean the MAV will not have its sensor properly aimed during most of the overflight.

To create a path from the MAV's initial position to the first waypoint for either the overflight or the orbit paths, a Dubins path generator is used. A Dubins path is the shortest path between two points given an initial point and heading and a final point and heading. It consists of two turns, from the initial point and into the final point, and a straight line connecting the two. (Dubins, 1957) While working with Diamond

et al. (2009) on the route surveillance research, Booth (2009) developed a MATLAB routine to generate a Dubins path as a series of latitude and longitude waypoints. This code was modified to fit the needs of the sensor aimpoint research. Booth's code output the latitude, longitude, and distance remaining for each waypoint. To standardize the waypoints with the waypoints generated by the orbit and overflight algorithms, the Dubins algorithm was changed to output latitude, longitude, altitude, and airspeed for each waypoint. The initial and final turning circle radii are calculated by Equation 3.12 using the maximum bank angle for the MAV, making for the tightest circle possible.

An additional algorithm, a wrapper algorithm, was created for both the orbit and overflight situations. These wrapper algorithms generate the sensor aimpoint waypoints and an associated Dubins path to get the MAV from its current position into the orbit or overflight path, as appropriate. In addition, each wrapper algorithm contains logic to determine the direction, clockwise or counter-clockwise, of the initial and final turns for the Dubins path. The initial turn is calculated based on taking the difference of the heading from the MAV's initial position to the initial point of the sensor aimpoint path, either overflight or orbit, and the MAV's initial heading at the time the algorithm is called. If the absolute value of the difference is between 0° and 180°, the initial turn is clockwise, otherwise it is counter-clockwise. These turns allow for the most expeditious transition for the MAV from its current heading to the sensor aimpoint path.

The final turn is calculated as well by each wrapper. For the case of the orbit algorithm, the final turn depends on sensor orientation. The final turn is concurrent with the direction the sensor is aimed, thus concurrent with the turn direction of the orbit. If the sensor is aimed to the right, the MAV makes right-hand or clockwise turns; if it is aimed to the left, the final Dubins turn is to the left or counter-clockwise. For the case of the overflight algorithm, the final turn is based on the difference between the heading from the MAV to the initial point and the heading along the overflight path. After ensuring the difference is greater than 0°, if the difference is between 0° and 180°, the initial turn is counter-clockwise, otherwise it is clockwise.

Unique to the orbit situation, the wrapper algorithm calculates the initial ground track as the MAV enters the orbit. This initial ground track, χ_g, is the first ground track

heading used to calculate the orbit position, as discussed in section 3.5.4. The direct heading from the current position of the MAV to the POI is used as the initial ground track heading. While the MAV will not fly that direct heading to intercept the orbit, the MAV will fly a parallel path to intercept the orbit.

3.6 User Interface

A user interface is required to allow an operator to access the algorithms and have them automatically plug the waypoints into the flight management software. It serves as the view class, and likely the controller class as well, as discussed in section 3.3.1. The user interface provides an easy way to input the coordinates for a POI as well as interact with the flight management system. The interaction with the flight management system automatically passes variables such as current MAV position, sensor attitude and field of view, wind heading and speed, MAV attitude, MAV cruise speed, MAV stall speed, maximum bank angle, and other pertinent MAV parameters. This limits the required inputs on behalf of the operator, keeping the system simpler. Additionally, this interface allows the operator to end the orbit or overflight and resume the previously planned route. The user interface would be a necessary step to making this research ready for an operational environment. The development of a user interface should follow the logic layed out in sections 3.2 and 3.3. It would have to interface with a specific flight management software, such as Virtual Cockpit, which is used in the route surveillance research. The user interface is not a primary component of the sensor aimpoint research; therefore, it is not fully developed beyond a simple MATLAB command script which was usable in testing, but would not be desireable for an operational environment.

3.7 Conclusion

This chapter covered the development of the sensor aimpoint waypoint generation algorithms from the system use cases, to software development diagrams, to support algorithms, through the actual waypoint generation algorithms themselves. The algorithms generates waypoints to keep the sensor aimed at a desired POI while in the presence of wind. Algorithms for the support of the sensor aimpoint algorithms were

also discussed. The next chapter examines the testing, implementation, and results of the implementation of these algorithms.

IV. Results and Discussion

This chapter reviews the software and hardware equipment used to test and evaluate the sensor aimpoint algorithms. In addition to the test equipment, basic test procedures and results are discussed along with results and preliminary conclusions about those results.

4.1 Algorithm Implementation

The algorithms discussed in section 3.5 were implemented into software code in the engineering programming tool MATLAB. MATLAB provided powerful computational functions in an easy to understand and implement software package. The orbit and overflight algorithms were the main focus of the software coding, but they each had common supporting functions such as determining the sensor aimpoint, converting coordinates from geodetic to Cartesian systems, and estimating wind, to name a few. Additionally, the Dubins path algorithm was coded to transition the MAV from its initial position to the first sensor aimpoint waypoint. Appendices D and E present the code as written in MATLAB for the main and supporting algorithms.

4.2 User Interface for Testing

A user interface, as discussed in section 3.6, was not coded in MATLAB. MATLAB has limited functionality to develop graphical user interfaces. Additionally, while MATLAB is a standard software tool in engineering communities, it is not a standard tool for operational environments. A user interface that is a stand-alone software program with its own installation is more ideal. A user interface of this sort could be written in a programming language such as C++. Previous efforts of Sakryd and Ericson (2008) had created a user interface using this method. They transformed the completed MATLAB algorithms into C++ support files and were able to successfully call the algorithms through their user interface.

Since the user interface had not been completed for testing of the sensor aimpoint algorithms, a MATLAB script was written to create a crude, but useable, interface. In this interface, the test operator provides the necessary algorithm inputs into the MATLAB script. The script then calls the sensor aimpoint algorithms and support

algorithms as required for the given test. The script outputs a file of waypoints for the MAV to fly, formatted for Virtual Cockpit. Virtual Cockpit uses a plain text file for its waypoints, such as the one presented in Appendix F, so the style was very easy to create with MATLAB. The operator manually loads this waypoint file into Virtual Cockpit, manually commands Virtual Cockpit to upload the file to the Kestrel Autopilot on the MAV, and then manually commands the MAV to begin flying that set of waypoints. This method relies heavily on appropriate timing to get the MAV to the initial start position and heading when the operator commands the MAV to begin flying the waypoints. A C++ user interface that communicates directly with Virtual Cockpit can avoid the need for such careful timing by passing data such as MAV position and orientation directly into the algorithms. While not optimal, the MATLAB script method sufficiently tested how the algorithms functioned.

4.3 Computer Simulation

The hardware-in-the-loop (HIL) test equipment, discussed below, is very easy to set up and use. As the HIL equipment is maintained by the AFIT Advanced Navigation Technologies Center, the equipment is readily available and free of cost. It can be operated by a single person. Since it uses the same autopilot hardware as used in flight, no estimations of how the Kestrel autopilot works are required for a software simulation. This eliminates the use of software-in-the-loop testing as part of a build-up test approach to test and evaluate the sensor aimpoint algorithms. During the development and coding of the algorithms, however, MATLAB was used to generate graphical plots of where the waypoints would be based on changes in wind, look angle, sensor attitude, or aircraft attitude. Detailed plots and descriptions are given in Appendix G.

4.4 Test Equipment

To evaluate the orbit and overflight algorithms, as well as supporting algorithms, static software models were run in MATLAB, dynamic hardware-in-the-loop tests were conducted, and flight tests were accomplished. The following sections discuss the equipment used in each of those situations.

4.4.1 Autopilot and Flight Management Software. A commercial off-the-shelf (COTS) system was used for test and evaluation of the algorithms. The COTS system came from Procerus Technologies. It consists of a Kestrel autopilot, a communications box that translates and transmits computer commands to the Kestrel, and the Virtual Cockpit software. The Kestrel is a fully capable autopilot with three-axis accelerometers and MEMS gyros. It has a pitot static system and a GPS antenna input. The flight management software, Virtual Cockpit version 2.4, communicates with and controls multiple MAVs that use the Kestrel. Virtual Cockpit provides a basic "click-and-fly" operation allowing a user to easily input waypoints for the MAV, as well as displaying the location of the MAV above a map of the local flight area. Virtual Cockpit also has the capability to pass flight control variables to and from external software programs. This can allow the creation of additional software programs to run external algorithms, such as those developed for this research, and take outputs from those algorithms as waypoint inputs to Virtual Cockpit and the Kestrel.

Two variables in Virtual Cockpit deserve additional discussion due to their impact on how the MAV reacted when commanded to a sensor aimpoint flight path. The first is waypoint radius which is input into Virtual Cockpit as a distance in meters. Virtual Cockpit commands the MAV to fly from one waypoint to another in sequence. Typically, this results in the MAV flying a direct line of sight path between the waypoints. When the distance between the MAV and the waypoint became less than or equal to the waypoint radius, Virtual Cockpit marks that waypoint as having been reached, and sets the next waypoint in the command list as the goto waypoint for the MAV. Since the sensor aimpoint algorithms rely on the MAV to fly as close as possible to the waypoints commanded by the sensor aimpoint algorithms, the initial assumption was that the waypoint radius must remain small. This is considered especially important in the orbit case where the orbit radius is small and waypoints are close together. Too small of a waypoint radius, however, can establish a situation where the MAV misses a waypoint, perhaps because it is blown off course due to wind, and then it has to fly back to that waypoint until Virtual Cockpit marks it as reached.

The second important variable is cross track distance, also defined in meters. Virtual Cockpit defines this variable as the point in front of the MAV along the projected flight path to which the MAV would steer to intercept the flight path. The initial assumption was that the MAV should try to stick to the flight path commanded by the sensor aimpoint algorithms as best possible; therefore, the cross track value should be small. If the cross track value is large, the MAV might never get back on course before the next waypoint is commanded; therefore, the MAV might not actually fly the commanded course. Early testing, particularly flight testing on 13 November 2008, found that the MAV would make a hard bank to turn to reach the intercept point commanded by a small cross track value, which led to the MAV continually overshooting the course. This was a particular problem with the overflight algorithm. Future testing increased the cross track value to avoid this issue.

4.4.2 Simulated Flight Environment. Procerus Technologies recommends the Aviones simulator(Procerus Technologies, 2008), an open source unmanned aerial vehicle flight environment simulator developed by the Brigham Young University Human Centered Machine Intelligence. Aviones allows for a hardware-in-the-loop (HIL) configuration to conduct MAV flight simulations in the laboratory. MAV performance characteristics are input into a physics parameter file to model the MAV in Aviones. Aviones connect to the Kestrel autopilot via serial connections and provides simulated GPS signals and aircraft dynamics data. As long as the proper physics parameters are loaded, Aviones accurately simulates the MAV. To verify Aviones properly simulated the MAV in flight, profiles were created and flown both in flight test and HIL test to compare the MAV response.

Because no physics parameters were readily available for the BATCAM, it was not properly modeled in Aviones and could not be used as a HIL model. DeLuca (2004) had previously characterized the aerodynamic performance of the BATCAM; however, these values were not easily translated into the parameters required by Aviones, predominately due to the v-tail on the BATCAM. Aviones physics parameters for the Procerus testbed, the Unicorn UAV, were provided by Procerus, which allowed another simulated MAV to

be flown in the HIL tests. The AFIT Advanced Navigation Technologies (ANT) Center did not own a Unicorn UAV, so it could not be flown in flight tests.

Based on the work of Jodeh (2006), Sakryd and Ericson (2008), and Vantrease (2008), valid physics parameters were available for the SIG Rascal test MAV, which was available as an airframe for flight testing. This meant the SIG could both be flown and simulated. This allowed the same test profile to be flown during flight test and HIL tests to verify the accuracy of Aviones as a simulator, thus allowing future tests to only be conducted via HIL. These comparisons are discussed below.

One limitation of the Aviones is that it could not simulate variable winds. The physics parameter file can be manually changed while the simulation is running, including winds, but this is a tedious and slow process. An external program, e.g. MATLAB, can not change the file and have Aviones reread the physics parameters. This meant that only constant winds were available for HIL testing.

4.4.3 Hardware-in-the-Loop. As mentioned above, the hardware-in-the-loop (HIL) includes a Kestrel autopilot, a computer to run Aviones and Virtual Cockpit, serial connections to link the computer to the Kestrel and pass along the Aviones data, a communications box connected to the computer to send and receive communications between Virtual Cockpit and the Kestrel, along with the necessary power sources for all hardware involved. The HIL system ran the simulations in real time. All communications between Virtual Cockpit and the Kestrel are the same as they would be in flight, with the exception of the potential for less noise that might interfere with signal transmission and reception.

4.4.4 Test Airframes. As mentioned above, both the BATCAM and the SIG Rascal were available for flight testing purposes. The BATCAM had been used by Sakryd and Ericson (2008) and was used by Diamond *et al.* (2009). BATCAM systems had been made available to the ANT Center by the Air Force Research Laboratories Air Vehicles Directorate. The SIG Rascal 110 radio-controlled aircraft had been acquired and developed for the ANT Center by Jodeh (2006). The two airframes are discussed below.

4.4.4.1 BATCAM Description. The BATCAM is a small MAV used by the Air Force Special Operations Command for small-unit intelligence, surveillance, and reconnaissance purposes. It weighs less than a pound, is 24 in. long, and has a 21 in. wingspan. The wings are designed to flex and roll around the body of the airframe so the BATCAM can be easily transported in a 5 in. diameter tube that attaches to a rucksack. (Office of the Secretary of Defense, 2005) The BATCAM uses an electric motor as a power plant along with the Kestrel autopilot that connects to a pitot-static system and a GPS antenna. It has a cruise speed of 20 kts and a stall speed of 8 kts. The maximum bank angle was established as 30°. The BATCAM has two video cameras that transmit color video. The cameras are fixed in the body looking out the front of the BATCAM and out the left wing. The front camera is declined $-49°$ from aircraft centerline and the side camera is declined $-39°$. Each camera has a 48° horizontal field of view (FOV). Assuming a 640x480 resolution, this means a 40° vertical FOV. A more detailed description of the BATCAM is available in Diamond *et al.* (2009).

4.4.4.2 SIG Rascal 110 Description. The SIG Rascal 110 is a COTS radio-controlled aircraft that employs a high-wing and conventional landing gear, more commonly known as a tail-dragger configuration. It has a four-cycle aircraft engine as the main power plant. The Rascal uses a Kestrel autopilot that connects to a pitot-static system and a GPS antenna. It has a cruise speed of 40 kts and a stall speed of 20 kts. The maximum bank angle was established as 40°. The testbed Rascal has two Black Widow KX141 video cameras that transmit color video. The cameras are mounted on uniaxial gimbals, allowing them to pan in elevation, θ_s. The cameras on the Rascal were fixed at an elevation to keep with the intent of the research into the sensor aimpoint algorithms. The same elevation angles and FOVs used in the BATCAM were assumed for the Rascal flight tests. This is a valid assumption, since any sensor can be mounted into the Rascal for the purposes of the sensor aimpoint algorithms. Ultimately, the positioning of the cameras on the Rascals did not matter, due to an unfortunate accident with the Rascal during flight testing on 3 December 08, as discussed in section 4.10. A more detailed description of the SIG Rascal is available in Jodeh (2006).

4.5 Flight Testing General Information

Flight testing was conducted for the sensor aimpoint research along with other flight testing related to the route surveillance research conducted by Diamond *et al.* (2009). Due to restrictions on MAV operations, flight testing could not be conducted at Wright-Patterson AFB, Ohio, the location of AFIT. The Federal Aviation Administration required all Air Force operated MAVs to be flown in restricted airspace. The closest restricted airspace was located at Camp Atterbury, Indiana, a three-hour drive from AFIT. This meant that flight testing was of limited availability and could only be conducted sparsely. The limited number of flight test sorties increased the importance of HIL testing. To be able to see the MAVs, the flight testing was conducted only during daylight. Flight testing was conducted in the months of November and December, so the available daylight was limited to a few scant hours. In addition, sharing the test aircraft and test time with other route surveillance research meant very few test flights available to capture flight test data. The following sections describe the flight testing done for the sensor aimpoint research.

Testing evaluated the orbit and overflight algorithms without the Dubins path, as the Dubins code was not ready in time for either flight test. The test procedure had the operator interface with MATLAB code, inputting the POI location, desired flight altitude, number of waypoints, wind estimate, and initial ground track heading. MATLAB computed the sensor aimpoint flight path waypoints and saved them as a Virtual Cockpit flight plan file. The operator then loaded the flight plan file through Virtual Cockpit and uploaded it to the Kestrel autopilot. The operator would start the MAV in a rally orbit at or about the altitude calculated for the sensor aimpoint flight plan. When the MAV was approximately at the ideal location in the rally orbit to transition from the rally to the sensor aimpoint flight plan, the operator would command the MAV to fly the flight plan. The MAV would fly the plan while data was recorded. Upon completion of the overflight path or approximately 3 minutes of elapsed time in the orbit path, the operator would command the MAV to return to the rally orbit in preparation for the next sensor aimpoint test point.

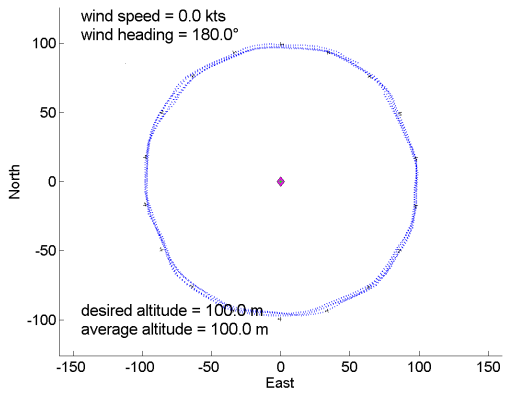
For the operator to input the wind, a wind estimate was necessary. Virtual Cockpit displayed a wind arrow with its estimation of the wind speed. The arrow shifted as Virtual Cockpit revised its wind estimation, which meant that the arrow was nearly always moving. The operator looked at the arrow for a short period of time before inputting the winds into the algorithm by estimating the heading of the arrow. The wind speed was also observed over a short period of time and a best guess average for the wind speed, usually within 0.5 kts, was used as the wind speed input.

Data was recorded using the telemetry data logging function in Virtual Cockpit. The variables logged included airspeed, aircraft attitude, aircraft position, and wind estimation. They were logged at 1 Hz. During data reduction, only the data points captured while the MAV was moving along the commanded sensor aimpoint flight path were kept, while the others were discarded.

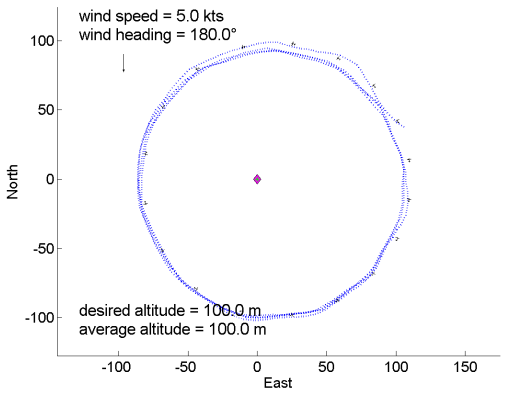
4.6 BATCAM Initial Hardware-in-the-Loop Testing

Initial hardware-in-the-loop (HIL) testing occurred on 7 November 2008. The BATCAM was simulated with the test equipment. At the time of this test, it was unknown that the physics parameters did not exist for the BATCAM. The data showed believable results for MAV movement in a two-dimensional plane. Data was not reviewed at the time to see how the MAV performed in a vertical field. The data from the HIL test was only looked at to see how the planned orbit waypoints and the resulting orbit path were affected by winds. This was especially pertinent since a flight test using the BATCAMs was due the following week.

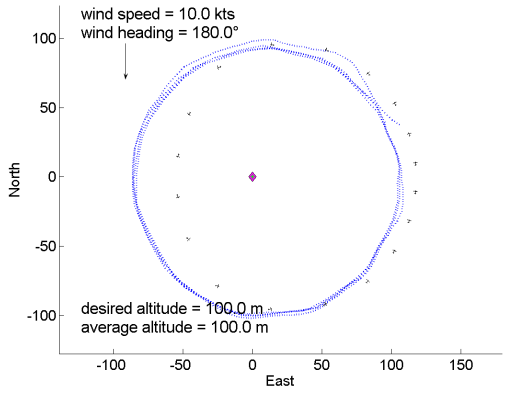
Since the BATCAM had a cruise speed of 20 kts, it was known that it could not fly an orbit with wind speed of 20 kts. Four test points were considered from 0 kts to 15 kts of wind in 5 kts increments. In each case, the wind was out of the north with a heading of 180°. Figure 4.1 illustrates the results from this testing. Based on the results from this testing, it was assumed that the BATCAM would work acceptably in winds up to 10 kts, with limited acceptable functionality in winds higher than that, with little benefit in winds exceeding 15 kts.



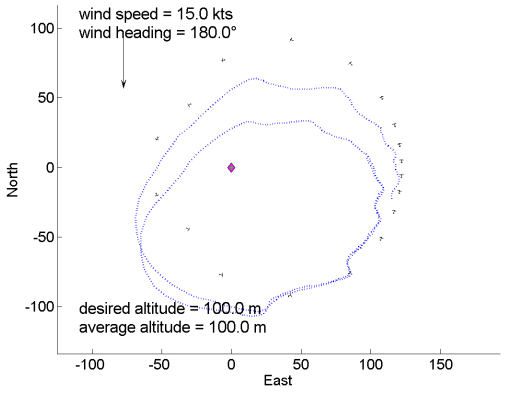
(a) $V_w = 0$ kts



(b) $V_w = 5$ kts



(c) $V_w = 10$ kts



(d) $V_w = 15$ kts

Figure 4.1: Initial HIL Testing of Orbits in Different Winds

4.7 BATCAM Flight Testing

On 13 November 2008, initial flight testing of the sensor aimpoint algorithms occurred, with the BATCAM as the test bed. The purposes of this testing were to observe the sensor aimpoint algorithm in a realistic environment as well as to gather data to validate the BATCAM hardware-in-the-loop (HIL) model. Data to serve both purposes was captured, although the BATCAM HIL model could not later be validated due to problems with the model, as discussed in section 4.4.2.

Additional route surveillance flight testing occured prior to the sensor aimpoint tests, moving the sensor aimpoint tests to later in the day. This reduced the available test time; still, a total of eight test points were gathered for the orbit and overflight algorithms. The orbit algorithm was tested with different altitudes and different numbers of waypoints. The overflight algorithm was tested with different altitudes, different look angles, and with both the front and the side sensor. A fixed POI was established at a latitude of 39.34360°N, longitude of −86.02980°E, and MSL altitude of 216 m. A waypoint radius of 10 m and a cross track value of 25 m were used.

Winds were out of the south-by-southwest, ranging from 10.0 kts to 11.5 kts, with headings ranging from 000° to 020°. Based on the HIL testing from 7 November, it was known that these winds were higher than desired for the first flight testing of the algorithms; however, given the difficulties in getting flight test sorties, the testing continued.

Tables 4.1 and 4.2 provide details of the test points captured during flight testing. The orbit test points were chosen to see how the algorithm reacted at various altitudes and number of waypoints. The overflight look angles were chosen to keep the MAV in view of the ground control station and to minimize flight time over buildings and roads near the test site. Initially, the overflight test points were to be conducted at various altitudes to see how altitude affected performance; however, due to time limitations, multiple passes along the same look angle but at different altitudes were not able to be accomplished.

For both test types, a standard rally point was established at the geodetic coordinates of 39.34426°N, −86.02945°E, with a radius of 75 m. A Virtual Cockpit rally

point was altitude independent; when commanded to go to the rally point and orbit counterclockwise, the MAV maintained the altitude it had at the time of the command. In addition, the operator could command a different altitude as desired. During testing, between each test point, the operator commanded the MAV to go to the rally point and orbit at the altitude needed for the subsequent test point. The MAV made two orbits to get the best estimation of the wind. During this orbit, the operator used the MATLAB interface to input the pertinent parameters for the algorithm to be tested, including the wind. When the MAV reached the northernmost point in the orbit, with an approximate heading of 270°, the operator commanded the MAV to begin the sensor aimpoint flight path.

4.7.1 Orbit Testing. Plots of the orbit waypoints, actual MAV path, and sensor footprint are given in Figure 4.2. The MAV did pretty well reaching the waypoints except at the 50 m altitude. At this lower altitude, the MAV had to make tighter turns than it did at the other altitudes. When the MAV flew with a tailwind, it blew well off course and then tried to make tighter turns to get back on course. Part of the error in this process was in the control loops in the Kestrel. The Kestrel only knew the next waypoint and attempted to use a straight path to get from its current position to the position of the waypoint. If the Kestrel knew that the flight path would continue to curve, it could do a better job of planning to reach the next few waypoints and likely not have as significant of a problem with being blown off course as was seen in the 50 m altitude orbit.

Table 4.3 lists statistics for the distance from the sensor aimpoint to the POI for each of the orbit test points. The sensor aimpoint was calculated for each telemetry point using the algorithms created for this research. The FOV at each telemetry point was similarly calculated using the footprint algorithm for this research. The footprint area

Table 4.1: Orbit Test Points from 13 Nov 08

Altitude	**n**	χ_g	V_w	χ_w
100 m	12	210°	11.5 kts	015°
50 m	36	210°	10.0 kts	000°
100 m	36	210°	11.5 kts	020°
150 m	36	210°	10.0 kts	015°

Table 4.2: Overflight Test Points from 13 Nov 08

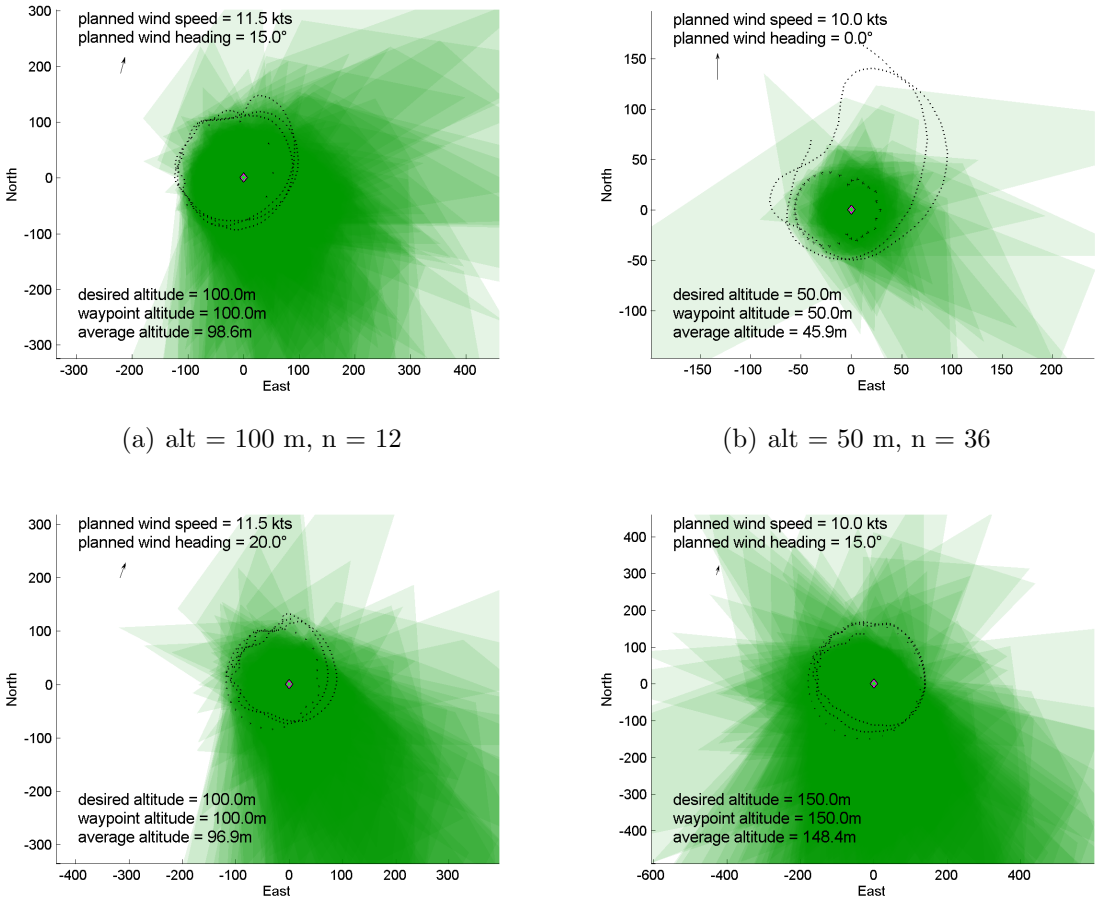
Altitude	**Sensor**	χ_L	V_w	χ_w
50 m	Side	045°	10.5 kts	010°
50 m	Side	095°	10.0 kts	010°
50 m	Front	135°	10.0 kts	010°
75 m	Front	190°	11.5 kts	015°

was checked to see if the POI lay within the footprint. The total number of telemetry data points where the POI lay within the footprint was summed and divided by the total number of data points to determine the percentage of the time that the POI was in the FOV. Additionally, the radial distance along a two-dimensional horizontal plane was then calculated from the sensor aimpoint to the POI. The statistics for this radial distance are listed in the table.

Table 4.3: Orbit Aimpoint Statistics

Alt	**n**	**Data Pts**	**% In FOV**	**RMS**
100 m	12	214	87.85%	55.67 m
50 m	36	100	50.00%	66.74 m
100 m	36	188	60.64%	126.57 m
150 m	36	319	53.29%	125.40 m

An interesting difference appeared at the 100 m altitude orbits with respect to the number of waypoints and the sensor aimpoint accuracy. Since the altitude and wind speed were the same for both test runs, the only difference in test procedure was the number of waypoints. Considering how pitch (θ_a) and roll (ϕ_a) changed during the orbit can help illustrate why the two similar orbits achieved such different aimpoint statistics. Tables 4.4 and 4.5 show the pitch and roll angles for each orbit. The orbit algorithm assumes 0° pitch, so the commanded pitch for both orbits is 0°. The algorithm calculates a different roll angle for each waypoint. This angle changes based on wind speed and heading, as indicated in Equation 3.19. The commanded roll listed in Tables 4.4 and 4.5, therefore, is calculated for the 100 m orbits by taking the mean of ϕ_a calculated for each waypoint. The mean roll is then compared to the mean commanded roll for the orbit. Figure 4.3 shows the pitch and roll angle data plotted for each 100 m orbit. The dotted line is the actual angle from the telemetry captured by the Kestrel. The solid line is the mean angle and the dashed line is the commanded angle.



