IMPROVING UNMANNED AERIAL VEHICLE FORMATION FLIGHT AND SWARM COHESION BY USING COMMERCIAL OFF THE SHELF SONAR SENSORS

THESIS

Robert L. McClanahan, Captain, USAF

AFIT-ENV-MS-17-M-202

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENV-MS-17-M-202

IMPROVING UNMANNED AERIAL VEHICLE FORMATION FLIGHT AND SWARM COHESION BY USING COMMERCIAL OFF THE SHELF SONAR SENSORS

THESIS

Presented to the Faculty

Department of Systems Engineering and Management

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 Systems Engineering

Robert L. McClanahan, BS

Captain, USAF

March 2017

AFIT-ENV-MS-17-M-202

IMPROVING UNMANNED AERIAL VEHICLE FORMATION FLIGHT AND SWARM COHESION BY USING COMMERCIAL OFF THE SHELF SONAR SENSORS

Robert L. McClanahan, BS

Captain, USAF

Committee Membership:

Dr. David Jacques, PhD
Chair

Dr. John Colombi, PhD
Member

Lt Col Amy Cox, PhD
Member

Abstract

Unmanned Aerial Vehicle (UAV) formation flight and swarming are active areas of research within the Department of Defense (DoD). These areas of study cover multiple engineering disciplines; from mechanical and aeronautical to computer science and human factors. The current use of low cost commercial off the shelf (COTS) components to architect UAV formation flights results in insufficient position accuracy of the UAVs in the formation. Latency in communication between autonomous vehicles degrades formation cohesion. This research aims to demonstrate the position error of formation flights decreases by using onboard sonar sensors to accurately measure the distance the follower UAV is from the leader UAV. The sensor enables the follower UAV to appropriately and quickly respond to errors in position by adjusting the followers velocity. The UAV architecture, using onboard sensors, demonstrated tighter formation cohesion by measuring the average position error during multiple flight tests of UAVs and compare these results with previous formation flight tests that did not utilize onboard sensors. The previous flight tests, used the same guided position algorithm, the same X-8 airframes, but no onboard sensor for real time distance error measurement. Since the previous flight test had a similar configuration. This assessment shows how position error was effected by incorporating the sonar sensor. This research effort was able to reduce the Root Mean Square Deviation (RMSD) by 37.3% and the average position error by 70.9% when compared to the previous flight tests without sonar.

Acknowledgments

I would like to express my sincere appreciation and gratitude to my faculty advisor, Dr. David Jacques, for not only his guidance, but his willingness to let loose of the reigns and let me run. This experience has been not only an academic challenge, but an absolute blast thanks to his leadership and support. I would also like to thank Mr. Rick Patton for always being ready to pounce and lend an eager, encouraging hand whenever something was asked of him or if I was just trying to pick his brain for his wealth of knowledge and experience. In addition, I would like to thank Jeremy Gray for all his help and guidance. Jeremy's expertise in writing python scripts and interfacing with the Pixhawk was instrumental to the success of this research effort. Every interaction with Dr. Jacques, Rick, or Jeremy was always met with a handshake, a smile, and always a few laughs. I could not have asked for a more positive experience in graduate school and I know I have made a few more friends along the way.

Finally, I would like to thank my loving wife for her unwavering support. Without her hard work and dedication to the family none of this would have been possible. She allowed me to spend countless hours locked in the basement, struggling, but determined not to let Python get the better of me. I love you Lauren and thank you!

Robert L. McClanahan, Capt, USAF

Table of Contents

Page

List of Figures

Page

List of Tables

Page

IMPROVING UNMANNED AERIAL VEHICLE FORMATION FLIGHT AND SWARM COHESION BY USING COMMERCIAL OFF THE SHELF SONAR SENSORS

I. Introduction

1.1 Background

Unmanned Aerial Vehicle (UAV) formation flight and swarming are active areas of research within the Department of Defense (DoD). These areas of study cover multiple engineering disciplines; from mechanical and aeronautical to computer science and human factors. The Strategic Capabilities Office (SCO) recently unclassified information on their UAV swarm test and the SCO's proposed fiscal year 2017 (FY17) budget has doubled from FY16 to $902 million in FY17 [1]. This budget increase shows the resources which the DoD is dedicating to UAV swarm and formation flight technology.

There are numerous instances where close proximity flight is advantageous, to include, flying in a space constrained environment, controlling radar signature, and even electronic warfare beam shaping. Formation flight, like that of geese, would allow the UAVs to fly more efficiently, by dividing the induced aerodynamic drag among the formation [2]. A UAV swarm is a group of UAVs working together to accomplish a common goal. Multiple UAVs working together to accomplish a task can be extremely efficient in various scenarios. These scenarios include Search, Identify, Engage and Assess (SIEA) missions, area mapping, and even air refueling [3]. The DoD has an invested interest in all of these swarm areas. One could imagine military scenarios that would fall into each of these categories. Using teamwork, the swarm can efficiently and more effectively accomplish the mission. UAV swarms can provide strength in numbers

and quickly overrun the enemy, much like a swarm of locust attacking a crop. According to the Air Force Chief Scientist, Gregory Zacharias, “Groups of coordinated small drones could also be used to confuse enemy radar systems and overwhelm advanced enemy air defenses by providing so many targets that they cannot be dealt with all at once” [4]. The enemy may be able to see the swarm coming, but there is nothing they can do about it, as there is no way to engage all of the UAVs.

Controlling a UAV formation flight or swarm presents many challenges. These challenges range from organization of the formation or swarm, task allocation, inter-swarm communication, and operator work load. Many control algorithms have been developed and are constantly evolving to address the numerous issues with swarming behavior. Algorithms are designed with a certain mission in mind. The final swarm algorithm solution should be dynamic, allow various mission types, and even have the ability to switch mission type during execution.

A single operator cannot have situational awareness of the health and status of all the UAVs in a formation flight or swarm, and it obviously is not cost effective to have numerous operators controlling a swarm, with each operator responsible for a couple of the UAVs in the swarm. Therefore, some inherent risk exists in formation flight and swarm systems since the operator cannot have full situational awareness of each aircraft.

1.2 Problem

The current use of low cost commercial off the shelf (COTS) components to architect UAV formation flights results in insufficient position accuracy of the UAVs in the formation. Latency in communication between autonomous vehicles degrades

formation cohesion. Current COTS open-air formation architectures have separation variances of approximately 10m with an average position error of 3.10m during formation quadrotor test flights [5]. These vast position errors degrade and eliminate the swarm's ability to accomplish the mission and remove the aerodynamic or signature control advantages gained by the formation flight.

1.3 Objective

Demonstrate that the use of onboard sonar sensors reduces the desired position error and by doing so, allows the commanded vehicle to vehicle offset or separation distance to be reduced. This will be accomplished by using onboard sonar sensors to both accurately measure the distance the follower UAV is from the leader UAV and enable the follower UAV to appropriately and quickly respond to errors in position. The UAV architecture, using onboard sensors, will demonstrate tighter formation cohesion by measuring the average position error during multiple flight tests of UAVs and compare these results with previous formation flight tests, without using onboard sensors. The previous formation flight tests and data collected by Gray will be used for direct comparison, since his flight tests utilized the identical airframe and components minus the sonar sensor [5].

1.4 Justification

In the current fiscal environment, it is important to remain cost conscious and to design systems that allow the greatest flexibility in the operational environment. With the reduction in manpower, the Air Force must come up with new ways of multiplying the force. According to the former Air Force Chief of Staff, Gen Mark Welsh, "Virtually

every mission area faces critical manning shortages… we have got to figure out different ways of using our people in a more efficient way or we will wear them out. And if we lose them, we lose everything" [6]. The Air Force is currently 41% smaller than it was during the first gulf war, but has increasing responsibilities [6].

UAV swarms allow a group of low cost, autonomous vehicles to work together as a cohesive unit to accomplish a single task or multiple tasks. Using this teamwork, the swarm can efficiently and more effectively accomplish the mission. According to the Department of Defense's (DoD) Unmanned Systems Integrated Roadmap, "Operating in swarms of 'intelligent munitions' weapons…can autonomously search for and destroy critical mobile targets while aiming over a wide combat area" [7]. The DOD has realized the importance of groups of unmanned systems cooperatively working together towards a common goal and plans to incorporate this technology into future systems [7].

1.5 Scope

This research is built off of Jeremy Gray's thesis, *Design and Implementation of a Unified Command and Control Architecture for Multiple Cooperative Unmanned Vehicles Utilizing Commercial Off the Shelf Components,* by adding onboard sonar sensors to the follower vehicle [5]. The addition of the onboard sonar sensors allows the follower UAV to sense the location of the lead vehicle, giving it greater situational awareness, and allowing it to autonomously correct for position errors.

The onboard sensors were limited to a single sonar and the algorithm was only modified to incorporate the addition of the sonar sensor. This allowed the effect of the sonar sensor to be directly compared to the system architecture without the onboard

sonar. Further, all test UAVs were equipped with Pixhawk autopilots, chosen since it is a COTS, low cost, open source, autopilot. The Pixhawk allows for software manipulation through Python scripting, which can be run either directly in the ground station (Mission Planner) graphical user interface (GUI) or through various command prompt based programs (MAVProxy and DroneKit).

All flight tests were limited to only two unmanned vehicles, one leader and one follower. Flight tests were conducted using quadrotor aircraft. Quadrotor UAVs were selected since they allow the test to be slowed down or even paused by commanding the quadrotors to hover in place. The choice of rotary platforms reduced the risk of the vehicles colliding during flight tests.

1.6 Methodology

A literature search was conducted to determine the appropriate type of sensor as well as the sensor model to be used in this effort. A sonar sensor was selected based on the size, weight, power consumption, and the ease of integration with the Pixhawk autopilot. The flight control algorithm developed by Jeremy Gray was modified to be used with DroneKit [5]. The sonar algorithm was run directly on Mission Planner, separate from Gray's algorithm.

A series of flight tests were conducted to analyze the effect of using the onboard sensor during formation flights with two vehicles. The lead UAV flew a series of known flight paths via a way point pattern while the follower UAV was commanded to follow the lead UAV at a set distance. Flight data was collected, including GPS position,

altitude, and airspeed on both aircraft during the duration of all flight test. Additionally, the sonar data was collected on the follower aircraft.

A root mean square error analysis was run on the data to determine the separation distance error. This error was then compared to previous formation flights to determine the effect of the sonar sensor.

1.7 Research Questions

The research questions that this investigation attempts to answer are listed below.

- What onboard sensors are available with low weight and low power consumption for use in UAV swarms?
- What swarm algorithm modifications are required to incorporate onboard sensing?
- How are the onboard sensors integrated with the Pixhawk autopilot?
- How can the Pixhawk autopilot take advantage of the onboard sensor data?
- How much can onboard sensors reduce required offset and desired separation distance errors?

1.8 Materials/Equipment

This research required multiple quadrotor UAVs, multiple 3D Robotics telemetry radios, sonar sensors, flight test approval, and range time to conduct flight testing. The required equipment is outlined in detail below.

The two COTS quadrotors that this research used where the X-8s, manufactured by 3D Robotics. These quadrotors have been used extensively by the Air Force Institute of Technology and are a proven testing platform. A detailed description of these quadrotors can be found in chapter III.

Two Pixhawk autopilots were required for this research, one for each quadrotor. The use of the Pixhawk autopilot allowed for seamless integration with the sonar sensor. A description of the Pixhawk can be found in chapter II, with specific implementation details in chapter III.

Four sonar sensors were tested in this research to determine the sensor that produced the best balance of accuracy, distance, and beam width. All sonar sensors were manufactured by Maxtronics and the sensor model numbers tested were the MB 1202, MB1260, MB1240, and MB 1020. The MB 1202 was selected after all ground testing was completed. A description of the Maxtronics sonar can be found in chapter II and chapter III.

1.9 Thesis Summary

This section describes the other chapters of this thesis. In the next chapter, an extensive literature review was completed to understand the current state of formation flight research and to explore available sensors. Chapter III outlines the system architecture and describes the algorithms used and how they are incorporated into the system. Chapter IV analyzes the results of this research and compares these results to previous flight tests conducted without the use of sonar sensors. Finally, chapter V states the conclusion of this research and recommends future work.

II. Literature Review

2.1 Chapter 2 Overview

The DoD is investing heavily into Unmanned Aerial Vehicle (UAV) formation flight and swarm technology. Swarms of low cost, attritable, autonomous vehicles, working together as a cohesive unit, can more efficiently accomplish a task or mission than a single, high value, asset. With the Air Force being undermanned and with more responsibility than ever, the DoD must push to accomplish its mission more efficiently and at a lower cost [6]. The use of low cost, commercial off the shelf (COTS) components, could achieve this objective. COTS components currently used to architect UAV formation flights results in insufficient position accuracy of the UAVs in the formation. This inaccuracy in position negates many of the positive effects and uses of close formation flying. The literature review outlines previous work, and covers the following subject areas in order to understand the current state of research:

- Swarm and Formation Flight Algorithms
- Onboard Sensors
- Overview of Pixhawk Autopilot and Ground Station

2.2 Swarming and Formation Flight Algorithms

Before we analyze the efficiency of a swarm algorithm we must first understand some basic rules about swarm behavior: "Rules must be drawn together in order for an agent or computer to accomplish a coherent response" [8]. Craig Reynolds is known for developing the three basic laws governing flocking behavior [9]. These rules are collision avoidance, velocity matching, and flock centering [9]. Flocking and formation flight

algorithm research has been conducted at the Air Force Institute of Technology (AFIT) by manipulating these three basic rules. Both Kaiser and Lambach developed algorithms by modifying these basic rules for use in UAV formations flight [10], [11]. In 2007, Nowak, Price, and Lamont from AFIT, expanded on Reynold's original rules and developed a simulation called SWARMFARE to evaluate behaviors of autonomous control [8], [9]. The rules they utilized are:

- *Flat Align – vector align with neighbors*
- *Target Orbit – orbit target at safe distance*
- *Cluster range towards - cohesion*
- *Cluster Range away - separation*
- *Attract – towards center of mass of all targets*
- *Weighted Attract – towards closest target*
- *Target Repel – repel if with 90% of UAV sensor range*
- *Weighted Target Repel – repulsion based on proximity to target*
- *Evade – collision detection and avoidance*
- *Obstacle Avoidance*

By applying different weights to each of these rules, the behavior of the swarm can be modified or altered to achieve the appropriate response for the mission [8].

As shown in the rules, some issues with swarming include the swarm agents getting too close together (cohesion), too far apart (separation) for their sensors, or the creation of an articulation point. An articulation point is most easily described as a single point failure. This occurs when a single agent is the link between two larger masses of the

swarm. If this single agent loses contact with either half of the swarm, then the swarm will be separated. Figure 1 visualizes an articulation point.

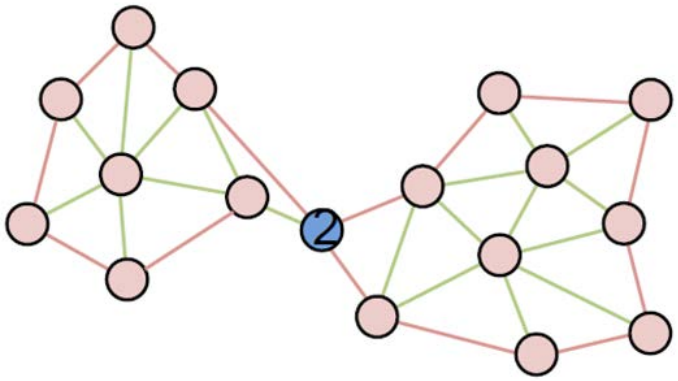

Figure 1. Articulation Point [13]

Seoung and MckLurkin developed a mode switching algorithm that uses network sensing to detect the health of the swarm. This algorithm changes the weights on the rules based on the vulnerability of the swarm. If all is well, the swarm will flock in the same direction (Flat Align). If an articulation point develops, the swarm will cluster to eliminate the Articulation Point. Figure 2 shows the resulting swarm configuration after being in a vulnerable state both with and without mode switching. The three configurations on the left are without mode switching, the three on the right are with mode switching. The mode switching allowed the swarms to recover from a vulnerable state [13].

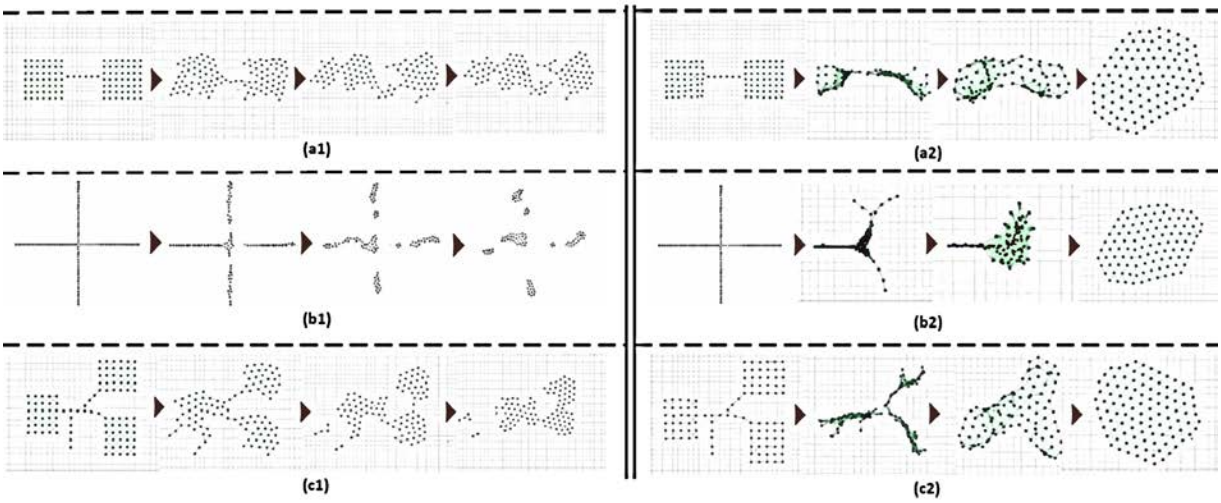

Figure 2.Comparison of swarm in vulnerable states without (left) and with (right) mode switching [13]

Seoung and MckLurkin validated these simulation results using six robots (Figure 3). Again the top row (a1-a6) is without the mode switching algorithm whereas the bottom row (b1-b6) is with the mode switching algorithm [13].

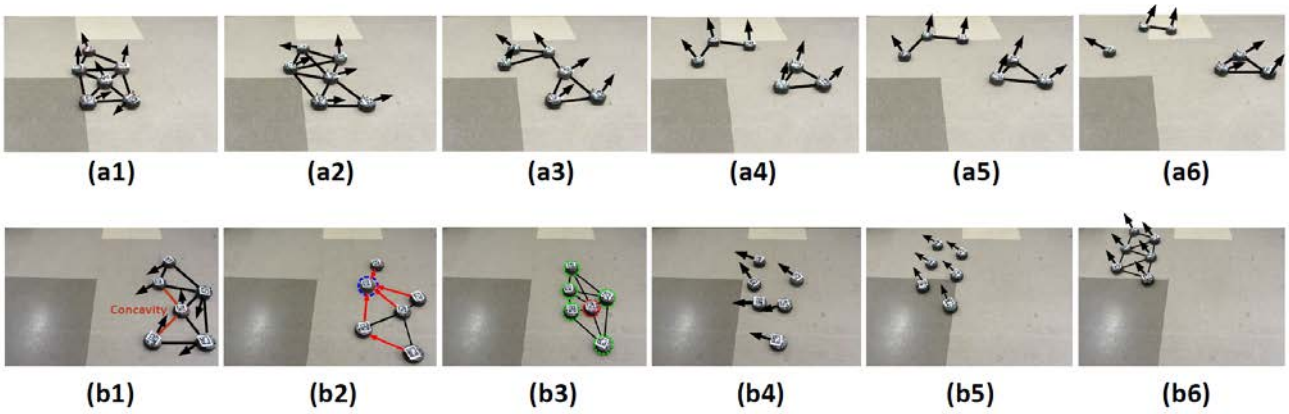

Figure 3. Simulation Validation Using Sic Robots[13]

This work showed the importance of having an algorithm that is dynamic and can change based on the changing environment of the swarm.

A new bio-inspired pigeon algorithm has recently been developed by Hao, Luo, and Duan [3]. The main tools that pigeons use to find their way are maps, compass, and landmarks. They can sense the magnetic field and use the sun to develop a mental map. Pigeons also recognize and remember landmarks to aid in their navigation. One issue that comes up with most swarming algorithms is premature convergence. The swarm can get too clustered together reducing the effectiveness of certain missions. The pigeon algorithm has been modified to reduce the clustering. The modified pigeon inspired algorithm produces a subpopulation of superior position pigeons for the others to track [3]. This modified pigeon algorithm is depicted in Figure 4.

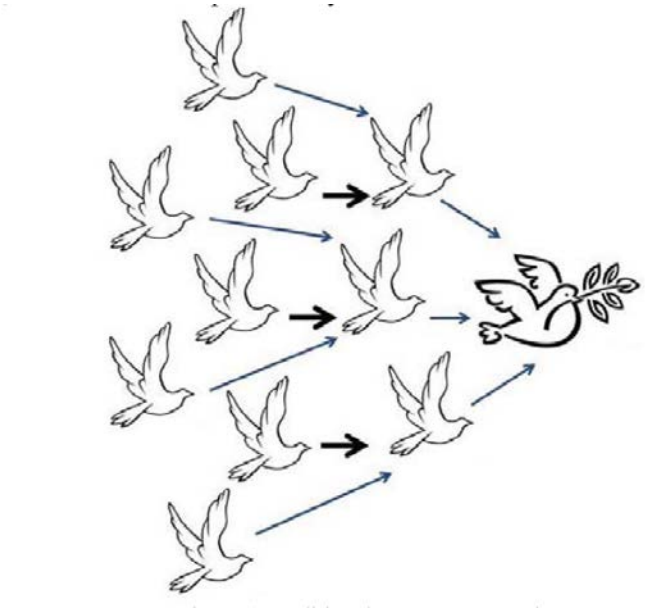

Figure 4. Modified Pigeon Inspired Algorithm [3]

The pigeon algorithm was just one of many algorithms that were studied in this research. There are numerous swarm algorithms in existence and more being continuously developed. To generate the most robust swarm, it is necessary to have an

algorithm that can react to the instantaneous health of the swarm and dynamically change its priorities to keep the swarm operating as one cohesive and effective unit.

Through the present research, it was discovered that the majority of swarm and formation flight algorithms are only simulated, not flight tested, and include major assumptions. There are numerous recent articles on formation flight algorithms [14], [15], [16]. Each article incorporates a different approach to keeping the UAVs aligned in the formation. What these articles lack is real world flight testing. They mostly employ simulated environments to test their formation flight algorithms; the simulations incorporate major assumptions which are not in line with current technology limitations. For instance, in one formation flight conference paper, the assumption was used that all vehicles in the formation could communicate with each other [16]. In another article, it was assumed that all vehicles had "good" [14] onboard computing power. These formation flight architectures and algorithms were all impressive, just not validated with real world flight test.

With the numerous swarm and close formation flight algorithms being developed and simulated, only one seems to be flight tested. While other algorithms have been tested in indoor controlled ranges, it is only the leader-follower algorithm that is being flight tested out in the real environment: "The advantage of the 'leader-follower' approach lies in its conceptual simplicity, where the formation flight problem is reduced to a set of tracking problems that can be analyzed and solved using standard control techniques" [2]. The leader-follower approach requires hardware that is already flown on UAVs, the global positioning system (GPS) receiver, and an onboard inertial

measurement unit (IMU). In its basic form, the leader-follower approach does not require extra sensors or computing power.

In August 2015, the Advanced Robotic Systems Engineering Laboratory (ARSENL) from the Navy Postgraduate School completed a then world record swarm of 50 UAVs using the leader-follower algorithm [17]. This test utilized 2 mini swarms of 25 UAVs each, with 1 UAV designated the leader of each mini swarm. The leaders flew at the highest altitude of each swarm with each of its 24 followers spaced at 15m increments below to ensure no midair collisions. This entire 50 UAV swarm was controlled by a single ground station operator [17]. This test demonstrated the utility and scalability of the leader-follower scheme.

In Gray's research, a formation flight algorithm was developed by capturing the lead UAVs telemetry (GPS position, ground course vector, and ground speed), computing a follower vehicle waypoint, and sending this new waypoint to the follower UAV. In the computation of the follower's waypoint, the following variables could be adjusted:

- r_{Offset}: radial distance from the desired follower position to the leader vehicle's position
- Θ_{Offset}: angular offset from the leader's ground course vector
- L_1: forward offset along the leader's ground course vector
- Θ_{GC}: ground course angle of the leader relative to north

All of these variables are depicted in Figure 5.

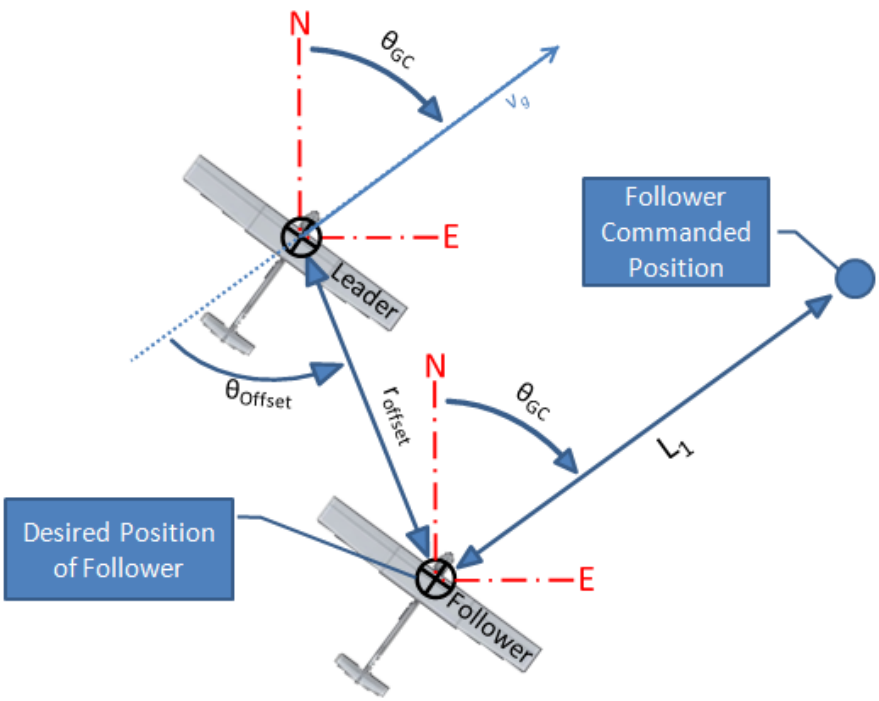

Figure 5. Follower Commanded Position Calculation Method [5]

In Gray's architecture, the data from the leader is transmitted to the ground station, where the algorithm calculates the desired follower's position, prior to this new position being sent to the follower. One factor affecting the accuracy of the follower position is the latency of this command and control architecture. This latency in Gray's test was measured to be approximately 0.46s. For formation flights using slower moving vehicles (quad rotors), the communication latency is less of a factor than it is on faster moving (fixed wing) vehicles. This can be seen directly in the average positon error of 3.1 meters for the quadrotor tests and 130.7 meters for the fixed wing tests [5].

Another factor affecting the accuracy of the follower position is the lack of feedback. The follower UAV does not know if it is in the correct position and neither

vehicle airspeed nor groundspeed were variables that were controlled in Gray's formation flight algorithm [5].

2.3 Onboard Sensors

Inter-vehicle communication also plays an important role in the fidelity of swarm behavior. By having the UAVs in the swarm communicate directly with each other, the latency of commands is significantly reduced compared to sending all communications signals routed through the ground station. The behavior and formation of the swarm needs to be guided by individual agent's reactions with its environment [8]. This is where onboard sensing comes into play. For a UAV to react with its environment, it must first be able to sense and understand its environment.

For a swarm to stay organized, the individual UAVs need to have the ability to sense and react to their environment. They need to know their distance from other aircraft in the swarm and from potential hazards. Some common sensors used on UAVs include optical, sonar, and light detecting and ranging (LiDAR).

Optical sensors offer several advantages including low weight and power consumption, but there are a few key disadvantages. Optical sensors can easily detect an object in the clear sky since the contrast between the object and the sky is great. However, on an overcast day or when the object is below the horizon, optical sensors have a harder time detecting the object. Perhaps the greatest disadvantage to the optical sensor is determining the distance to the object [18]. Without knowing the size of the object, the distance to the object cannot be determined. A work-around may exist for

swarms with all the same size aircraft, but the distance measurement will still not be accurate.

Sonar sensors are another way to accurately measure distance. Sonar sensors are an extremely popular choice for measuring distance since they are cheap, robust, and accurate. Sonar sensors work by emitting a pulse and then measuring the time it takes for the echo to return to the sensor [19]. Sonar sensors are available in a variety of beam widths and detection distances. In addition, the open source autopilot (Pixhawk) that this research used, already has integration documentation for these simple sensors. This facilitated sensor integration.

LiDAR sensors, like sonar, can also accurately measure distance. LiDAR sensors work by pulsing a laser at high frequency and measuring the time it takes for the light to be reflected back [20]. This is much like a sonar sensor, but light is used versus sound. One of the advantages of LiDAR is the resolution. A scanning or flash LiDAR can create high resolution contour maps of terrain. The disadvantage of LiDAR, for the application of this research, is that is has a narrow beam and would need to be pointed precisely at the lead UAV to get a return signal.

Each sensor was then compared based on a list of important factors to determine the appropriate sensor. These categories are: Cost, Field of View, Detection Distance, Distance Accuracy, and ease of Pixhawk Integration. The results of this sensor comparison are displayed in Table 1.

Table 1. Sensor Comparison

Sensor Comparison					
	Cost	Field of View	Detection Distance	Distance Accuracy	Pixhawk Integration
Optical	+	+	+	-	-
Sonar	+	+	-	+	+
Lidar	-	-	+	+	+

2.4 Pixhawk Autopilot

The Pixhawk autopilot is an open source autopilot that allows any user to download and modify the source code or run Python scripts in conjunction with the ground station. Pixhawk utilizes Mission Planner (other software options are also available) as the ground station graphical user interface (GUI). The Pixhawk also has a large user community with forums that allow for quick reference to any problems that may arise. The Pixhawk autopilot is designed to be plug and play. The basic set up for Pixhawk is shown in Figure 6. This figure illustrates basic layout of the Pixhawk system and describes how the components are connected to the autopilot.

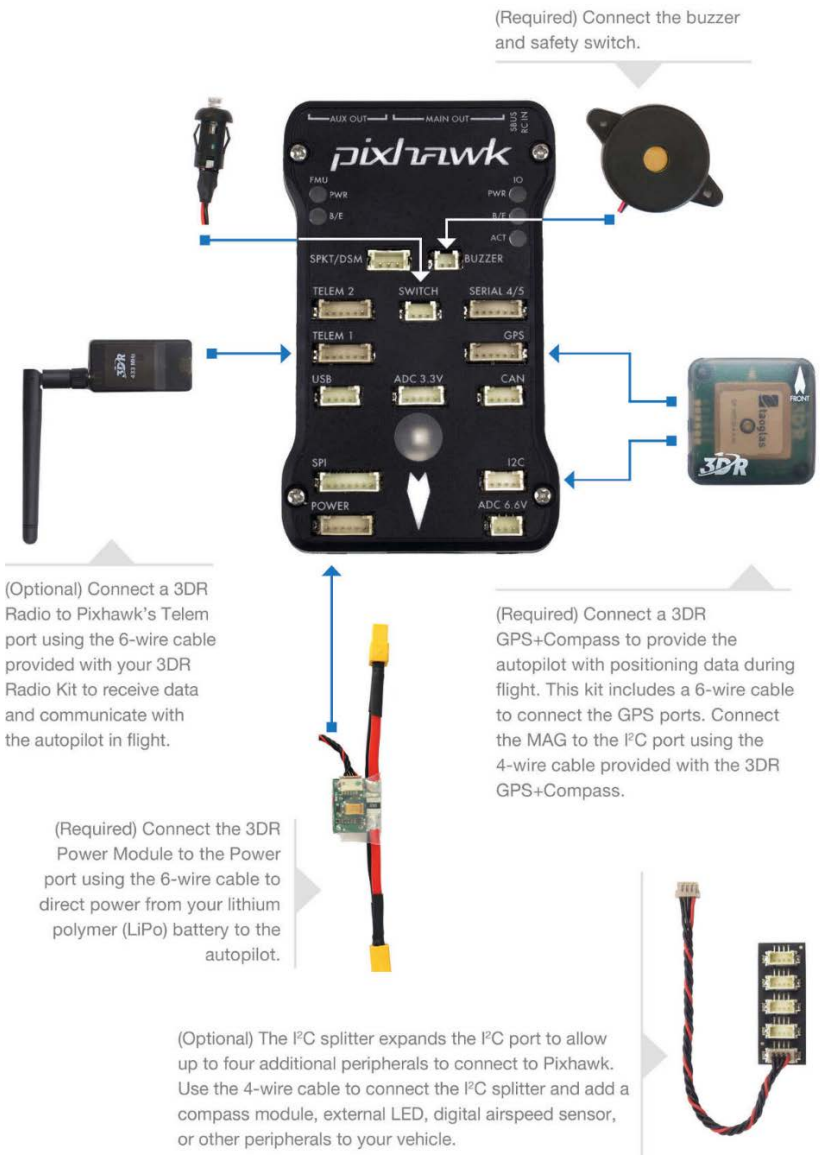

Figure 6. Pixhawk Autopilot Basic Set Up [21]

The Pixhawk auto pilot allows for the incorporation of sensors. The Pixhawk manual has existing procedures for connecting a sonar to the Pixhawk autopilot. The sensors can utilize either the analog to digital port (ADC) or the I2C port. Parameters must then be adjusted via Mission Planner in order to incorporate the use of the sensor. These parameters are RNGFND_PIN, RNGFND_MAX_CM, RNGFND_SCALING, and

RNGFND_TYPE [21]. Table 2 shows a brief description of each of these parameters. Once all parameters are set, the sensor voltage and distance can be read in the Mission Planner GUI or incorporated into custom Python scripts.

Table 2. Sonar Parameters

Parameter	Description
RNGFND_PIN	Analog pin that rangefinder is connected to.
RNGFND_MAX_CM	Maximum distance rangefinder can reliably read
RNGFND_SCALING	Scaling factor between rangefinder reading and distance
RNGFND_TYPE	Type of rangefinder device connected

The Mission Planner GUI allows the user to easily incorporate custom scripts. In the lower left region of Mission Planner, there is a tab that permits the user to run Python scripts. This allows easy integration of the custom scripts with Mission Planner and allows the user to read any of the numerous autopilot variables or set the control parameters to a desired state or position. A complete list of the variable names and example code is located in the user manual for Mission Planner and are defined by the MAVLink protocol [22]. An example screen shot of the Mission Planner GUI is shown in Figure 7. This screen shot illustrates the standard layout of the GUI. The heads up display with all flight data is located in the top left. The moving map, that displays the vehicles current position and way points, is on the right side of the screen. Also, a list of tabs can be seen on the bottom left hand side of the screen. The tab, titled “Scripts”, is used to load custom Python scripts.