(a) alt = 100 m, n = 12

(b) alt = 50 m, n = 36

(c) alt = 100 m, n = 36

(d) alt = 150 m, n = 36

Figure 4.2: Orbit Flight Path and Sensor Footprints

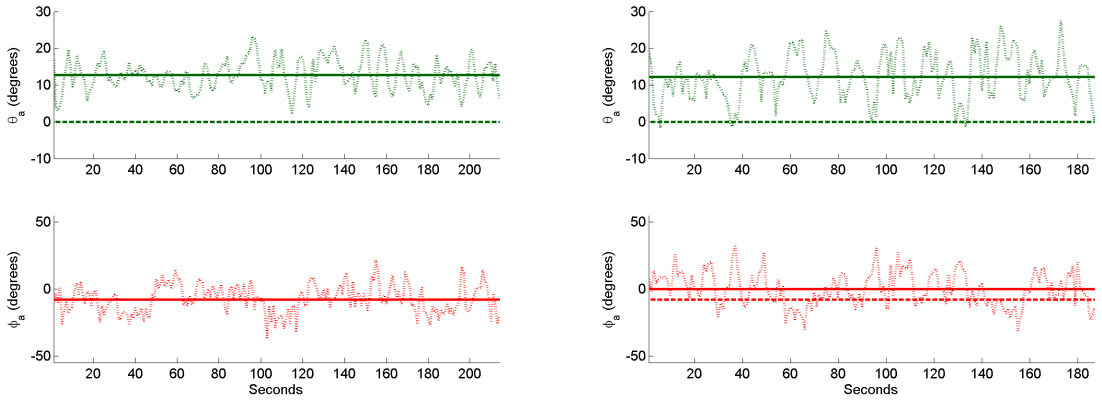
Between the tables and the figure, two conclusions can easily be drawn. The first is the small angle approximation assumed by the orbit algorithm for pitch is inaccurate for the BATCAM airframe. When the BATCAM was in a sustained orbit, it pitched up to maintain altitude. This resulted in the sensor aimpoint being further away from the POI. Figure 4.4 shows the difference in aimpoint location relative to a POI for a MAV in an 8° left bank with pitch of 0° (in magenta) and pitch of 12° (in cyan). The MAV is located so the 0° perfectly aligns the aimpoint on the POI. The aimpoint for the MAV in

Table 4.4: Attitude Stats for Orbit: alt = 100 m, n=12

	Planned	**Mean**	**Min**	**Max**	**StDev**	**Var**
Pitch, θ_a	0.0°	12.7°	2.1°	22.9°	4.3	18.4
Roll, ϕ_a	−8.0°	−7.7°	−37.3°	22.0°	10.7	113.5

Table 4.5: Attitude Stats for Orbit: alt = 100 m, n=36

	Planned	**Mean**	**Min**	**Max**	**StDev**	**Var**
Pitch, θ_a	0.0°	12.2°	−1.8°	27.6°	6.4	40.7
Roll, ϕ_a	−7.9°	−0.1°	−31.3°	33.1°	12.7	162.4



(a) n = 12 (b) n = 36

Figure 4.3: Attitude Angles for 100 m Orbits

the 12° pitch up attitude is 21.4 m radially away from the POI. In both cases, however, the POI is covered by the FOV of the sensor. To an operator, this may mean that the POI is not centered in the sensor display, but the operator could still see the POI in the display.

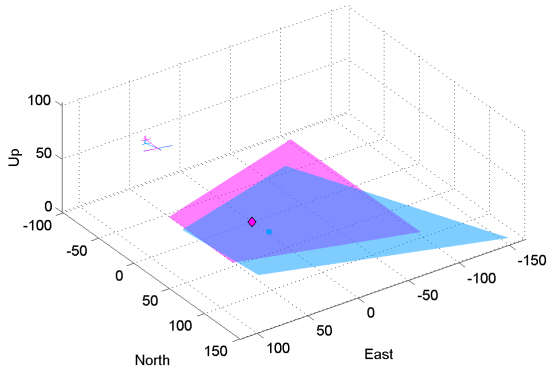


Figure 4.4: Comparison of Aimpoint in an Orbit Roll at Different Pitches

The second conclusion drawn from the information in Tables 4.4 and 4.5 as well as Figure 4.3 is that the BATCAM rolled about the longitudinal axis much more during

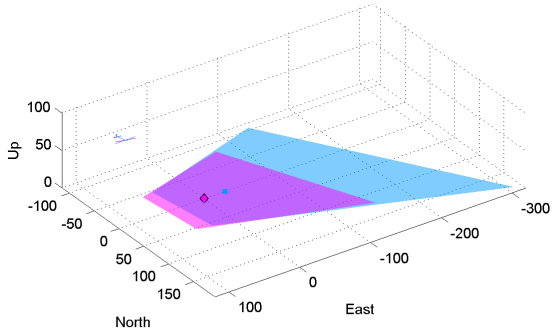
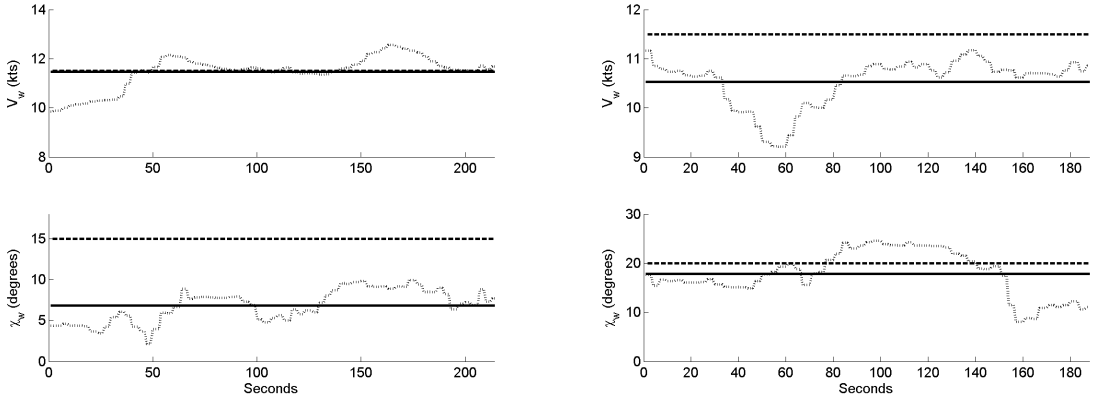


Figure 4.5: Comparison of Aimpoint with $\phi_a = -0.1°$ and $\phi_a = -7.7°$

the orbit with 36 waypoints than in the 12 waypoint orbit. In the 12 waypoint orbit, the mean roll was nearly equal to the commanded roll. In the 36 waypoint orbit, the mean roll was nearly 0°, much different than the commanded roll. Figure 4.5 shows the difference of the two sensor aimpoints and footprints, assuming the bank of $\phi_a = -7.7°$ would put the sensor aimpoint on the POI. In this case, the bank of $\phi_a = -0.1°$ would put the aimpoint 29.5 m away, radially, from the POI. Figure 4.3(b) further illustrates the drastic change in the rolling of the BATCAM, especially compared to the orbit with fewer waypoints. The rolling back and forth could have been caused by a tight cross track value. This may have caused the MAV to overshoot a projected path to reach the next waypoint, then it would roll the other direction to reach the next waypoint. This would be more prominent in the case with the higher number of waypoints because the MAV would have more legs in the flight plan and more turns between legs. Additionally, the rolling may have been caused by gusty winds.

The wind data captured by the Kestrel is plotted in Figure 4.6. Tables 4.6 and 4.7 show the statistics for that data. The winds were very similar for the two test points. For the first, the mean of the wind speed remained the same as planned. The mean wind heading was different than planned wind heading, but with a small variance. The wind speed for the second test, with 36 waypoints, was lower than planned, but by a marginal amount. The mean wind heading was close to the planned wind heading, but with a large variance. Such a variance in the wind heading could have caused the MAV to roll



(a) n = 12 (b) n = 36

Figure 4.6: Wind Data for 100 m Orbits

more to counter the change in wind direction, but it cannot be conclusively determined that this would cause the roll. While the difference in mean roll angles may have caused the difference in the percentage of the time the POI was in the FOV, the cause for the difference in roll angles cannot be decisively decided.

Table 4.6: Wind Stats for Orbit: alt = 100 m, n=12

	Planned	**Mean**	**Min**	**Max**	**StDev**	**Var**
Speed, V_w	11.5 kts	11.5 kts	9.8 kts	12.6 kts	0.7	0.4
Heading, χ_w	15.0°	6.8°	2.2°	9.9°	2.0	3.8

Table 4.7: Wind Stats for Orbit: alt = 100m, n=36

	Planned	**Mean**	**Min**	**Max**	**StDev**	**Var**
Speed, V_w	11.5 kts	10.5 kts	9.2 kts	11.2 kts	0.5	0.2
Heading, χ_w	20.0°	17.9°	8.1°	24.5°	4.6	21.5

Tables 4.8 and 4.9 give the MAV attitude statistics for the 50 m and 150 m orbits. At the 50 m altitude, the MAV experienced many more perturbations in attitude. At the 150 m altitude, the MAV did not have to make as tight of turns and it had more time to correct its course before reaching the next waypoint. The trouble the MAV experienced at the lower altitude is further evident in Figure 4.2(b), where the flight path track shows the MAV consistently blown off course when it flew with a tail wind.

Table 4.8: Attitude Stats for Orbit: alt = 50 m, n=36

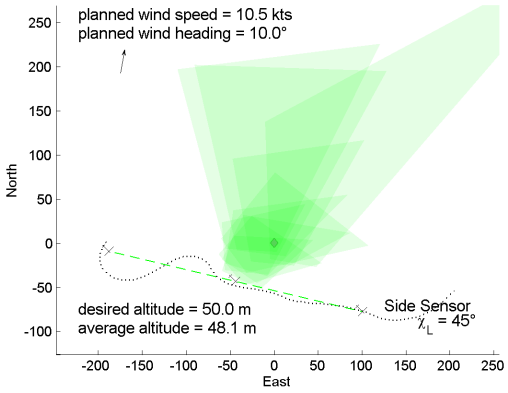
	Planned	**Mean**	**Min**	**Max**	**StDev**	**Var**
Pitch, θ_a	0.0°	12.4°	−15.3°	22.7°	7.9	61.6
Roll, ϕ_a	−12.6°	−9.2°	−35.9°	27.3°	12.4	154.9

Table 4.9: Attitude Stats for Orbit: alt = 150 m, n=36

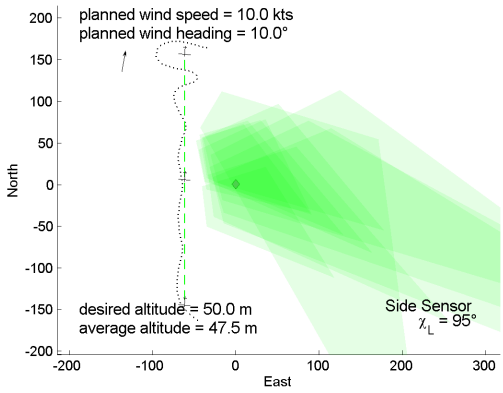
	Planned	**Mean**	**Min**	**Max**	**StDev**	**Var**
Pitch, θ_a	0.0°	11.3°	−2.5°	24.7°	5.6	31.2
Roll, ϕ_a	−4.2°	1.1°	−18.8°	22.1°	7.4	54.6

4.7.2 Overflight Testing. The MAV was rarely lined up with the overflight path by the first waypoint because it flew line of sight from the time it was commanded to fly the overflight path until it reached the first waypoint. Figure 4.7 shows the flight paths and footprints for each overflight case. Fewer data points were collected for each overflight test point because the MAV flew a short flight path along the waypoints, opposed to the orbit case where the MAV continually circled the POI, in theory keeping the POI in the FOV throughout the orbit. The telemetry data point where the POI first entered the FOV and the last data point before the POI exited the FOV were taken as the first and last possible points the POI could have been in the FOV.

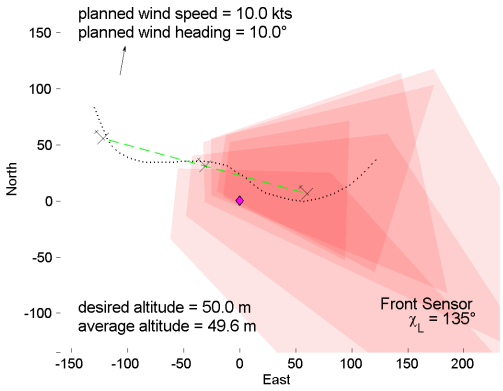
Statistics for the distance from the sensor aimpoint to the POI and percentage of time the POI was in the FOV are listed in Table 4.10. All but the overflight with the side sensor and a look angle of 95° share high rates of the POI being in the FOV. Consideration of the flight path for the exceptional test point, as shown in Figure 4.7(b), shows that the MAV had two opportunities for the POI to enter the FOV before the MAV aligned its track with the planned track. One plotted footprint has a corner at the approximate location of 110 m North and -15 m East. This is the footprint that first had the POI in the FOV. It occurred at 21 s into the test data when the MAV had just turned from a heading of approximately 105° to a heading of 250°. The next data point where the POI was in the FOV occurred at 35 s, just after the MAV crossed the overflight planned route at the approximate location of 58 m North and -64 m East. If this second instance was considered the first time the POI was in the FOV, this test run would have had an 83.33% rate for the POI being in the FOV. An interesting affect of the MAV turning in and out from the commanded path was that the POI entered the FOV



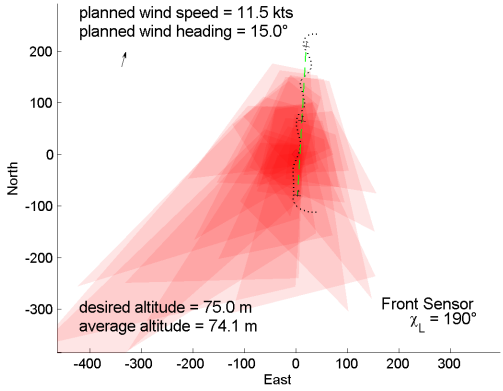
(a) Side Sensor, $\chi_L = 045°$



(b) Side Sensor, $\chi_L = 095°$



(c) Front Sensor, $\chi_L = 135°$



(d) Front Sensor, $\chi_L = 190°$

Figure 4.7: Overflight Flight Path and Sensor Footprints

when the MAV was further North than if it had been flying the path of the algorithm perfectly. If it had been flying the overflight commanded path, the POI would not have entered the FOV until the MAV was at the approximate location of 35 m North and -64 m East. Because the MAV was turning left and right, it managed to get the POI in the FOV earlier; however, this also allowed the POI to not be in the FOV during some turns.

As indicated in Table 4.10, the overflight with the side sensor and a look angle of 95° had more than twice as many data points as the other overflight passes at 50 m. This resulted from an outlier instance when the POI entered the FOV, occuring 24 s into the data. This instance occurred when the MAV overshot the initial waypoint in the flight plan and then oscillated across the intended flight path, as indicated in Figure 4.7(b).

If this outlier is excluded, then this overflight pass would only have 11 s of data, similar to other two passes at 50 m. The overflight at 75 m had more than twice as many data points, a function of the increased altitude.

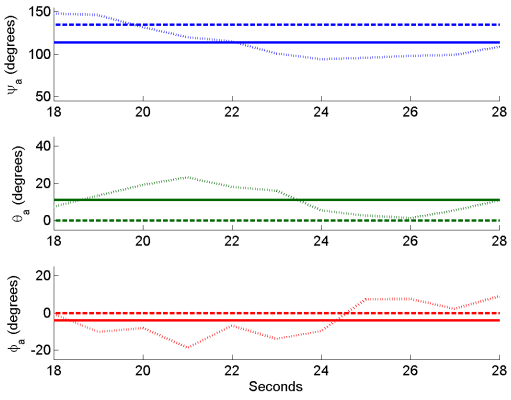
Table 4.10: Overflight Aimpoint Statistics

Alt	χ_L	**Data Pts**	**% In FOV**
50 m	45°	11	90.91%
50 m	95°	26	42.31%
50 m	135°	6	100.00%
75 m	190°	25	80.00%

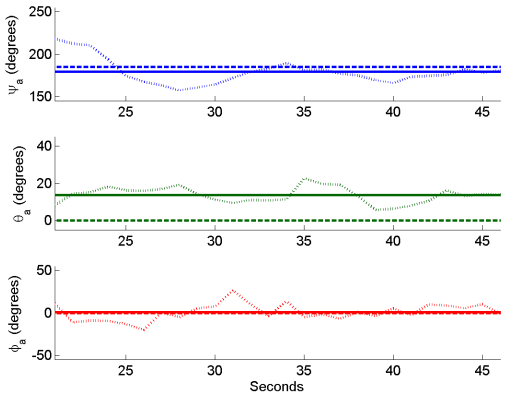
In all cases, the POI may have been in the FOV a large percentage of the time, but the sensor aimpoint was usually not near the POI. Figure 4.8 shows the attitude angles of the MAV during the time frame that the data points were collected, along with the commanded attitude and mean attitude during that time. Attitudes in all three axes are presented, since the yaw attitude or aircraft heading (ψ_a) should have remained the same according to the overflight path planned by the algorithm. In every instance, the mean pitch was significantly higher than planned, as discussed above for the orbit test points. In all cases, the fact that the MAV kept overshooting and overcorrecting to get back on course clearly affected the heading angles and likely the percentage of time the POI was in the FOV.

Because the MAV was on each overflight path for a much shorter time than any given orbit path, the wind affected the overflight paths much less. Figure 4.9 shows the wind data from the entire overflight telemetry, not just during the time period with the POI in the FOV. While the mean wind speed and mean wind heading differed in all cases from the planned wind, the differences were minor. The difference in mean wind speed from planned wind speed were always less than 1.5 kts, and the heading was always within 10° of the planned heading.

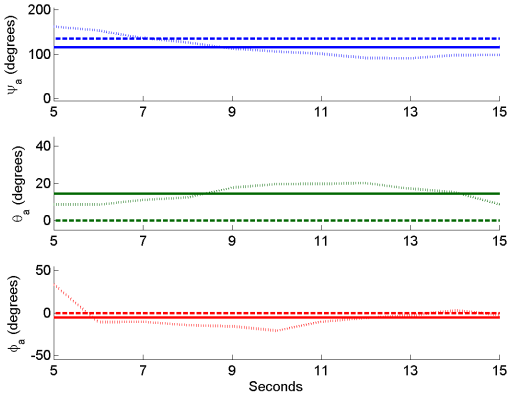
The BATCAM flight testing was not in the most favorable conditions, with winds greater than or equal to 50% of the cruise speed for the MAV. Such high winds likely affected flight performance of the MAV, causing the MAV to toss and turn in the sky. Additionally, the MAV, either by aerodynamic design or due to the winds, flew at a higher pitch angle than expected. For these reasons, the percent of time the POI was in the



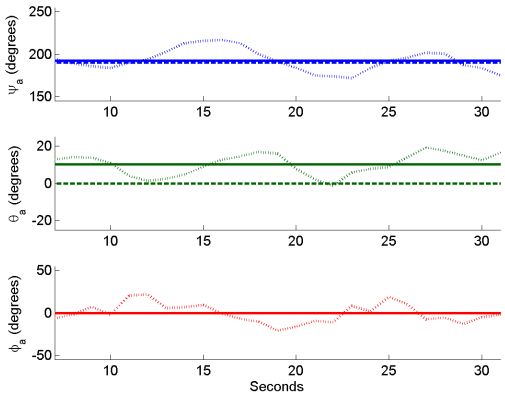
(a) Side Sensor, $\chi_L = 045°$



(b) Side Sensor, $\chi_L = 095°$



(c) Front Sensor, $\chi_L = 135°$



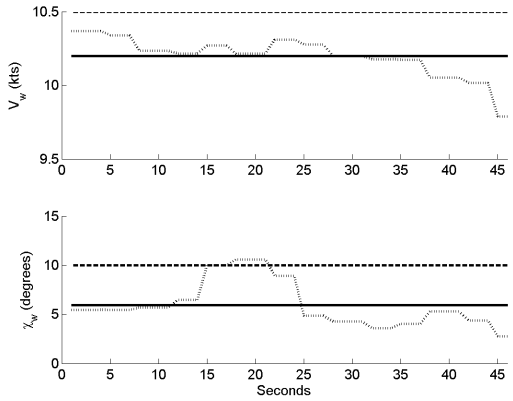
(d) Front Sensor, $\chi_L = 190°$

Figure 4.8: Attitude Angles for the Overflight Paths

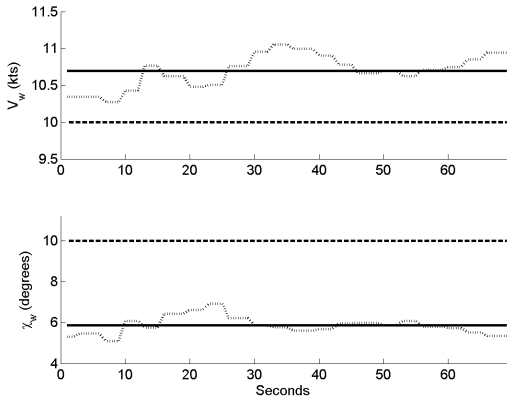
FOV was less than 100%. The testing met the first objective, however, by providing data for the sensor aimpoint algorithms in a realistic environment. Such a windy environment may not be well suited for flying the BATCAM in general, especially for a surveillance mission. The second objective of the BATCAM flight testing, to gather data to validate the HIL model, was also achieved, although challenges to validation of the BATCAM HIL model were discovered during follow-up HIL testing, as discussed below.

4.8 BATCAM Follow-Up Hardware-in-the-Loop Testing

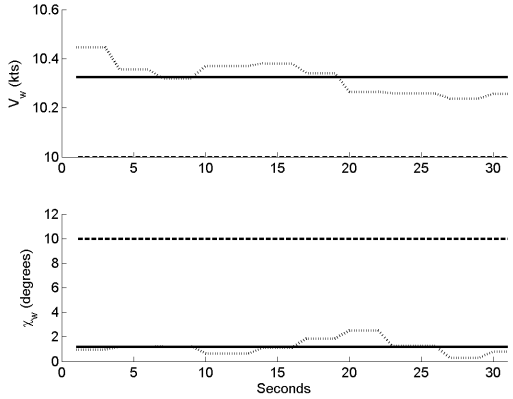
Hardware-in-the-loop (HIL) testing on 24 November 2008 followed the BATCAM flight testing in an attempt to validate the HIL model of the BATCAM. During the HIL testing, the same test points that were flown for the flight testing were simulated in the



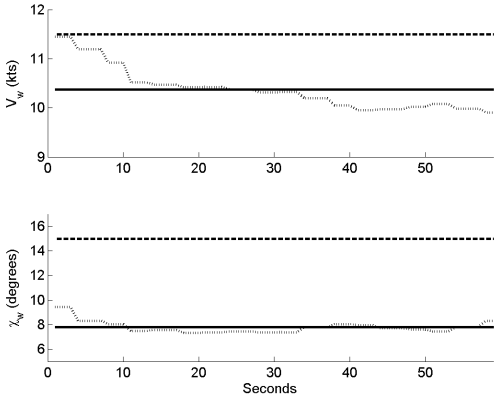
(a) Side Sensor, $\chi_L = 045°$



(b) Side Sensor, $\chi_L = 095°$



(c) Front Sensor, $\chi_L = 135°$

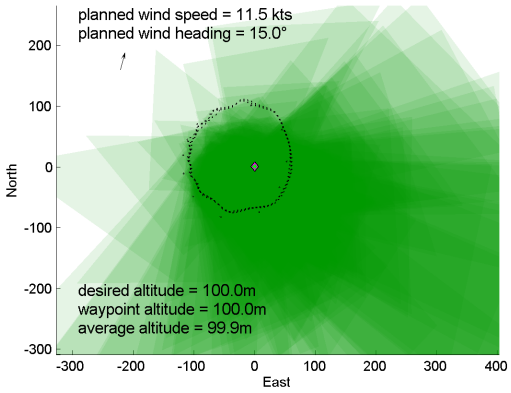


(d) Front Sensor, $\chi_L = 190°$

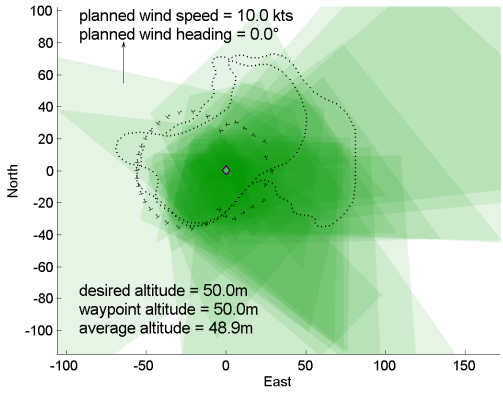
Figure 4.9: Wind Data for the Overflight Paths

test laboratory. The validation of the BATCAM HIL model was unsuccessful due to an inaccurate BATCAM physics model necessary for the flight environment simulator, Aviones. From an overhead perspective, the BATCAM flew very well, as indicated in Figures 4.10 and 4.11. Only in the 50 m altitude orbit did the MAV not keep to the planned path at least as well as BATCAM did during flight testing. When the pitch commands were analyzed, however, the problem with the physics model became apparent.

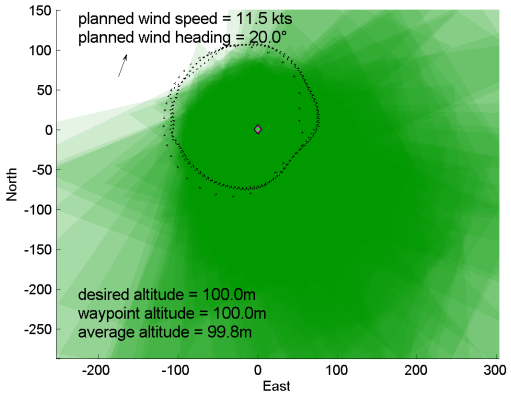
Figure 4.12 shows the MAV pitch attitude and roll altitudes for two of the test points conducted for the BATCAM HIL tests. The pitch angle repeatedly oscillated in a slow sinusoidal pattern with a high frequency of rapid changes in pitch up and pitch down. The roll also had a sinusoidal pattern with an even higher frequency and amplitude



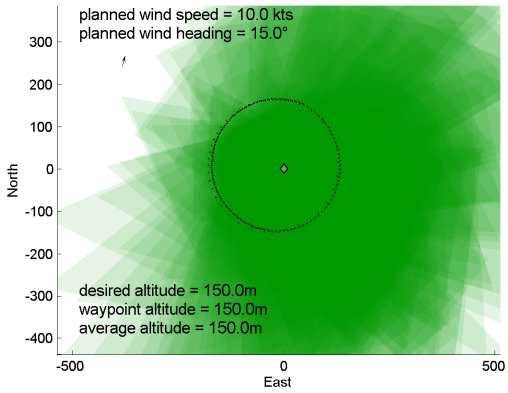
(a) alt = 100 m, n = 12



(b) alt = 50 m, n = 36



(c) alt = 100 m, n = 36

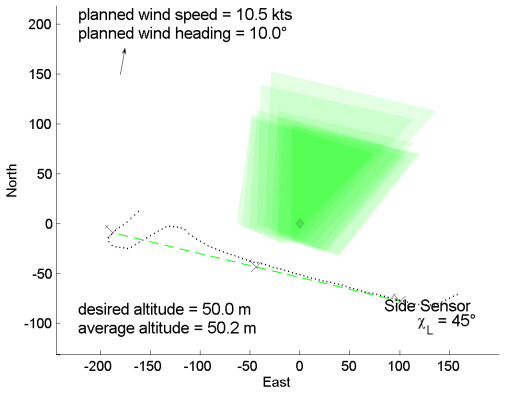


(d) alt = 150 m, n = 36

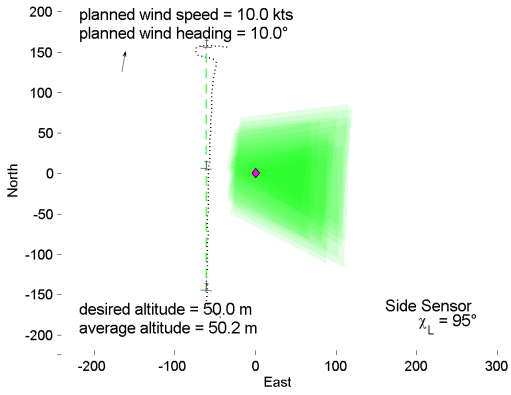
Figure 4.10: Orbit Flight Path and Sensor Footprints

oscillation, banking from left to right and back again. Because the mean roll was nearly the commanded roll, the overhead image of the track of the BATCAM appeared correct; however, the actual performance of the model was inaccurate. Figure 4.13 displays the altitude over the entire test run for the same two test cases. These each show the MAV slowly lost altitude despite having a positive pitch.