Figure 7. Mission Planner GUI [23]

2.5 Conclusion

Chapter II reviewed swarm algorithms and the rules that guide them. Also, an overview of applicable sensors that are currently being used in UAV applications were explored. Finally, this chapter outlined the Pixhawk autopilot hardware, software, and its various interfaces to understand how system hardware components are incorporated.

This literature search did not uncover any open-air flight tests conducted using a feedback loop with onboard sensors. In all open-air flight tests, the follower vehicle or vehicles were unaware of any position error. This research is targeting the gap that exist by combining low-cost formation flight architecture with low-cost onboard sensors to reduce formation flight position error.

III. Methodology

3.1 Introduction

This research aims to determine the feasibility of incorporating low-cost sonar sensors to aid in UAV formation flights using a COTS open source autopilot. Currently the use of COTS components in UAV formation flights results in insufficient position accuracy of the aircraft in the formation [5]. This research uses sonar sensors on board the following UAV to increase the position accuracy of the follower UAV with respect to the leader. The sonar sensors allow for an accurate distance measurement to be collected and then used in adjusting the follower UAV's velocity.

3.2 Overview

In chapter II, this research reviewed swarm algorithms and the rules that guide them. Also, an overview of applicable sensors that are currently being used in UAV applications were explored. Finally, chapter II outlined the Pixhawk autopilot hardware, software, and its various interfaces.

By researching the literature, it was determined that the leader-follower algorithm is an appropriate algorithm to incorporate into this project. The other algorithms that were studied provide a higher degree of formation cohesion in simulation; however, major assumptions were made and are not practical to incorporate into the physical system architecture. The leader-follower algorithm only requires the leader's current heading and global positions system (GPS) data to be passed to the following vehicle. This data is already calculated in the lead vehicle and can be passed to the follower vehicle via the ground station.

The Maxbotics MB1202 sonar sensor was selected after conducting background research on the various types of sensors that are available. This sensor was selected for this project based on size, weight, power, accuracy, beam width, and ease of integration with the Pixhawk autopilot.

3.3 Materials and Equipment

This research required multiple UAVs, two quadrotors, Pixhawk autopilots, and sonar sensors. The required equipment is outlined in this section.

3.3.1 Unmanned Aerial Vehicles

The COTS quadrotors used in this research are the X8s manufactured by 3D Robotics. The X8 has an X frame configuration consisting of a total of 8 motors. Two X8s were used during flight testing for this research effort. One of the X8s used in flight testing is shown in Figure 8 and the X8 specifications are shown in Table 3.Table 3. 3D Robotics X8 Specifications

Figure 8. 3DR X8 Quadcopter

Table 3. 3D Robotics X8 Specifications [24]

Frame	X
Propellers	APC 10X4.7
Battery	4S 10000 mAh
Weight (with battery)	7.7 lbs
Aircraft dimensions	13.7 in x 20.1 in x 11.8 in
Payload weight	< 2 lbs
Flight time	14 min

A block diagram of both the leader and follower X-8 components are shown in Figure 9 and Figure 10 respectively. These diagrams illustrate which components come stock on the X-8 and which were added for this research effort. Stock items are shown in blue, whereas the added components are highlighted in yellow. The lead vehicle required modifications to enhance the acoustic signature to give the follower's sonar a greater effective range. This modification is discussed in detail in chapter 4, section 2.1. In addition, the lead vehicle was fitted with an extra modem to communicate with a backup

ground station. It should be noted that this extra modem was not required for the formation flight architecture; it was required to satisfy the flight safety review board. This extra modem allows the backup ground station operation to have situational awareness of the health of the lead UAV.

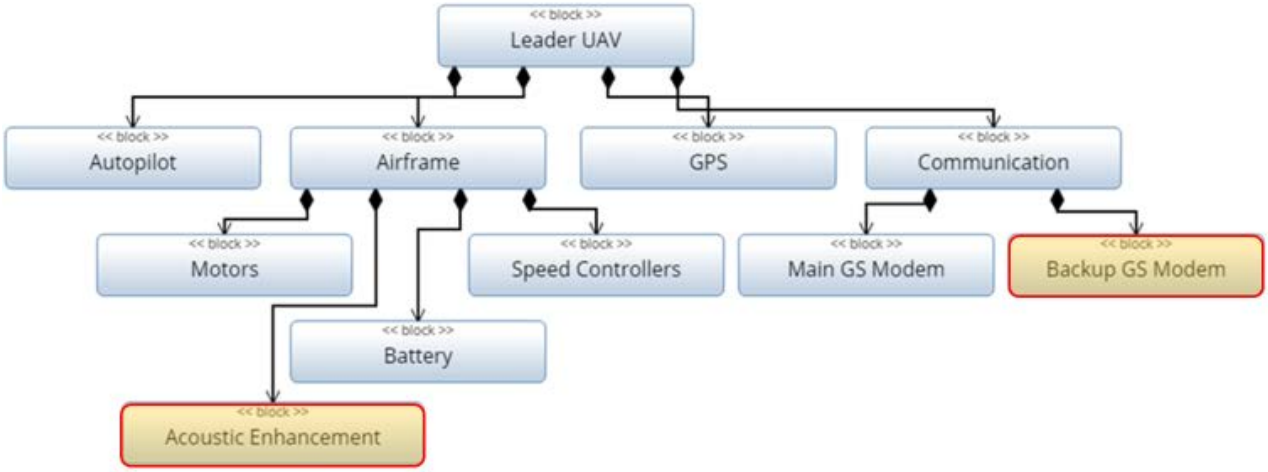

Figure 9. Leader Block Diagram

The follower UAV required the addition of the sensor package. The sensor package included the sonar, gimbal, and the algorithm to operate each of them. The sonar and gimbal set up is discussed, in detail, later in this chapter.

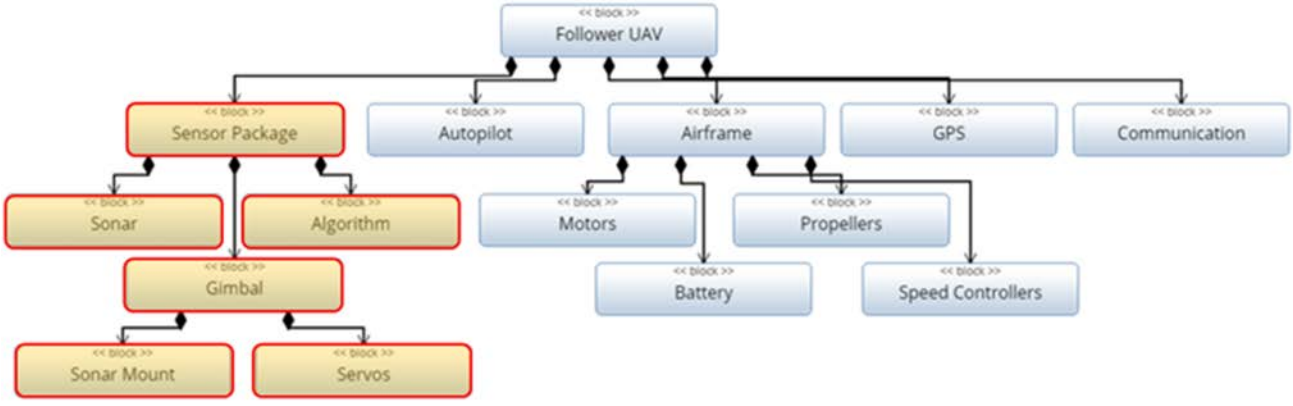

Figure 10. Follower Block Diagram

3.3.2 Pixhawk Autopilot

The Pixhawk Autopilot is a low-cost ($250), COTS system developed and sold by 3D Robotics. It is an open source platform that permits access to the source code. The ground station GUI, Mission Planner, contains simple interfaces to allow custom python scripts to be ran in conjunction with the standard autopilot software. The Pixhawk Autopilot is shown in Figure 11.

Figure 11. **Pixhawk Autopilot** [21]

3.3.3 Sonar Sensor

The sonar sensor selected for this research effort was the MaxSonar MB1202 (shown in Figure 12). This model sensor was selected due to its wide beam width and through ground testing proved to give a consistent return on an X8 quad out to 4.5 meters. This sensor also incorporates the use of the I2C communication protocol which allowed seamless integration with the Pixhawk Autopilot.

Figure 12. **I2CXL-MaxSonar®- EZ™ Series MB1202** [25]

3.4 Procedures and Processes

This section describes how the hardware and software work together to control the follower UAVs position and separation distance. Also, a description of the flight tests that were conducted and the analysis technique used to verify the system performance is discussed in this section. This research project required a formation flight algorithm, a gimbal mounted sonar sensor and sensor control algorithm, an algorithm to control the Pixhawk autopilot's response to the sonar input, flight test verification, and data analysis.

3.4.1 Algorithm

This research began with the algorithm developed by Gray as a baseline [5]. In Gray's algorithm, the lead UAVs telemetry is transmitted down to the ground station, where the algorithm uses this data to calculate the follower UAVs new flight path. Once the new flight path is calculated, the algorithm sends a new way point to the follower UAV, which aligns the follower UAV correctly with the lead UAV. This new way point is set far enough ahead of the follower UAV that it never reaches the way point before the next update is sent from the ground station. This algorithm sends an updated way

point at a frequency of 8 Hz [5]. Since Gray's code was written to work with MAVProxy, it had to be modified to work with the new developer software called DroneKit [5].

With the modified Gray algorithm as the foundation, new code had to be written to incorporate the use of the follower UAV's on board sonar sensor [5]. A detailed discussion of how the sensor is integrated both physically on the UAV and into the software will be discussed in later sections. This system architecture is graphically depicted in the operational view (OV-1). This OV-1 was modified from Gray to incorporate the sonar sensing capability of the follower UAV and is depicted in Figure 13 [5].

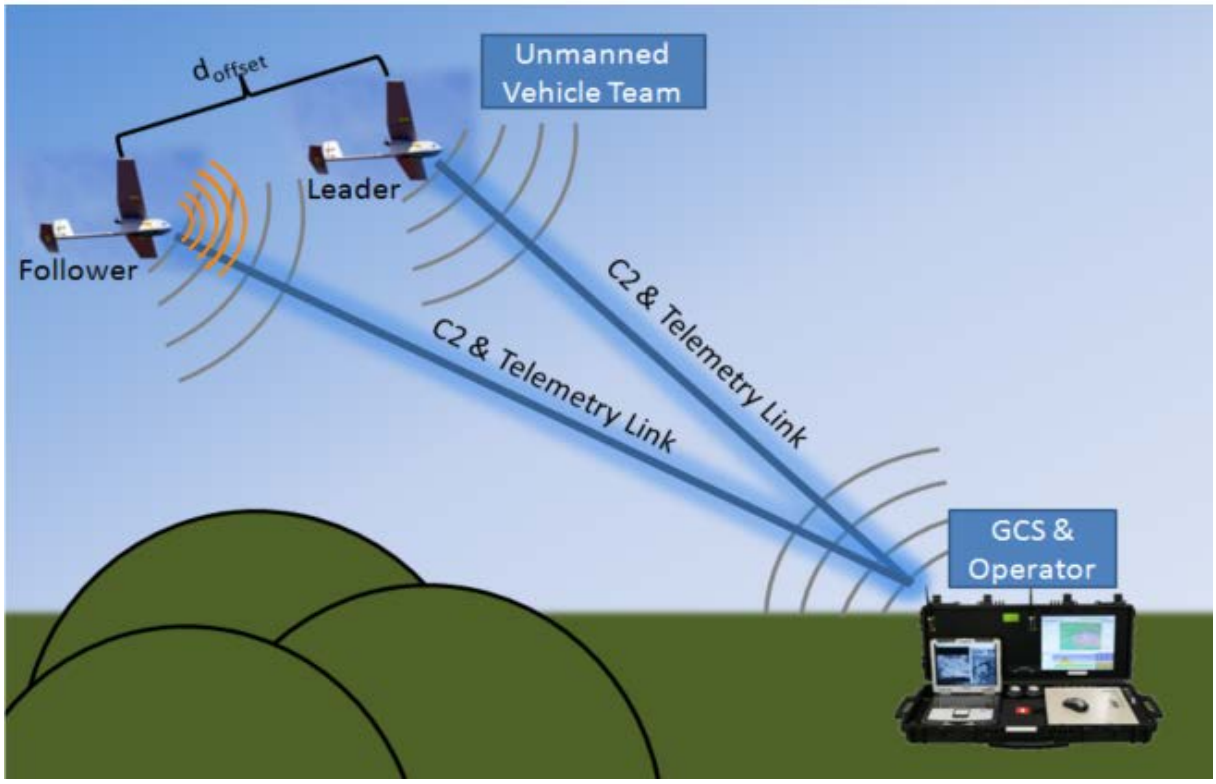

Figure 13. Modified OV-1 from Gray [5]

The communication architecture that was used in this research is detailed in the Department of Defense Architecture Framework (DoDAF) System Interface Description (SV-1) shown in Figure 14.

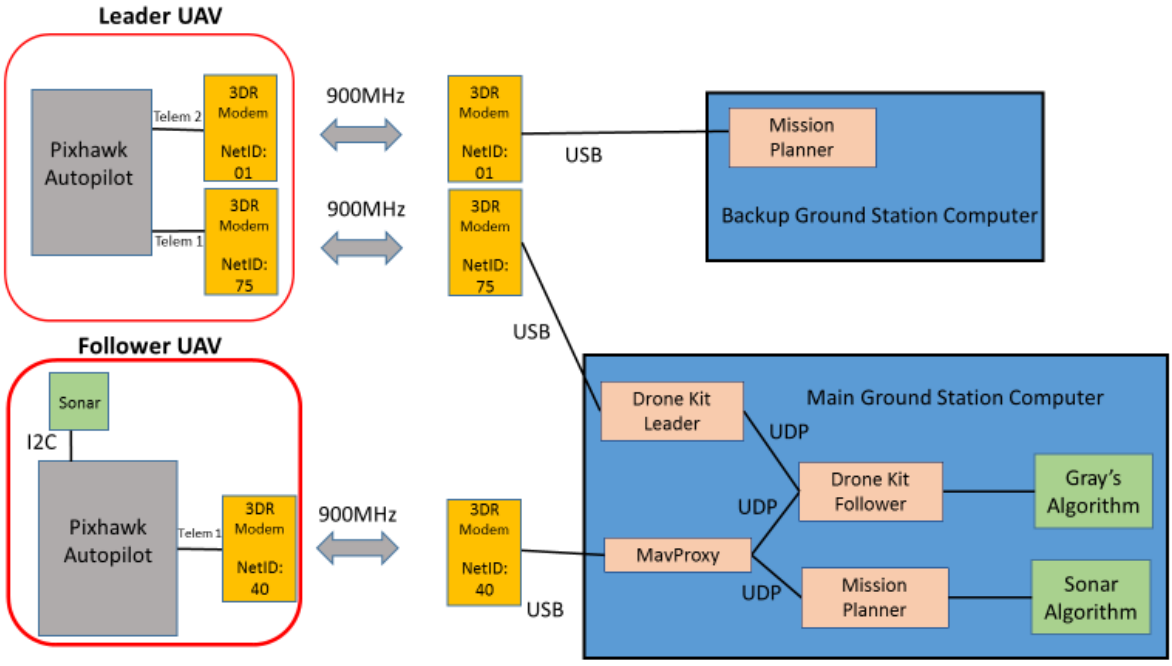

Figure 14. SV-1 (System Interface Description)

For the lead UAV, this architecture used two 900MHz telemetry radio links, utilizing both the telemetry 1 and telemetry 2 ports on the Pixhawk autopilot. The telemetry 1 port was connected to DroneKit on the ground station computer and communicates the leaders telemetry with the follower UAV's instance of DroneKit. The telemetry 2 port was connected to Mission Planner on a backup ground station computer. This link was a safety requirement to allow a dedicated operator to monitor the health of the lead UAV, and if necessary provide control inputs.

The follower UAV utilized a single 900 MHz link to the main ground station computer. MAVProxy was then used to split this telemetry stream to the follower's instance of DroneKit and Mission Planner. This instance of Dronekit was used to run the modified version of Gray's algorithm and communicate with the second instance of DroneKit [5]. Mission Planner was used to run the sonar algorithm script.

Each of the 3 scripts, Leader, Follower, and Sonar, were run at different frequencies. The Follower script was run twice as fast as the Leader script. This was done to keep the leaders GPS points from stacking up in the UDP. The Sonar script was run at 2.2 Hz. This frequency allowed the sonar adequate time to take the 10 sonar readings with a 50ms pause between each reading. Table 4 shows the 3 scripts with their specific control loop frequencies.

Table 4. Control Loop Frequencies

Control Loop Frequency	
Script	Frequency (Hz)
Leader	4
Follower	8
Sonar	2.2

3.4.2 Sonar Sensor Mounting

The sonar is mounted on the follower UAV and is set up on a 2 axis gimbal. The gimbal set up allows the sonar to search for the lead UAV. With the dynamic nature of flight, compounded with the error associated with GPS and the beam width of the sonar sensor, it was necessary to develop a way to have the sonar search the area in front of the UAV. This set up is shown in Figure 15.

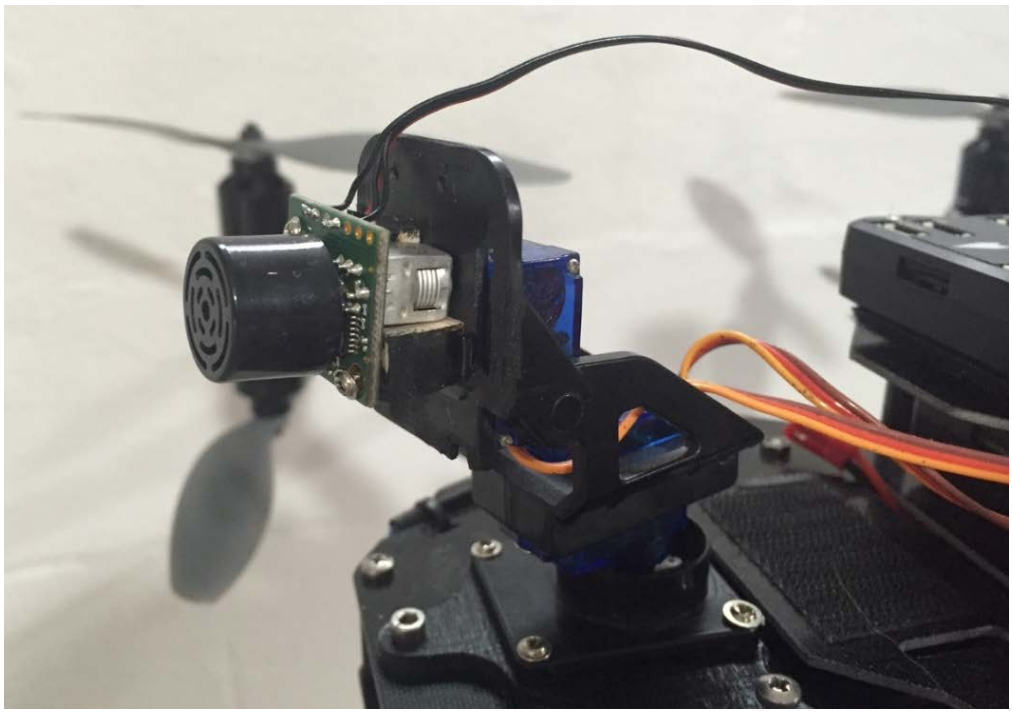

Figure 15. Sonar Gimbal Set Up

When the sonar sensor is receiving a return from the lead UAV, the gimbal is stationary. It is only when the sensor fails to receive a return does the gimbal activate. When no return is received, the gimbal begins a figure 8 search pattern. During this search, the algorithm does not command Pixhawk to adjust the velocity of the UAV. This figure 8 search pattern continues until the sonar receives a return. Once a return is received, the gimbal stops moving, holds its current position, and the Pixhawk begins adjusting the UAV velocity accordingly.

The gimbal figure 8 search pattern consist of 12 points. Each of these 12 points is reached by commanding a specific pulse width for both the pan and tilt gimbal servos. This search pattern is described graphically in Figure 16. Each point, represented by a

star, is the position that the gimbal stops to allow the sonar to take a reading. Once the gimbal hits point 12, the pattern repeats.

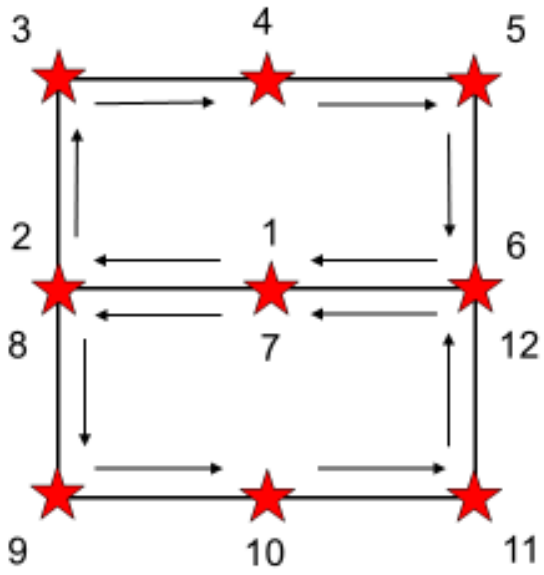

Figure 16. Gimbal Figure 8 Search Pattern

The gimbal allowed the sonar coverage areas to be increase by 43%. This increase in coverage area is shown in Figure 17. The green circle represents the sonar's field of view at any given time. The orange area represents the field of view that is covered by gimballing the sonar. The field of view was restricted to this orange area due to the location of the gimbal on the X8. This restriction ensured that the sonar was not affected by X8 airframe or propellers.

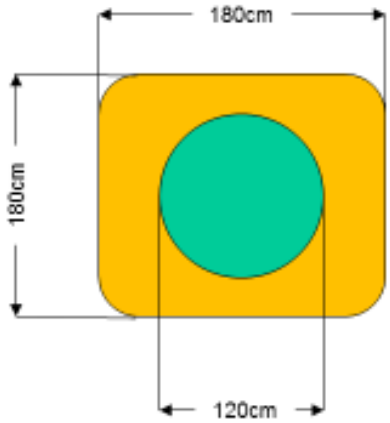

Figure 17. Sonar Coverage Area

3.4.3 Sonar Algorithm and Pixhawk Autopilot Response

The Pixhawk Autopilot monitors the reading from the sonar sensor, and based on the reading, it responds by controlling the airspeed or groundspeed of the follower UAV. The Pixhawk autopilot response to the sonar sensor input was divided into three basic possibilities. These three possibilities are: 1) attract, 2) monitor, and 3) repel. When the sonar sensor indicates that the UAV is too far from the leader, the Pixhawk commands the UAV to increase velocity (1. Attract). When the sonar senor indicates that the following UAV is within a specified range from the leader, the Pixhawk maintains the current velocity (2. Monitor). When the sonar sensor indicates that the UAV is too close, the Pixhawk commands the UAV to decrease velocity (3. Repel). This algorithm is depicted in Figure 18.

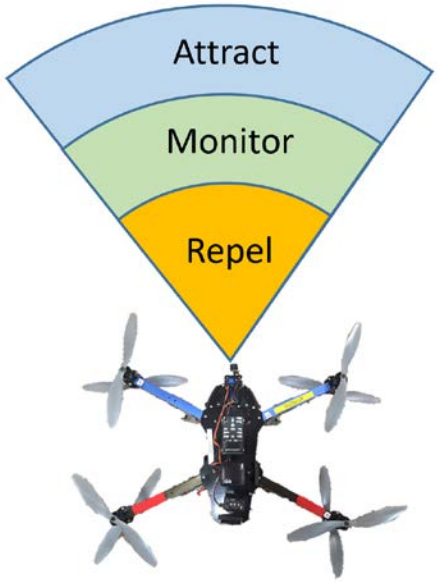

Figure 18. Pixhawk Response Algorithm

The idea for this algorithm came from the work of Pendelton and Goodrich who used Couzin's flocking model [26], [27]. The levels at which the follower's velocity is adjusted is a function of the distance follower is from the leader. The DoDAF Operational View, OV-5b, of the sonar algorithm can be seen in Figure 19. Figure 19 shows the complete algorithm architecture with more detailed views of sections of the diagram shown in Figure 20, Figure 21, and Figure 22.

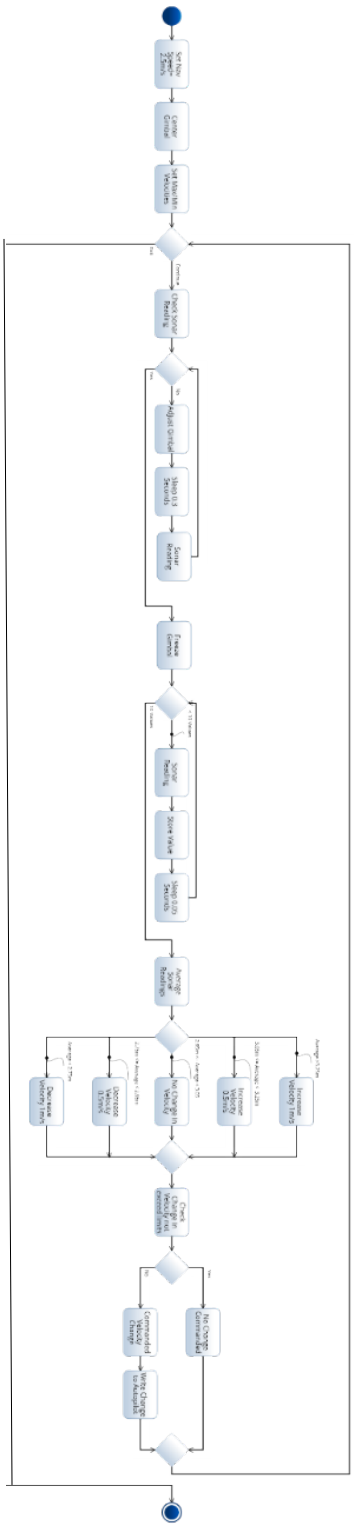

Figure 19. OV-5b (Sonar Algorithm Activity Diagram)

In Figure 20, the initiation of the script is depicted by the blue dot. Once the script is started, the algorithm sets an initial waypoint navigation speed of 2.5m/s, centers the gimbal, and sets the minimum and maximum allowable navigation speeds, 0m/s and 5m/s respectively. Once these variables are set, the control loop begins. The sonar reading is checked and if the reading is greater than 4.5m, the gimbal begins to step the sonar around a figure eight pattern, pausing at each step to check the sonar reading. This loop is exited once the sonar reading drops below 4.5m.

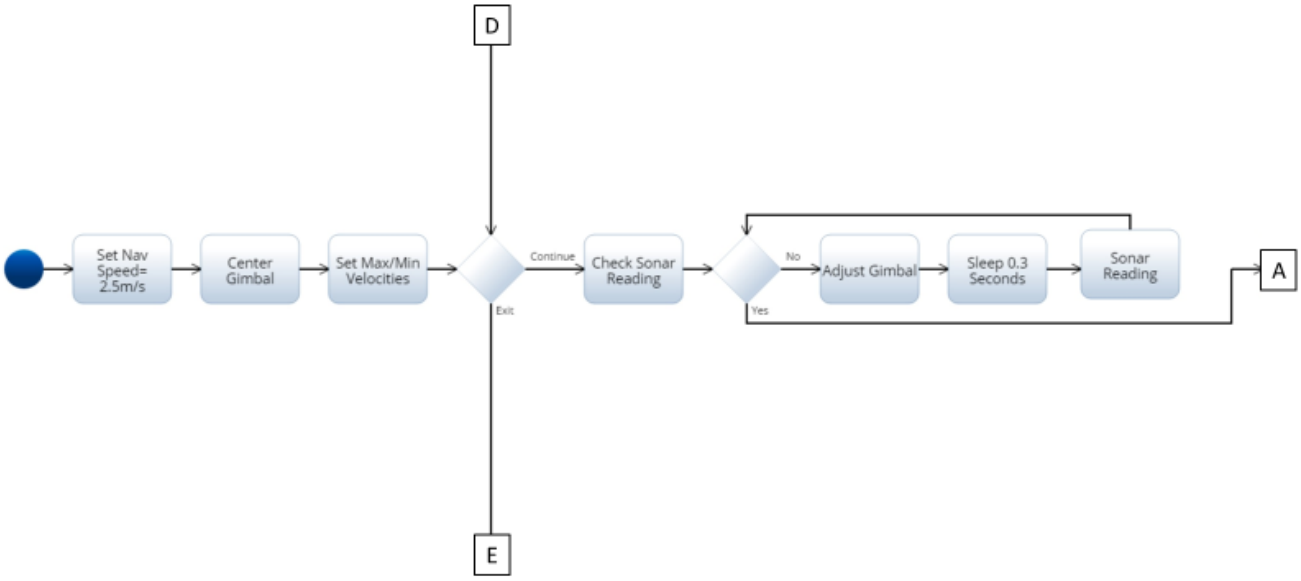

Figure 20. OV-5b Detail View 1

Now that the gimbal has adjusted the sonar to locate the leader, the gimbal is frozen in that position. Now an average of 10 sonar readings are taken over a 0.45 second period. These steps are shown in Figure 21.

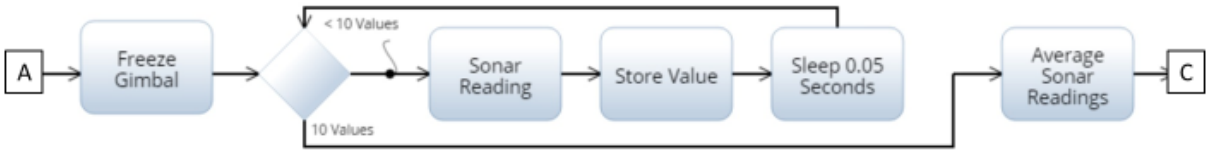

Figure 21. OV-5b Detail View 2

This average sonar reading falls into one of five categories, shown in Figure 22. The speed adjustments were divided into these categories, versus creating a linear function, due to the Pixhawk restricting the velocity adjustment resolution to 0.5m/s. After the appropriate speed adjustment is determined, a check is completed to ensure this adjustment will not cause the new navigation speed to fall outside of the set limits. If this desired change is within limits, the new commanded navigation speed is then written to the autopilot. If the new desired speed is outside the predefined limits, no change in navigation speed will be written to the autopilot.

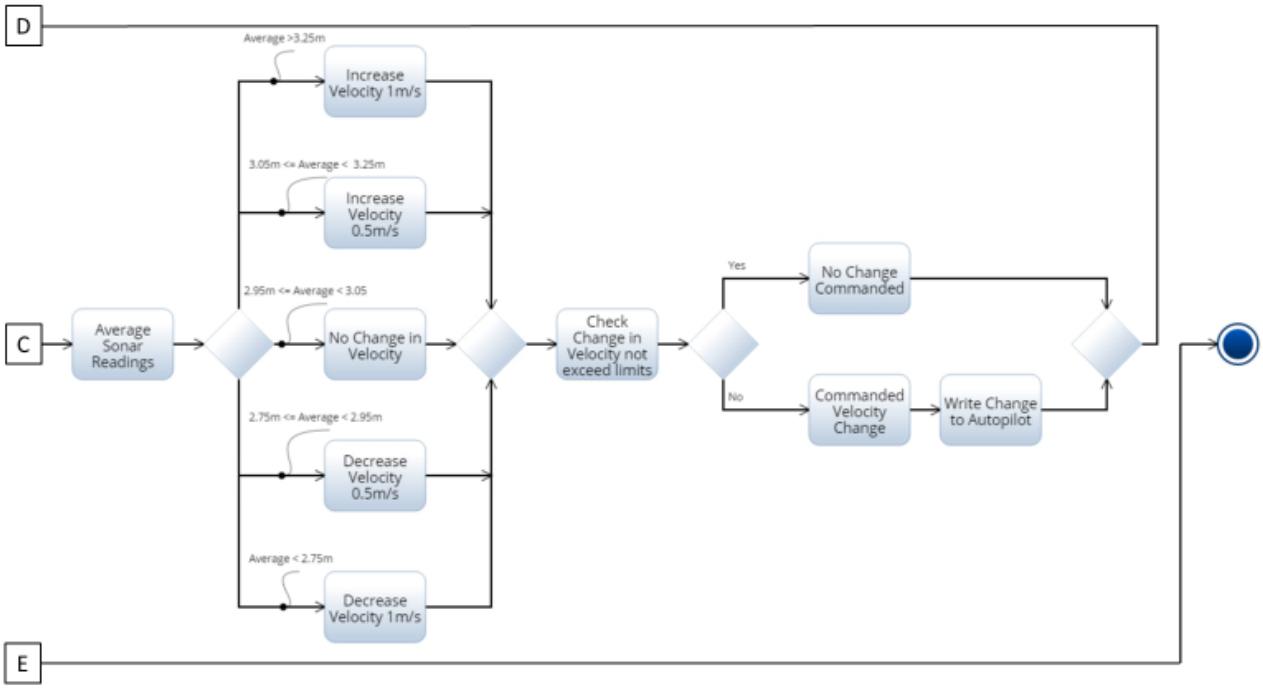

Figure 22. OV-5b Detail View 3

3.4.4 Flight test

A series of flight tests were conducted at an altitude of 10m above the ground. The lead UAV was commanded to fly a basic rectangular waypoint pattern, shown in Figure 23, with a velocity of 2 m/s. The followers route and velocity was controlled autonomously by the algorithms.