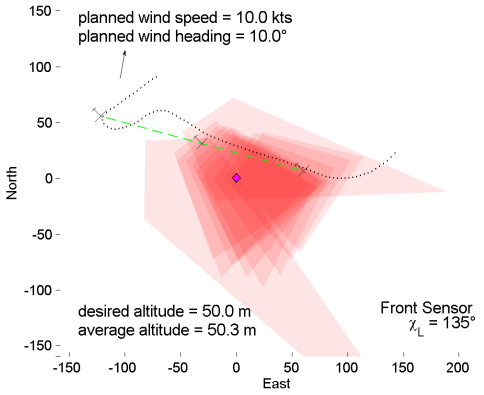
Investigation into the problems with attitude and holding altitude revealed that an accurate physics model for the BATCAM was lacking for use in Aviones, as discussed in 4.4.2. Troubleshooting was conducted using the physics model for the Procerus Unicorn UAV and the SIG Rascal, both of which had been previously proven, the former by Procerus and the latter during testing conducted by Sakryd and Ericson (2008) and Vantrease (2008). Neither of these physics models exhibited similar problems. De-



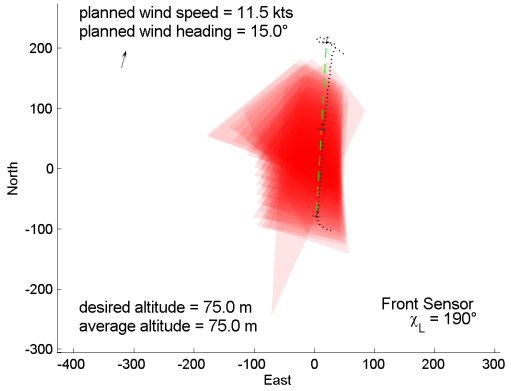
(a) Side Sensor, $\chi_L = 045°$



(b) Side Sensor, $\chi_L = 095°$



(c) Front Sensor, $\chi_L = 135°$



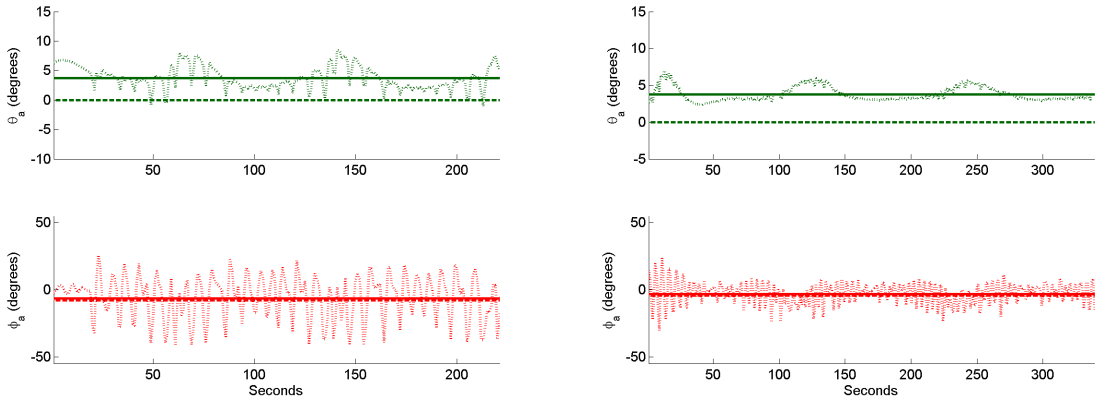
(d) Front Sensor, $\chi_L = 190°$

Figure 4.11: Overflight Flight Path and Sensor Footprints

constructing the work accomplished by DeLuca (2004) was not deemed time-efficient, particularly when HIL and flight testing could both be accomplished using the Rascal. BATCAM testing of the sensor aimpoint algorithm was suspended following the failures of the BATCAM HIL model.

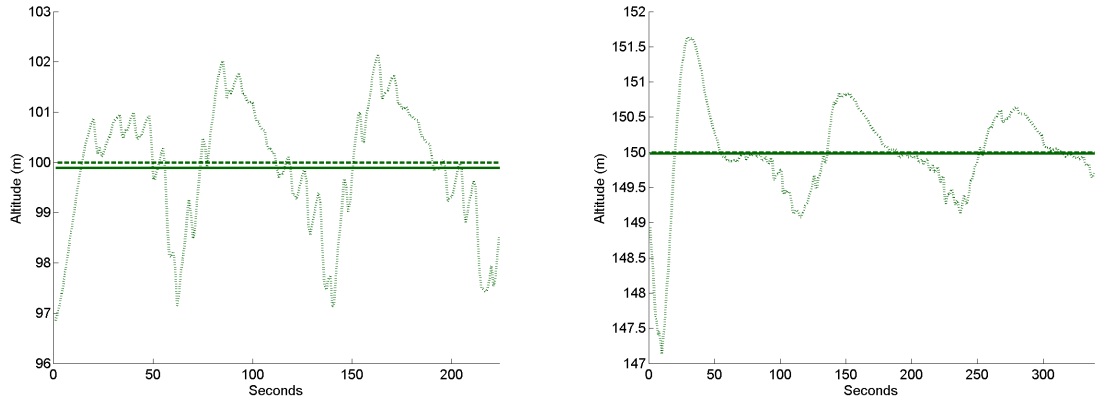
4.9 SIG Rascal Initial Hardware-in-the-Loop Testing

Initial hardware-in-the-loop (HIL) testing for the SIG Rascal began on 24 November 2004. The main test objective of the Rascal HIL testing was to see how the Rascal flew with two different sets of cross track, waypoint radius, and stall speed values. The same test points that were used for the BATCAM flight test on 13 November were used for the Rascal HIL test and listed in Tables 4.1 and 4.2. Initially, a waypoint radius of



(a) alt = 100 m, n = 12 (b) alt = 150 m, n = 36

Figure 4.12: Attitude Angles Representative of HIL Testing



(a) alt = 100 m, n = 12 (b) alt = 150 m, n = 36

Figure 4.13: Altitudes Representative of HIL Testing

10 m and a cross track value of 35 m were used for the testing. This was followed by a second round of testing at the same test points but with a waypoint radius of 50 m and a cross track value of 300 m. This second set of values were the same as those used by previous testing accomplished with the SIG Rascal by Sakryd and Ericson (2008). The stall speed used for the first test was 8 kts, the same as the BATCAM. During HIL testing, it was discovered the Rascal never flew less than 20 kts, so for the testing with the 300 m cross track, the stall speed was increased to 20 kts. This higher stall speed, along with the winds used in the testing, would not allow the Rascal to fly lower than 150 m. For example, with a 10 kts wind, the orbit algorithm calculated an altitude of

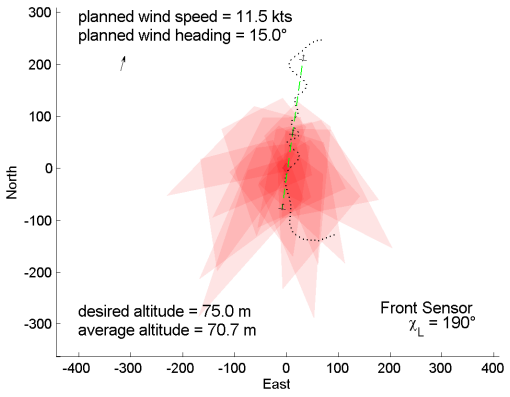
150 m. With an 11.5 kts wind, the calculated altitude was 170 m. Because the orbit test points with the higher stall speed were so dissimilar to the BATCAM test points, the Rascal HIL test was not run for the orbit. Without orbit data for both sets of stall speed and cross track values, analyzing the orbit provided little value added. Comparing the overflight data from the two cases, however, showed the importance of running the algorithms using the correct inputs for the MAV.

The main difference between the data from the different tests with the cross track values of 35 m and 300 m was that the latter had a much smoother flight path. When using the 35 m cross track, the Rascal overshot the planned flight plan more than the BATCAM did, likely because it could not turn as quickly as the BATCAM. With the 300 m cross track, the Rascal had few if any oscillations across the intended flight path. Figure 4.14 displays the differences of the two paths and footprints based on the different cross track values.

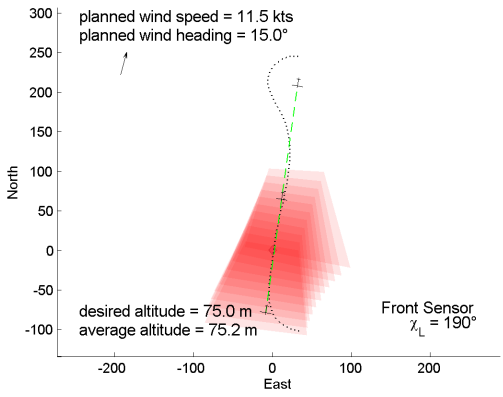
The different cross track values also affected the statistics for the aimpoint in the overflight scenario. Table 4.11 shows the statistics for the 35 m and 300 m cross tracks. One reason the former has more data points for each of the overflight test points was that the POI first entered the FOV earlier, as the MAV flew across the intended flight path. The reason for the greater number of data points at the higher altitude was that the sensor cast a larger footprint, so as the MAV moved at the same airspeed, the POI could stay in the series of footprints for a longer time. What the tables indicate is that the larger cross track and waypoint radius better suited the Rascal and should be used for future testing. Additionally, the BATCAM would likely benefit from a larger cross track value; unfortunately, additional BATCAM flight testing, as the HIL testing was inaccurate, was not pursued further during this research effort.

Table 4.11: Compare 35 m to 300 m Cross Tracks

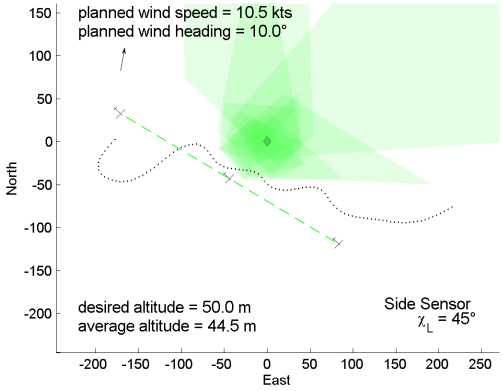
		35m		**300m**	
Alt	χ_L	**Data Pts**	**% In FOV**	**Data Pts**	**% In FOV**
50 m	45°	20	45.00%	1	100.00%
50 m	95°	33	24.24%	6	100.00%
50 m	135°	11	45.45%	4	100.00%
75 m	190°	30	46.67%	11	100.00%



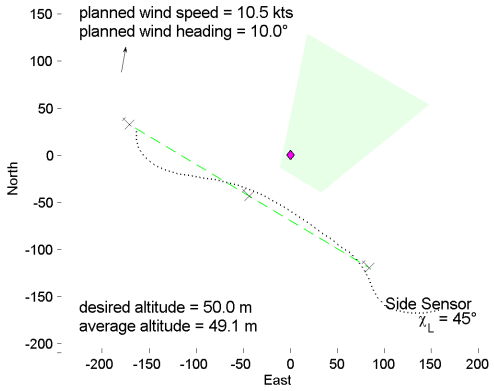
(a) 35 m Cross Track



(b) 300 m Cross Track



(c) 35 m Cross Track



(d) 300 m Cross Track

Figure 4.14: Overflight Flight Path and Sensor Footprints Comparing Cross Tracks

4.10 SIG Rascal Flight Testing

On 2 December 2008, flight testing was conducted using the SIG Rascal. The main purpose of this testing was to gather data to validate the HIL model for the Rascal. Once the HIL model was validated, all subsequent testing of the sensor aimpoint algorithms could be accomplished in that manner with a high degree of confidence in the results of the tests. A second objective of testing the Rascal was to gather data from an actual MAV. The first objective was met while the second objective was hampered by problems encountered during testing.

The Rascal was flown later in the day after other route surveillance flight tests with the BATCAM. Since the prior Fleeting Target research flew the Rascal with Virtual Cockpit 2.3, the first flight of the Rascal on 2 December was used as a functional check

flight to confirm the Rascal was usable with Virtual Cockpit version 2.4. As the pilot brought the Rascal in for its first landing, the Rascal experienced a wingtip stall and it crashed on the runway. The crash sheared off the landing gear and damaged the camera housing. The Rascal was able to be repaired and flown later that afternoon, but the available time to collect data was severely limited. As a result, only a few test points were collected; these few points, however, provided enough data to compare to HIL test data and verify the HIL could adequately simulate the Rascal.

This testing was an extension of the BATCAM testing completed on 13 November and followed the same basic procedure. One change was that the operator moved the rally point to better transition from rally to overflight path. This was only the case for the overflight test. During the tests, winds varied between 8.4 and 9.4 kts, out of the southeast, with an estimated heading (wind arrow pointed toward) of 340°. Three good data points were captured: an orbit at a planned altitude of 50 m, an orbit at a planned altitude of 150 m, and an overflight with the front sensor at an altitude of 50 m. Two other overflight test points were attempted, but telemetry data was not properly captured. Tables 4.12 and 4.13 detail the test points. Flight testing with the Rascal used a cross track of 300 m and a waypoint radius of 50 m, the same as that used by Sakryd and Ericson (2008) during their testing. Similarly, the maximum bank angle was 0.7 rad or 40° and the stall speed was set at 20 kts.

Table 4.12: Orbit Test Points from 2 Dec 08

Des Alt	**Cmd Alt**	**n**	χ_g	V_w	χ_w
50 m	100 m	36	160°	8.4 kts	340°
150 m	150 m	36	160°	9.4 kts	340°

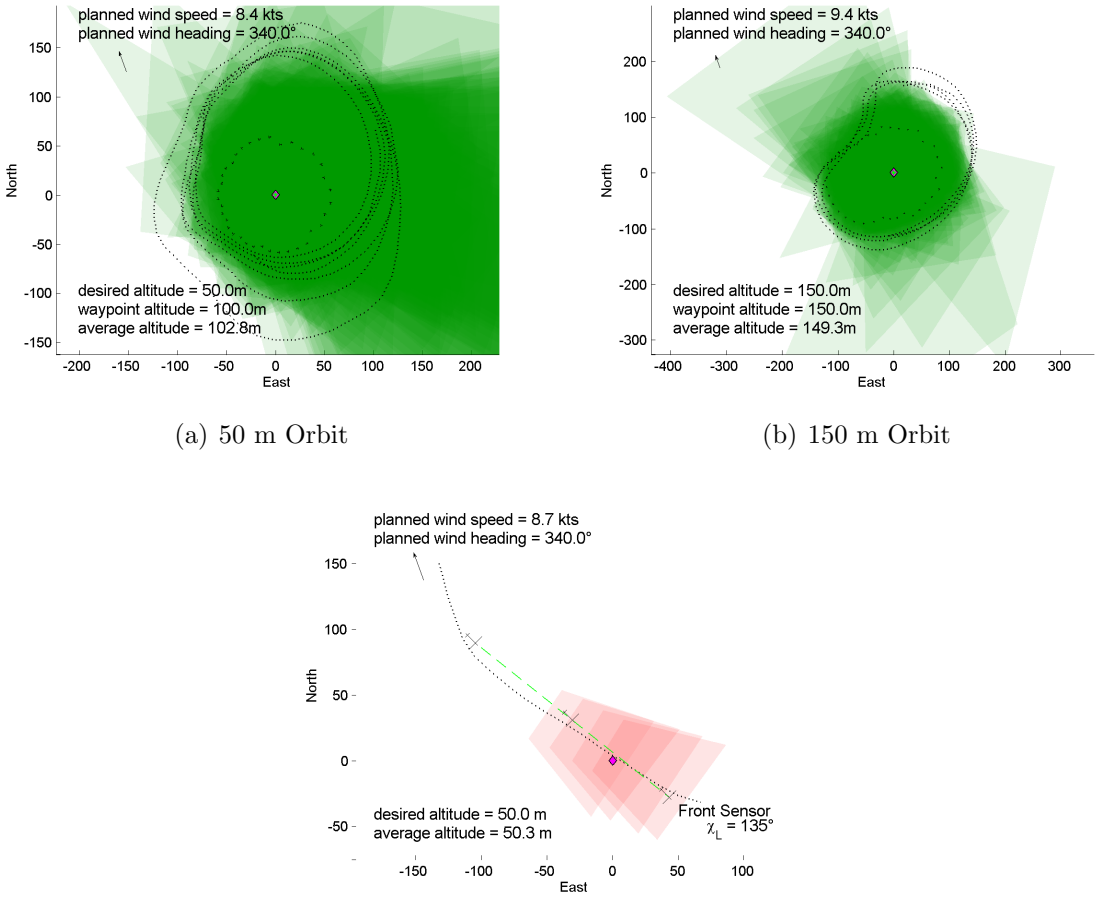
Table 4.13: Overflight Test Point from 2 Dec 08

Altitude	**Sensor**	χ_L	V_w	χ_w
50 m	Front	135°	8.7 kts	340°

The input into the orbit algorithm for the first orbit test point was a desired altitude of 50 m. Based on Rascal limitations such as bank angle and stall speed, along with the wind, the algorithm output waypoints at 100 m. The Rascal flew very well

at this altitude, averaging 102.8 m during the orbit. This demonstrated the algorithm properly incremented altitude as designed.

In the case of the orbit test points, the POI remained in the FOV at least 70% of the time. For the overflight test point at 50 m altitude and with a cruise speed of 40 kts, the POI was only visible for 4 seconds. Figure 4.15 shows the three plots of flight path, waypoints, and sensor footprints from the flight tests. Table 4.14 shows the aimpoint statistics for the three flight test data points.



(a) 50 m Orbit

(b) 150 m Orbit

(c) Overflight

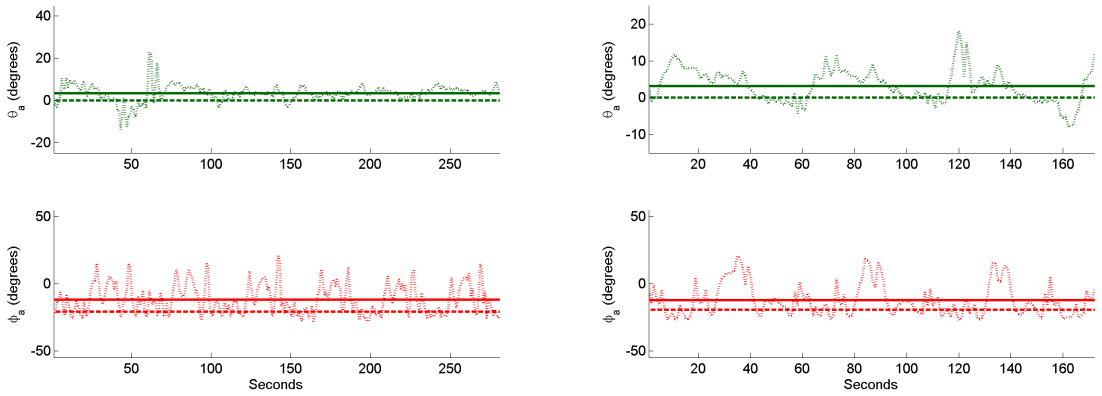
Figure 4.15: Flight Paths and Sensor Footprints from Flight Test

The plots of the aircraft attitudes for the orbits help to indicate why the percent of time the POI was in the FOV for the two orbits was less than 100%. Figure 4.16 shows the pitch and roll angle data from the Rascal orbits. The roll angle, in both orbits, cyclically repeated. This change in bank angle was the reason the POI slipped out of the

Table 4.14: Aimpoint Statistics

Alt	n or χ_L	Data Pts	% In FOV	RMS
50 m	36	284	71.83%	66.80 m
150 m	36	172	70.35%	111.78 m
50 m	135°	4	100.00%	N/A

FOV repeatedly during the orbit. Figure 4.17 shows the locations of the Rascal when the POI was not in the FOV. In both orbits, the Rascal was always in the same general area when the POI was not in the FOV; these were the times when the Rascal had the steepest bank angle to attempt to get back to the commanded set of waypoints.

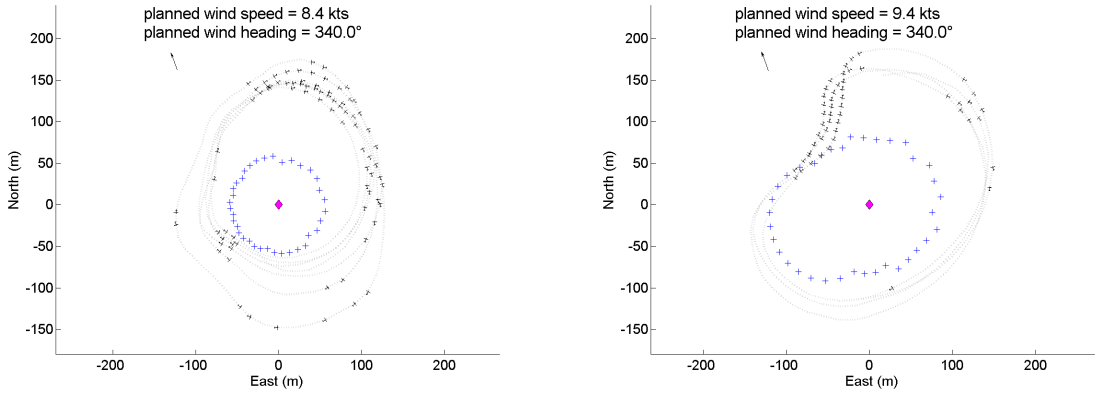


(a) 50 m Orbit

(b) 150 m Orbit

Figure 4.16: Attitudes Angles from Orbits

The flight testing provided both data about the use of the sensor aimpoint algorithms in a real world environment as well as data with which to compare the HIL model of the SIG Rascal. The flight tests demonstrated the Rascal did a good job of keeping the POI in the FOV in both the orbit and the overflight cases. The orbit case showed difficulty in keeping the POI in the FOV when the MAV had to contend with winds approximately 25% of its cruise speed because the MAV had to bank steeply to attempt to reach the commanded waypoints. The overflight case indicated a problem with flying at low altitudes and higher cruise speeds because the FOV quickly passes over the POI, allowing very little time for the operator to view the POI in the sensor display.



(a) 50 m Orbit

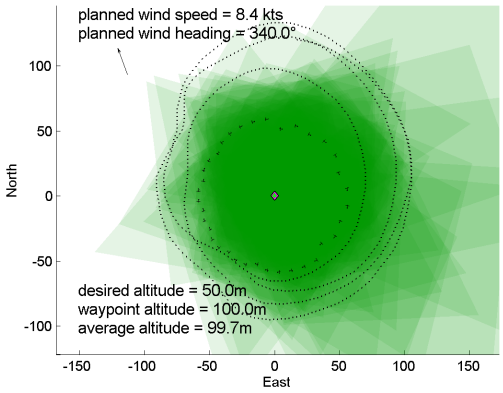
(b) 150 m Orbit

Figure 4.17: Missed Aimpoint Positions from Orbits

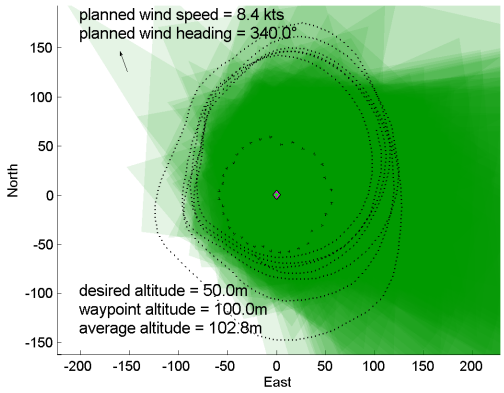
4.11 SIG Rascal Hardware-in-the-Loop Validation Testing

The purpose of the hardware-in-the-loop (HIL) testing on 9 December 2008 was to determine the accuracy of the HIL model for the SIG Rascal. Two sets of tests were accomplished. The first mimicked the flight tests of the Rascal from 2 December with all the same Kestrel settings, such as cross track of 300 m and waypoint radius of 50 m. The second accomplished the same test points but modified the cross track and waypoint radius to 100 m and 25 m, respectively. This second test was done to attempt to improve the results from the sensor aimpoint algorithms and determine proper values for the cross track and waypoint radius for future testing.

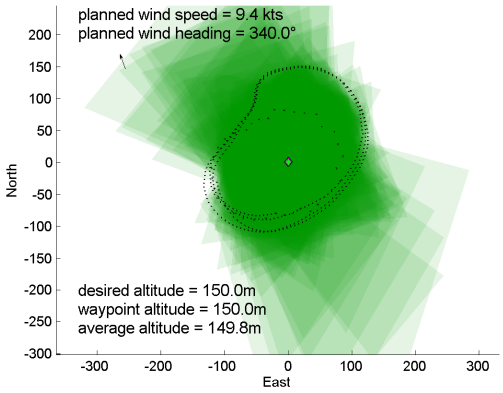
Comparing the HIL to the flight test results indicated the HIL was properly tuned for the Rascal. Figure 4.18 shows the flight path and sensor footprints from both the flight test and the HIL test. The HIL test shows the MAV kept to the intended sensor aimpoint path better than achieved in flight test, for the most part, but this was likely due to a constant wind instead of a variable wind experienced in the flight test. Table 4.15 shows the aimpoint statistical data for the three test points which indicated better POI in FOV rates for both orbit cases and the same for the overflight case. This data demonstrated the HIL adequately simulated a real flight environment and could be used for all future testing in lieu of additional flight tests.



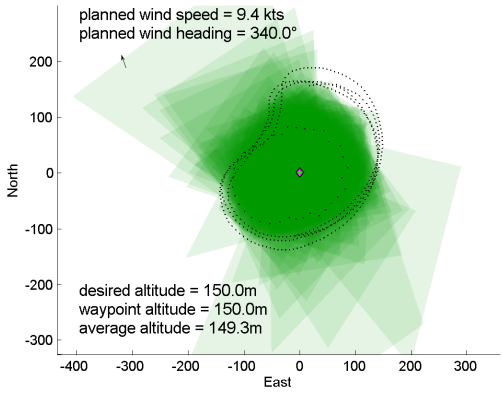
(a) 50 m Orbit - HIL Test



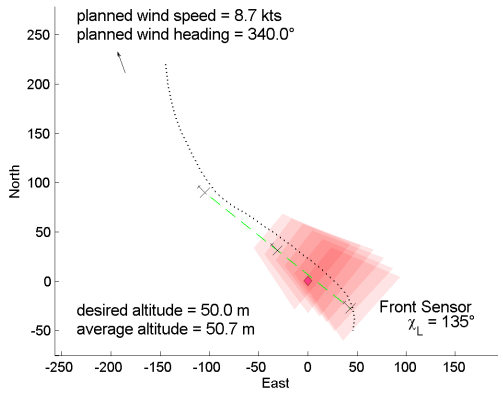
(b) 50 m Orbit - Flight Test



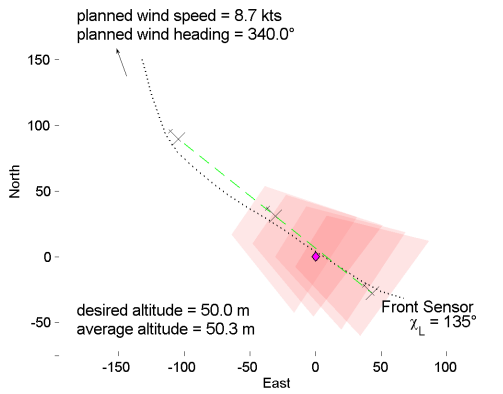
(c) 150 m Orbit - HIL Test



(d) 150 m Orbit - Flight Test



(e) 50 m Overflight - HIL Test



(f) 50 m Overflight - Flight Test

Figure 4.18: Comparison of Rascal Flight Test to HIL Test

Table 4.15: 300 m Cross Track Aimpoint Statistics

Alt	n or χ_L	Data Pts	% In FOV	RMS
50 m	36	174	80.46%	53.60 m
150 m	36	180	75.56%	98.24 m
50 m	135°	6	100.00%	N/A

Table 4.16: 100 m Cross Track Aimpoint Statistics

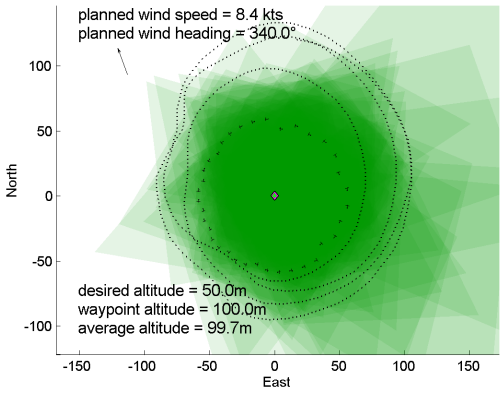
Alt	n or χ_L	Data Pts	% In FOV	RMS
50 m	36	179	78.21%	52.05 m
150 m	36	179	69.27%	109.24 m
50 m	135°	6	100.00%	N/A

The second set of HIL tests accomplished the same test points but with smaller values for cross track and waypoint radius. The results of these tests are shown in Figure 4.19 and Table 4.16, which indicate a 100 m cross track and 25 m waypoint radius kept to the commanded path better, but at the slight expense to the percent of time that the POI was in the FOV. At the time the HIL tests were run and the data was captured, only the flight paths were compared, with the assumption that if the MAV kept to the commanded flight path better, the sensor would more often capture the POI in the FOV. Therefore, the remainder of HIL tests for this research used the 100 m cross track and 25m waypoint radius instead of the 300 m cross track and 50 m waypoint radius settings.

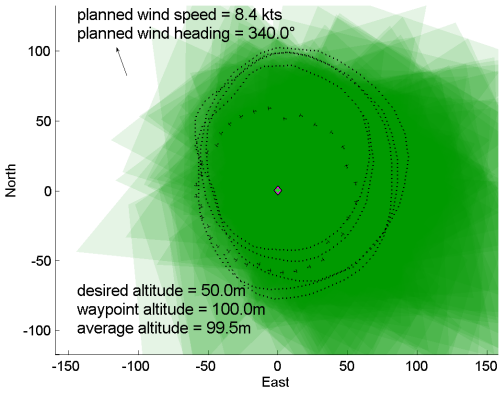
4.12 Dubins Path Testing

The creation and testing of the Dubins path generation algorithms, as described in section 3.5.7, occurred after the flight testing of the BATCAM and SIG Rascal. Since the hardware-in-the-loop (HIL) had been demonstrated to adequately simulate the Rascal, HIL evaluation of the Dubins path algorithms was deemed sufficient to demonstrate the Dubins path. The user interface described in section 3.6 would have provided the optimal testing of the Dubins path since it would have allowed the input of the current position of the MAV from which to calculate the path, but the user interface had not been created. Instead, the Dubins path testing was accomplished using a simple MATLAB interface.