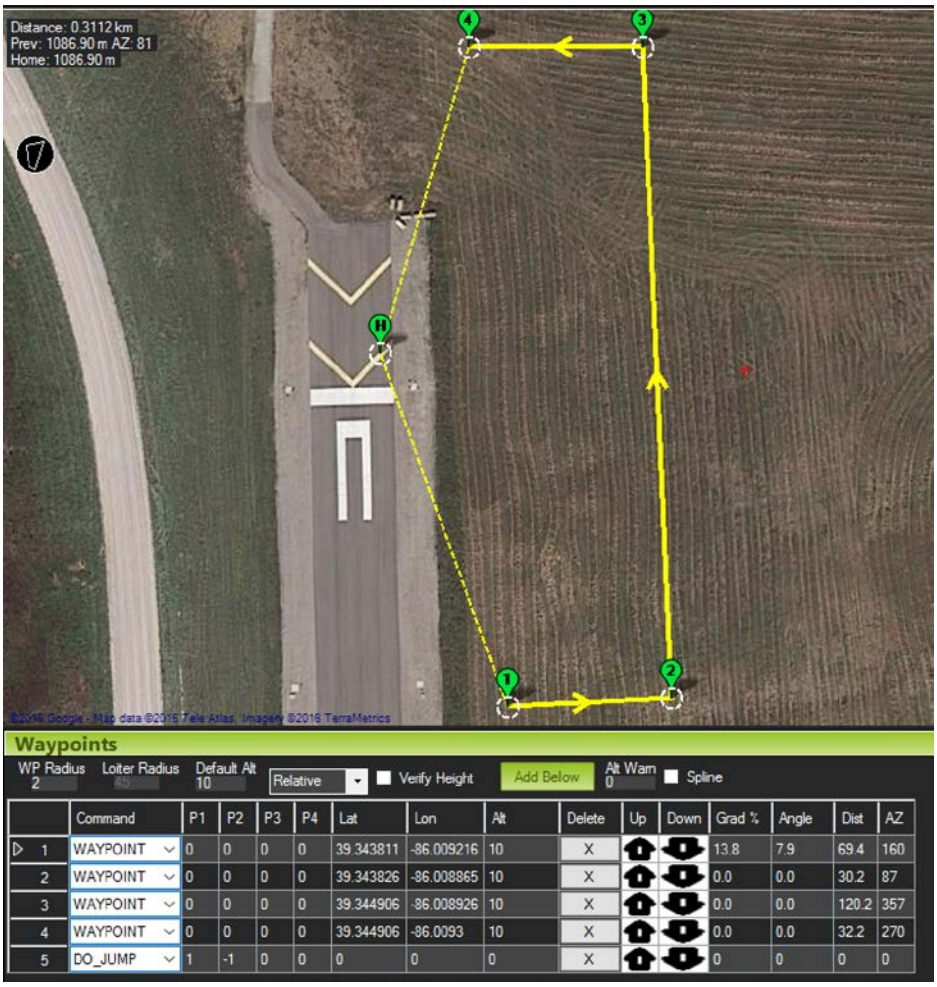

Figure 23. Leader Waypoint Pattern

During all flight test, data was collected on the GPS position, altitude, and velocity of the leader, as well as the GPS position, altitude, velocity, sonar reading, and commanded velocity of the follower. This data was collected at a rate of 2Hz and was time stamped for comparison between the two vehicles.

3.4.5 Data Analysis

The data collected was used to determine the position error of the follower UAV. The analysis technique used was Root Mean Square Deviation (RMSD). RMSD allows

the average position error to be calculated by using the UAVs actual position and the UAVs desired position for each time step. The equation for RMSD is shown in equation 1.

$$RMSD = \sqrt{\frac{\sum_{t=1}^{n}(y(t)-y(t\_obs))^2}{n}} \quad (1)$$

In the RMSD equations, *y(t)* is the desired position of the follower UAV at that time, *y(t_obs)* is the actual position of the follower UAV at that time, and *n* is the number of data points. The RMSD results were then compared to the RMSD results from Gray's test flights to determine if the incorporation of the sonar sensor decreased the observed position error of the follower UAV [5].

3.5 Summary

This chapter detailed the materials and equipment needed for this research and discussed the procedures and processes used in order to conduct this research. Chapter 4 will discuss the ground tests, flight tests, and analyze the results by comparing the position error of this research with the previous flight test conducted by Gray [5].

IV. Results and Analysis

4.1 Chapter 4 Overview

This chapter describes the ground tests that were accomplished, the multiple flight tests using two X-8 quadrotors, analyzes the position error results of each flight, and compares this research flight test results with the sonar, to the previous work without the sonar.

4.2 Ground Tests

This section describes the ground tests that where conducted leading up to the flight tests. The ground tests included a sonar range test, a sonar algorithm test, and a guided position algorithm test.

4.2.1 Sonar Range Test

Ground tests of the sonar was conducted to determine the max effective range of the sonar when getting the return from the lead X-8 quadrotor. It was theorized from the beginning of this research that the sonar's range would be reduced due to the limited amount of reflective surface area of the lead quadrotor. Three Maxbotics sonar models were tested during the ground test, the MB1020, MB1260, and MB1202.

The ground test was conducted by hanging an X-8 quadrotor from a tree approximately 5 feet off the ground. The quadrotor needed to be off the ground to ensure the sonar return was from the quadrotor itself and not the ground. Two variations of this test where conducted. The first was using the stock X-8 and the second test modified the X-8 by adding an additional 50 square inches of aluminum foil. These tests

configurations can be seen in Figure 24 and Figure 25 respectively. The ground test results are shown in Table 5.

Figure 24. Stock X-8 Quadrotor

Figure 25. Modified X-8 Quadrotor

Table 5. Sonar Ground Test Results

Sonar	MB 1020	MB 1202/1260
Stock X8	1.6m	3.2m
Modified X8	2.1m	4.6m

The range at which the sonar could detect the X-8 was reduced from the published ranges for each model. This was due to the limited amount of reflective surface area on the X-8 and the published max ranges for the sonar are for man sized targets. The MB1020 was only able to see the stock X-8 out to 1.6m and the modified X-8 only increased the max effective range to 2.1m. This test verified that the MB1020 was not going to have sufficient range for this research effort. The stock X-8 allowed the MB1202 and 1260 sonars to have a max effective range of 3.2m, whereas the modified X-8 increased the sonars effective range to 4.6m. The MB1202 sonar was selected over the MB1260 because the MB1202 allows for I2C communication with the Pixhawk, whereas the MB1260 requires the use of the analog port.

The aluminum foil increased the sonars range by 30.4%. Due to this increase in the sonars effective range, the lead X-8 quadrotor was modified for flight test by wrapping aluminum tape around the legs to increase the reflective surface area. This flight configuration can be seen in Figure 26. Ground test were not duplicated using the aluminum tape configuration, but flight test show that this configuration was comparable to the aluminum foil ground test.

Figure 26. Lead X-8 with Aluminum Tape

4.2.2 Sonar Algorithm Ground Tests

Once the sonar algorithm was written, it was ground tested to ensure proper function. A ground test set up utilizing the sonar, gimbal, and the Pixhawk autopilot was built for this purpose. This set up can be seen in Figure 27. The verification of the algorithm was accomplished by systematically stepping through the response at various sonar readings. The ground test matrix is shown in Table 6.

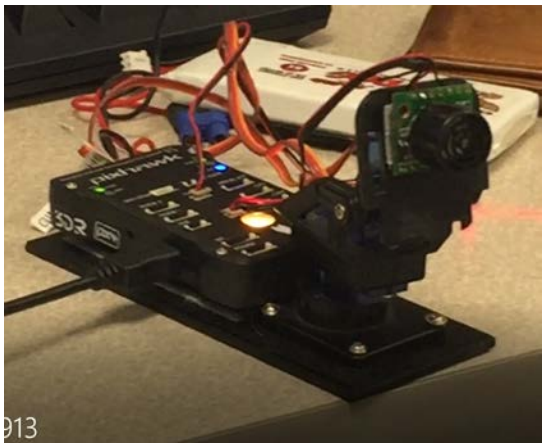

Figure 27. Sonar Algorithm Ground Test Set Up

Table 6. Sonar Algorithm Ground Test Matrix

Sonar Ground Test Matrix	
Target Range	**Response**
Sonar <= 4.5 m (Target Within Range)	Gimbal Stationary
	Average of 10 readings over 0.5 Seconds
	Range >= 3.25m 1. Increase Speed by 100cm/s 2. Check to ensure new speed does not exceed max airspeed 3. Write new airspeed to autopilot
	3.05m <= Range < 3.25m 1. Increase Speed by 50cm/s 2. Check to ensure new speed does not exceed max airspeed 3. Write new airspeed to autopilot
	2.95m <= Range < 3.05m Continue to Monitor Sonar Range
	2.75m <= Range < 2.95m 1. Decrease Speed by 50cm/s 2. Check to ensure new speed does not go negative 3. Write new airspeed to autopilot
	Range < 2.75m 1. Decrease Speed by 100cm/s 2. Check to ensure new speed does not go negative 3. Write new airspeed to autopilot
Sonar > 4.5 m (Target Out of Range)	Move Gimbal in figure 8 pattern until Sonar <= 4.5 m

The output of each test was printed to the Mission Planner command screen for verification. Screen shots of one of these tests can be seen in Figure 28 and Figure 29. From Figure 28, it can be seen that as the sonar reading changes, the new assigned trim airspeed is changed based on the distance. Figure 29 shows that the minimum airspeed

check was working properly. Even though the sonar indicates the following vehicle is too close to the leader, the algorithm does not allow the vehicle to be slowed anymore.

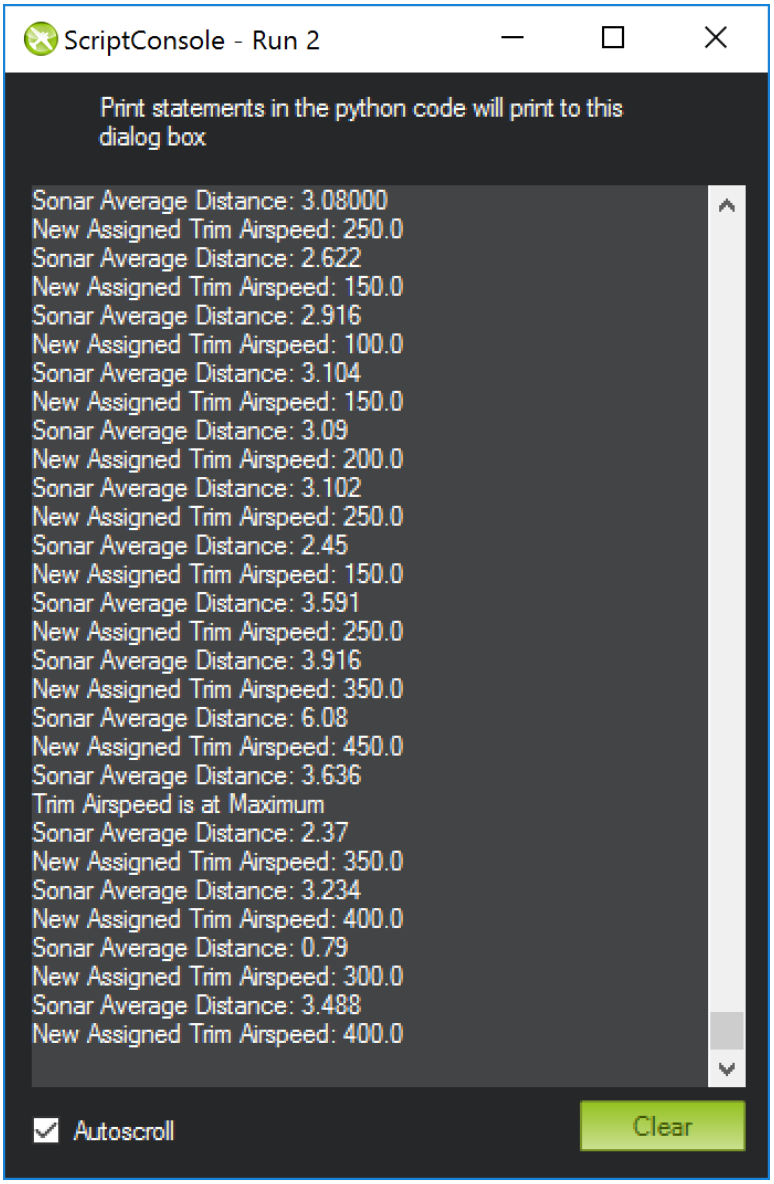

Figure 28. Sonar Algorithm Outputs

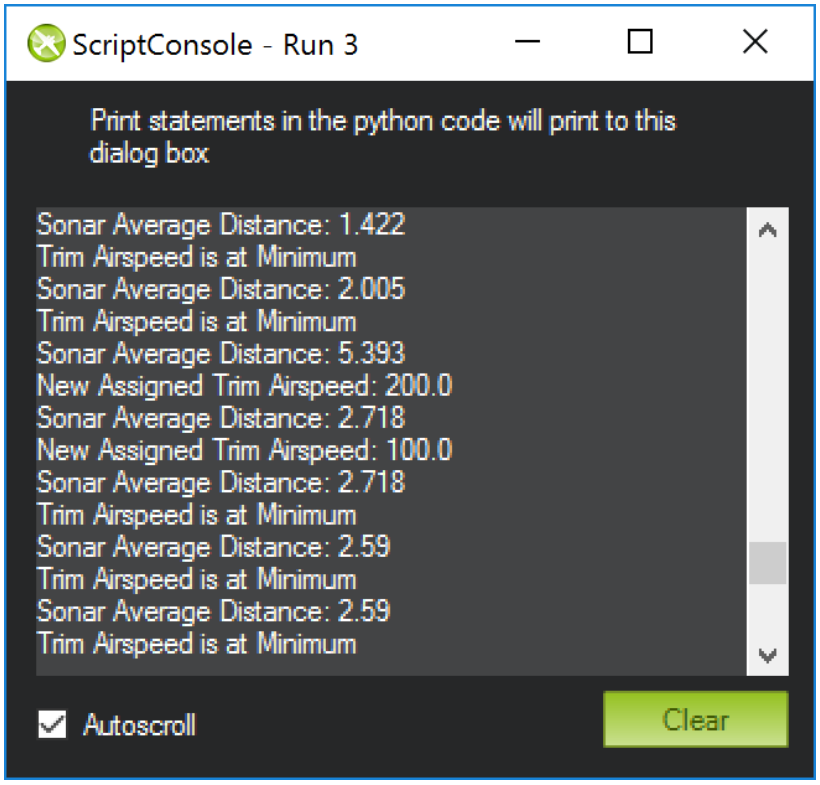

Figure 29. Sonar Minimum Airspeed

The sonar algorithm was verified again in a flight test to ensure when the sonar commanded a change in speed, the aircraft would respond. This flight test hovered one X-8 quadrotor, 2m off the ground, and commanded it to navigate towards a wall. Once the sonar detected the wall, the algorithm was able to slow the X-8 quadrotor, avoiding a collision with the wall. The minimum speed was set to 0m/s for all flight test. Quadrotor aircraft have the ability to backup and this characteristic could prove useful is some test, but this was not required for this research since Gray's guided point algorithm is a function of velocity [5]. If the leader stops, the follower will slow to a stop as well and position itself the desired distance behind the leader.

4.2.3 Guided Position Algorithm Ground Test

The guided position algorithm was originally developed by Jeremy Gray for use with MAVProxy, but had to be modified to be used with DroneKit [5]. Once the code modification was complete, it was ground tested in two phases to verify functionality. The first phase was a desktop test to ensure data was being passed from the leader to the follower. The second phase tested the accuracy of the new guided position commanded by the algorithm.

For the first phase, two autopilots with their associated modems were connected to the ground station computer. The ground station operator then ran the code to split the follower's telemetry using MAVProxy. Once the signal was split, the follower's script was executed in DroneKit and the MissionPlanner was connected to the other signal. Finally, the leader's script was run in the second instance of DroneKit. When the leader was connected to DroneKit, it began sending its telemetry to the follower and the follower began sending new commands to the autopilot. A screenshot of this test can be seen in Figure 30. The lower right command window is the leader's DroneKit instance. The lower left command window is the follower's DroneKit instance.

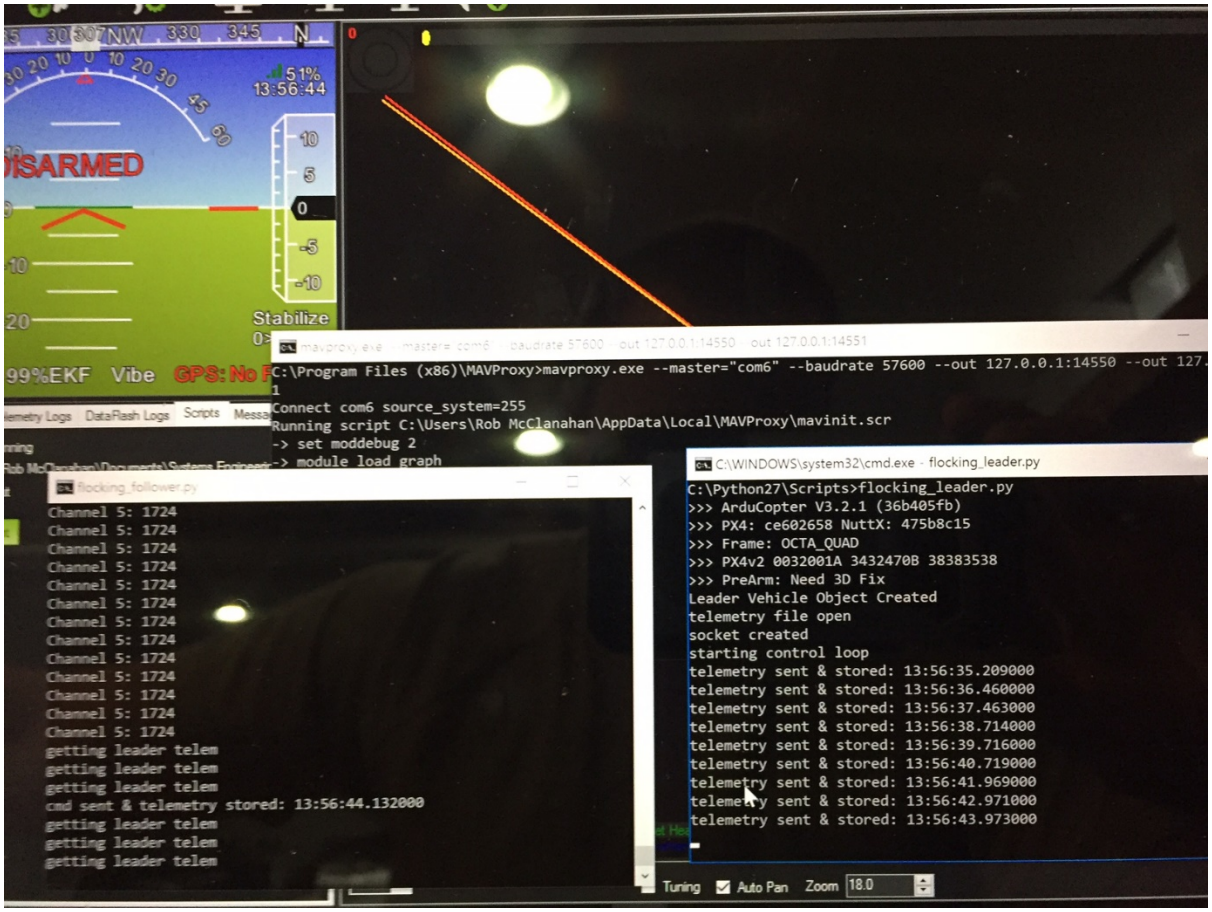

Figure 30. Phase 1 Guided Algorithm Ground Test

Phase two of the guided algorithm ground test was used to verify the follower's script commanded the correct guided position based off the leader's telemetry. To accomplish this test, the leader and follower X-8 quadrotors where placed outside. The follower algorithm was set to keep a 3m standoff from the leader. Once the algorithms were running, the lead quad was walked around an area as if it was flying a waypoint pattern. The leader's position was then compared with the commanded guided position of the follower. These results can be seen in Figure 31. The commanded guided position had an average separation distance of 3.005m with a standard deviation of 0.279m. One would expect the commanded guided position to always be exactly 3m from the leader,

but this was not the case. This error is a function of the lag in the system, measured by Gray at 0.46s [5]. As expected, the largest error in the commanded guided position occurs when the lead UAV makes a sharp turn.

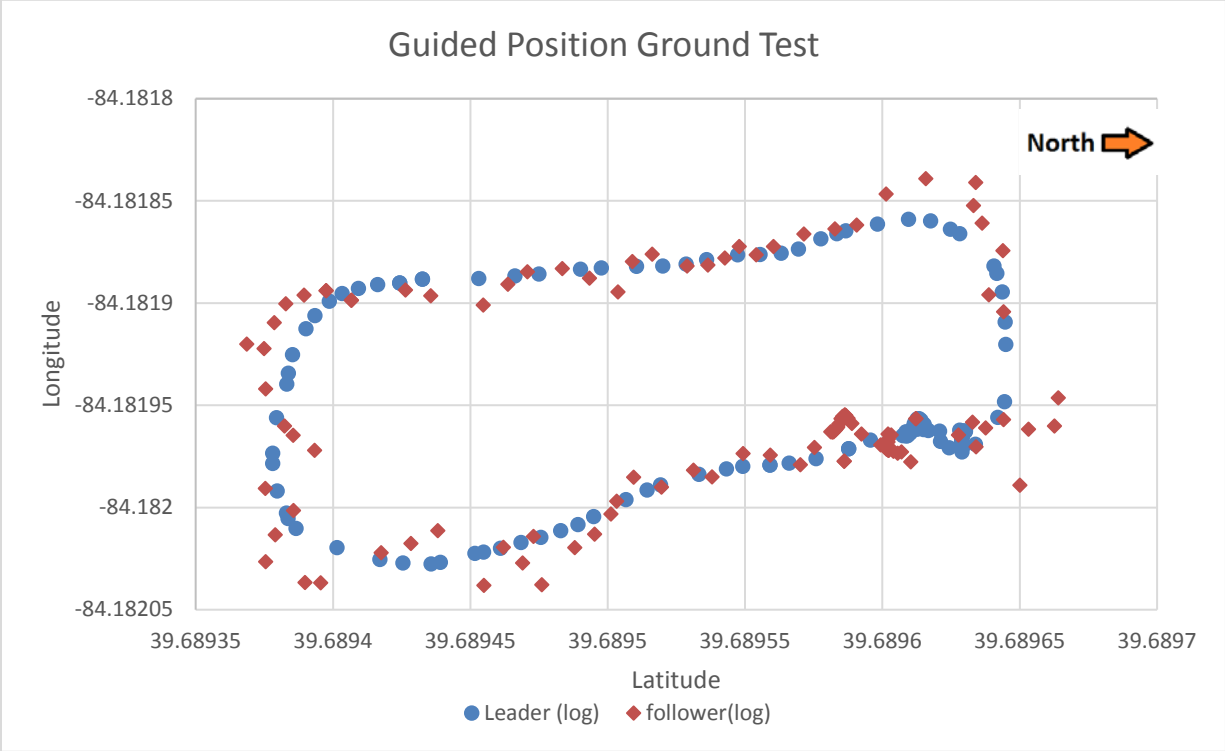

Figure 31. Guided Position Ground Test

4.3 Flight Tests and Results

Multiple flight tests were conducted during the course of this research. The initial flight was flown with a 4m desired separation distance between the leader and follower to ensure the system was functioning properly. Once the system was proven, the remaining flight tests were flown with a 3m desired separation distance between the leader and follower. All flight tests were flown with a typical racetrack pattern at an altitude of 10m

above ground level (AGL). The average separation distance, RMSD, and standard deviation of the follower X-8 is analyzed for each flight.

4.3.1 Initial 4m Separation Flight Test

It was known from the beginning that the 4m flight test would be pushing the boundaries of what the sonar could detect. However, it was important to test the full system functionality at this greater separation distance to give the safety pilots more time to respond and keep the quadrotors from having a mid-air collision. No sonar data was collected during this flight due to the python script not saving that particular variable. It was initially thought the sonar data could be taken directly from the Pixhawk telemetry file for comparison; however, this procedure proved difficult to match the sonar output with the corresponding GPS position. This code was updated in later flight tests to include the sonar data and commanded way point navigation speed. It was visually noted that the sonar was getting a return from the leader at times during this 4m flight test. A picture taken during the 4m flight test can be seen in Figure 32.

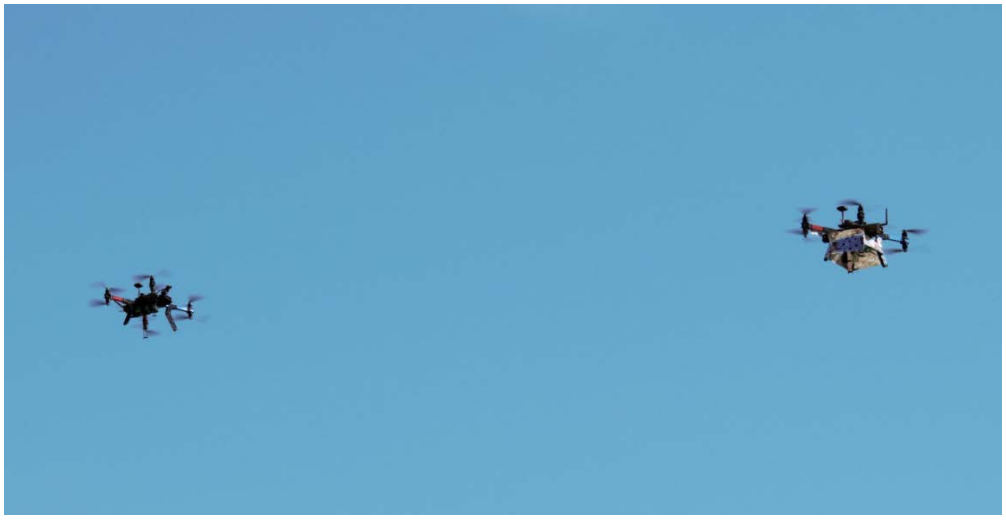

Figure 32. 4m Test Flight

The results of the 4m flight test can be seen in Figure 33 and

Table 7. Figure 33 is a plot of the GPS location of both the leader (blue) and the follower (orange). This plot visualizes the location of each X-8 during the flight and shows that the follower X-8 was indeed able to follow the leader. Note the undesired offset in the follower's position when the leader was traveling on a North/South vector. This offset was not initially noticed during the 4m flight test, but was recognized during the following 3m flight test and is discussed in that section.

Table 7 displays the key results of this test. It was expected that the RSMD for this flight would be highest, since the sonar was not playing an active role during this test.

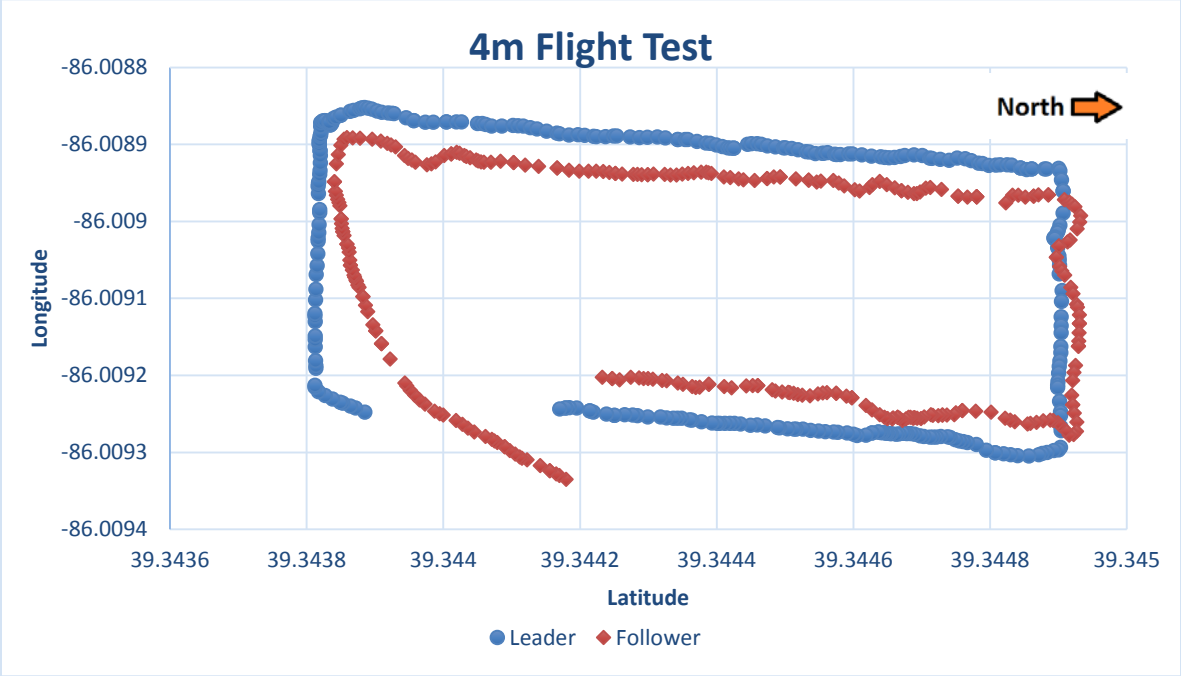

Figure 33. GPS Track from 4m Flight Test

Table 7. 4m Flight Test Results

4m Flight Test	
Average Separation Distance	7.67m
RMSD	4.76m
Standard Deviation	3.02m

The X-8s were never in danger of a mid-air collision. With this successful test, the desired separation distance was reduced from 4m to 3m for the remainder of the flight tests to allow the sonar to influence the position of the follower X-8.

4.3.2 3m Separation Flight Tests

Flying with a desired separation distance of 3m allowed the sonar to frequently calculate the distance the follower was from the leader resulting in constant velocity adjustments to keep the follower at the set separation distance. A series of flight tests were conducted at 3m. The initial 3m flight test uncovered a problem with the guided point algorithm. A picture of a 3m flight test can be seen in Figure 34.

Figure 34. 3m Test Flight

4.3.2.1 Initial 3m Flight Test

The first flight test flown with a 3m desired separation distance uncovered a problem not noticed during the previous test or ground testing. The follower vehicle was positioned offset from the leader, but only when traveling North or South. When the leader was traveling East or West, the follower was tucked in correctly behind the leader. This phenomenon can be seen in the GPS plot of each vehicle shown in Figure 35.

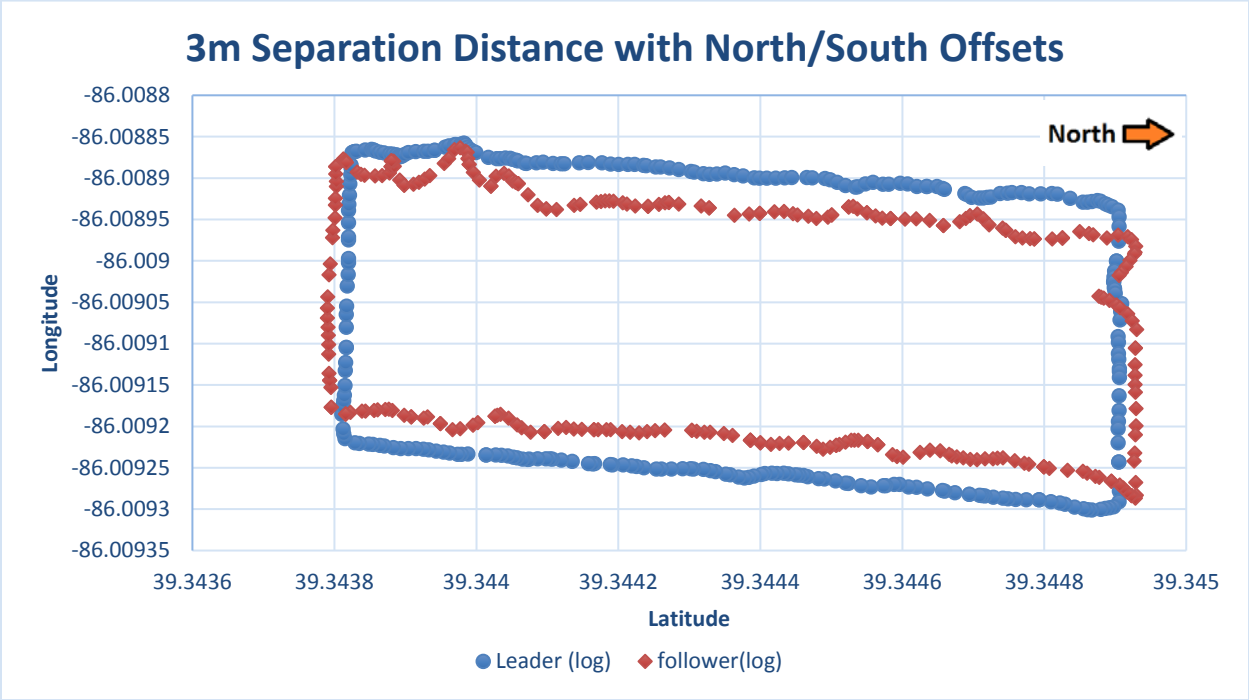

Figure 35. North/South Offsets

This offset created two problems with the system architecture. The first issue was the safety pilot had to take control of the following vehicle frequently when the lead vehicle initiated a turn due to the fact that the lead vehicle would turn in front of or cut off the follower. The second issue that arose was that with the north/south offset, the sonar was not able to frequently detect the distance from the lead vehicle unless the

vehicles were traveling east or west. Despite these challenges, the results shown in Table 8, do show a decrease in the RMSD when compared to the 4m flight test. This is due to the fact that the sonar was able to adjust the velocity of the following X-8 on more occasions.

Table 8 Initial 3m Flight Test

Initial 3m Flight Test	
Average Separation Distance	6.51m
RMSD	2.55m
Standard Deviation	2.51m

Two other issues were also noted during this test flight. First, at times the follower X-8 would fly sideways behind the lead X-8. With the follower X-8 flying sideways, there was no chance for the front mounted sonar to measure the distance to the leader. The other problem noted was the lead X-8 was not holding the desired altitude of 10m. Randomly it would lose over a meter of altitude which put it out of the field of view of the follower's sonar.

The root cause of this North/South offset was discovered and corrected in the guided algorithm code. The error was in dealing with the way the offset angle was defined in Gray's code [5]. The reference needed to be rotated by 90 degrees to make it relative to East. The reason this error only showed itself when traveling North or South was because for this research the offset angle was set at 0 degrees. In addition, a fresh compass calibration as well as a gain tuning flight was completed on both X-8s.

4.3.2.2 Final 3m Flight Test

With the guided point algorithm corrected and the other adjustments complete, the final 3m flight test was conducted. The GPS plot of this flight can be seen in Figure 36. From this figure, notice that the follower is now directly in line with the leader at all times throughout the flight.