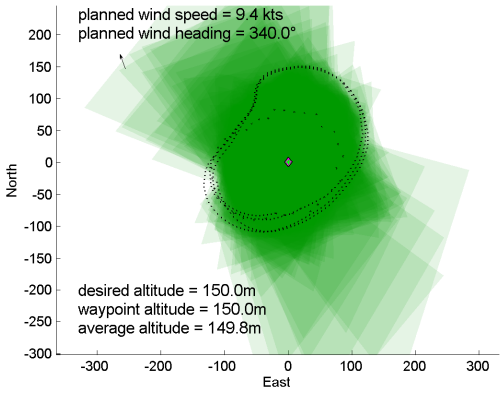
The cross track was set to 100 m and the waypoint radius set to 25 m, based on previous Rascal hardware-in-the-loop tests. The altitude was set to 100 m for both orbit



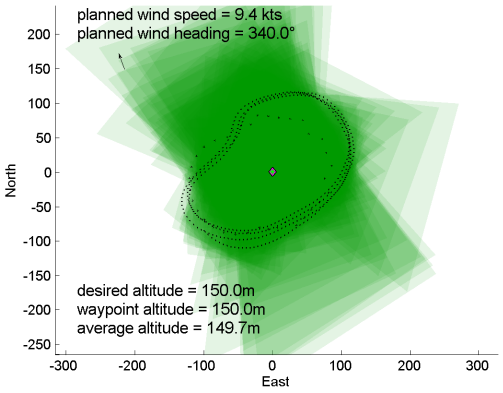
(a) 50 m Orbit - 300 m Cross Track



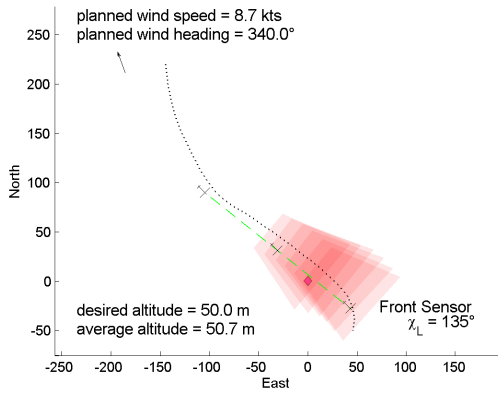
(b) 50 m Orbit - 100 m Cross Track



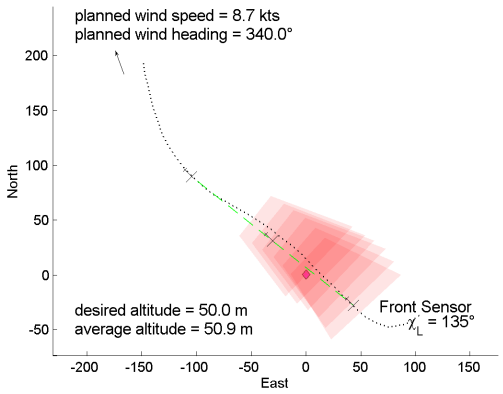
(c) 150 m Orbit - 300 m Cross Track



(d) 150 m Orbit - 100 m Cross Track



(e) 50 m Overflight - 300 m Cross Track



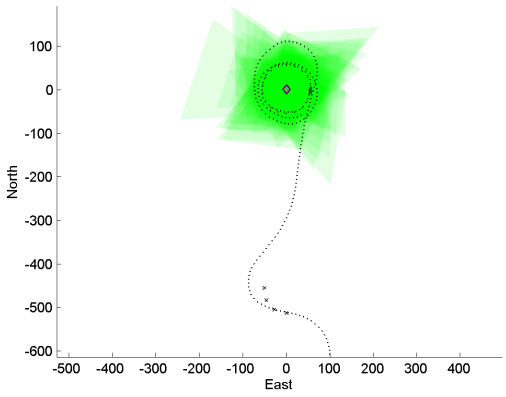
(f) 50 m Overflight - 100 m Cross Track

Figure 4.19: Comparison of Rascal Cross Track Values

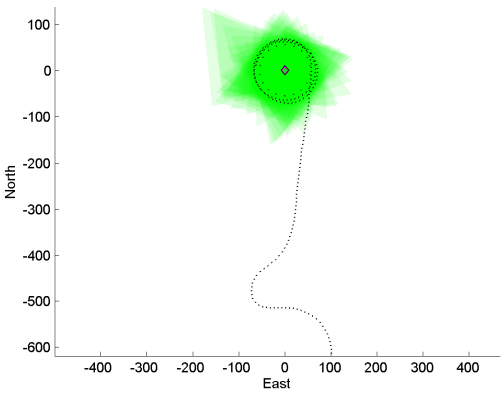
and overflight tests. The orbit simulations used 18 waypoints for the sensor aimpoint flight path. The simulated Rascal was flown around a rally point at a set altitude. A position (latitude, longitude, and altitude) at the top of the rally was used as the starting position of the MAV for the Dubins path algorithm. The approximate heading of the MAV was also known at this point to be 270° since the orbit was counter-clockwise. This collection of inputs could be used instead of capturing the values through the user interface since the simulated MAV would arrive very close to the point and heading. Wind speed and heading, sensor attitude and FOV, MAV cruise speed, stall speed, and maximum bank angle were manually input into the MATLAB script as well. If a user interface existed, these variables would be automatically passed into the algorithm from Virtual Cockpit. These values were constant for each HIL test. The wind speed and heading was changed for different tests; both were input into Aviones in north and east components and into the MATLAB script as a heading and speed. Lastly, the look angle or number of orbit waypoints were manually input into the algorithm, as appropriate. If a user interface existed, these would be variables the operator would change along with altitude and POI location. To simplify the Dubins path testing and focus on how well the Dubins path algorithm functioned, the Dubins path testing was conducted in a simulation without winds.

4.12.1 Dubins Path with Orbit Testing. In the case of the transition to orbit, eight POIs were spaced approximately 45° apart to create a ground track heading, χ_g from the MAV position to the POI. For comparison, the same test points were completed with only the sensor aimpoint algorithms, relying on Virtual Cockpit to command the MAV to fly from the current position to the initial point of the sensor aimpoint flight path. This second scenario was the same as was used during flight testing. Figure 4.20 shows representational orbits comparing the two test scenarios while Table 4.17 shows a complete comparison of the two test scenarios.

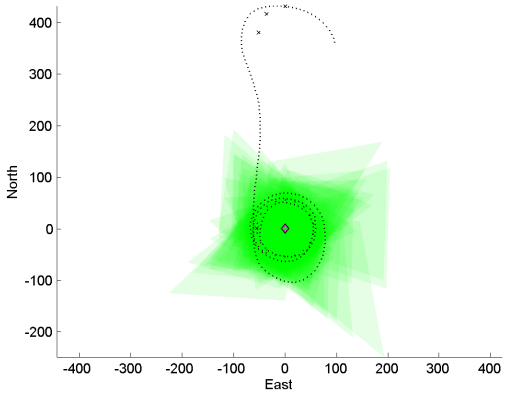
Three of the eight orbit comparison cases were chosen as representational. These three were the cases in which the POI was north, south, and east of the starting position of the MAV, as shown in Figure 4.20. The case where the POI was to the east of



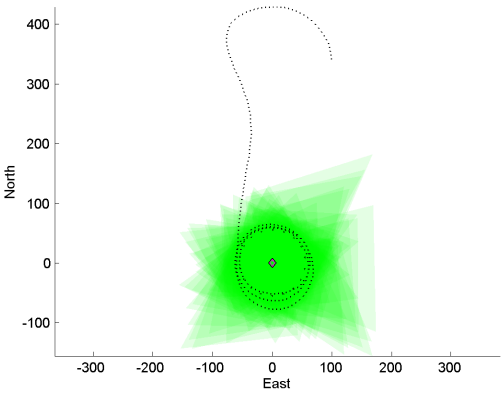
(a) North - Dubins



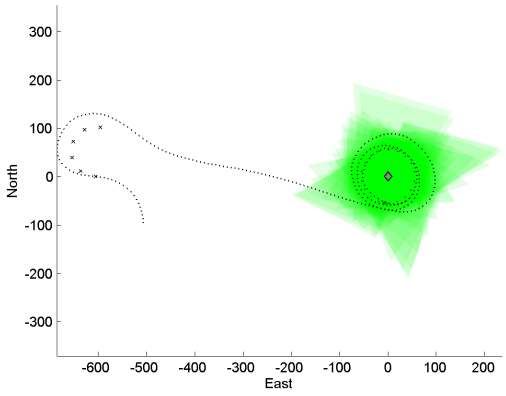
(b) North - Virtual Cockpit



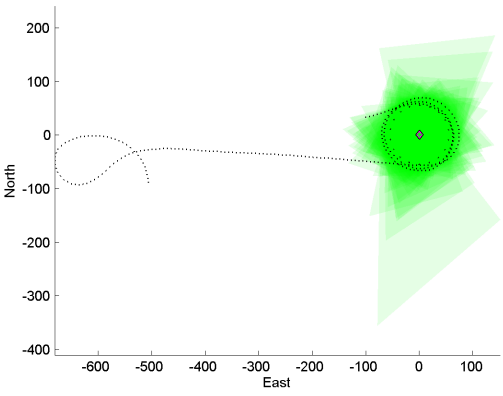
(c) South - Dubins



(d) South - Virtual Cockpit



(e) East - Dubins



(f) East - Virtual Cockpit

Figure 4.20: Compare Dubins Flight Path for Orbit Case

Table 4.17: Compare Dubins Flight Path for Orbit Case

	Dubins			**Virtual Cockpit**		
χ_g	**Data Pts**	1^{st} **Pt**	**% In FOV**	**Data Pts**	1^{st} **Pt**	**% In FOV**
320°	99	52 s	94.95%	83	69 s	98.80%
270°	116	36 s	93.10%	109	42 s	100.00%
225°	105	49 s	91.43%	94	58 s	98.94%
0°	106	45 s	93.40%	92	59 s	100.00%
180°	111	40 s	95.50%	107	44 s	100.00%
50°	118	63 s	96.61%	88	94 s	98.86%
90°	118	63 s	94.92%	100	82 s	94.00%
125°	116	66 s	93.97%	98	84 s	98.98%

the MAV was viewed as the most challenging case for the orbit algorithm, since the MAV had to turn completely around from its initial heading to get to the POI. It was only in this case that the initial turn direction of the MAV differed between the Dubins path algorithm and path commanded by Virtual Cockpit. Otherwise, as expected, the difference between using the Dubins path generator and not had little impact on the end result of the flight path. The large initial overshoot when using the Dubins path was not expected and actually made the use of the Dubins path to enter the orbit less desirable than the path automatically flown by Virtual Cockpit.

While the path commanded by Virtual Cockpit always took longer to get the POI into the FOV (the **Enters FOV** column on Table 4.17 lists elapsed seconds of data from the start until the POI first enters the FOV), using Virtual Cockpit to command the transition better kept the POI in the FOV throughout the orbit. The three plots of the MAV track in the Dubins case shown in Figure 4.20 show the MAV always overshot the turn which led into the orbit. The MAV banked hard to get back on the orbit course. During the hard bank, the POI slipped outside the FOV. Results therefore indicated Virtual Cockpit did a better job than the Dubins algorithm for the orbit case.

4.12.2 Dubins Path with Overflight Testing. For the overflight case, a single POI was located due south of the rally point. Look angles, χ_L, were commanded to go in the same eight headings, spaced approximately 45° apart. As in the orbit case, this evaluated the transition to the overflight path in nearly every heading compared to the initial heading of the MAV. These eight look angles were repeated for both the front

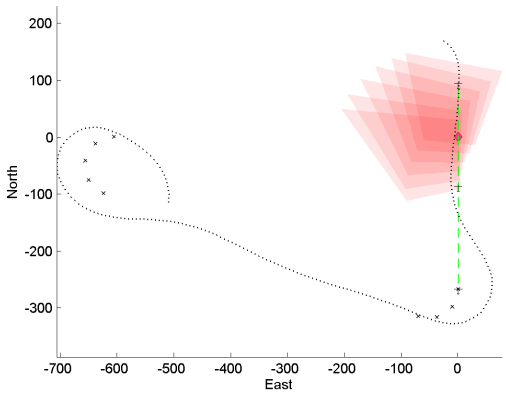
and the side sensor, for a total of 16 test points for the overflight path. As expected, the Dubins Path provided a benefit of getting the MAV aligned to the commanded path in the overflight cases. Tables 4.18 and 4.19 indicate that this benefit was not that the Dubins Path always got the POI in the FOV earlier or kept the POI in the FOV for a longer duration as the transition path commanded by Virtual Cockpit. The real benefit from the Dubins Path is indicated by the overflight test point with the front sensor and look angle of 180°, where the path commanded by Virtual Cockpit made the MAV overshoot the initial point of the overflight path, thus causing the MAV to have a hard time keeping the POI in the FOV. Figure 21(d) best illustrates this situation. In fact, all of the subfigures shown in Figure 4.21 show that the Dubins Path better aligned the MAV near the initial point of the overflight path. Also indicated in Figure 4.21 was that the MAV could not keep to the exact path commanded by the Dubins Path algorithm, likely because the Kestrel had no knowledge of the future waypoints in a turn.

Table 4.18: Compare Dubins Flight Path for Overflight Case - Front Sensor

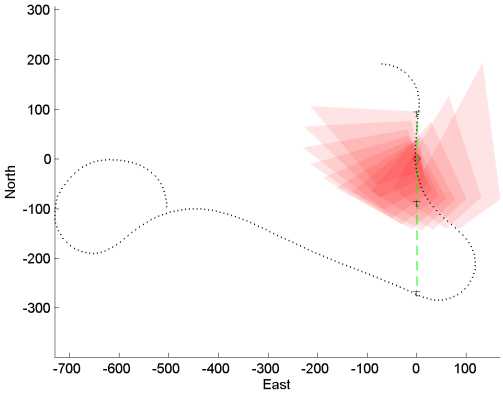
	Dubins			**Virtual Cockpit**		
χ_L	**Data Pts**	1^{st} **Pt**	**% In FOV**	**Data Pts**	1^{st} **Pt**	**% In FOV**
315°	9	92 s	100.00%	4	99 s	100.00%
270°	8	103 s	100.00%	13	49 s	100.00%
225°	8	88 s	100.00%	2	98 s	100.00%
180°	7	79 s	100.00%	11	81 s	63.64%
135°	11	58 s	100.00%	9	61 s	100.00%
90°	8	57 s	100.00%	13	53 s	100.00%
45°	9	56 s	100.00%	10	53 s	100.00%
360°	6	81 s	100.00%	10	85 s	100.00%

Table 4.19: Compare Dubins Flight Path for Overflight Case - Side Sensor

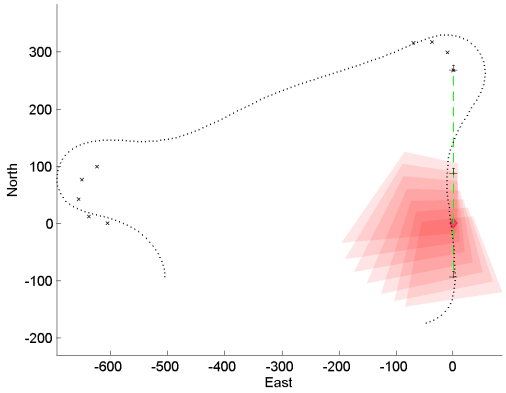
	Dubins			**Virtual Cockpit**		
χ_L	**Data Pts**	1^{st} **Pt**	**% In FOV**	**Data Pts**	1^{st} **Pt**	**% In FOV**
315°	9	74 s	100.00%	7	78 s	100.00%
270°	9	90 s	100.00%	6	97 s	100.00%
225°	10	106 s	100.00%	8	110 s	100.00%
180°	9	113 s	100.00%	7	114 s	100.00%
135°	9	97 s	100.00%	11	102 s	100.00%
90°	8	78 s	100.00%	16	76 s	100.00%
45°	8	62 s	100.00%	12	62 s	100.00%
360°	7	60 s	100.00%	7	63 s	100.00%



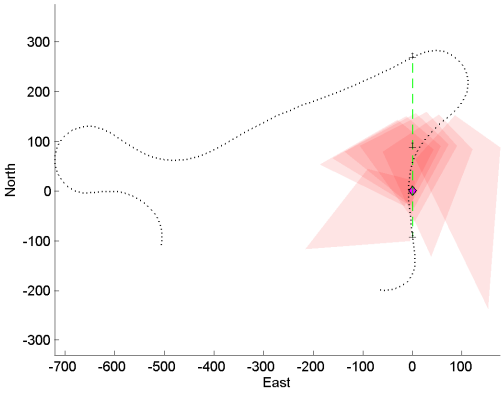
(a) $\chi_L = 000°$ - Dubins



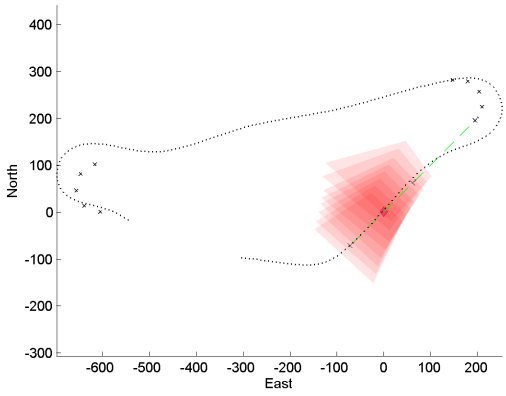
(b) $\chi_L = 000°$ - Virtual Cockpit



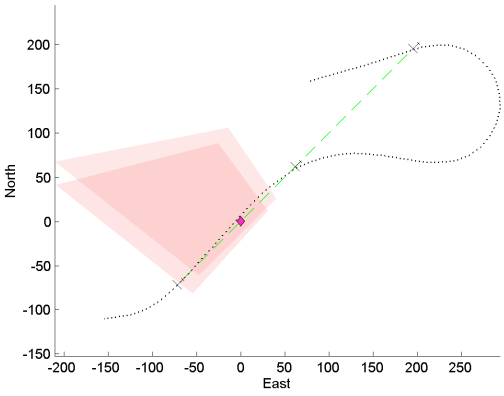
(c) $\chi_L = 180°$ - Dubins



(d) $\chi_L = 180°$ - Virtual Cockpit



(e) $\chi_L = 225°$ - Dubins



(f) $\chi_L = 225°$ - Virtual Cockpit

Figure 4.21: Compare Dubins Flight Path for Overflight Case

Based on the results of the tests comparing the Dubins algorithm to the commands natively generated by Virtual Cockpit, the Dubins algorithm did not benefit sufficiently to justify its use in the orbit case, but it did for the overflight case. For this reason, the remaining orbit tests for this research relied upon Virtual Cockpit instead of the Dubins algorithm. Alternatively, the overflight tests were conducted using the Dubins algorithm to calculate a transition path.

4.13 Final HIL Testing

The sensor aimpoint algorithms were extensively tested using the hardware-in-the-loop (HIL) equipment and the model for the SIG Rascal. The purpose of this testing was to thoroughly evaluate the benefit and accuracy of the sensor aimpoint algorithms.

The orbit algorithm was tested through 15 different configurations, varying altitude, from 50 m to 200 m in 50 m increments, and wind speed, from 0 kts to 25 kts in 5 kts increments. In all cases, the number of waypoints was set to 18 and the wind was out of the east with a heading, χ_w, of 270°. Since wind speed can force the orbit algorithm to iterate altitude, certain test points were not necessary to evaluate. For example, for a desired altitude of 50 m and a wind speed of 10 kts, the algorithm generates waypoints at 110 m altitude. The same is true for a desired altitude of 100 m and a 10 kts wind; therefore, the 50 m desired altitude test case was not evaluated. In between each change in wind speed, the MAV was instructed to fly at least 2 orbits at the rally point to allow the Kestrel to recalibrate for the new wind speed. The POI was established at a latitude of 39.34558°N, longitude of −86.02290°E, and MSL altitude of 216 m. The rally was established at a latitude of 39.34467°N and longitude of −86.02992°E, with a radius of 100 m and an altitude input as required for each test point. Table 4.20 shows the aimpoint statistics for each test point accomplished for the orbit testing.

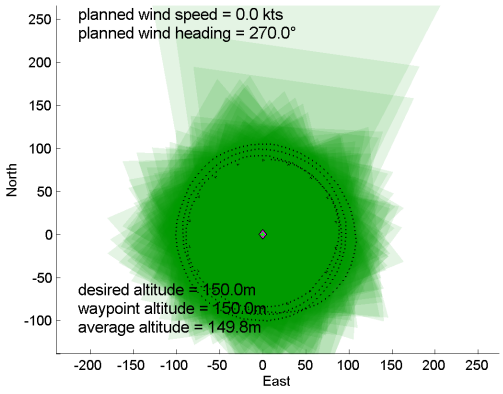
The test results shown in Table 4.20 demonstrate how wind speed and altitude affect the ability for the MAV to keep the POI in the FOV. The trend is that, for a given altitude, as wind speed increased, the ability to keep the POI in the FOV decreased. When the MAV had a tail wind at the top of the orbit, it continually overshot its turn and had to correct to get back on the commanded path. The correction required a bank

Table 4.20: Orbit Aimpoint Statistics

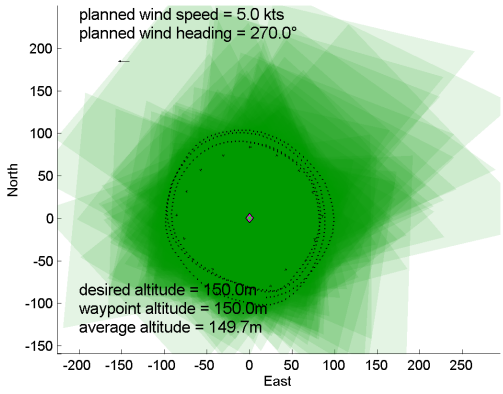
Des Alt	**Cmd Alt**	V_w	**Data Pts**	**% In FOV**	**RMS**
50 m	50 m	0 kts	106	98.11%	14.46 m
50 m	80 m	5 kts	121	86.78%	31.58 m
100 m	100 m	0 kts	92	100.00%	18.02 m
100 m	100 m	5 kts	124	82.26%	47.68 m
100 m	110 m	10 kts	138	67.39%	86.23 m
150 m	150 m	0 kts	146	100.00%	18.03 m
150 m	150 m	5 kts	139	97.12%	34.86 m
150 m	150 m	10 kts	171	71.35%	113.32 m
150 m	150 m	15 kts	150	55.33%	128.28 m
200 m	200 m	0 kts	178	99.44%	21.70 m
200 m	200 m	5 kts	147	99.32%	49.12 m
200 m	200 m	10 kts	171	95.32%	68.66 m
200 m	200 m	15 kts	182	88.46%	80.53 m
200 m	200 m	20 kts	212	80.19%	94.94 m
200 m	250 m	25 kts	336	71.43%	181.14 m

larger than the waypoint generation algorithm calculated and the rate of the POI being in the FOV decreased. Figure 4.22 illustrates this for the 150 m altitude orbit case. The second trend indicated by Table 4.20 is that as altitude increased, the POI remained in the FOV at a higher rate in higher winds. Because the sensor footprint increased with altitude, the likelihood that the POI would be in the FOV also increased. The final conclusion from the HIL orbit tests was that the algorithms kept the POI in the FOV at least two-thirds of the time as long as the wind speed was less than one-quarter of the cruise speed.

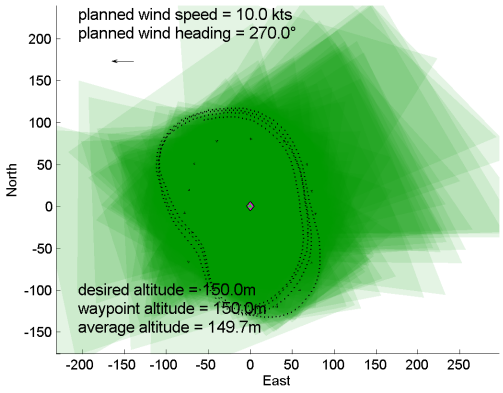
The overflight algorithm was tested at nine different test points with the front sensor only. The algorithm generates a flight plan based on the selected sensor, but that does not affect how the MAV flies the commanded path. Capturing telemetry for the MAV for both the front and the side as the MAV flies the same path through the air would be equivalent to post-processing the data; therefore, testing was simplified by only considering the front sensor. All test points were flown at 100 m altitude, at the cruise speed for the Rascal of 40 kts, and with a wind out of the east with a heading of 270°. A total of nine test points considered three wind speeds, 0 kts, 10 kts, and 20 kts, and three look angles, 045°, 225°, and 360°. The look angles of 090° and 270° were not considered



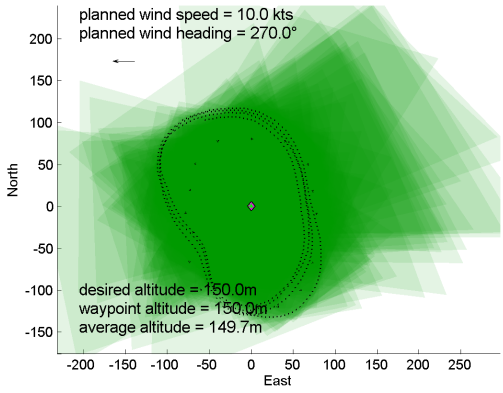
(a) $V_w = 0kts$



(b) $V_w = 5kts$



(c) $V_w = 10kts$



(d) $V_w = 15kts$

Figure 4.22: Comparison of Orbit Paths at Different Winds at 150 m Altitude

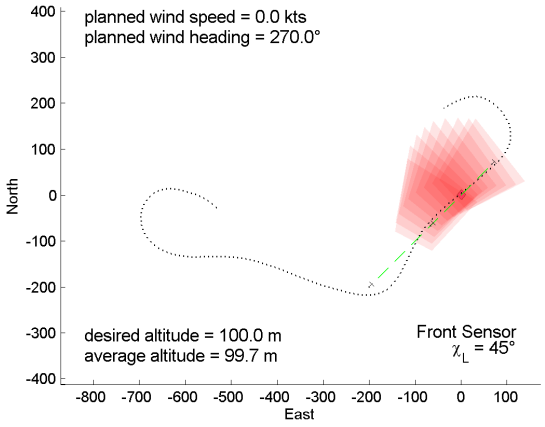
as they would provide a pure headwind and tailwind respectively. This would slow or speed up the MAV, but not change the output of the overflight path. The look angles of 135°, 180°, and 315° would provide similar crab angles to the path as those from the three look angles used. As discussed above, all the transition paths for the overflight tests were accomplished using the Dubins Path algorithm.

As with the orbit testing, in between each change in wind speed, the MAV was instructed to fly at least 2 orbits at the rally point to allow the Kestrel to recalibrate for the new wind speed. The POI was established at a latitude of 39.34558°N, longitude of −86.02290°E, and MSL altitude of 216 m. The rally was established at a latitude of 39.34467°N and longitude of −86.02992°E, with a radius of 100 m, and an altitude of 100 m, the same as the test points. Table 4.21 lists the aimpoint statistics for each test point accomplished during the overflight testing, while Figures 4.23-4.25 shows the commanded flight path, actual flight path, and sensor footprints for each test point.

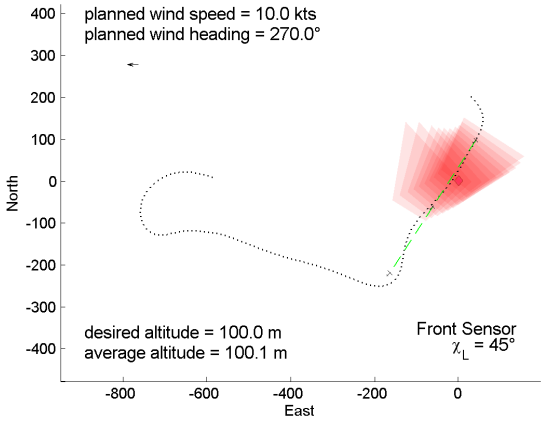
Table 4.21: Overflight Aimpoint Statistics

χ_L	V_w	**Data Pts**	**% In FOV**
45°	0 kts	9	100.00%
225°	0 kts	8	100.00%
360°	0 kts	6	100.00%
45°	10 kts	10	100.00%
225°	10 kts	8	100.00%
360°	10 kts	8	100.00%
45°	20 kts	12	100.00%
225°	20 kts	6	100.00%
360°	20 kts	6	100.00%

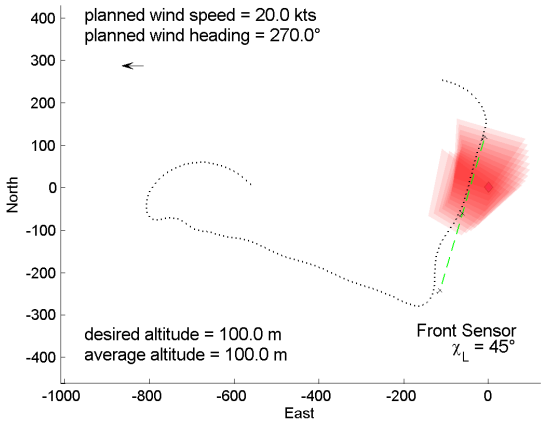
The data and plots in Table 4.21 and Figures 4.23-4.25 indicate that the overflight algorithm can capture the POI in the FOV in each of the wind situations. Data was not collected at higher winds, however, because the MAV could not reach the commanded flight path. Depending on the relation between the wind and the flight path angle, the wind can actually help the MAV turn and keep the POI in the FOV for a longer duration. Additionally, the relation between the approach heading and the flight path heading affects how soon the POI enters the FOV. The later the POI enters the FOV, the shorter duration an operator can view the POI.



(a) $\chi_L = 045°, V_w = 0kts$

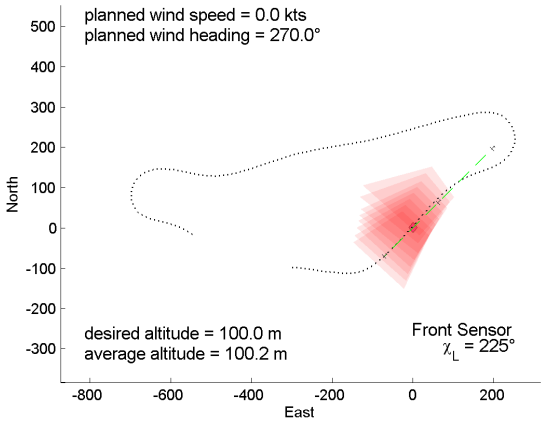


(b) $\chi_L = 045°, V_w = 10kts$

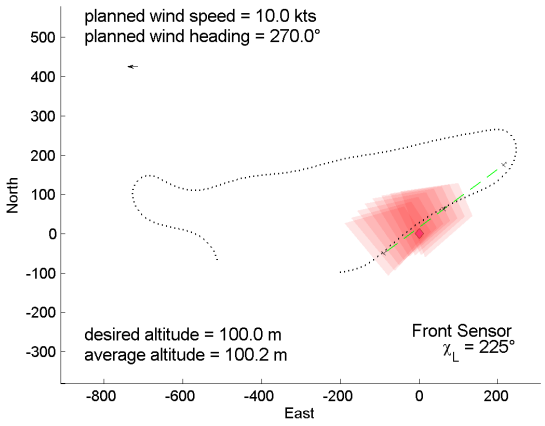


(c) $\chi_L = 045°, V_w = 20kts$

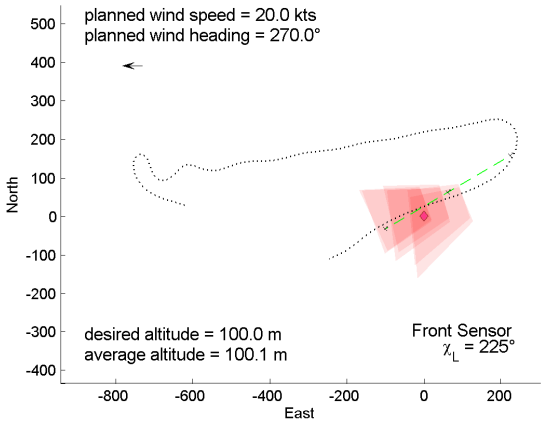
Figure 4.23: Comparison of Overflight Paths at Different Winds at 100 m Altitude



(a) $\chi_L = 225°, V_w = 0kts$

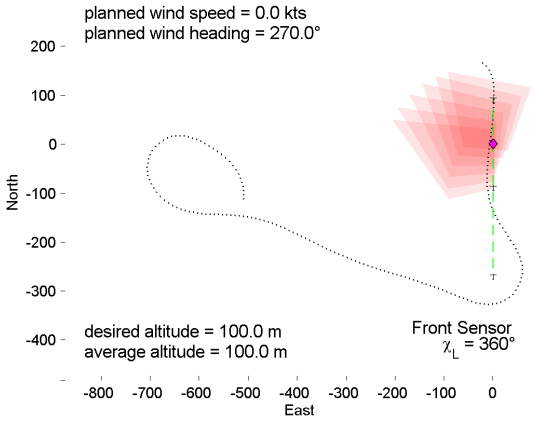


(b) $\chi_L = 225°, V_w = 10kts$

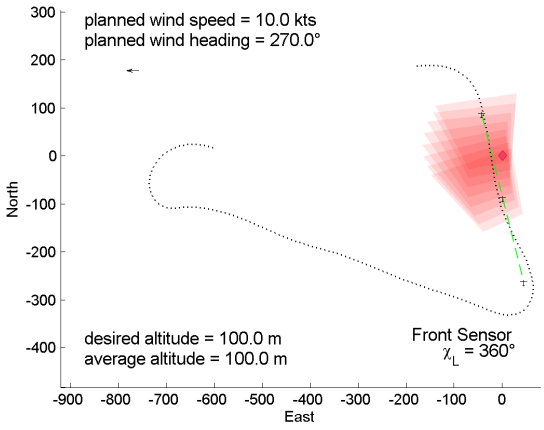


(c) $\chi_L = 225°, V_w = 20kts$

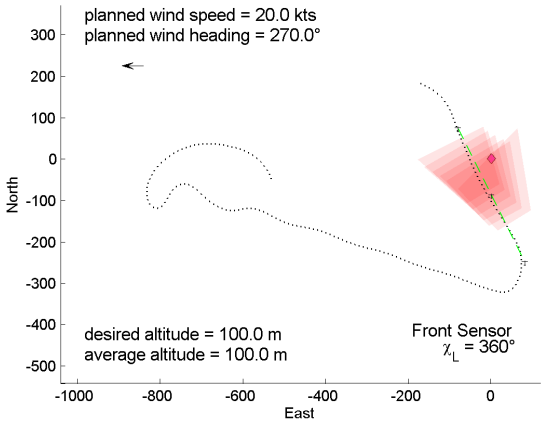
Figure 4.24: Comparison of Overflight Paths at Different Winds at 100 m Altitude



(a) $\chi_L = 360°, V_w = 0kts$



(b) $\chi_L = 360°, V_w = 10kts$



(c) $\chi_L = 360°, V_w = 20kts$

Figure 4.25: Comparison of Overflight Paths at Different Winds at 100 m Altitude

The lack of time of the POI in the FOV may reduce the utility of the algorithm to operators. As mentioned above in the orbit section, the POI can be kept in the FOV longer by flying the MAV at a higher altitude. The trade off in this case is a reduction in pixel density on the POI. This trade space was not a consideration in the development of the algorithm, but it is explored below in section 4.14. Another potential factor to increase the amount of time the POI remained in the FOV for the overflight case would be to reduce the cruise speed for the MAV, an option not explored by this research.

4.14 Sensor Resolution Study

Several of the tests found improvements in the results from the algorithms at higher altitudes. As noted above, as altitude increased, pixel density on the POI decreased. The algorithms never accounted for this trade space, but if the MAV climbs to a higher altitude the sensor output may not be acceptable to the operator. As discussed in section 4.4.4, the sensors were assumed to have 640x480 lines of resolution and a 48° by 40° field of view.

The pixel density was calculated using equations adapted from Abbott *et al.* (2007). Using Equation 3.3 from the sensor aimpoint algorithm, the angle, θ_{aim}, between the ground and the slant range line to the MAV was found. The slant range is found by dividing the altitude of the MAV by the sine of this angle, as in Equation 4.1. The angular pixel resolution is found using Equation 4.2 and is the approximate angle in radius encompassed by each pixel at the sensor aimpoint. Since the lines of resolution and FOV angle differ for vertical (y) and horizontal (x), they are calculated independently. Angular pixel resolution is lower at the leading edge, or furthest away from the MAV, of the sensor footprint, and higher at the trailing edge, or closest to the MAV. The pixel density is given by Equation 4.3 and represents the number of pixels per square meter.

$$R = \frac{\text{alt}}{\sin\theta_s} \tag{4.1}$$

$$\theta_{pix} = \frac{\text{FOV}}{\text{lines}} \tag{4.2}$$

$$\text{pixel density} = \left(\frac{1}{R\theta_{pix,x}}\right)\left(\frac{1}{R\theta_{pix,y}}\right) \tag{4.3}$$

For a given sensor and assuming straight and level flight, the pixel resolution is a function of altitude and sensor depression angle (θ_s). As the MAV changes its attitude, θ_{aim} will also change. As an example, if the mean bank angle in an orbit was 10°, the side sensor would have an equivalent −49° depression. Figure 4.26 plots the change in pixel density as altitude increases for the two sensor depressions, −39° and −49° used in the testing of this research . To compare the affect of sensor depression and altitude, straight and level flight is assumed. Lines that indicate pixel resolution for both detection ($3pix/m^2$) and identification ($15pix/m^2$), based on the conservative cases explored by Abbott *et al.*, are plotted as well.

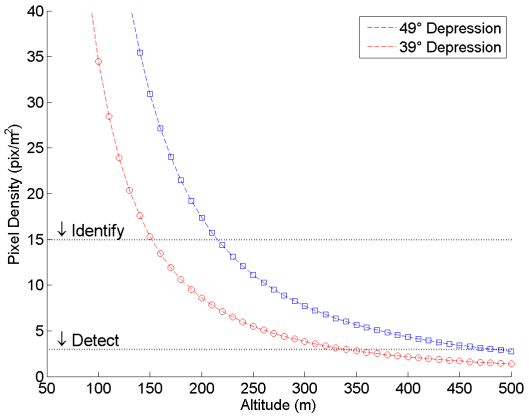


Figure 4.26: Pixel Density vs. Altitude

Since nearly all the tests flown or simulated for this research occurred below 200 m, an operator should have been able to identify the POI in every case. Additionally, given the results of the sensor resolution study, the overflight test points accomplished in the hardware-in-the-loop testing described in section 4.13 could have been flown at 200 m and still allowed the operator to view the POI. The time the POI would remain in the FOV would vary with winds, but in a no wind situation, with a look angle of 225°, and using the front sensor, 27 seconds of data was collected. At this altitude, the pixel density would be 17.4 pix/m^2 for a −49° sensor depression and 8.6 pix/m^2 for a −39°

sensor depression. At a minimum, this would allow an operator to identify the POI and opt to descend for a closer look, if necessary.

4.15 Results

The testing of the waypoint generation algorithms indicated that the algorithms create flight plans that keep the POI in the FOV at least two-thirds of the time as long as the wind speed was less than one-quarter of the cruise speed. The flight dynamics of the MAV greatly affect the accuracy of the algorithms, as the small and very maneuverable BATCAM had less optimal results than the larger and more stable SIG Rascal. The hardware-in-the-loop (HIL) testing both confirmed the results seen in test flight as well as evaluated the algorithms at many more conditions than available in test flight. One limitation of the HIL testing was the inability to simulate variable winds; therefore, outside of flight test, the algorithms were only tested in a constant wind environment. The next chapter reviews the research effort in full and makes recommendations for future research.

V. Conclusions and Recommendations

5.1 Testing Conclusions

Through hardware-in-the-loop (HIL) testing and flight testing, the sensor aimpoint waypoint generation algorithms were evaluated and determined to function as designed. As wind speeds can often approach the cruise speed for a MAV, wind can quickly limit the utility of flying a MAV, with or without the sensor aimpoint algorithms. Since the sensor aimpoint algorithms only provide open-loop controls, they do not dampen the banking of the MAV when it tries to correct its attitude after it has been perturbed by the wind. Additionally, limitations of the Kestrel autopilot, namely that the Kestrel does not anticipate a projected path based on the locations of future waypoint, keeps the MAV from making the desired turns in time, as seen with the Dubins Path testing. The results indicate the algorithms keep the POI in the FOV at least 66% of the time as long as the wind speed was less than 25% of the cruise speed; at higher winds, the MAV has a difficult time tracking the flight path generated by the sensor aimpoint algorithm.

5.2 Recommendations

Based on the assumptions listed in section 1.4 as well as other assumptions or situations encountered during the research, several areas were identified for future research.

5.2.1 Multiple MAVs or Cooperative Control. Incorporating this research into a system that commands and controls multiple MAVs simultaneously, including managing video returns from multiple MAVs, serves as the basis for various future research. The human systems integration aspects of controlling multiple MAVs from a single base station will likely prove very challenging from a task-management perspective of a single operator. Controlling multiple MAVs with the ability to parse them out to other operators may prove technically challenging.

5.2.2 Targets. Moving targets and camouflaged targets should be pursued, potentially with the assistance of target-cueing software. Adapting the overflight algorithm to intercept a moving target yet still provide sensor data from a specified look angle would increase its utility.

5.2.3 Object Avoidance. Incorporating object collision avoidance along with the sensor aimpoint technology would make the MAV more adaptable for operational uses.

5.2.4 Variable Winds. The HIL system used for this research could not accommodate variable winds. Future testing should modify the Aviones flight environment simulator to allow for variable winds, or an alternate flight environment simulator should be used.

5.2.5 Airframes. While the BATCAM was used for some initial testing, the lack of a valid HIL model limited its utility for this research. A HIL model should be created and validated. Such a model would not only allow testing of this research on an additional airframe in HIL, but would be useful for other MAV research. While access to the test range limits the ability to run tests both in the HIL and in flight test for validation purposes, several research efforts, along with this one, have captured flight telemetry data for the BATCAM which could be used as part of the validation process. In addition to continued testing of this research on the BATCAM, testing the algorithms created by this research on additional MAV airframes would further evaluate the robustness of the algorithms. Since the SIG Rascal was the only airframe available for both flight testing and HIL testing, the research has only been thoroughly studied for one vehicle.

5.2.6 Closed-Loop Control. Two efforts for closed-loop control would benefit the sensor aimpoint algorithms. The first would be to provide feedback to dampen the banking of the MAV as it attempts to return to its desired attitude after being disturbed by the wind or after deviating from the course. This would reduce the oscillating back and forth across the flight path that was seen during the BATCAM flight tests. It would further help to stabilize the POI in the center of the FOV as the MAV adjusts its attitude. The second closed-loop control would allow the MAV to anticipate upcoming turns based on the locations of waypoints it will encounter in the near-future. This would help the MAV keep to the commanded turns it encounters in the Dubins Path and orbit path.

Anticipation of future turns should reduce the overshooting of the first sensor aimpoint path waypoint that was seen in many of the tests.

5.2.7 User Interface. A user interface, particularly a graphical user interface, that interfaces with both the flight management system and the algorithms, would greatly enhance utility and reduce operator workload and operator training. The concepts for how to develop such a user interface was explored by this research, although the user interface itself was not created. A specific feature might include a point-and-click selection of a POI that automatically deduces the location of the POI and feeds that into the algorithms.

5.2.8 Sensor Footprint. The sensor footprint in this research was found as a function of the sensor field of view. While this calculates the image returned by the sensor to the sensor display, it does not account for the usability of the sensor footprint. A sensor footprint map, based on sensor pixel density, discussed in sections 2.3 and 4.14, would provide a more useful concept of the sensor footprint. A pixel density mapping would calculate a footprint only where the pixel density exceeded a base threshold, such as a detection threshold of $3pix/m^2$. Such an application would be useful in a situation where a formation of MAVs was determined by the location of their sensor footprints, such as determining MAV formation position to minimize the overlap of the sensor footprints. A pixel density mapping footprint would ignore such situations as those in which the state of the aircraft places a corner of the geometrically-determined sensor footprint above the horizon. A pixel density mapping footprint would most likely be smaller than the geometrically-determined footprint, but it would only consider areas useful to the operator.

5.3 Conclusion

The waypoint generation algorithms generate flight plans for both an orbit and an overflight case. These flight plans keep the POI in the FOV during the orbit and overflight. As winds increase beyond one-quarter of the cruise speed of the MAV, the percent of time that the POI stays in the FOV decreases; however, the test results

from this research indicate that the POI will stay in the FOV at least two-thirds of the time. The overflight algorithm, particularly when used with the Dubins Path transition algorithm, lines up the MAV to keep the POI in the FOV as the MAV flies over or along the side of the POI. With the development of a graphical user interface, the usability of the waypoint generation algorithms can be improved and made operational.

Appendix A. Glossary

This appendix contains a list of keywords and definitions for this research.

Aimpoint – a center point of the sensor FOV in both height and width. This is the point of the sensor that should remain aimed at the POI during an orbit.

Base Station – a computer platform, likely portable, and associated communications equipment designed to provide input to and receive output from one or more MAVs in flight. At a minimum, the base station must run Virtual Cockpit and a sensor interface software to interact with MAVs.

BATCAM – a small MAV that weighs less than a pound, is 24in long, and has a 21in wingspan.

Field of Regard – the angular range through which the sensor may move. In this instance, the field of regard is fixed.

Field of View – the width and height of the sensor capture area. This is typically measured in angular units, since length units will change based on the height of the MAV as well as the angle of declination of the camera from the MAV's longitudinal or lateral axes.

Hardware-in-the-Loop – a simulation system that uses the actual hardware to be used in a flight test or operational environment. It allows low-risk representational testing of the system prior to flight testing.

Micro Air Vehicle – a small, radio controlled-scale aircraft that serves as a sensor platform.

Operator – the human user of the MAV system. The operator observes sensor images on a base station. Using the sensor data and other software (Virtual Cockpit), the operator provides input commands to the MAV.

Point of Interest – an object which the operator wants to review more thoroughly.

Return to Base – the MAV flies back to the recovery point at the end of a mission, to refuel, or for maintenance.

Sensor – a device designed to return electromagnetic data in a field of view for the MAV. For this testing effort, the sensor used was a visible spectrum, color video camera. Other options could include sensors that operate in the infrared range or laser radar sensors, among other types. For this testing effort, the MAV had two sensors: a front view and a side view. These sensors were independent and operated at a fixed field of regard.

Use Case – a text-based description of how a system operates that helps translate system required capabilities into programmable software functions.

Virtual Cockpit – a commercial, off-the-shelf software package used to communicate with a MAV that contains an integrated Kestrel autopilot. This software allows flight or route planning, as well as the ability to launch and recover a MAV. Additionally, this software provides health and communication status of MAVs in flight.

Waypoint – a latitude, longitude, altiude above ground level, and airspeed for the MAV to achieve. A series of waypoints constitutes a flight plan.

USE CASE		*Orbit POI*
Goal in Context		Operator directs the MAV to orbit a POI identified through the sensor display.
Scope and Level		Primary Task
Preconditions		The MAV is on a pre-determined flight path or route. The MAV is in communication with the operator's base station, which displays MAV status and sensor information.
Success End Condition		The MAV resumes the planned route after orbiting the POI, keeping the sensor aimpoint centered on aimed at the POI at all times while in the orbit.
Failed End Condition		The MAV does not resume the planned route. The MAV does not orbit the POI. The sensor aimpoint does not remain centered on the POI during the orbit.
Primary Actors		Operator. MAV.
Secondary Actors		Base Station.
Trigger		The operator sees a POI about which he wants to orbit.
Description	**Step**	**Action**
	1	While viewing sensor data from the side sensor, the Operator designates the POI as a point to orbit.
	2	The MAV exits the planned route. Include: *Exit Planned Route*.
	3	The MAV flies to an intercept to the POI orbit in the most expeditious manner possible.
	4	The MAV intercepts the POI orbit and orbits the POI, keeping the sensor aimpoint centered on the POI.
	5	The Operator commands the MAV to resume the planned route. Include: *Resume Planned Route*.

Extensions	**Step**	**Branching Action**
	4a	The MAV reaches bingo fuel during the orbit and must return to base. Include: *Return to Base*.
	4b	The MAV crashes and cannot continue mission. Terminate use case.
	4c	Objects or terrain mask the point of interest from certain aspect angles. The MAV must adapt the orbit flight plan due to object collision avoidance.
	5a	The Operator wants the MAV to overfly the POI. Include: *Overfly POI*.
	5b	The Operator takes over manual control of the MAV. Include: *Control MAV Manually*.
Related Information		*Overfly POI* *Control MAV Manually* (Parent Use Case) *Exit Planned Route* (Parent Use Case) *Resume Planned Route* (Parent Use Case) *Return to Base* (Parent Use Case)
Performance		Various times based on distance to POI and desired orbit duration.
Channels to Actors		Virtual Cockpit and Sensor Interface software.
Open Issues		Determine CEP to allow the aimpoint to be considered "centered" on the POI.
Due Date		30 Oct 08
Superordinates		**Fly Planned Route**.
Subordinates		None.

USE CASE		*Overfly POI*
Goal in Context		Operator directs the MAV to overfly a POI identified through the sensor display.
Scope and Level		Primary Task
Preconditions		The MAV is on a pre-determined flight path or route. The MAV is in communication with the operator's base station, which displays MAV status and sensor information.
Success End Condition		The MAV resumes the planned route after overflying the POI, the sensor aimpoint track remained aligned on the POI.
Failed End Condition		The MAV does not resume the planned route. The MAV sensor aimpoint is not aligned with the POI during the overflight.
Primary Actors		Operator. MAV.
Secondary Actors		Base Station.
Trigger		The operator sees a POI about which he wants to overfly.
Description	**Step**	**Action**
	1	While viewing sensor data from the front sensor, the Operator designates the POI as a point to overfly from a specified sensor aimpoint view angle.
	2	The MAV exits the planned route. Include: *Exit Planned Route*.
	3	The MAV establishes overflight path to keep sensor aimpoint track aligned on POI.
	4	The MAV overflies the POI.
	5	The MAV resumes the planned route. Include: *Resume Planned Route*.

Extensions	**Step**	**Branching Action**
	4a	The MAV reaches bingo fuel during the overflight and must return to base. Include: *Return to Base*.
	4b	The MAV crashes and cannot continue mission. Terminate use case.
	4c	Objects or terrain mask the point of interest from certain aspect angles. The MAV must adapt the overflight path due to object collision avoidance.
	5a	The Operator wants the MAV to overfly the POI again. Repeat use case.
	5b	The Operator takes over manual control of the MAV. Include: *Control MAV Manually*.
	5c	The Operator wants the MAV to orbit the POI. Include: *Overfly POI*.
Related Information		*Orbit POI* *Control MAV Manually* (Parent Use Case) *Exit Planned Route* (Parent Use Case) *Resume Planned Route* (Parent Use Case) *Return to Base* (Parent Use Case)
Performance		Various times based on distance to POI and overflight altitude.
Channels to Actors		Virtual Cockpit and Sensor Interface software.
Open Issues		Determine CEP to allow the sensor track to be considered "centered" on the POI.
Due Date		30 Oct 08
Superordinates		**Fly Planned Route**.
Subordinates		None.

C.1 Orbit Algorithm Code

Listing C.1: MATLAB/orbit.m

```
function waypoints = orbit(POI,acft_pos,sens_att,wind,hdg_g,V_a,V_stall,...
    phi_max,n)
%waypoints = orbit(POI,acft_pos,sens_att,wind,hdg_g,V_a,V_stall,phi_max,n)
% Generates offset waypoints for the aircraft to fly such that the sensor
%   will aim at the target while the aircraft orbits the point of interest
%   in the presence of wind.
%
% INPUTS:
%   POI         : a 1x3 vector of the point of interest's geodetic location
%                 (lat, lon, altitude). Lat and Lon are in decimal
%                 degrees. Altitude is in meters MSL.
%   acft_pos    : a 1x3 vector of the aircraft's current geodetic location
%                 (lat, lon, alt), assuming AGL altitude.
%   sens_att    : a 1x3 vector of the sensor's attitude angles relative to
%                 the aircraft (yaw, pitch, roll)
%   wind        : a 1x2 vector for wind
%                 wind(1) = windspeed in m/s
%                 wind(2) = wind heading in radians from north
%   hdg_g       : initial (approach) ground track in radians from north
%                 This is the ground track heading for the first waypoint
%                 in the orbit.  Subsequent waypoints will be based off
%                 of this heading, increasing by 360deg/n, to complete an
%                 orbit around the POI.  The sign of the heading
%                 increment is based on left (-) or right (+) turns,
%                 which is based on the direction (left or right) that
%                 the side sensor is aimed.
%   V_a         : desired airspeed in m/s for the aircraft during the orbit
%                 This may change to allow the aircraft to fly the orbit
%                 path while maintaining altitude and sensor on target
%   V_stall     : the stall speed of the aircraft in m/s.  V_a may be
%                 iterated in -0.25m/s increments to generate a set of
%                 waypoints if wind speed is high enough.  If the
%                 iteration drops below the V_stall, the orbit altitude
```

```
%                   incrementally increases.
%   phi_max     : the maximum bank angle the aircraft can handle.  If the
%                   calculated bank angle is greater than phi_max, the
%                   algorithm will iterate the altitude so that the
%                   algorithm does not command a flight plan that required
%                   a bank angle which exceeds phi_max
%   n           : the number of desired waypoints in the orbit path
%
% OUTPUTS:
%   waypoints   : a nx4 vector of waypoints for the aircraft to fly through
%                   to aim the sensor at the point of interest.  The
%                   waypoints should form a rough orbit about the point,
%                   but will not be circular in the presence of wind.
%                   Assumes alt = acft_pos(3), such that the aircraft
%                   attempts to retain the same altitude it is starts at.
%                   Each row is the (lat, lon, alt, airspeed) vector.
%                       Lat is in deg N
%                       Lon is in deg E
%                       Alt is in meters AGL
%                       Airspeed is in m/s
%
% NOTES:
%   x-axis - positive out the nose
%   y-axis - positive out the RIGHT wing
%   z-axis - positive TOWARDS the GROUND
%   Yaw    - positive as nose goes to the right from pilot's perspective
%               0 out the nose; +90deg out right wing; -90deg out left wing
%   Pitch  - positive nose up
%   Roll   - positive as left wing rises
%
%   Sensor Roll should always be 0.  It just changes with Azmith (Yaw) and
%       Elevation (Pitch).
%
%   This algorithm assumes that the sensor is directly pointed out the left
%       or the right wing.  If the side sensor is only aimed at -45deg, the
%       algorithm then builds the orbit waypoints as if the sensor is aimed
```

```
%       at -90deg instead.

%% If there are no inputs
if nargin==0
    alt         = 50; % meters
    wind        = [convvel(10.0,'kts','m/s'),deg2rad(015)];
    hdg_g       = deg2rad(210); % initial approach heading
    V_a         = convvel(40,'kts','m/s'); % desired airspeed
    V_stall     = convvel(13,'kts','m/s'); % aircraft stall speed
    phi_max     = deg2rad(26);
    n           = 36; % n=12 waypoints to form the orbit path.
    %% Don't change these
    POI         = [39.34360,-86.02980,216];
    sens_att    = [deg2rad(-90) deg2rad(-39) deg2rad(00)]; % Side Sensor
    acft_pos    = [39.34319,-86.02916,alt];
end

%% Setup the original information
POI_GPS = POI;
POI = [0,0,0]; % sets the point of interest as the x,y,z origin
flight_alt = acft_pos(3); % Set the original flying altitude

%% Ensure at least one waypoint is generated
if n<1
    n=1;
end

%% Set up constants and variables
waypoints = ones(n,4); % initialize this variable
g = 9.81; % gravity constant in m/s^2
theta = 0; % first iteration of waypoint loop assumes 0deg pitch angle
phi = 0; % first iteration of waypoint loop assumes 0deg bank angle
vel_des = V_a; % The desired airspeed at each waypoint
alt_step = 10; % altitude step size in meters

%% Prepare the sensor data
```

```
    % Check to see if the sensor is the side sensor
if sens_att(1)~=0
        % Then it is not aimed out the front
    else % it is aimed out the front
        disp('Use the side sensor');
    end

    % If the sensor azmith is < 0 (i.e., -90deg), make it positive.
    if sens_att(1) < 0
        sens_att(1) = 2*pi+sens_att(1);
    end
% Determine which wing the sensor points out.  Assume the sensor azmith is
    %   either +90deg or -90deg.  This assumption removes the need for inducing
    %   a slideslip into the situation or to spiral into the POI location.
    if sens_att(1) < pi
        % Right wing sensor
        sens_att(1) = pi/2;
    else
        % Left wing sensor
        sens_att(1) = 3*pi/2;
    end

    %% Iterate to solve bank angle and airspeed by comparing radii
    % % Iteration is required because the sensor aimpoint offset is dependent on
    % %   bank angle (phi).  Phi also determines the orbit distance.  This
    % %   iteration increments phi in a step size relative to the aircraft's
    % %   altitude.  If the step size was hard coded, the iteration would not
    % %   work for high altitudes, as the changes in phi must be very small.
    % %   First, a radius (R1) is found based on the sensor aimpoint, which is a
    % %   function of phi.  Next, a radius (R2) is found based on an aircraft
    % %   dynamics equation to find the radius of a turn based on aircraft
    % %   velocity and bank angle.  These two radii are compared.  Until the
    % %   difference between the two radii is less than some error distance, phi
    % %   will increment based on the established step size.  R1 increases with
    % %   phi and R2 decreases with phi.  This algorithm assumes that altitude
    % %   (acft_pos(3)) is fixed.  Furthermore, it will fix velocity, unless phi
```

```
% %   exceeds a maximum bank angle, at which point it will decrease velocity
% %   for all waypoints in the orbit, until a sustainable airspeed can be
% %   maintained.
% %
% % Use phi to solve for a sensor aimpoint and from that a radius (R1)
% % Use phi to solve for a radius given a velocity (R2)
% % Iterate until the radii are very close to each other (err_dist)
%
% % Set the constants for this algorithm
% err_dist = 2.5; % for the iteration to find the waypoints, measured in m
% maxbank = pi/2 - abs(sens_att(2)); % maximum bank angle
% stepsize = 1/(10*acft_pos(3)); % increment phi (rad), based on the acft alt
% R1 = 0; % horizontal distance from the aircraft to the sensor aimpoint
% % Next line ensures the loop goes through at least once
% R2 = 2*err_dist; % radius of orbit based on the aircraft dynamics equation
%
% while abs(R1-R2)>err_dist
%     % Increment bank angle. If right hand turns, phi_new > phi_old
%     if sens_att(1) < pi % Right wing sensor --> right hand turns
%         phi = phi + stepsize; % increment bank angle
%     else % if left hand turns, phi_new < phi_old because LHT are neg bank
%         phi = phi - stepsize; % increment bank angle
%     end
%
%     % Calculate the aircraft heading at the projected waypoint, using
%     %   V_a, V_w, hdg_w, and hdg_g
%     hdg_a = find_hdg(V_a,wind(1),wind(2),hdg_g);
%
%     % Get the sensor aimpoint offsets.  The return value is a 1x3 vector of
%     %   (del_x,del_y,0).  It indicates where the sensor will point in ...
   relation
%     %   to the aircraft.  To make the offset relative to the aircraft, the
%     %   function assumes the aircraft position is (0,0,z).  The sensor
%     %   aimpoint radius (R1) is independent of heading, only depending
%     %   on pitch (theta), bank (phi), height (acft_pos(3)), and sensor
%     %   orientation.  Therefore, hdg_a can equal 0.  Heading WILL
```

```
%     %   affect, however, the offset waypoint for the aircraft.
%     sensor_aimpoint = sensoraimpoint([0,0,acft_pos(3)],[hdg_a,theta,phi],...
    sens_att);
%     % Use the sensor aiming distance to solve for R1
%     R1 = sqrt(sensor_aimpoint(1)^2+sensor_aimpoint(2)^2);
%     % Find R2, the radius based on the bank angle and velocity
%     R2 = abs(V_a^2 / (g * tan(phi))); % need abs in case phi is neg
%     % If phi is greater than maximum bank angle, change ground speed
%     %   instead
%     if abs(phi) > abs(maxbank) % if bank is greater than max bank angle
% %         disp('changing velocity');
%         phi = 0;
%         V_a = V_a - 1; % increment desired airspeed by 1m/s
%         R1 = 0;R2 = 2*err_dist;
%     end
% end

%% Prep to keep altitude level throughout the orbit
uneven_alt = true;
while uneven_alt

%% Determine minimum altitude for orbit in wind
% Determine the minimum altitude the aircraft can fly the orbit at in the
%   given wind.  This is determined by evaluating the required bank angle
%   at the waypoint where the ground track would correspond to the wind
%   heading.  At this waypoint, the aircraft has a direct tailwind and
%   would require the lowest airspeed to maintain the same ground speed in
%   the circle.  Simply put, this is the point of failure for the algorithm
%   that does not increment altitude for the solution.

% Find the bank angle, airspeed, and altitude
check   = [1 1]; % set an imag component to ensure iteration
while check ~= 0
    % Calculate aircraft and ground speed when the aircraft has a direct
    %   tail wind
    [hdg_a,V_g] = windcorr(V_stall,wind(1),wind(2),wind(2));
```

```
        phi(1) = atan(((-V_g^2+acft_pos(3)*g)+sqrt((V_g^2-acft_pos(3)*g)^2-4*(...
            acft_pos(3)*g*tan(sens_att(2)))*(V_g^2*tan(sens_att(2)))))/(2*acft_pos...
            (3)*g*tan(sens_att(2))));
        phi(2) = atan(((-V_g^2+acft_pos(3)*g)-sqrt((V_g^2-acft_pos(3)*g)^2-4*(...
            acft_pos(3)*g*tan(sens_att(2)))*(V_g^2*tan(sens_att(2)))))/(2*acft_pos...
            (3)*g*tan(sens_att(2))));
        check = imag(phi);
        % Vary airspeed and altitude if bank angle has imaginary component
        if check ~= 0
            % iterate until bank angle has no imaginary components
            acft_pos(3) = acft_pos(3) + alt_step;
        elseif (abs(phi) > abs(phi_max))
        % Ensure that the calculated bank angle is not greater than the
        %   maximum allowed bank angle.  If the bank angle is greater than the
        %   maximum bank angle, increment altitude accordingly.
            disp('initial check: phi exceeds max phi');
            acft_pos(3) = acft_pos(3)+alt_step;
        end
    end

    %% Calculate Waypoints
    for ii=1:n
        % Takes as "inputs" V_a (calculated above), wind information, desired
        %   initial ground heading, aircraft altitude, aircraft orientation,
        %   sensor orientation, and POI location.  Outputs n waypoints around
        %   the POI.
        % Use the current ground vector to find the first waypoint.  Build all
        %   other waypoints as a series of waypoints based on that initial
        %   vector.

        % Reset V_a if changed on a previous loop
        V_a = vel_des;

        % Iterate until phi is reasonable
        phi_check = false;
        while phi_check==false
```

```
% Find the bank angle, airspeed, and altitude
check   = [1 1]; % set an imag component to ensure iteration
while check ~= 0
    % Calculate aircraft heading and ground speed at the waypoint
    [hdg_a,V_g] = windcorr(V_a,wind(1),wind(2),hdg_g);

    phi(1) = atan(((-V_g^2+acft_pos(3)*g)+sqrt((V_g^2-acft_pos(3)*g)...
        ^2-4*(acft_pos(3)*g*tan(sens_att(2)))*(V_g^2*tan(sens_att(2)))...
        ))/(2*acft_pos(3)*g*tan(sens_att(2))));
    phi(2) = atan(((-V_g^2+acft_pos(3)*g)-sqrt((V_g^2-acft_pos(3)*g)...
        ^2-4*(acft_pos(3)*g*tan(sens_att(2)))*(V_g^2*tan(sens_att(2)))...
        ))/(2*acft_pos(3)*g*tan(sens_att(2))));
    check = imag(phi);
    % Vary airspeed and altitude if bank angle has imaginary ...
        component
    if check ~= 0
        % iterate until bank angle has no imaginary components
        V_a = V_a - 0.25;
        % If V_a iterates to < V_stall, increment altitude
        if V_a < V_stall
            acft_pos(3) = acft_pos(3)+alt_step;
        end
    end
end

% Take the value closest to zero, for the minimum bank.
if sens_att(1) < pi % Right wing sensor --> right hand turns
    phi = min(phi);
else
    phi = max(phi);
end

if abs(phi) < pi/2+sens_att(2)
    phi_check = true;
else
```

```
            acft_pos(3) = acft_pos(3)+alt_step;
        end

        % Ensure that the calculated bank angle is not greater than the
        %   maximum allowed bank angle.  If the bank angle is less than the
        %   maximum bank angle, phi is acceptable.  If it is greater than
        %   the maximum bank angle, increment altitude accordingly.
        if abs(phi) < abs(phi_max)
            phi_check = true;
        else
            disp('phi exceeds max phi');
            phi_check = false;
            acft_pos(3) = acft_pos(3)+alt_step;
        end
    end

    %% Generate Error if aircraft cannot go slow enough to maintain orbit
    if V_g <= 0
        error('Orbit:negVg','Windspeed meets or exceeds commanded airspeed ...
           and is too high to maintain an orbit.');
    end

    % Get the sensor aimpoint offsets.  The return value is a 1x3 vector of
    %   (del_x,del_y,0).  It indicates where the sensor will point in ...
       relation
    %   to the aircraft.  To make the offset relative to the aircraft, the
    %   function assumes the aircraft position is (0,0,z).  Additionally, the
    %   to plot the point, assume the aircraft has no pitch and no roll ...
       during
    %   the overflight.  Therefore, the aimpoint only depends on heading.
    sensor_aimpoint = sensoraimpoint([0,0,acft_pos(3)],[hdg_a,theta,phi],...
       sens_att);

    % Save the waypoint
    POI_Offset = [POI(1)-sensor_aimpoint(1),POI(2)-sensor_aimpoint(2),...
       acft_pos(3)];
```

```
        [waypoints(ii,1),waypoints(ii,2)]=cart2geod([POI_GPS(1),POI_GPS(2)], ...
           POI_Offset(1), POI_Offset(2), flight_alt);
306     waypoints(ii,3) = acft_pos(3);

        waypoints(ii,4) = V_a;
    %     waypoints(ii,5) = V_g;
    %     waypoints(ii,6:8) = rad2deg([hdg_a,theta,phi]);
311     if sens_att(1) < pi % Right wing sensor --> right hand turns
            hdg_g = hdg_g+deg2rad(360/n);
        else
            hdg_g = hdg_g-deg2rad(360/n);
        end
316 end

    %% Ensure the entire orbit is at the same altitude
    % Check the first waypoint altitude and the last waypoint altitude. If
    %   different, repeat the algorithm.
321 first_alt = waypoints(1,3);
    last_alt = waypoints(n,3);
    if first_alt ~= last_alt
        acft_pos(3) = waypoints(n,3);
    else
326     uneven_alt = false;
    end
    end
```