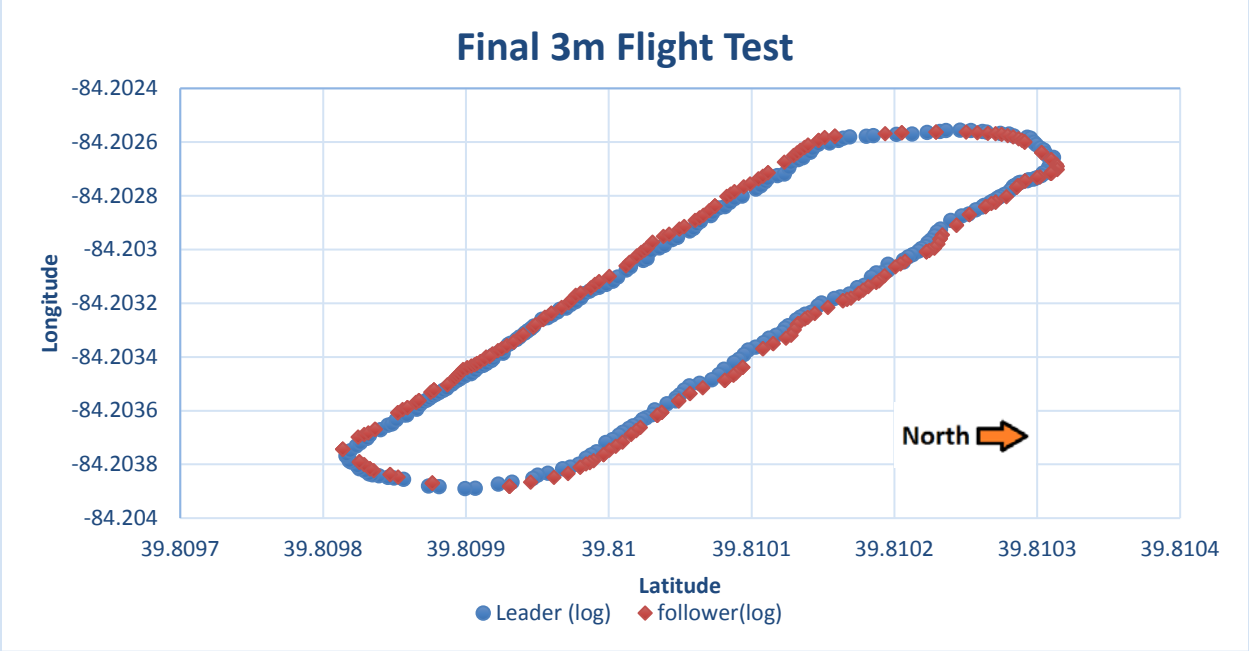

Figure 36. Final 3m Flight Test

In Figure 36 it is impossible to graphically discern how close the follower is to the leader at any point during the flight. This is due to the 2hz frequency that the GPS position data was collected and plotted. In Figure 37, the frequency of the data displayed in Figure 36 was reduced to approximately one data point every 10 seconds and the distance between the two vehicles is displayed between the data points. This reduction in the data rate allows for an easy visual comparison of the leader and follower's position during the flight. From Figure 37, the follower was able to closely follow the leader and

remained in line directly behind the leader to give the sonar the best chance at detecting the distance to the lead vehicle.

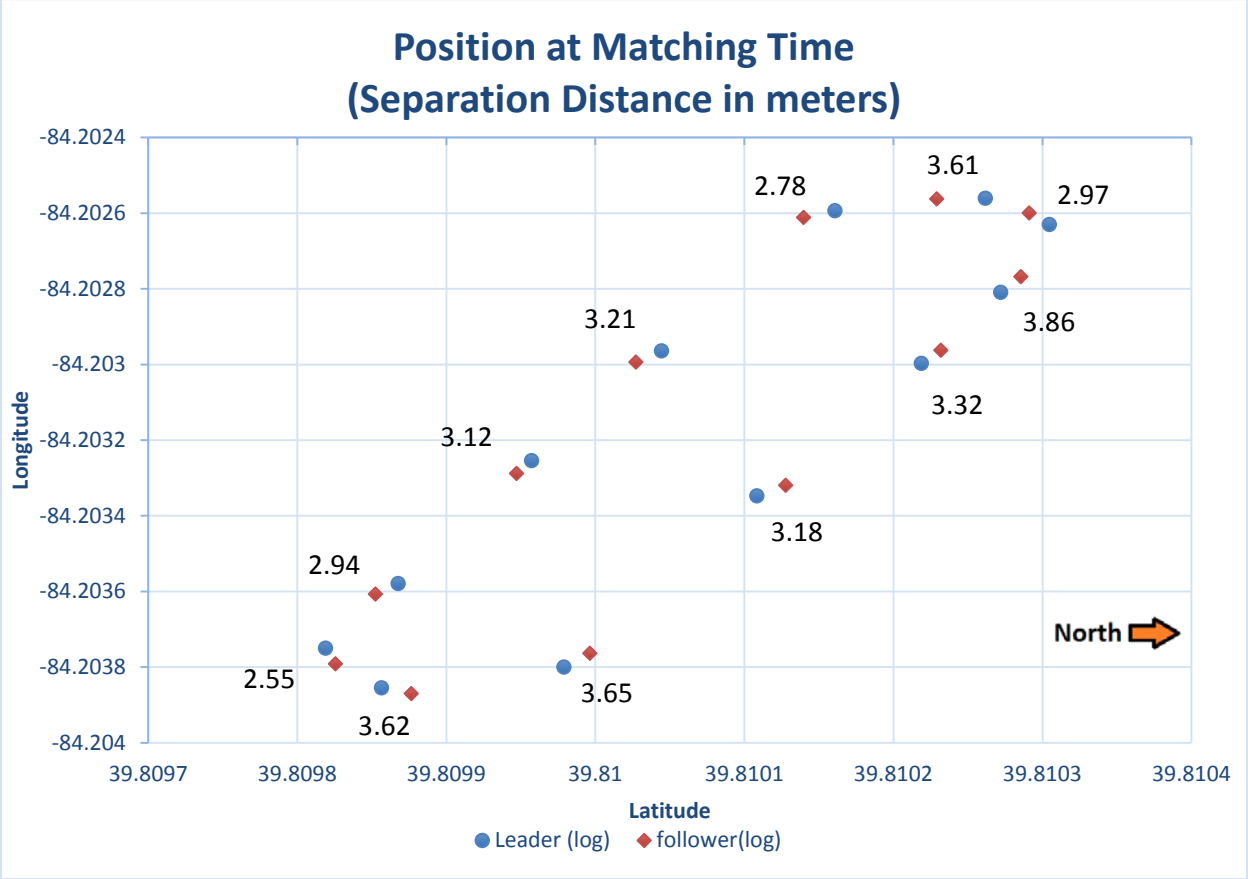

Figure 37. Position at Matching Time

With the follower now directly behind the leader, the follower's sonar was able to more reliably detect the distance from the lead X-8. The raw sonar data from this flight is shown in Figure 38. The sonar defaults to the value of 7.6m when no signal return is detected and therefore no distance is measured. Any value below 7.6m, is the measured distance from the leader. From this figure, the sonar was able to detect the distance from the lead X-8 throughout the flight.

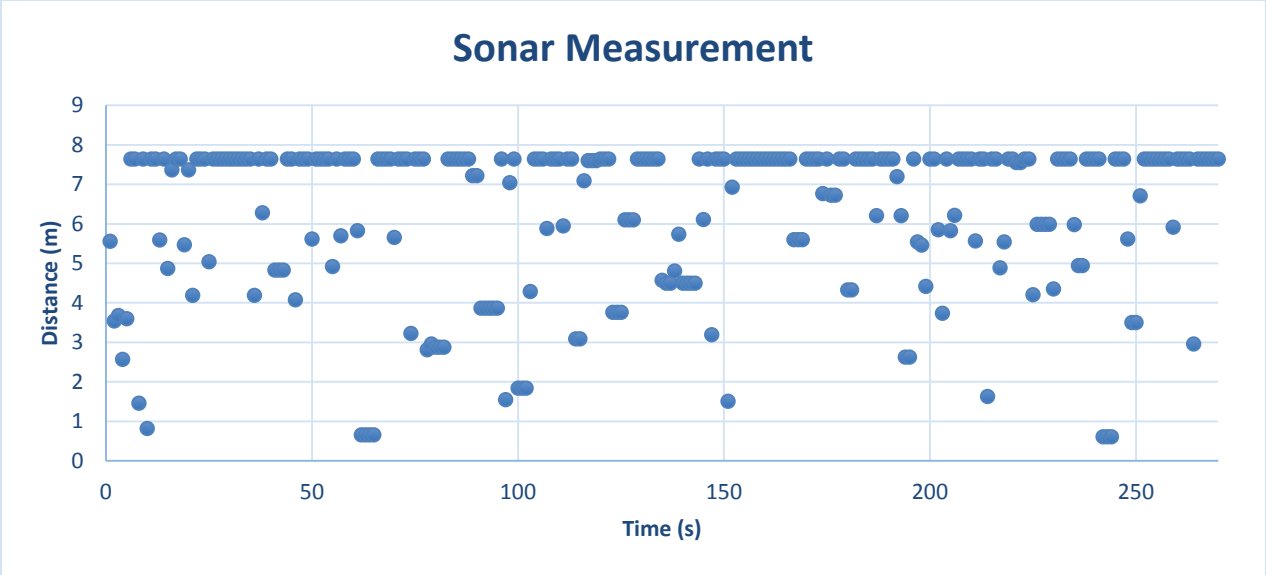

Figure 38. Raw Sonar Data

Due to the ability of the sonar to measure the distance the follower was from the leader, the sonar algorithm was able to adjust the follower's velocity in an attempt to maintain the desired separation distance of 3m. This adjustment in the follower's velocity is shown in Figure 39 and Figure 40. In these figures, the blue-dashed line is the commanded way point nav speed that is adjusted by the sonar. The red line is the actual ground velocity of the follower vehicle. From these plots, by adjusting the way point nav speed of the follower vehicle, the sonar algorithm is able to modify the velocity of the follower vehicle to maintain the desired separation distance from the leader. It is also noted that the actual ground speed occasionally lags behind the commanded speed. This is caused by a couple factors. First, the follower vehicle is occasionally way point limited. This occurs when the leader slows down and causes the desired follower position to be close enough to its current position that the follower reaches the desired waypoint and begins to slow to a hover prior to getting an updated desired GPS point. This

phenomenon can be seen graphically in the middle of the plot on Figure 40. The second reason for this lag is internal to the autopilot. It was noted during testing that at times the follower would not adjust its velocity even though it had a commanded change that was written to the Pixhawk. It was as if the Pixhawk was busy with other tasks and would take a few seconds before it executed the commanded change in velocity.

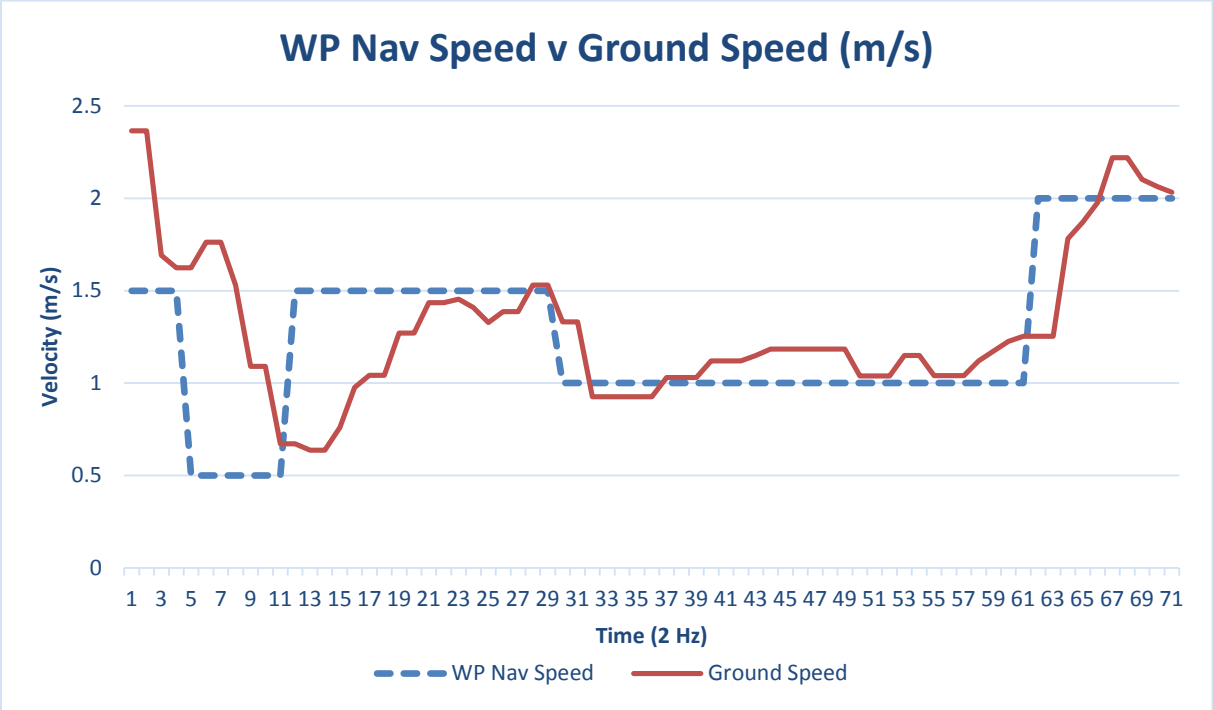

Figure 39. Waypoint Nav Speed versus Ground Speed (Plot 1)

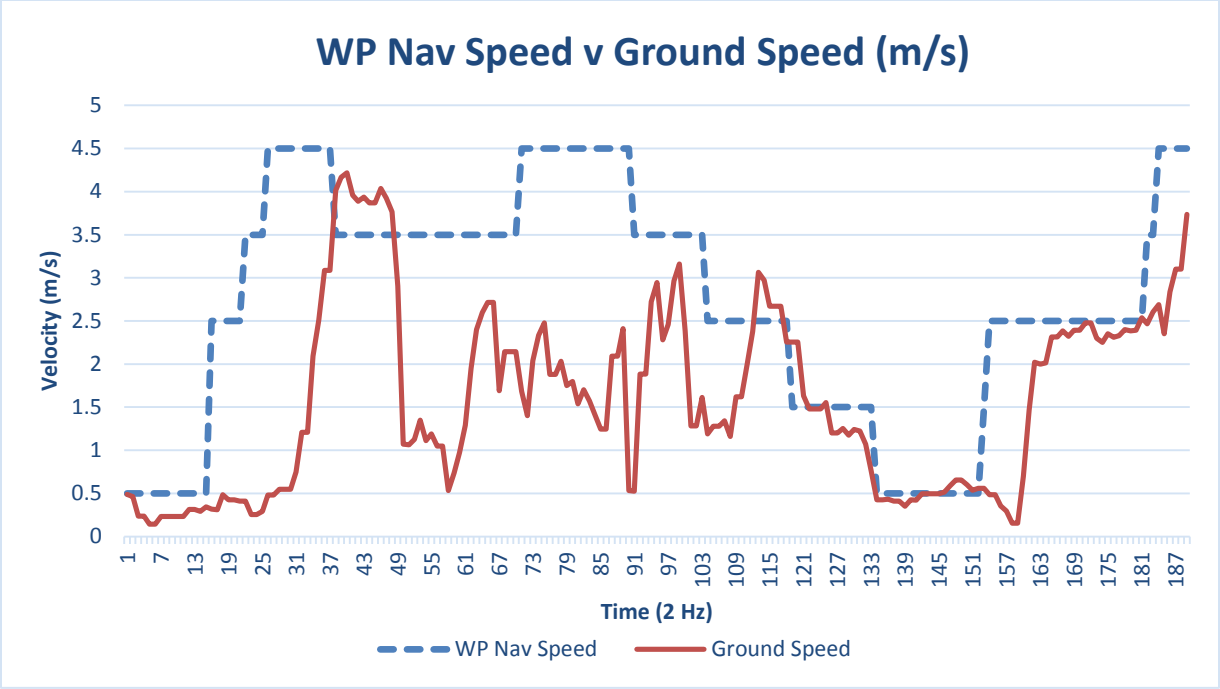

Figure 40. Waypoint Nav Speed versus Ground Speed (Plot 2)

The position error, is the difference in the actual separation distance from the desired separation distance. The follower's position error throughout the flight can be seen in Figure 41. This is the deviation from desired offset of 3m. The desired position error value is 0m; when the position error is positive the follower quad is too far from the leader and when the position error is negative, the follower is too close to the leader.

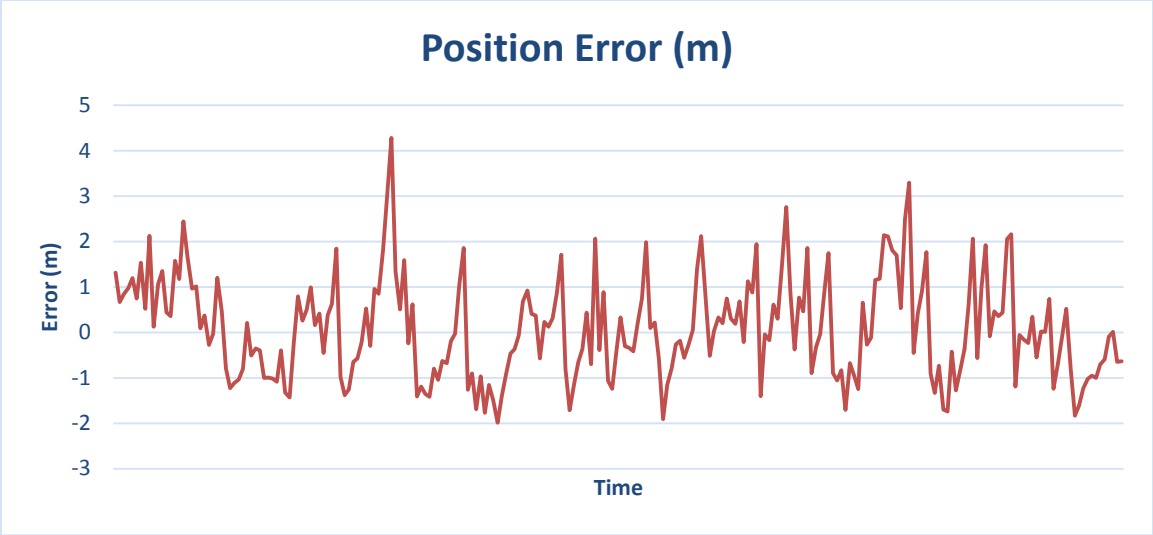

Figure 41. Follower Position Error

With the sonar now playing an active role in the system. The results were much improved over the previous flight tests. The RMSD was reduced by 56.5% from the initial 3m flight test and the average separation distance was reduced from 6.51m to 3.08m. The results are shown in Table 9.