C.2 Overflight Algorithm Code

Listing C.2: MATLAB/overfly.m

```
function waypoints = overfly(POI,acft_pos,sens_att,fov,wind,look_angle,V_a)
%waypoints = overfly(POI,acft_pos,sens_att,fov,wind,look_angle,V_a)
% Generates offset waypoints for the aircraft to fly such that the sensor
%   will aim at the target while the aircraft approaches from a specific
%   orientation and in the presence of wind.
%
% INPUTS:
%   POI          : a 1x3 vector of the point of interest's geodetic location
%                  (lat, lon, altitude). Lat and Lon are in decimal
%                  degrees. Altitude is in meters MSL.
%   acft_pos     : a 1x3 vector of the aircraft's current geodetic location
%                  (lat, lon, alt), assuming AGL altitude.
%   sens_att     : a 1x3 vector of the sensor's attitude angles relative to
%                  the aircraft (yaw, pitch, roll)
%   fov          : a 1x2 vector for the sensor field of view in radians
%                  fov(1) = horizontal field of view
%                  fov(2) = vertical field of view
%   wind         : a 1x2 vector for wind
%                  wind(1) = windspeed in m/s
%                  wind(2) = wind heading in radians from north
%   look_angle   : the desired look angle, no matter which sensor is used,
%                  in radians from north
%   V_a          : desired airspeed for the aircraft during the overflight
%                  This may change to allow the aircraft to fly the
%                  path while maintaining altitude and sensor on target
%
% OUTPUTS:
%   waypoints    : a 3x3 vector of waypoints for the aircraft to fly through
%                  to aim the sensor at the target along the preferred
%                  ground vector.  Assumes z = acft_pos(3), such that the
%                  aircraft retains the same altitude it is currently at.
%                  Each row is the (lat, lon, alt, airspeed) vector.
%                      Lat is in deg N
%                      Lon is in deg E
```

```
%                       Alt is in meters AGL
%                       Airspeed is in m/s
%                   waypoints(1) = upstream waypoint
%                   waypoints(2) = POI Offset waypoint
%                   waypoints(3) = downstream waypoint
%
% NOTES:
%   x-axis - positive out the nose
%   y-axis - positive out the RIGHT wing
%   z-axis - positive TOWARDS the GROUND
%   Yaw    - positive as nose goes to the right from pilot's perspective
%               0 out the nose; +90deg out right wing; -90deg out left wing
%   Pitch  - positive nose up
%   Roll   - positive as left wing rises
%
%   Sensor Roll should always be 0.  It just changes with Azmith (Yaw) and
%       Elevation (Pitch).

%% If there are no inputs
if nargin==0
    POI         = [39.34336,-86.03189,216];
    acft_pos    = [0,00,100];
    sens_att    = [deg2rad(00) deg2rad(-49) deg2rad(00)]; % Front Sensor
%     sens_att    = [deg2rad(-90) deg2rad(-39) deg2rad(00)]; % Side Sensor
    fov         = [deg2rad(48),deg2rad(40)]; % h_fov and v_fov
    wind        = [convvel(05,'kts','m/s'),deg2rad(180)];
    look_angle  = deg2rad(020);
    V_a         = convvel(20,'kts','m/s');
end

%% Setup the original information
POI_GPS = POI;
POI = [0,0,0]; % sets the point of interest as the x,y,z origin

%% Find the aircraft heading based on look angle and sensor
% Aircraft heading is dependant on look angle and irrelavent to wind, in
```

```
%   this case, since no ground track has yet been determined.
hdg_a    = look_angle - sens_att(1);
% Establish aircraft attitude in this steady-level flight
acft_att = [hdg_a,0,0];

%% Find groundtrack for the additional waypoints
[V_g,hdg_g] = gnd(V_a,hdg_a,wind(1), wind(2));

% Ground speed cannot go negative.  However, the aircraft can fly in a
%   backwards direction, such that it has a "negative" ground speed, that
%   results as a reverse ground track than the aircraft nose is pointing.

%% Find Waypoints

% Get the sensor aimpoint offsets.  The return value is a 1x3 vector of
%   (del_x,del_y,0).  It indicates where the sensor will point in relation
%   to the aircraft.  To make the offset relative to the aircraft, the
%   function assumes the aircraft position is (0,0,z).  Additionally, the
%   to plot the point, assume the aircraft has no pitch and no roll during
%   the overflight.  Therefore, the aimpoint only depends on heading.
sensor_aimpoint = sensoraimpoint([0,0,acft_pos(3)],acft_att,sens_att);

% Save the waypoint
POI_Offset = [POI(1)-sensor_aimpoint(1),POI(2)-sensor_aimpoint(2),acft_pos(3)...
   ];
waypoints(2,1:3) = POI_Offset;
waypoints(2,4) = V_a;

% Find the sensor footprint
ftprint = footprint([0,0,acft_pos(3)],acft_att,sens_att,fov);

% Offset the up-stream (before the POI) waypoint by a value equal to the
%   maximum sensor footprint distance away from the aircraft.  This should
%   mean the POI will be centered as soon as the sensor footprint covers
%   the POI.
q = max(max(abs(ftprint)));
```

```
    upstream(1)      = POI_Offset(1)-q*cos(hdg_g);
107 upstream(2)      = POI_Offset(2)-q*sin(hdg_g);
    upstream(3)      = acft_pos(3);
    waypoints(1,1:3) = upstream;
    waypoints(1,4)   = V_a;

112 % Offset the down-stream (past the POI) waypoint by a value equal to twice
    %   the aircraft's altitude.  This doesn't really matter, so it makes a
    %   convienent, arbitrary number.
    % q = 2*acft_pos(3);
    downstream(1)    = POI_Offset(1)+q*cos(hdg_g);
117 downstream(2)    = POI_Offset(2)+q*sin(hdg_g);
    downstream(3)    = acft_pos(3);
    waypoints(3,1:3) = downstream;
    waypoints(3,4)   = V_a;

122 [waypoints(:,1),waypoints(:,2)]=cart2geod([POI_GPS(1),POI_GPS(2)], waypoints...
       (:,1), waypoints(:,2), acft_pos(3));
```

D.1 Dubins Path Algorithm Code

Listing D.1: MATLAB/dubins.m

```
function waypoints = dubins(acft_pos, hdg, alt_MSL, V_a, phi_max, turn_mat)
%waypoints = dubins(acft_pos, hdg, alt_MSL, V_a, phi_max, turn_mat)
%   Calculates a set of GPS waypoints for a turn-straight-turn path
%   (Dubins path) given an initial aircraft position and heading, a final
%   desired aircraft position and heading (for the start of another
%   algorithm), a turn radius for each of the turns,
%
% INPUTS:
%   acft_pos  : 2x3 matrix of geodetic positions for initial and final
%                   coordinates.  Lat and Lon are in decimal degrees.
%                   Altitude is AGL altitude in meters
%                   [Lat_init Lon_init  Altitude]
%                   [Lat_fin  Lon_fin   Altitude]
%   hdg       : 2x1 vector with initial and final headings in radians from
%                   north.
%                   [hdg_init] - aircraft's current heading
%                   [hdg_fin ] - desired heading at end of Dubins path
%   alt_MSL   : 2x1 vector of the MSL altitude in meters of the aircraft's
%                   current and final positions
%   V_a         : desired airspeed for the aircraft during the Dubins path
%   turn_mat  : 2x2 vector denoting direction of turns.
%                   The first row designates direction of turns with first
%                   number for initial turn_mat and second for final turn_mat...
    .
%                       +1 is CCW, -1 is CW, EX: [-1 -1] = CW CW turns
%                   The second row adds an extra turn_mat to either end.
%                   First number for initial turn_mat and second for
%                   final turn_mat.
%                       0 is no extra, 1 is extra. EX: [0 0] = no extra
%                       turns
%   phi_max   : maximum bank angle for the aircraft in radians
%
% OUTPUTS:
```

```
%   waypoints   : a nx4 vector of geodetic waypoints for the aircraft to
%                 fly the Dubins path.  Waypoints will start at the
%                 aircraft's position, one waypoint for each 90deg of
%                 each turn circle the aircraft will sweep through, one
%                 at the start and end of the straight line, and one for
%                 the aircraft's final position.
%               Each row is the (lat, lon, alt, airspeed) vector.
%                     Lat is in deg N
%                     Lon is in deg E
%                     Alt is in meters AGL
%                     Airspeed is in m/s

%% If NARGIN==0
if nargin==0
    acft_pos(1,:) = [39.34100,-86.02400,110];
    acft_pos(2,:) = [39.34388,-86.03111,100];
    hdg(1)      = deg2rad(000);
    hdg(2)      = deg2rad(210);
    alt_MSL(1)  = 216+acft_pos(1,3);
    alt_MSL(2)  = 216+acft_pos(2,3);
    V_a         = convvel(20,'kts','m/s'); % desired airspeed
    turn_mat(1,1:2) = [-1 +1];
    turn_mat(2,1:2) = [ 0  0];
        phi_max     = deg2rad(30);
end

%% Variables
% Number of sweep partitions, in radians, to divide the total swept angles
%   in each turn_mat of the Dubins path
sweep_part  = pi/4;
% Set the flight altitude for the Dubins path based on the altitude desired
%   at the end of the Dubins path
flight_alt   = acft_pos(2,3);
g            = 9.81; % gravity constant in m/s^2

%% Calculate turn radius based on V_a and phi_max
```

```
% finds the turn radius achievable by flying at the cruise airspeed and the
%   maximum bank angle.  A tighter turn could be made by slowing the cruise
%   airspeed to just above stall airspeed in the turn; however, this is not
%   done for simplicity.  Inputting the radius instead of calculating it
%   may result in a radius tighter than the aircraft can actually turn.
turn_rad = V_a^2/(tan(phi_max)*g);

%% Convert geodetic position into cartesian positions for easy calculations
% Use +y = North; +x = East; 0 rad from north is pi/2
geo_origin = acft_pos(2,:);
geo_origin(:,3) = alt_MSL(2);
[y x]=geod2cart(geo_origin, acft_pos(:,1), acft_pos(:,2));

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

% find turn circle centers.  The abs(turn(1,x)) term allows for a 0 to be
%   passed in so the turn is not made and the path ends at the x,y pair
    %circle 1
    x_c1=x(1) + turn_rad*cos(NEDHdg(1)+turn_mat(1,1)*pi/2);
    y_c1=y(1) + turn_rad*sin(NEDHdg(1)+turn_mat(1,1)*pi/2);
    %circle 2
    x_c2=x(2) + turn_rad*cos(NEDHdg(2)+turn_mat(1,2)*pi/2);
    y_c2=y(2) + turn_rad*sin(NEDHdg(2)+turn_mat(1,2)*pi/2);

% find points of tangency between two circles
if turn_mat(1,1)*turn_mat(1,2) == 1 %for init and final circles in same dir
    if turn_mat(1,1) == 1  %for ccw on circle 1 and circle 2
        Ai_c1=Zeroto2pi(NEDHdg(1)+3*pi/2);
        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_mat(2,1)*2*pi;
        A2=Zeroto2pi(Af_c2-Ai_c2)+turn_mat(2,2)*2*pi;
    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);
            A1=-m2piToZero(Af_c1-Ai_c1)+turn_mat(2,1)*2*pi;
            A2=-m2piToZero(Af_c2-Ai_c2)+turn_mat(2,2)*2*pi;
        end
    else  %for circles in opposite direction
        %check if points are too close for this
        if norm([x_c2-x_c1 y_c2-y_c1]) < 2*turn_rad
            error('Points are too close for cw/ccw turn_mat--abort.');
        end
        if turn_mat(1,1) == 1 % for ccw for circle 1, cw for circle 2
            %compute angle btwn circle centers and tangent line
            th=atan(turn_rad/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);
            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_mat(2,1)*2*pi;
            A2=-m2piToZero(Af_c2-Ai_c2)+turn_mat(2,2)*2*pi;
        else  %find angles for cw circle 1 and ccw circle 2
            th=atan(turn_rad/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_mat(2,1)*2*pi;
            A2=Zeroto2pi(Af_c2-Ai_c2)+turn_mat(2,2)*2*pi;
        end
    end

    %% Calculate waypoints
    % calculate points on circle 1
    A1div=ceil(A1/(sweep_part));
    ii=1:(A1div+1);
```

```
    x_wp=x_c1+turn_rad*cos(Ai_c1+A1*turn_mat(1,1)*(ii-1)/A1div);
    y_wp=y_c1+turn_rad*sin(Ai_c1+A1*turn_mat(1,1)*(ii-1)/A1div);
143 % keyboard;
    rise = turn_rad*sin(Ai_c2)-y_wp(1,length(y_wp));
    run  = turn_rad*cos(Ai_c2)-x_wp(1,length(x_wp));
    hdg_path = atan2(run,rise);
    if hdg_path < 0
148     hdg_path = 2*pi+hdg_path;
    end
    if abs(hdg(1) - hdg_path) < sweep_part/2
        % there should be no initial circle, as the final heading is very close
        %   to the initial heading and the circle then is a waste of time
153     x_wp(2:length(x_wp)) = []; y_wp(2:length(y_wp)) = [];
    else
        % keep the circle as it was calculated
    end

158 while hdg(2) > 2*pi
        hdg(2) = hdg(2) - 2*pi;
    end
    % calculate points for circle 2 (including endpoint)
    x_wp_sofar = length(x_wp)+2;
163 y_wp_sofar = length(y_wp)+2;
    A2div=ceil(A2/(sweep_part));
    ii=1:(A2div+1);
    x_wp=[x_wp x_c2+turn_rad*cos(Ai_c2+A2*turn_mat(1,2)*(ii-1)/A2div)];
    y_wp=[y_wp y_c2+turn_rad*sin(Ai_c2+A2*turn_mat(1,2)*(ii-1)/A2div)];
168 if abs(hdg(2) - hdg_path) < sweep_part/2
        % there should be no final circle, as the final heading is very close
        %   to the heading along the path and the circle then is a waste of time
        x_wp(x_wp_sofar:length(x_wp)) = []; y_wp(y_wp_sofar:length(y_wp)) = [];
    else
173     % keep the circle as it was calculated
    end

    %% Convert to GPS coordinates
```

```
    waypoints = ones(4,length(x_wp));
178 [waypoints(1,:) waypoints(2,:)]=cart2geod(geo_origin, y_wp, x_wp, alt_MSL(2))...
        ;
    waypoints = waypoints';

    %% Format waypoints for output
    waypoints(:,3) = flight_alt;
183 waypoints(:,4) = V_a;

    %% Zero to 2Pi Function
    function A=Zeroto2pi(Ai)
    % This function converts all input angles to between 0 and 2*pi
188 %   input is a vector
    A=Ai;
    for ii=1:length(Ai)
        while A(ii) < 0
            A(ii)=A(ii)+2*pi;
193     end
        while A(ii) > 2*pi
            A(ii)=A(ii)-2*pi;
        end
    end
198
    %% -2Pi to Zero Function
    function A=m2piToZero(Ai)
    % This function converts all input angles to between 0 and 2*pi
    %   input is a vector
203 A=Ai;
    for ii=1:length(Ai)
        while A(ii) > 0
            A(ii)=A(ii)-2*pi;
        end
208     while A(ii) < -2*pi
            A(ii)=A(ii)+2*pi;
        end
    end
```

D.2 Dubins Orbit Wrapper Code

Listing D.2: MATLAB/orbit_dubins.m

```
function [path_wpts, orb_wpts] = orbit_dubins(POI,acft_pos,sens_att,wind,...
    hdg_a,V_a,V_stall,phi_max,n)
%[path_wpts, orb_wpts] = orbit_dubins(POI,acft_pos,sens_att,wind,hdg_a,V_a,...
    V_stall,phi_max,n)
% Finds both the orbit around a POI and a Dubins path to get from the
%   aircraft's current position to the orbit.  Takes the aircraft's current
%   position and heading, has it run a CW/CCW circle to get on a path to
%   intercept the orbit.
%
% INPUTS:
%   POI         : a 1x3 vector of the point of interest's geodetic location
%                 (lat, lon, altitude). Lat and Lon are in decimal
%                 degrees. Altitude is in meters MSL.
%   acft_pos    : a 1x3 vector of the aircraft's current geodetic location
%                 (lat, lon, alt), assuming AGL altitude.
%   sens_att    : a 1x3 vector of the sensor's attitude angles relative to
%                 the aircraft (yaw, pitch, roll)
%   wind        : a 1x2 vector for wind
%                 wind(1) = windspeed in m/s
%                 wind(2) = wind heading in radians from north
%   hdg_a       : current aircraft heading
%   V_a         : desired airspeed in m/s for the aircraft during the orbit
%                 This may change to allow the aircraft to fly the orbit
%                 path while maintaining altitude and sensor on target
%   V_stall     : the stall speed of the aircraft in m/s.  V_a may be
%                 iterated in -0.25m/s increments to generate a set of
%                 waypoints if wind speed is high enough.  If the
%                 iteration drops below the V_stall, the orbit altitude
%                 incrementally increases.
%   phi_max     : the maximum bank angle the aircraft can handle.  If the
%                 calculated bank angle is greater than phi_max, the
%                 algorithm will iterate the altitude so that the
%                 algorithm does not command a flight plan that required
%                 a bank angle which exceeds phi_max
```

```
%   n           : the number of desired waypoints in the orbit path
%
% OUTPUTS:
%   path_wpts   : a set of waypoints for a Dubins path that will get the
%                 aircraft from its current position to the sensor
%                 aimpoint orbit
%   orb_wpts    : a set of waypoints for the aircraft to fly such that the
%                 sensor will be aimed at the POI during the orbit


%% Set Variables
if nargin==0
    POI         = [39.34360,-86.02980,216];
    alt         = 100; % meters
    % acft_pos    = [39.34100,-86.03700,alt]; % SW
    acft_pos    = [39.34100,-86.02400,alt]; % SE
    % acft_pos    = [39.34700,-86.02400,alt]; % NE
    % acft_pos    = [39.34700,-86.03700,alt]; % NW
    % acft_pos    = [39.34100,-86.03040,alt]; % due south
    % acft_pos    = [39.34700,-86.03115,alt]; % due north
    % acft_pos    = [39.34379,-86.03700,alt]; % due west
    % acft_pos    = [39.34379,-86.02400,alt]; % due east
    sens_att  = [deg2rad(-90) deg2rad(-39) deg2rad(00)]; % Side Sensor
    wind      = [convvel(10.0,'kts','m/s'),deg2rad(015)];
    hdg_a       = deg2rad(315); % initial acft heading
    n         = 18; % n=12 waypoints to form the orbit path.
    % BATCAM Values
    V_a         = convvel(20,'kts','m/s'); % desired airspeed
    phi_max     = deg2rad(30);
    V_stall     = convvel(8,'kts','m/s'); % The stall speed for the aircraft.

    % % SIG Values
    % V_a         = convvel(40,'kts','m/s'); % desired airspeed
    % phi_max     = deg2rad(26);
    % V_stall     = convvel(13,'kts','m/s'); % The stall speed for the ...
       aircraft
```

```
end

%% Find hdg_g
% Get heading from current aircraft position to initial point
[N E] = geod2cart(acft_pos, POI(1), POI(2));
hdg_g = atan2(E,N);
if hdg_g < 0
    hdg_g = 2*pi+hdg_g;
end
% hdg_g is the ground track angle from the aircraft's initial position
%   to the initial point of the orbit path

%% Calculate orbit waypoints
orb_wpts  = orbit(POI,acft_pos,sens_att,wind,hdg_g,V_a,V_stall,phi_max,n);

%% Get Initial Turn
% Fix inputs if greater than 2*pi
while hdg_a > 2*pi
    hdg_a = hdg_a - 2*pi;
end
while hdg_g > 2*pi
    hdg_g = hdg_g - 2*pi;
end

% Fix headings so they all work together
if (hdg_a < 2*pi) && (hdg_g >= 0)
    hdg_g = hdg_g + 2*pi;
end
if (hdg_g < 2*pi) && (hdg_a >= 0)
    hdg_a = hdg_a + 2*pi;
end

% Find the difference in headings
diff = abs(hdg_g - hdg_a);
while diff > 2*pi
    diff = diff - 2*pi;
```

```
104 end

    % Determine which acft_quad acft heading is in
    if      (    0     < diff) && (diff <=    +pi   )
        turn_mat(1,1) = -1; % CW
109 else
        turn_mat(1,1) = +1; % CCW
    end

    %% Get Final Turn
114 % Check to see if the sensor is the side sensor
    if sens_att(1)~=0
        % Then it is not aimed out the front
    else % it is aimed out the front
        disp('Use the side sensor');
119 end

    % If the sensor azmith is < 0 (i.e., -90deg), make it positive.
    if sens_att(1) < 0
        sens_att(1) = 2*pi+sens_att(1);
124 end
    % Determine which wing the sensor points out.  Assume the sensor azmith is
    %   either +90deg or -90deg.  This assumption removes the need for inducing
    %   a slideslip into the situation or to spiral into the POI location.
    if sens_att(1) < pi
129     % Right wing sensor, therefore right-hand/clockwise turns
        turn_mat(1,2) = -1; %CW
    else
        % Left wing sensor, therefore left-hand/counter-clockwise turns
        turn_mat(1,2) = +1; %CCW
134 end

    %% Finish out turn matrix
    turn_mat(2,1:2) = [ 0   0];

139 %% Calculate Dubins Waypoints (New Algorithm)
```

```
    acft_pos(1,:) = acft_pos;
    acft_pos(2,:) = orb_wpts(1,1:3);
    hdg(1)        = hdg_a;
    hdg(2)        = hdg_g;
144 alt_MSL       = acft_pos(:,3) + POI(3);
    path_wpts     = dubins(acft_pos, hdg, alt_MSL, V_a, phi_max, turn_mat);
```

D.3 Dubins Overflight Wrapper Code

Listing D.3: MATLAB/overfly_dubins.m

```
function [path_wpts, over_wpts] = overfly_dubins(POI,acft_pos,sens_att,fov,...
    wind,look_angle,V_a,phi_max,hdg_a)
%[path_wpts, over_wpts] = overfly_dubins(POI,acft_pos,sens_att,fov,wind,hdg_a...
    ,V_a,V_stall,phi_max,n)
% Finds both the overflight path to keep a sensor aimed at a POI and a
%   Dubins path to get from the aircraft's current position to the
%   overflight.  Takes the aircraft's current position and heading, has it
%   run a CW/CCW circle to get on a path tointercept the orbit.
%
% INPUTS:
%   POI          : a 1x3 vector of the point of interest's geodetic location
%                  (lat, lon, altitude). Lat and Lon are in decimal
%                  degrees. Altitude is in meters MSL.
%   acft_pos     : a 1x3 vector of the aircraft's current geodetic location
%                  (lat, lon, alt), assuming AGL altitude.
%   sens_att     : a 1x3 vector of the sensor's attitude angles relative to
%                  the aircraft (yaw, pitch, roll)
%   fov          : a 1x2 vector for the sensor field of view in radians
%                  fov(1) = horizontal field of view
%                  fov(2) = vertical field of view
%   wind         : a 1x2 vector for wind
%                  wind(1) = windspeed in m/s
%                  wind(2) = wind heading in radians from north
%   look_angle   : the desired look angle, no matter which sensor is used,
%                  in radians from north
%   V_a          : desired airspeed for the aircraft during the overflight
%                  This may change to allow the aircraft to fly the
%                  path while maintaining altitude and sensor on target
%   phi_max      : the maximum bank angle the aircraft can handle.  This is
%                  used to calculate the minimum turn radius for the
%                  Dubins path assuming the aircraft maintains V_a
%   hdg_a        : current aircraft heading
%
% OUTPUTS:
```

```
%   path_wpts    : a set of waypoints for a Dubins path that will get the
%                  aircraft from its current position to the sensor
%                  aimpoint orbit
%   over_wpts    : a set of waypoints for the aircraft to fly such that the
%                  sensor will be aimed at the POI during the overflight

%% Set Variables
if nargin==0
    POI          = [39.34360,-86.02980,216];
    alt          = 100; % meters
    % acft_pos     = [39.34100,-86.03700,alt]; % SW
    acft_pos     = [39.34100,-86.02400,alt]; % SE
    % acft_pos     = [39.34700,-86.02400,alt]; % NE
    % acft_pos     = [39.34700,-86.03700,alt]; % NW
    % acft_pos     = [39.34100,-86.03040,alt]; % due south
    % acft_pos     = [39.34700,-86.03115,alt]; % due north
    % acft_pos     = [39.34379,-86.03700,alt]; % due west
    % acft_pos     = [39.34379,-86.02400,alt]; % due east
    sens_att     = [deg2rad(00) deg2rad(-49) deg2rad(00)]; % Front Sensor
%     sens_att     = [deg2rad(-90) deg2rad(-39) deg2rad(00)]; % Side Sensor
    fov          = [deg2rad(48),deg2rad(40)]; % h_fov and v_fov
    wind         = [convvel(05,'kts','m/s'),deg2rad(180)];
    look_angle   = deg2rad(020);
    hdg_a        = deg2rad(315); % initial acft heading

    % BATCAM Values
    V_a          = convvel(20,'kts','m/s'); % desired airspeed
    phi_max      = deg2rad(30);

    % % SIG Values
    % V_a          = convvel(40,'kts','m/s'); % desired airspeed
    % phi_max      = deg2rad(26);
end

%% Calculate overflight waypoints
over_wpts  = overfly(POI,acft_pos,sens_att,fov,wind,look_angle,V_a);
```

```
%% Find hdg_over_g
% The ground track the aircraft will fly while flying the overflight
%   waypoint path.  The N,E points are needed to find the path from the
%   MAV's current position to the initial point.  The hdg_g is necessary
%   to calculate the Dubins path.
[over_N over_E]=geod2cart(POI, over_wpts(:,1), over_wpts(:,2));
hdg_over_g = atan2(over_E(2)-over_E(1),over_N(2)-over_N(1));
if hdg_over_g < 0
    hdg_over_g = 2*pi+hdg_over_g;
end

%% Find hdg_g
% Get heading from current aircraft position to initial point
[N E] = geod2cart(POI,acft_pos(1),acft_pos(2));
hdg_g = atan2(over_E(1)-E,over_N(1)-N);
if hdg_g < 0
    hdg_g = 2*pi+hdg_g;
end
% hdg_g is the ground track angle from the aircraft's initial position
%   to the initial point of the overflight path

%% Get Initial Turn
% Fix inputs if greater than 2*pi
while hdg_a > 2*pi
    hdg_a = hdg_a - 2*pi;
end
while hdg_g > 2*pi
    hdg_g = hdg_g - 2*pi;
end

% Fix headings so they all work together
if (hdg_a < 2*pi) && (hdg_g >= 0)
    hdg_g = hdg_g + 2*pi;
end
if (hdg_g < 2*pi) && (hdg_a >= 0)
```

```
    hdg_a = hdg_a + 2*pi;
end

% Find the difference in headings
diff = abs(hdg_g - hdg_a);
while diff > 2*pi
    diff = diff - 2*pi;
end

% Determine which acft_quad acft heading is in
if      (   0    < diff) && (diff <=   +pi   )
    turn_mat(1,1) = -1; % CW
else
    turn_mat(1,1) = +1; % CCW
end

%% Get Final Turn
% Find final turn based on the difference between the hdg_g and the
%   hdg_over_g.
while hdg_g > 2*pi
    hdg_g = hdg_g - 2*pi;
end
while hdg_over_g > 2*pi
    hdg_over_g = hdg_over_g - 2*pi;
end
diff = hdg_g - hdg_over_g;
if diff < 0
    diff = diff + 2*pi;
end
if      (  0   <= diff ) && ( diff < pi )
    turn_mat(1,2) = +1; % CCW
else
    turn_mat(1,2) = -1; % CW
end

%% Finish out turn matrix
```

```
    turn_mat(2,1:2) = [ 0   0]; % no "extra" turns

    %% Calculate Dubins Waypoints (New Algorithm)
    acft_pos(1,:) = acft_pos;
145 acft_pos(2,:) = over_wpts(1,1:3);
    hdg(1)        = hdg_a;
    hdg(2)        = hdg_over_g;
    alt_MSL       = acft_pos(:,3) + POI(3);
    path_wpts     = dubins(acft_pos, hdg, alt_MSL, V_a, phi_max, turn_mat);
```

E.1 Cartesian to Geodetic Converter Code

Listing E.1: MATLAB/cart2geod.m

```
function [Lat Lon]=cart2geod(geo_origin, N, E, MSL_alt)
%[Lat Lon]=cart2geod(geo_origin, N, E, MSL_alt)
% Generates latitude and longitude coordinates in decimal degrees,
%   based on a given orgin in latitude and longitude, the MSL altitude of
%   that origin, and an North-East offset in meters from that origin.
%   The geodetic origin must correlate with the North-East origin.
%   This function is only good for small distances (>1000km)
%
% INPUTS:
%   geo_origin : a 1x2 vector of latitude and longitude in decimal degrees
%                   This must correspond with (N,E) = (0,0)
%   N          : an offset to the North in meters
%   E          : an offset to the East in meters
%   MSL_alt    : the altitude in meters above mean sea level (MSL) of the
%                   geodetic origin
%
% OUTPUTS:
%   Lat        : the geodetic latitude in decimal degrees
%   Lon        : the geodetic longitude in decimal degrees

%set the origin as the first data point passed in with the matrix
Lat_o = deg2rad(geo_origin(1));
Lon_o = deg2rad(geo_origin(2));

%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+N/(Rm+MSL_alt);
Lon=Lon_o+E/((Rp+MSL_alt)*cos(Lat_o));
```

```
% convert back to decimal degrees
Lat=rad2deg(Lat);
Lon=rad2deg(Lon);
```

E.2 Geodetic to Cartesian Converter Code

Listing E.2: MATLAB/geod2cart.m

```
function [N E]=geod2cart(geo_origin, Lat, Lon)
%[N E]=geod2cart(geo_origin, Lat, Lon)
% Generates North and East offset values in meters from the geodetic
%   origin to the input Latitude and Longitude.
%
% INPUTS:
%   geo_origin : a 1x3 vector for geodetic origin in latitude, longitude,
%                   and MSL altitude.  Latitude and longitude are in
%                   decimal degrees and MSL altitude is in meters.
%   Lat        : offset latitude point in decimal degrees
%   Lon        : offset longitude point in decimal degrees
%
% OUTPUTS:
%   N          : offset distance to the north in meters
%   E          : offset distance to the east in meters

%Convert to radians
Lat = deg2rad(Lat);
Lon = deg2rad(Lon);

%set the origin
Lat_o = deg2rad(geo_origin(1));
Lon_o = deg2rad(geo_origin(2));
Alt_o = geo_origin(3);

%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;
```

```
N = (Rm+Alt_o)*(Lat-Lat_o);
E = (Rp+Alt_o)*cos(Lat_o)*(Lon-Lon_o);
```

E.3 Sensor Aimpoint Code

Listing E.3: MATLAB/sensoraimpoint.m

```
function sensor_aimpoint = sensoraimpoint(acft_pos,acft_att,sens_att)
%sensor_aimpoint = sensoraimpoint(acft_pos,acft_att,sens_att)
% Determines the position on the ground where the sensor is currently
%   aimed, based on aircraft position, aircraft attitude, and sensor
%   attitude.  Assumes a North-East-Down system.
%
% INPUTS:
%   acft_pos : a 1x3 vector of the aircraft's current postion in m
%              (x, y, z).  Assumes z is same as AGL altitude.
%   acft_att : a 1x3 vector of the aircraft attitude angles in radians
%              (yaw, pitch, roll)
%   sens_att : a 1x3 vector of the sensor's attitude angles relative to the
%              aircraft (yaw, pitch, roll)
%   h_fov    : horizontal field of view in radians for the given sensor
%   v_fov    : vertical field of view in radians for the given sensor
%
% OUTPUTS:
%   sensor_aimpoint : a 1x3 vector of the sensor aimpoint, assuming z = 0
%       if the aircraft (x,y) position is given as (0,0), the sensor
%           aimpoint is relative to the aircraft position, not to the earth
%
% NOTES:
%   x-axis - positive out the nose
%   y-axis - positive out the RIGHT wing
%   z-axis - positive TOWARDS the GROUND
%   Yaw    - positive as nose goes to the right from pilot's perspective
%              0 out the nose; +90deg out right wing; -90deg out left wing
%   Pitch  - positive nose up
%   Roll   - positive as left wing rises
%
```

```
%   Sensor Roll should always be 0.  It just changes with Azmith (Yaw) and
%       Elevation (Pitch).

if nargin==0
    % Edit these to change the picture
    acft_pos = [00,00,100];
    acft_att = [deg2rad(00) deg2rad(00) deg2rad(-8.227)];
%     sens_att = [deg2rad(00) deg2rad(-49) deg2rad(00)]; % Front Sensor
    sens_att = [deg2rad(-90) deg2rad(-39) deg2rad(00)]; % Side Sensor
end

% Create a sensor unit vector that points out the nose of the aircraft.
% This can then be rotated to find the unit vector where the sensor is
% aimed.
sens_vec = [1 0 0]';

% angle2dcm is a MATLAB command that does the direction cosine matrix from
%   rotation angles.  It finds reference to body direction cosine matrix.
%   C = angle2dcm( yaw, pitch, roll ) where C is direction cosine matrix.
% angle2dcm' finds body to reference frames

% This produces a unit vector that points at the target from the sensor and
% is in NED coordinate system.
aim_vector = angle2dcm(acft_att(1),acft_att(2),acft_att(3))'*angle2dcm(...
   sens_att(1),sens_att(2),sens_att(3))'*sens_vec;

% Get the angle in the vertical realm from target to sensor unit vector.
tan_theta = aim_vector(3)/sqrt(aim_vector(1)^2+aim_vector(2)^2);

% Get the horizontal distance across an assumed 2-D flat-plane earth
horizontal_distance = acft_pos(3)/tan_theta;

% Get the angle in the horizontal plane from target to sensor unit vector.
psi = atan2(aim_vector(2),aim_vector(1));

% Get the (x,y) of the target
```

```
if aim_vector(3)>0 % if z is pointed towards the ground
    x = acft_pos(1)+horizontal_distance*cos(psi);
    y = acft_pos(2)+horizontal_distance*sin(psi);
    z = 0;
    sensor_aimpoint = [x,y,z];
else % if z does not aim towards the ground
    sensor_aimpoint = [NaN,NaN,NaN];
end
```

E.4 Footprint Code

Listing E.4: MATLAB/footprint.m

```
function footprint = footprint(acft_pos,acft_att,sens_att,fov)
%footprint = footprint(acft_pos,acft_att,sens_att,fov)
% Generates a sensor footprint for a given sensor and aircraft.
%
% INPUTS:
%   acft_pos    : a 1x3 vector of the aircraft's current postion in m
%                 (x,y,z).  Assumes z is same as AGL altitude.
%   sens_att    : a 1x3 vector of the sensor's attitude angles relative to
%                 the aircraft (yaw, pitch, roll)
%   acft_att : a 1x3 vector of the aircraft attitude angles in radians
%              (yaw, pitch, roll)
%   fov         : a 1x2 vector for the sensor field of view in radians
%                 fov(1) = horizontal field of view
%                 fov(2) = vertical field of view
% OUTPUTS:
%   footprint   : a 5x3 vector of the sensor footprint
%                 row 1 = bottom right corner
%                 row 2 = bottom left corner
%                 row 3 = top left corner
%                 row 4 = top right corner
%                 row 5 = bottom right corner

if nargin==0
    acft_pos    = [-173,00,100];
    acft_att    = [deg2rad(00) deg2rad(00) deg2rad(00)];
```

```
        sens_att    = [deg2rad(00) deg2rad(-49) deg2rad(00)]; % Front Sensor
%       sens_att    = [deg2rad(-90) deg2rad(-39) deg2rad(00)]; % Side Sensor
        fov         = [deg2rad(48),deg2rad(40)]; % h_fov and v_fov
    end

    % Set up the maximum distance allowed for the footprint to reach before
    % cutting it off
    max_dist = inf; % meters

    % Set up the field of view
    h_fov = fov(1);v_fov = fov(2);

    % Create the sensor footprint for a sensor out the front
    footprint = [ 1     +tan(h_fov/2)       +tan(-v_fov/2); % bottom right corner
                  1     -tan(h_fov/2)       +tan(-v_fov/2); % bottom left corner
                  1     -tan(h_fov/2)       +tan(+v_fov/2); % top left corner
                  1     +tan(h_fov/2)       +tan(+v_fov/2); % top right corner
                  1     +tan(h_fov/2)       +tan(-v_fov/2)];% bottom right corner

    % Rotate the sensor footprint
    for jj=1:5
        footprint(jj,:) = angle2dcm(acft_att(1),acft_att(2),acft_att(3))'*...
            angle2dcm(sens_att(1),sens_att(2),sens_att(3))'*footprint(jj,:)';
        % Parametericize the footprint
        t = acft_pos(3)/footprint(jj,3);
        t = acft_pos(3)/abs(footprint(jj,3));
        z = 0;
        y = acft_pos(2)+t*footprint(jj,2);
        x = acft_pos(1)+t*footprint(jj,1);
        if norm([x y z])>max_dist
            theta = atan2(y,x);
            x = max_dist*cos(theta);
            y = max_dist*sin(theta);
        end
        footprint(jj,:)=[x y z];
    end
```

E.5 Ground Speed and Aircraft Heading Code

Listing E.5: MATLAB/windcorr.m

```
function [hdg_a,V_g] = windcorr(V_a,V_w,hdg_w,hdg_g)
% [hdg_a,V_g] = windcorr(V_a,V_w,hdg_w,hdg_g)
%  Provides an estimate of aircraft heading and ground speed to input into
%   a path planning algorithm.  This algorithm calculates the heading angle
%   based on geometric relations and parrallel axis theorem as well as the
%   Law of Sines.  It calculates the ground speed based on vector
%   components.
%
% INPUTS:
%   V_a : airspeed in m/s
%   V_w : wind speed in m/s
%   hdg_w : wind heading in radians, relative to north
%   hdg_g : ground track in radians, relative to north
%
% OUTPUTS:
%   hdg_a : heading, in radians, relative to north
%   V_g : ground speed in m/s

% If passed in wind heading is greater than 360deg, change it to be <360deg
while (hdg_w>2*pi)
    hdg_w = hdg_w-(2*pi);
end

% If passed in ground track is greater than 360deg, change it to be <360deg
while (hdg_g>2*pi)
    hdg_g = hdg_g-(2*pi);
end

% Solve for the heading
if (hdg_w > hdg_g)
    hdg_a = hdg_g - asin((V_w/V_a)*sin(hdg_w - hdg_g));
else
    hdg_a = hdg_g - 2*pi() + asin((V_w/V_a)*sin(hdg_g - hdg_w));
end
```

```
35
   % Convert vectors into east and north components
   V_a_N = V_a*cos(hdg_a);
   V_a_E = V_a*sin(hdg_a);
   V_w_N = V_w*cos(hdg_w);
40 V_w_E = V_w*sin(hdg_w);
   % Find ground speed vector components
   V_g_N = V_a_N + V_w_N;
   V_g_E = V_a_E + V_w_E;
   % Convert components into a vector
45 V_g = sqrt(V_g_N^2 + V_g_E^2);

   % If the output is less than 0deg, change it to be >0deg and <360deg
   if hdg_a < 0
       hdg_a = 2*pi+hdg_a;
50 end

   if isnan(hdg_a)
       hdg_a = 0;
   end
```

E.6 Ground Speed and Track Code

Listing E.6: MATLAB/gnd.m

```
 1 function [V_g,hdg_g] = gnd(V_a,hdg_a,V_w, hdg_w)
   % [V_g,hdg_g] = gnd(V_a,hdg_a,V_w, hdg_w)
   %  Provides an estimate of ground speed and ground track to input into a
   %   path planning algorithm.
   %
 6 % INPUTS:
   %   V_a : airspeed in m/s
   %   hdg_a : heading, in radians, relative to north
   %   V_w : wind speed in m/s
   %   hdg_w : wind heading in radians, relative to north
11 %
   % OUTPUTS:
   %   V_g : ground speed in m/s
```

```
%   hdg_g : ground track in radians, relative to north

% If passed in wind heading is greater than 360deg, change it to be <360deg
while (hdg_w>2*pi)
    hdg_w = hdg_w-(2*pi);
end

% If passed in heading is greater than 360deg, change it to be <360deg
while (hdg_a>2*pi)
    hdg_a = hdg_a-(2*pi);
end

% Convert vectors into east and north components
V_a_N = V_a*cos(hdg_a);
V_a_E = V_a*sin(hdg_a);
V_w_N = V_w*cos(hdg_w);
V_w_E = V_w*sin(hdg_w);

% Find wind vector components
V_g_N = V_a_N + V_w_N;
V_g_E = V_a_E + V_w_E;
% Convert components into a vector
V_g = sqrt(V_g_N^2 + V_g_E^2);
hdg_g = atan2(V_g_E,V_g_N);
if hdg_g < 0
    hdg_g = 2*pi+hdg_g;
end

if isnan(hdg_g)
    hdg_g = 0;
end
```

Appendix F. Sample Virtual Cockpit Waypoint File

This flight plan for Virtual Cockpit was for the overflight case with the following parameters:

Table F.1: Sample Overflight Waypoint Flight Plan

POI Location	Lat: 39.34170°, Lon: −86.02290° MSL alt: 216 m
MAV AGL Alt	100 m
Sensor	Front, −49° elevation
Airspeed	20 kts
Look Angle	90°
Wind Heading and Speed	120° at 5 kts

```
# Note all lines beginning with a # are ignored

Takeoff  60.959999  10.288888  75.000000  39.344734  -86.028763  0

Land_Approach  10.288888  5.144444  75.000000  1.543333  30.480000
  3.048000  39.342159  -86.028473  39.343018  -86.029411  39.343018  -86.029411  0

Waypoint  100.000000  10.288889  39.341866  -86.025990 0

Waypoint  100.000000  10.288889  39.341700  -86.023908 0

Waypoint  100.000000  10.288889  39.341534  -86.021827 0
```

Appendix G. MATLAB Plots for Algorithm Development

This appendix includes the plots created and used during algorithm development, as discussed in chapter IV. These plots demonstrated the output from the algorithms and were used to refine how the algorithms worked. The plots are given in sections for each algorithm.

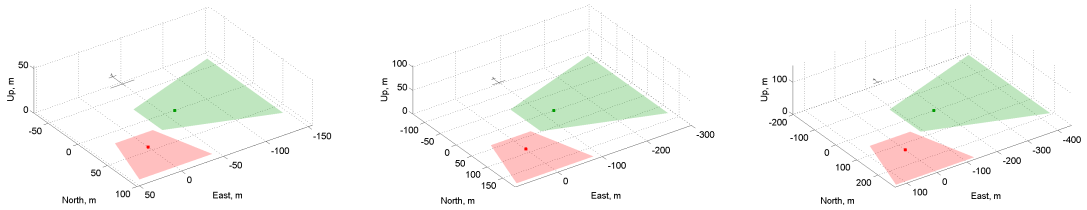
They are given in sections for each algorithm and the purpose they served for the testing of that algorithm.

G.1 Sensor Aimpoints and Footprints

Plotting the sensor aimpoints and footprints confirmed that the algorithms properly calculated where the sensor of the MAV was aimed and what would be in the field of view (FOV). This latter part was considered the sensor footprint on the ground. The aimpoints and footprints changed with altitude and attitude. In the plots below, the MAV is shown as ten times the actual size of the MAV, relative to the scale of the plot, to make it visible in the plots. Each plot shows both the front and the side sensor aimpoints and footprints, in red and green colors, respectively. Additionally, each plot assumes an inertial reference frame in which the MAV, the sensor aimpoints, and sensor footprints move. Each plot is viewed from the same vantage angle so the plots can easily be compared to each other to see how changes in flight altitude and attitude can affect the aimpoints and footprints. All plots assume a flat earth.

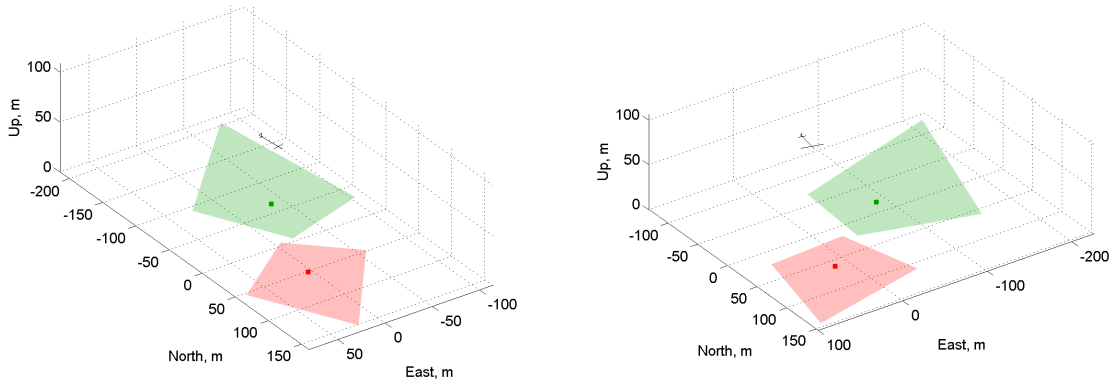
Assuming a level attitude, the altitude changed the projection of the point on the ground relative to the center of the MAV. This is demonstrated in Figure G.1. All three plots look similar, as expected, with the change being in the scale on the axes and in the plotted size of the MAV.

Figure G.2 shows how the aimpoints and footprints are affected by MAV attitude. Again, the MAV is scaled ten times larger than actual size to make it visible in the plot. Yaw, ψ_a, simply rotated the aimpoints and footprints along the earth. Pitch, θ_a, and roll, ϕ_a, each rotated the aimpoints and footprints in the respective axes. When combining both pitch and roll, the footprints take on shapes of elongated, irregular polygons, often where one corner may be extreme distances from the aimpoint.

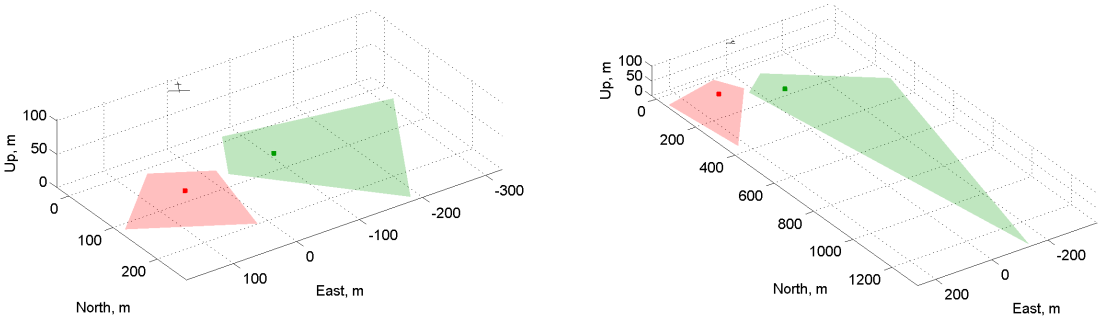


(a) AGL Altitude 50 m (b) AGL Altitude 100 m (c) AGL Altitude 150 m

Figure G.1: Sensor Aimpoints and Footprints as MAV Altitude Changes



(a) ψ_a 330°, θ_a −20°, ϕ_a −20° (b) ψ_a 360°, θ_a −10°, ϕ_a −10°



(c) ψ_a 30°, θ_a 0°, ϕ_a 0° (d) ψ_a 60°, θ_a +10°, ϕ_a +10°

Figure G.2: Sensor Aimpoints and Footprints as MAV Attitude Changes

G.2 Orbit Algorithm Analysis Plots

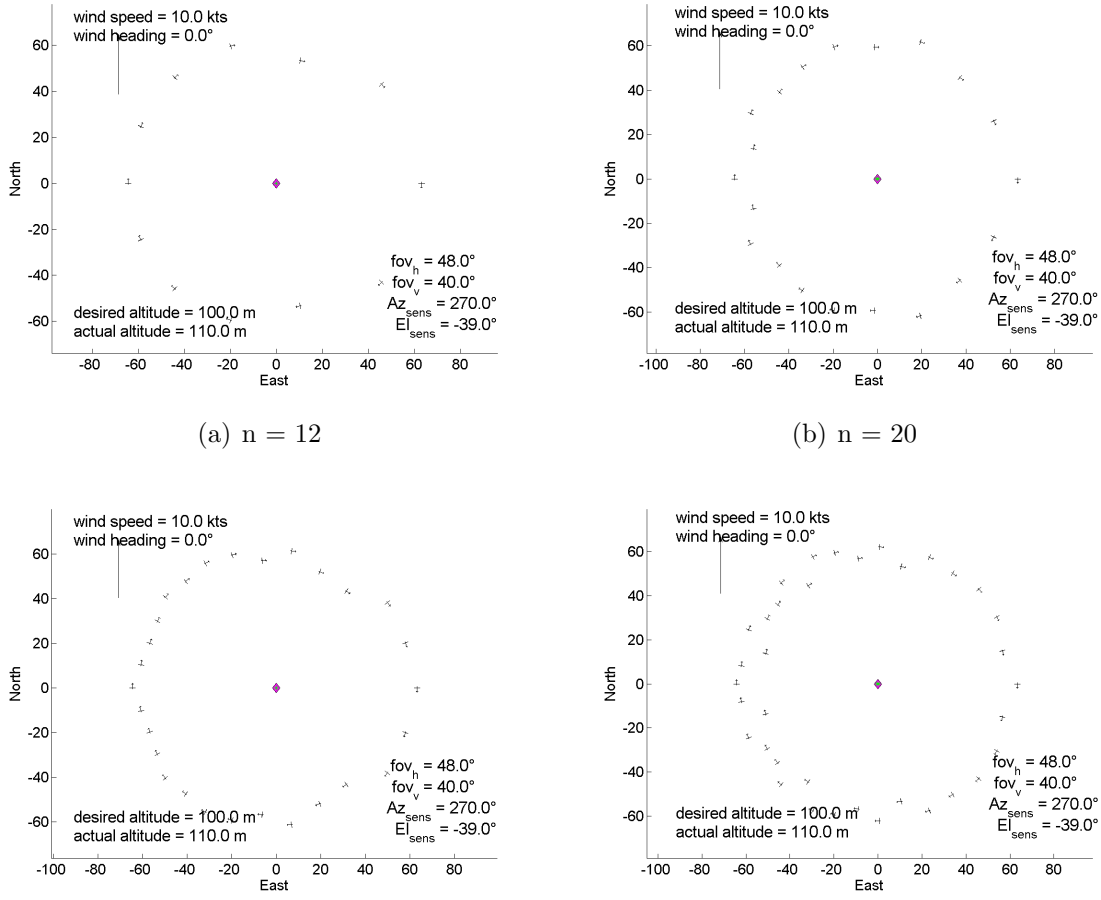
Static plots were used during the development of the orbit algorithm. These plots helped determine what happened as the number of desired waypoints in the orbit changed as well as how the orbit waypoints changed as wind varied. The MAV is depicted in the orbit algorithm plots as real size, relative to the scale of the plot. Larger size often cluttered the image. Additionally, the orientation was usually easily understood, so seeing a larger icon for the MAV provided little benefit. While a three-dimensional plot could show the bank of the MAV at each point, the plot became too cluttered in that view, so an overhead perspective was used to understand the orbit path. Pertinent wind, sensor, and altitude data is presented on each plot to demonstrate how data played an important role in the creation of the plot.

All plots shown for the orbit algorithm analysis used the SIG Rascal as the test bed. MAV parameters that were used as inputs to the orbit algorithm are not listed on the plots, as they cluttered the plots. The parameters included desired airspeed, V_a = 40 kts, stall speed, V_s = 20 kts, and maximum bank angle, $\phi_{max} = 40°$. The MAV is plotted at each waypoint to show where that waypoint was located.

Figure G.3 shows how the orbit course plan changes based on the number of waypoints. The number of waypoints actually made the orbit path clearer as the number increased. The same wind and desired altitude were used for all evaluations as the number of waypoints changed. The wind was 10 kts with a heading of due north. The desired altitude was 100 m AGL, but the algorithm output an altitude of 110m AGL, for reasons discussed in section 3.5.4. The number of waypoints were varied from 12 to 36 in 4 increments of 8 waypoints each. A greater number of waypoints took longer for the orbit algorithm to calculate a solution, but even at 36 waypoints, it took less than 0.15 s to compute a solution. It was also known that a higher number of waypoints would take longer to transmit through the test equipment flight management system to the MAV, but it was thought that a higher number of waypoints would yield a better solution.

One outcome was the appearance of corners in the solution, particularly in plots with a high number of waypoints. These corners were occasions where the orbit course

solution bent in sharply. The reason for these corners was the unique combination of ground speed at that heading and the bank angle required to maintain the turn at that speed while keeping the POI centered in the FOV. The MAV would most likely not be able to actually hit each of these corners without overshooting some other portion of the planned course, but moving towards the corners would be more likely to keep the POI close to the FOV as it transitioned to the next waypoint. Based on the outcome of these plots, it was determined to use the most waypoints that could reasonably be transmitted to the MAV.



(a) n = 12

(b) n = 20

(c) n = 28

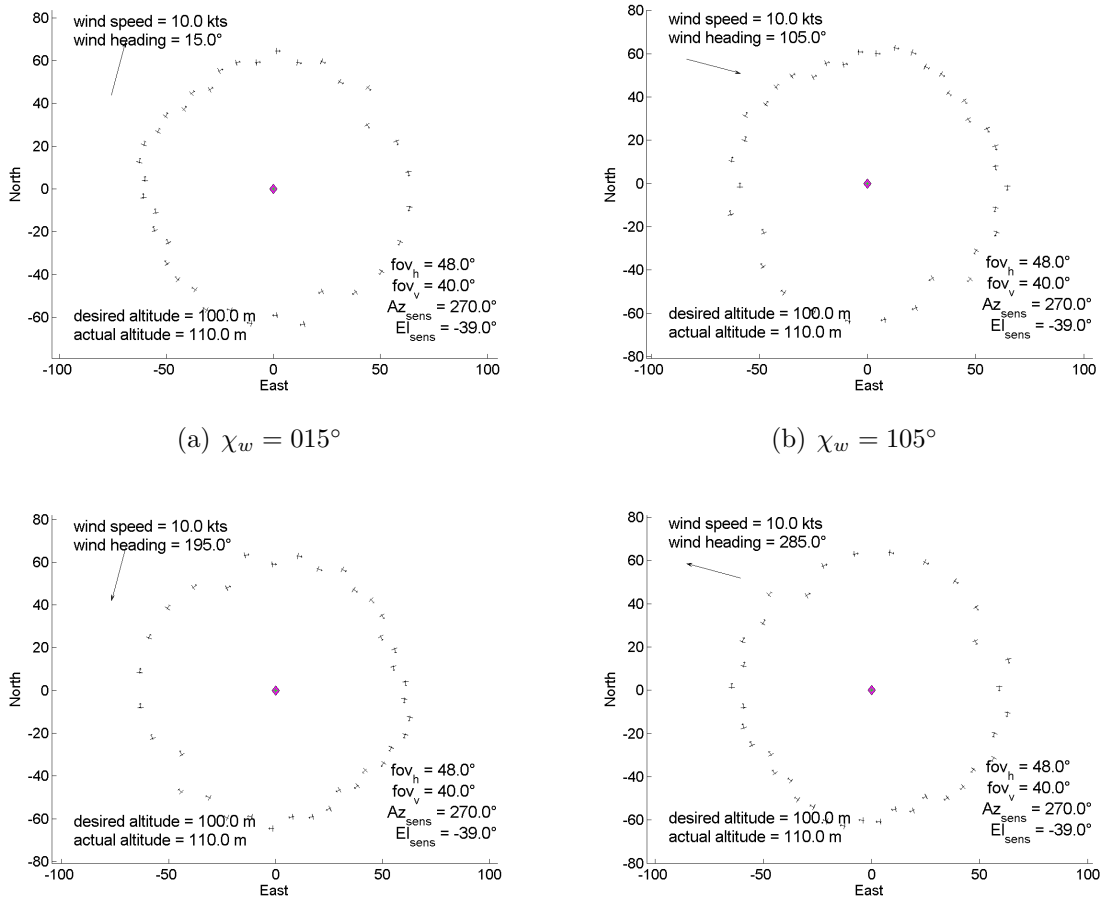
(d) n = 36

Figure G.3: Orbit Paths as Number of Waypoints Change

For the remainder of the plots, 36 waypoints were used for the orbit calculations because the plots showed the solution was better with the higher number of waypoints. Since the algorithm actually takes the number of waypoints in as a parameter, this could

be changed in the testing or operational environments, if needed, and the time to generate plots was minimal.

Changes based on wind were for direction or speed. Changes based on wind direction should simply rotate the orbit path as the wind heading changed. Figure G.4 confirmed this presumption. The wind heading was started at 15° and increased in 90° increments, while the wind speed was fixed at 10 kts.



(a) $\chi_w = 015°$

(b) $\chi_w = 105°$

(c) $\chi_w = 195°$

(d) $\chi_w = 285°$

Figure G.4: Orbit Paths for Different Wind Headings

Changes based on wind speed could change altitude, as discussed in section 3.5.4. The plots confirm that as wind speed increased, the altitude increased to maintain the bank angle for the turn and to keep the POI centered in the FOV. The wind heading was fixed at 15°, while the speed started at 0 kts and increased to 25 kts in 5 kts increments. The algorithm could calculate orbits for winds higher than 25 kts, but the

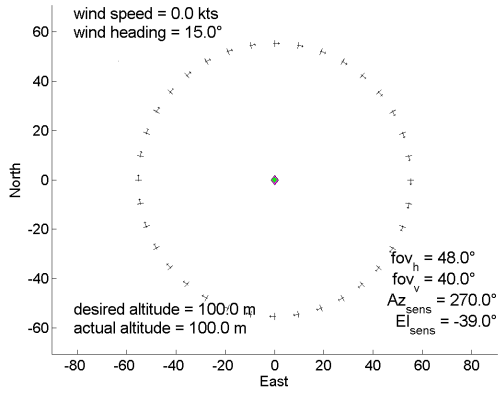
static plots demonstrated the output of the algorithm sufficiently, and the SIG Rascal most likely would not be able to fly in winds of that magnitude. As the wind speed increased, corners were also observed in the orbit waypoint plots. These illustrate how the MAV will react with a head wind, tail wind, and cross wind. Since the orbit algorithm calculates waypoints by varying ground track angle, a greater number of waypoints occur where the MAV has a headwind. This is due to the changing heading of the MAV, but not being able to move very much in the increased wind. The opposite occurs with a tail wind.

G.3 Overflight Algorithm Analysis Plots

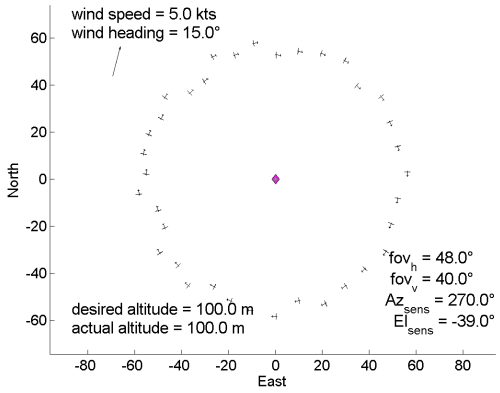
The output from the overflight algorithm was evaluated to determine if the sensor would be properly aimed at the POI. This was checked for both the front and side sensor orientations, in a variety of look angles, and in different winds. The MAV is depicted at five times its actual size to better illustrate the heading of the MAV during the course. The projected ground path is displayed with a dashed green line which helps to illustrate how the heading of the MAV differed from ground track. Each plot also has pertinent wind, sensor, and MAV information. The overflight plots were only considered from an overhead perspective because steady and level flight was assumed for the algorithm.

The first set of plots shown in Figure G.6 illustrate how the overflight course varied with look angle, χ_L, given a set wind and sensor. Four look angles in the cardinal directions were evaluated to demonstrate that the course simply rotated by 90°.

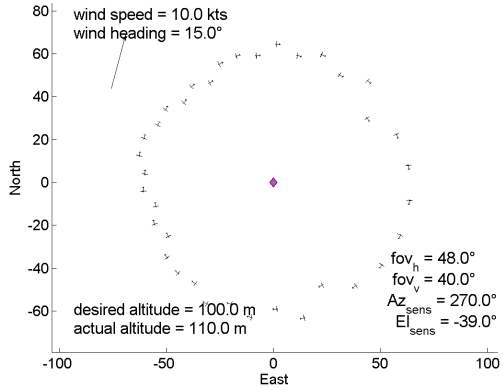
The overflight path changed based on sensor orientation, but the change was simply a rotation and offset in the reference frame. Changes based on wind, therefore, would affect the overflight path the same without dependence on the sensor orientation. Figures G.8 and G.9 respectively show how the overflight course changed based on wind heading and wind speed. As a reminder, wind heading was considered the direction the wind was headed towards, in the same manner as aircraft heading, despite the common aviation method of defining winds as the direction from which they came.



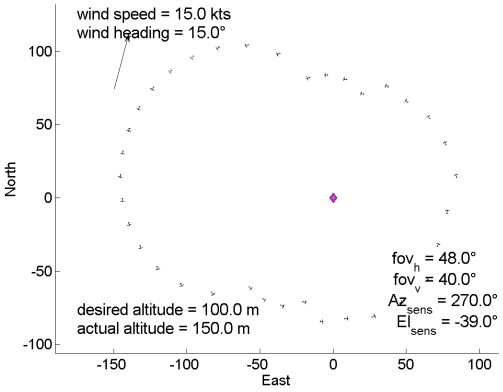
(a) $V_w = 0$ kts



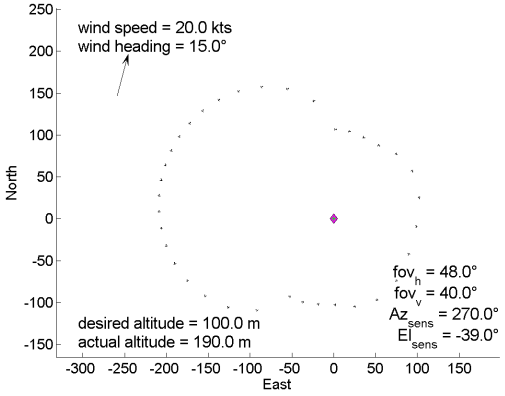
(b) $V_w = 05$ kts



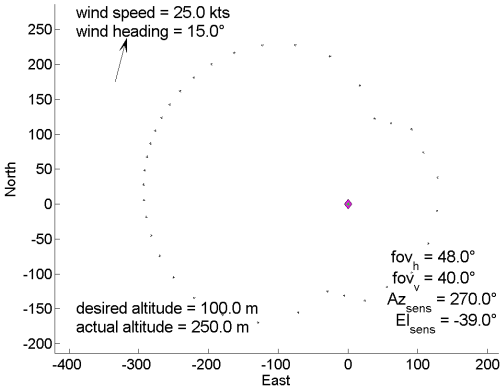
(c) $V_w = 10$ kts



(d) $V_w = 15$ kts

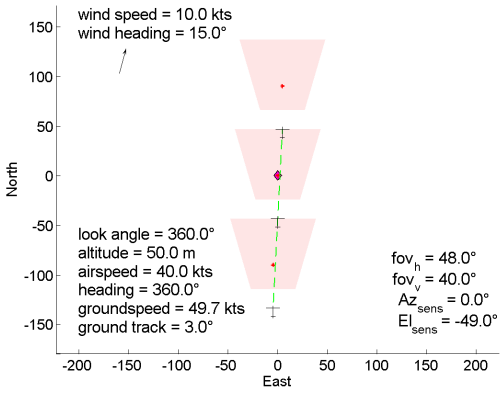


(e) $V_w = 20$ kts

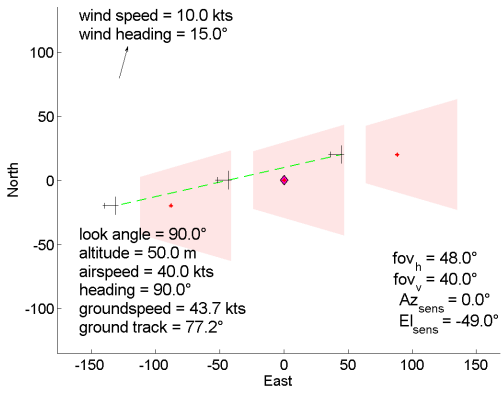


(f) $V_w = 25$ kts

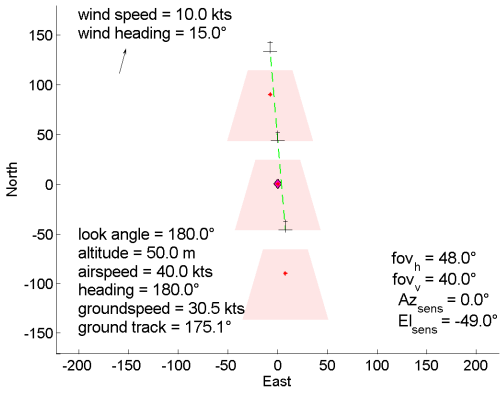
Figure G.5: Orbit Paths for Different Wind Speeds



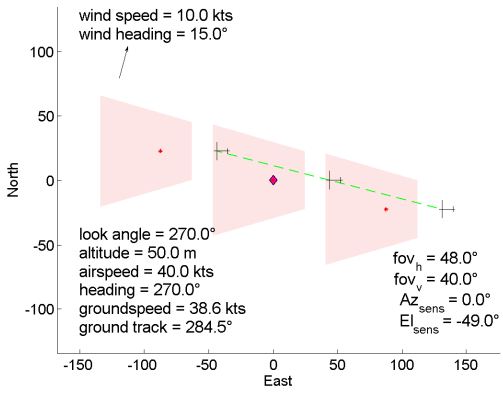
(a) $\chi_L = 360°$



(b) $\chi_L = 90°$

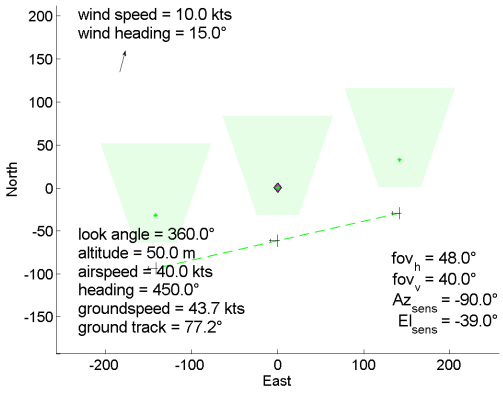


(c) $\chi_L = 180°$

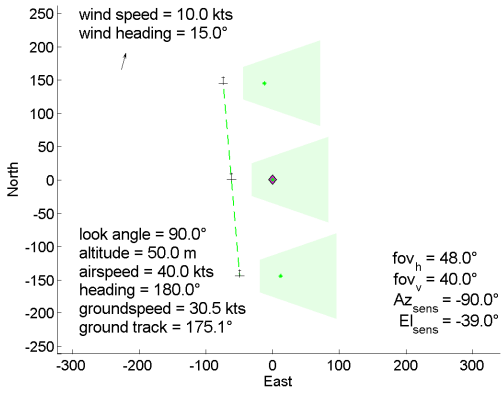


(d) $\chi_L = 270°$

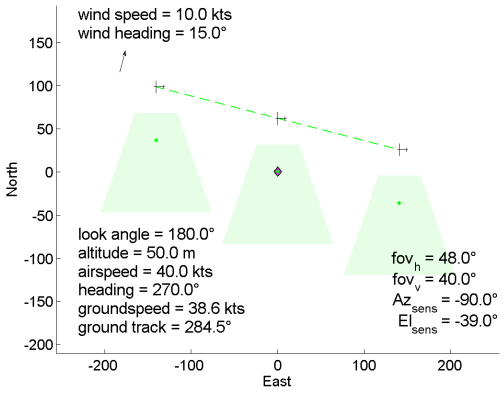
Figure G.6: Overflight Course Path for Front Sensor at Different Look Angles



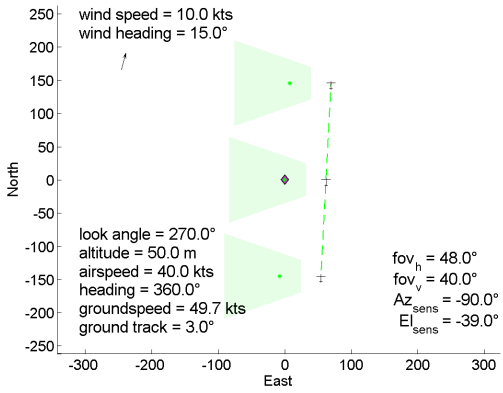
(a) $\chi_L = 360°$



(b) $\chi_L = 90°$



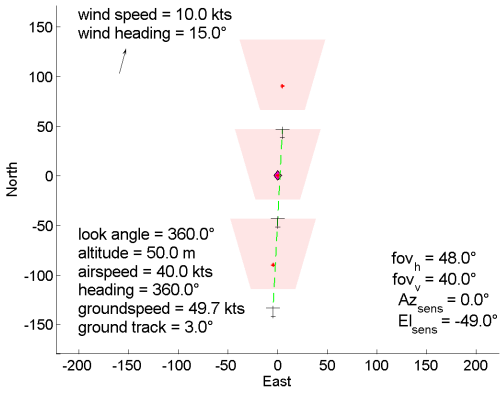
(c) $\chi_L = 180°$



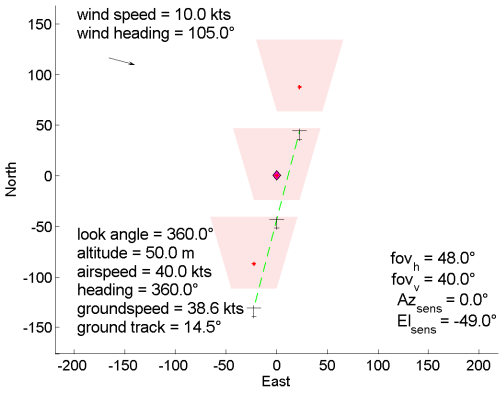
(d) $\chi_L = 270°$

Figure G.7: Overflight Course Path for Side Sensor at Different Look Angles

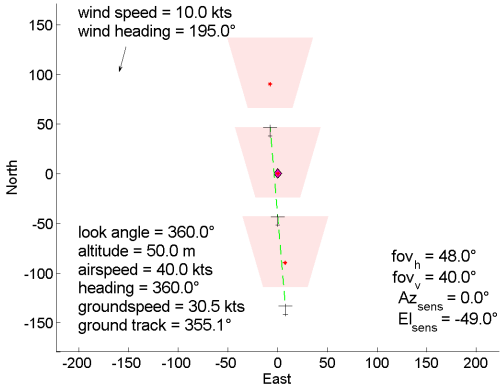
Wind speed was fixed at 10 kts and χ_w was changed by 90° increments for the evaluation of course changes based on changes in wind heading. This demonstrated how the course was affected with wind from all directions, starting from a 15° wind heading.



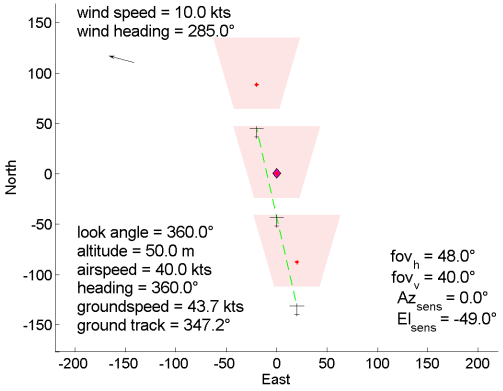
(a) $\chi_w = 015°$



(b) $\chi_w = 105°$



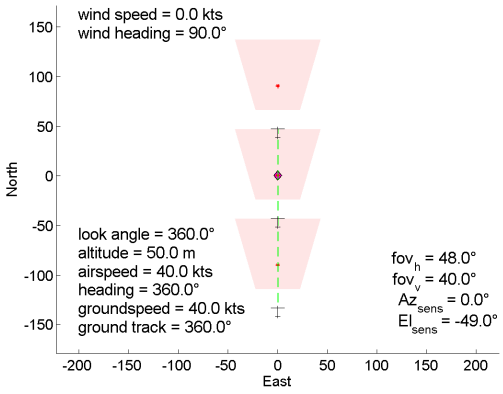
(c) $\chi_w = 195°$



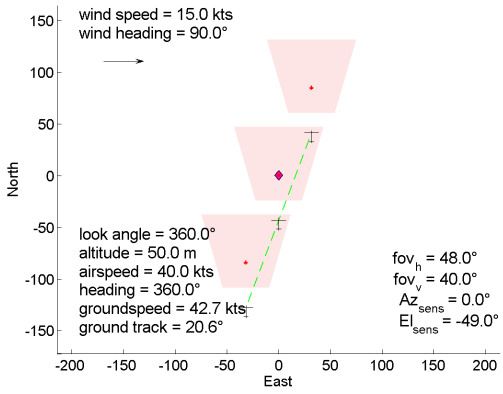
(d) $\chi_w = 285°$

Figure G.8: Overflight Course Path for Front Sensor at Different Wind Headings

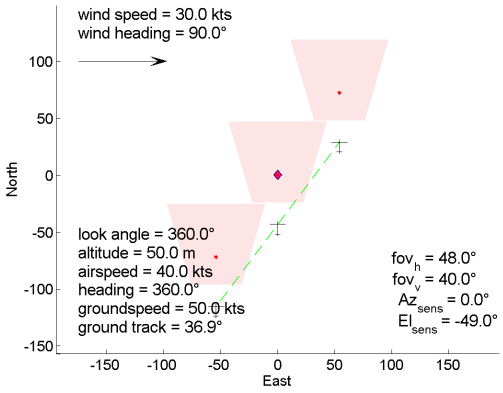
To evaluate changes based on wind speed, wind heading was fixed at 90° and V_w was varied by 15 kts increments, starting at 0kts. While the algorithm calculated a solution at 30 kts and 45 kts of wind speed, the likelihood of the MAV flying that was in serious doubt. These high winds, however, did show how the path would angle much better than the lower winds.



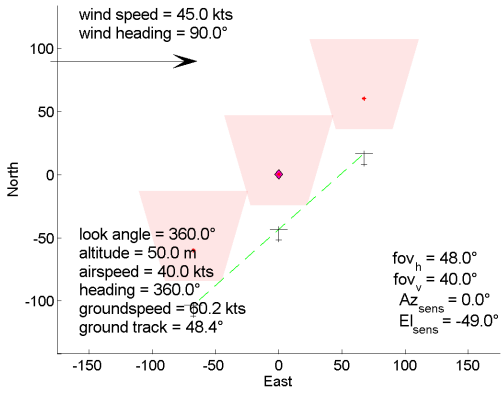
(a) $V_w = 0$kts



(b) $V_w = 15$kts



(c) $V_w = 30$kts



(d) $V_w = 45$kts

Figure G.9: Overflight Course Path for Front Sensor at Different Wind Speeds

G.4 Conclusion

These plots helped with the development of the algorithms, allowing the code to be refined before moving into more rigorous testing. The plots gave static displays of what the output from the algorithms would be to the flight management software. The algorithms were naturally modified in subsequent testing, but these plots still served a purpose in quickly demonstrating what would happen with the new modifications.

Bibliography

Abbott, L., C. Phillips, C. Stillings, and G. Knowlan (2007), Systems Engineering Analysis for Transition of the Fleeting Target Technology Demonstrator, Master's thesis, Air Force Institute of Technology, 2950 Hobson Way, WPAFB, OH, 45433-7765.

Booth, C. (2009), Surveillance Using Multiple Unmanned Aerial Vehicles, Master's thesis, Air Force Institute of Technology, 2950 Hobson Way, WPAFB, OH, 45433-7765.

Bosley, J. (2008), E-mail Correspondance Regarding Wind Estimation by the Kestrel Autopilot.

Burns, B. (2007), Autonomous Unmanned Aerial Vehicle Rendezvous for Automated Aerial Refueling, Master's thesis, Air Force Institute of Technology, 2950 Hobson Way, WPAFB, OH, 45433-7765.

Caveney, D., and J. K. Hedrick (2005), Path planning for targets in close proximity with a bounded turn-rate aircraft, *AIAA Guidance, Navigation, and Control Conference and Exhibit*.

Crouse, J. (2007), Development of Cursor-On-Target Control for Semi-Autonomous Unmanned Aircraft Systems, Master's thesis, Air Force Institute of Technology, 2950 Hobson Way, WPAFB, OH, 45433-7765.

DeLuca, A. M. (2004), Experimental Investigation into the Aerodynamic Performance of Both Rigid and Flexible Wing Structured Micro-Air-Vehicles, Master's thesis, Air Force Institute of Technology, 2950 Hobson Way, WPAFB, OH, 45433-7765.

Diamond, T., A. Rutherford, and B. Taylor (2009), Cooperative Unmanned Aerial Surveillance Control System Architecture, Master's thesis, Air Force Institute of Technology, 2950 Hobson Way, WPAFB, OH, 45433-7765.

Dubins, L. E. (1957), On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents, *American Journal of Mathematics*, *79*(3), 497–516.

Gates, R. M., Secretary of Defense (2008), Remarks to Air War College (Maxwell, AL).

Geiger, B. R., J. F. Horn, A. M. DeLullo, and L. N. Long (2006), Optimal Path Planning of UAVs Using Direct Collocation with Nonlinear Programming, *AIAA Guidance, Navigation, and Control Conference and Exhibit*.

Jodeh, N. (2006), Development of Autonomous Unmanned Aerial Vehicle Research Platform: Modeling, Simulating, and Flight Testing, Master's thesis, Air Force Institute of Technology, 2950 Hobson Way, WPAFB, OH, 45433-7765.

Kehoe, J., A. Watkins, and R. Lind (2007), Trajectory Generation for Effective Sensing, *AIAA Guidance, Navigation, and Control Conference and Exhibit*.

McGee, T. G., S. Spry, and J. K. Hedrick (2005), Optimal Path Planning in a Constant Wind with a Bounded Turning Rate, *AIAA Guidance, Navigation, and Control Conference and Exhibit*.

Office of the Secretary of Defense (2005), Unmanned Systems Roadmap 2005-2030.

Office of the Secretary of Defense (2007), Unmanned Systems Roadmap 2007-2032.

Osborne, J., and R. Rysdyk (2005), Waypoint Guidance for Small UAVs in Wind, *AIAA Infotech@Aerospace*, pp. 1–12.

Pachter, M., N. Ceccarelli, and P. R. Chandler (2008), Estimating MAV's Heading and the Wind Speed and Direction Using GPS, Inertial, and Air Speed Measurements, *AIAA*.

Procerus Technologies (2008), *Kestrel User Guide*, 478 South Geneva Road, Vineyard, UT 84058, document version 1.95 ed.

Robinson, B. (2006), An Investigation into Robust Wind Correction Algorithms for Off-the-Shelf Unmanned Aerial Vehicle Autopilots, Master's thesis, Air Force Institute of Technology, 2950 Hobson Way, WPAFB, OH, 45433-7765.

Rysdyk, R. (2003), UAV Path Following for Constant Line-of-Sight, *2nd AIAA "Unmanned Unlimited" Systems, Technologies, and Operations*.

Rysdyk, R. (2006), Unmanned Aerial Vehicle Path Following for Target Observation in Wind, *Journal of Guidance, Control, and Dynamics*, *29*(5), 1092–1100.

Rysdyk, R. (2007), Course and Heading Changes in Significant Wind, *Journal of Guidance, Control, and Dynamics*, *30*(4), 1168–1171.

Sakryd, G., and D. Ericson (2008), Systems Engineering Analysis for the Development of the Fleeting Target Technology Demonstrator, Master's thesis, Air Force Institute of Technology, 2950 Hobson Way, WPAFB, OH, 45433-7765.

Terning, N. (2008), Real-Time Flight Path Optimization for Tracking Stop-and-Go Targets with Micro Air Vehicles, Master's thesis, Air Force Institute of Technology, 2950 Hobson Way, WPAFB, OH, 45433-7765.

Vantrease, T. (2008), Development and Employment of a Semi-Autonomous Cursor on Target Navigation System for Micro Air Vehicles, Master's thesis, Air Force Institute of Technology, 2950 Hobson Way, WPAFB, OH, 45433-7765.

Yokoyama, N., and Y. Ochi (2008), Optimal Path Planning for Skid-to-Turn Unmanned Aerial Vehicle, *AIAA Guidance, Navigation, and Control Conference and Exhibit*.

Zollars, M. (2007), Optimal Wind Corrected Flight Path Planning for Autonomous Micro Air Vehicles, Master's thesis, Air Force Institute of Technology, 2950 Hobson Way, WPAFB, OH, 45433-7765.

Vita

A 2002 graduate of the University of Minnesota–Twin Cities, Capt Shannon Farrell was commissioned through the Air Force Reserve Officer Training program. He reported to Robins Air Force Base as an F-15 Avionics Engineer in the F-15 System Support Management Directorate, where he was responsible for the reliability and maintainability for four key avionics systems for all variants of the air superiority fighter. In July 2004, Capt Farrell joined the Warner Robins Center Test Authority as a Systems Test Engineer, where he managed all aspects of C-130 and C-5 developmental and qualification test and evaluation programs in support of aircraft sustainment and modifications.

In January 2007, he took over as the Engineering Directorate Executive Officer, Warner Robins Air Logistics Center (WR-ALC), Robins Air Force Base, Georgia. As Executive Officer, he supported a Senior Executive Service member and an Air Force Colonel as they led 1,300 engineers conducting four billion dollars of sustainment engineering efforts at the Air Logistics Center. Capt Farrell also served as Chief, Aircraft Battle Damage Repair Engineer for nearly 40 military structural engineers assigned to Robins AFB. Capt Farrell acted as Deputy Functional Area Manager for over 70 military engineers assigned to WR-ALC, as well as the Officer in Charge of the WR-ALC Engineer Replacement Training Unit.

In August 2007, he entered the aeronautical engineering program at the Air Force Institute of Technology, Wright Patterson Air Force Base, Ohio. Upon graduation, he will be assigned to Air Force Research Laboratory, Munitions Directorate, Eglin Air Force Base, Florida.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
13-03-2009	Master's Thesis	Mar 2008 - Mar 2009

4. TITLE AND SUBTITLE
WAYPOINT GENERATION BASED ON SENSOR AIMPOINT

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Shannon M. Farrell, Capt, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GAE/ENV/09-M01

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AFRL/RW, Dr. Robert Murphy, 101W Eglin Blvd, Eglin AFB, FL, 32542

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 research develops waypoint generation algorithms required to keep a point of interest (POI) in the field of view (FOV) of a fixed sensor on a micro air vehicle (MAV), in the presence of a constant wind. The waypoint generation algorithms establish waypoints such that a fixed sensor on a MAV at that waypoint and assumed attitude would be pointed directly at the POI. Two scenarios are explored: one where the MAV orbits the POI, and the second where the MAV flies to align the sensor at the POI along a preferred look angle. The first scenario relies on a sensor aimed out the side of the MAV, while the second considers both a side-viewing sensor and a forward-viewing sensor. The research explored each scenario through hardware-in-the-loop testing as well as flight testing. The results indicate the algorithms keep the POI in the FOV at least two-thirds of the time as long as the wind speed was less than one-quarter of the cruise speed; at higher winds, the MAV has a difficult time keeping to the flight path generated by the sensor aimpoint algorithm.

15. SUBJECT TERMS
waypoint generation, MAV, UAV navigation, outer-loop control, fixed sensor

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. David R. Jacques
U	U	U	UU	173	19b. TELEPHONE NUMBER *(Include area code)* (937) 255-3355x3329; David.Jacques@afit.edu