Table 9 Final 3m Flight Test Results

Final 3m Flight Test	
Average Separation Distance	3.08m
Average Position Error	0.896m
Standard Deviation	1.11m
RMSD	1.11m

With the system architecture functioning as designed, it was time to compare these flight test results with the flight test results accomplished by Gray [5].

4.4 Flight Test Results Comparison

The flight test results of this research were compared to the previous flight tests without onboard sensors to quantify how much the COTS sonar sensors increased the cohesion of the formation flight by reducing the follower's position error. Since Gray's flight tests use the same guided position algorithm, the same X-8 airframes, but no onboard sensor for real time distance error measurement, this assessment shows how position error was effected by incorporating the sonar sensor.

For this evaluation, the position error from Gray's flight with the least amount of position error was selected [5]. The complete results table from Gray's work is shown in Table 10 [5]. This research used the distance error root mean square (DRMS) from test 2. Test 2 was selected because it had the lowest position error of the six tests that Gray completed using two X-8 quadrotors [5].

Table 10 Gray's X-8 Quadrotor Results [5]

	Forward Error (m)			Right Error (m)			Position Error (m)		
Test	μ	σ	DRMS	μ	σ	DRMS	μ	σ	DRMS
1	-1.76	1.58	1.94	0.60	1.62	1.41	3.45	1.11	2.97
2	-1.80	1.18	1.21	0.70	0.85	0.62	3.08	0.63	1.77
3	-1.00	1.33	1.34	0.49	1.29	1.11	2.85	1.32	2.53
4	-2.92	1.92	2.64	-0.81	2.25	1.80	5.08	2.16	4.17
5	-2.54	1.56	2.44	-0.01	1.18	0.96	3.94	1.68	3.51
6	-2.44	1.78	2.21	-0.67	1.97	1.52	4.47	2.29	3.67

For visual flight test comparison, Figure 42 displays the position error of both Gray's flight and the 3m sonar flight [5]. The blue line is the position error of the follower X-8 quad during Gray's test, the red line in the position error of the follower X-8 quad during

the sonar test, the dashed black lines represent the mean position error of each flight test. Note that the desired position error is 0m for both tests. From this figure, Gray's flight test resulted in a mean position error of 3.08m and was always too far away from the leader. At no time during Gray's test was the position error negative, indicating that the follower was too close. The 3m sonar flight test resulted in a mean position error of 0.89m.

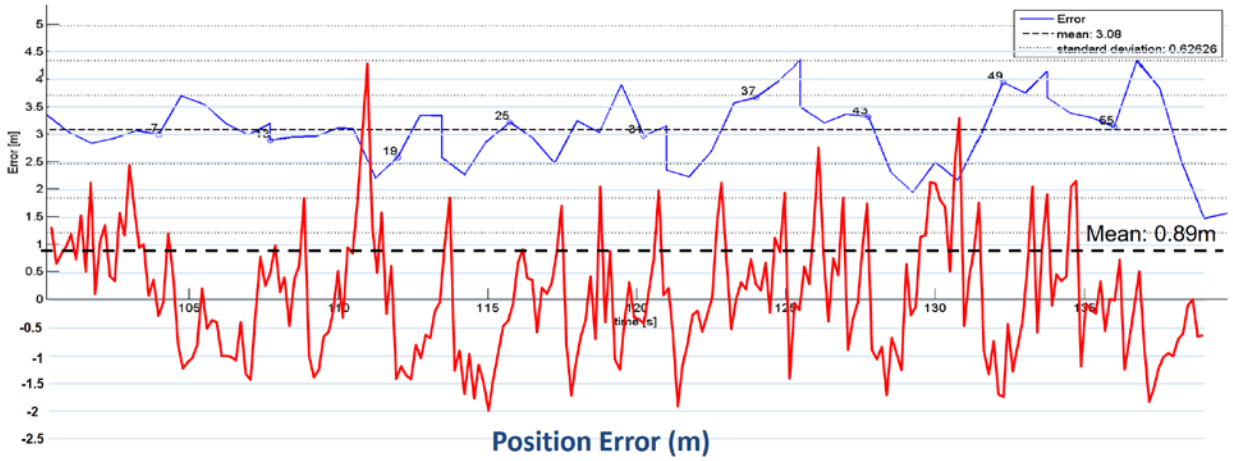

Figure 42. Position Error Comparison

This research effort was able to reduce the RMSD by 37.3% and the average position error by 70.9% when compared to Gray's flight test [5]. These results are tabulated in Table 11.

Table 11. Flight Test Comparison

	Gray	Sonar	% Error Reduced
RMSD (m)	1.77	1.11	37.3
Average Position Error (m)	3.08	0.896	70.9

V. Conclusions and Recommendations

5.1 Chapter Overview

This chapter answers the initial research questions, discusses the system limitations, outlines the recommendations for future research, and finally explores what steps would be needed or what technology gaps exist in current COTS components to allow this low-cost system to be implemented.

5.2 Research Questions Answered

The research questions that this investigation initially set out to answer are listed and discussed in this section.

- **What onboard sensors are available with low weight and low power consumption for use in UAV swarms?**

  Available sensors include optical, sonar, and light detecting and ranging (LiDAR). Each of these sensors have strengths and weakness that were discussed in detail in chapter 2, section 2.4. Sonar was chosen due to its wider beam width, and ease of integration with Pixhawk autopilot. This low power sonar was extremely limited in range. For this research, the sonar was only able to reliably detect the leader out to a distance of 4.5m. The sonar was extremely effective at close range, as seen in Figure 41, the sonar and algorithm never allowed the follower to get closer than 1m from the leader. Unfortunately, the sonar's limited detection range allowed the leader to drift far away from the follower. At one point, the follower was just over 7m away from the leader. Since the sonar could

not detect the leader at these distances, no change in the follower's velocity was commanded. Ideally the sensor range would need a to be doubled to allow the sensor algorithm adequate time to adjust the follower's velocity.

- **What swarm algorithm modifications are required to incorporate onboard sensing?**

  The onboard sensing algorithm was run separate from the guided positon algorithm directly in Mission Planner. This was done to allow the sonar algorithm to run at a higher frequency and permit the sonar to constantly be searching for the lead vehicle. This architecture required an extra script to be ran to control the sensor package subsystem. One limitation with this setup is that DroneKit would not run in conjunction with MissionPlanner. MavProxy was used to split the follower's communication stream to DroneKit and MissionPlanner. Using MavProxy as the middle man, allowed DroneKit and MissionPlanner to run at the same time.

- **How are the onboard sensors integrated with the Pixhawk autopilot?**

  The sonar can integrate directly with the Pixhawk either through the ADC (Analog Digital Converter) port or through the I2C port. For this research both the ADC and the I2C port were used during ground testing, but the I2C port was utilized for all flight test. This integration was straight forward. The MissionPlanner GUI has built in functionality to work directly the MaxSonar.

- **How can the Pixhawk autopilot take advantage of the onboard sensor data?**

  The Pixhawk used the onboard sensor (sonar) data to adjust the X-8s velocity in an attempt to keep it at the desired separation distance. The sonar range reading was divided into three sectors; attract, monitor, and repel. Each sector provided a different response for the Pixhawk. This is discussed in detail in chapter 3, section 3.4.3. A limiting factor in this setup is that it is assumed that the follower is always directly behind the leader. Future iterations could use the gimbal position to get a vector to the leader and adjust more than just the follower's velocity. This approach would require a sensor with a narrower beam width.

- **How much can onboard sensors reduce offset distances and desired separation distance errors?**

  With the use of the sonar sensors, the UAVs where able to fly at the same altitude, which was not attempted in the previous flights conducted by Gray for fear of a midair collision. The use of the sonar reduced the RMSD position error by 37.3% when compared to the previous flight test conducted by Gray [5]. On average Gray's position error was 3.08m, whereas the average position error for this research was just 0.896m, an improvement of 70.9%. This result was discussed in detail in section 5.2 of this chapter. Even with the limitations of this system, the onboard sensor demonstrated an improvement in the follower's position error.

5.3 Recommendations for Future Research

There are many areas where this research can be further refined and developed. However, this research effort can most greatly be influenced in the following two areas: vehicle to vehicle communication and increasing the range of the onboard sensor.

One limiting factor of this research was the frequency at which the control loops could be run. The control loops for this research were limited to approximately 8 hz. A faster control loop frequency would lead to more precise navigation and could reduce position error, especially during turns. The lag associated with passing all the communications through the ground station could be reduced by allowing the vehicles to communicate directly with each other. This would require an additional, low cost, COTS, onboard processor, like a BeagleBone, on each vehicle.

The other limiting factor in this research was the sonar. The sonar had a max effective range of approximately 4.5m on the lead quadrotor. This meant that the vehicles had to be close together before the sonar could influence the formation. By increasing the max effective range of the onboard sensor, the control algorithm could more precisely regulate the separation distance, particularly when the closure rate is high or when the vehicles drifted apart. A more powerful sonar could be a viable solution.

5.4 System Implications

This research provided a proof of concept that a low-cost solution for UAV formation flight and swarming is within reach with current COTS technology. This low-cost COTS formation flight architecture could allow groups of small, attritable, UAVs to perform formation flights and swarming behavior. This particular architecture is suitable

for missions that allow or tolerate position errors of approximately one meter. To fully migrate this research architecture into the Air Force's arsenal there are a few performance shortfalls that would need to be addressed. The shortfalls include the sonar sensor, vehicle to vehicle communication, and GPS error.

For this architecture to be fully mission capable, a more robust sensor would be required. The greatest limiting factor of the sensor was the range it could accurately detect the lead UAV. The sonar sensor that was flown for this research effort was limited to a 4.5m standoff distance. Once the leader was beyond this distance, the sonar was unreliable. With the sensor being unreliable over 4.5m, no velocity changes were commanded during this time and thus the position error was not being corrected. The ideal sensor could detect the leader out to 10m and would have a wider beam width or use multiple sensors to allow the leader to be tracked no matter its position relative to the follower.

The position error of the follower vehicle was greatly influenced by the latency of the communication architecture. The system latency was measured by Gray to be 0.46s [5]. This latency was most noticeable when the leader completed a turn. By allowing the UAV's to communicate directly with each other, the position error could be reduced in two ways. First, the leader's GPS position would be more accurate. By the time the leader's positon is transmitted to the ground station then analyzed to calculate the correct follower position, the leader is no longer in that position and the calculated follower position is inherently incorrect. This is shown graphically in Figure 31. The calculated desired guided position had a maximum error of 0.9m during a turn. Second, the control loop could be run at a higher frequency to allow for more accurate follower commanded

position. With the leader's GPS position being more accurate and the control loop running at a higher frequency, the followers position error and standard deviation would be reduced.

In addition to the communication latency resulting in desired GPS position error, the GPS itself has error associated with it. Both the leader's and the follower's GPS sensor error contributes to the position error of the follower. A Real Time Kinematic (RTK) GPS solution could prove to improve the followers position by reducing the error associated with both the leader's and follower's GPS. Current low cost, COTS, RTK GPS solutions are not ready to be integrated into a formation flight or swarm architecture. Ground testing has shown that these GPS units work well in stationary environments, but quickly lose the RTK solution when one of the receivers is placed on a moving platform. Once this technology gap is closed, an RTK GPS solution could prove to be invaluable to low cost formation flight and swarming.

Even with the limitations expressed in this section, this research did show that a formation flight or swarm architecture could be developed utilizing nothing but low-cost COTS components.

5.5 Summary

The objective of this research was to investigate the effects of onboard sensing on UAV formation flight cohesion by comparing flight test results with and without onboard sonar sensing capability.

The guided algorithm developed by Gray along with his flight testing results were used in this comparison [5]. From these results, it can be concluded that the sonar was

successful in increasing the cohesion of the formation flight by reducing the RMSD by 37.3% and the average position error by 70.9%. While this system is not perfect, this research did demonstrate that UAV formation and swarm cohesion can be improved using low cost, commercial off the shelf sonar sensors.

Bibliography

[1] D. Lamothe, “Veil of secrecy lifted on Pentagon office planning ‘Avatar’ fighters and drone swarms,” *The Washington Post*, 2016. [Online]. Available: https://www.washingtonpost.com/news/checkpoint/wp/2016/03/08/inside-the-secretive-pentagon-office-planning-skyborg-fighters-and-drone-swarms/.

[2] Y. Gu, G. Campa, B. Seanor, S. Gururajan, and M. R. Napolitano, “Autonomous Formation Flight – Design and Experiments,” *Aer. Veh.*, pp. 233–256, 2009.

[3] R. Hao, D. Luo, and H. Duan, “Multiple UAVs mission assignment based on modified Pigeon-inspired optimization algorithm,” *2014 IEEE Chinese Guid. Navig. Control Conf. CGNCC 2014*, pp. 2692–2697, 2015.

[4] K. Osborn, “Swarming Mini-Drones: Inside the Pentagon’s Plan to Overwhelm Russian and Chinese Air Defenses,” *The National Interest*, 2016. [Online]. Available: http://nationalinterest.org/blog/the-buzz/swarming-mini-drones-inside-the-pentagons-plan-overwhelm-16135. [Accessed: 14-Dec-2016].

[5] J. Gray, “Design and Implementation of a Unified Command and Control Architecture for Multiple Cooperative Unmanned Vehicles Utilizing Commercial Off the Shelf Components,” Air Force Institute of Technology, 2015.

[6] S. Losey, “Gen. Mark Welsh sounds alarm on undermanned Air Force,” *Air Force Times*, 2015. [Online]. Available: http://www.airforcetimes.com/story/military/2015/12/01/welsh-sounds-alarm-on-undermanned-air-force/76617202/. [Accessed: 15-Jul-2016].

[7] “Unmanned Systems Integrated Roadmap FY2013-2038,” 2013.

[8] D. J. Nowak, I. Price, and G. B. Lamont, “Proceedings of the 2007 Winter Simulation Conference S. G. Henderson, B. Biller, M.-H. Hsieh, J. Shortle, J. D. Tew, and R. R. Barton, eds.,” pp. 1315–1323, 2007.

[9] C. W. Reynolds, *Flocks , Herds , and Schools : A Distributed Behavioral Model*. 1987.

[10] J. N. Kaiser, “Effects of Dynamically Weighting Autonomous Rules In an Unmanned Aircraft System (UAS) Flocking Model.”

[11] J. L. Lambach, “Integrating UAS Flocking Operations With Formation Drag.”

[12] C. W. Reynolds, Z. Chao, L. Ming et al. “Outdoor flocking and formation flight with autonomous aerial robots,” *… Robot. Syst. 2005.(IROS 2005). …*, vol. 9, no. 5, pp. 287–300, 2013.

[13] S. K. Lee and J. McLurkin, “Distributed cohesive configuration control for swarm robots with boundary information and network sensing,” *IEEE Int. Conf. Intell. Robot. Syst.*, no. Iros, pp. 1161–1167, 2014.

[14] Z. Chao, L. Ming, Z. Shao-lei, and Z. Wen-guang, “Collision-free UAV formation flight control based on nonlinear MPC,” *Int. Conf. Electron. Commun. Control*, pp. 1–18, 2011.

[15] D. Luo, T. Zhou, and S. Wu, “Obstacle avoidance and formation regrouping strategy and control for UAV formation flight,” *IEEE Int. Conf. Control Autom. ICCA*, pp. 1921–1926, 2013.

[16] A. Kim, Seungkeun; Kim, Youdan; Tsourdos, “Optimized Behavioural UAV Formation Flight Controller Design,” in *European Control Conference*, 2009.
[17] M. Clement, “From Zero to Fifty Planes in Twenty-Seven Minutes,” *DIY Drones*, 2015. [Online]. Available: http://diydrones.com/profiles/blogs/from-zero-to-fifty-planes-in-twenty-seven-minutes. [Accessed: 21-Jan-2016].
[18] A. Zarandy, T. Zsedrovits, B. Pencz, M. Nameth, and B. Vanek, “A novel algorithm for distant aircraft detection,” *2015 Int. Conf. Unmanned Aircr. Syst. ICUAS 2015*, pp. 774–783, 2015.
[19] N. Cen, K. Cheng, and B. Fidan, “Formation control of robotic swarms based on sonar sensing,” *ISSNIP 2009 - Proc. 2009 5th Int. Conf. Intell. Sensors, Sens. Networks Inf. Process.*, pp. 31–36, 2009.
[20] “How does LiDAR work? The science behind the technology,” 2016. [Online]. Available: http://www.lidar-uk.com/how-lidar-works/. [Accessed: 05-Aug-2016].
[21] *Pixhawk Autopilot Quick Start Guide*. 3D Robotics, 2014.
[22] “Using Python Scripts in Mission Planner,” 2016. [Online]. Available: http://ardupilot.org/planner/docs/using-python-scripts-in-mission-planner.html. [Accessed: 30-Jun-2016].
[23] “Mission Planner,” 2016. [Online]. Available: http://ardupilot.org/planner/docs/mission-planner-overview.html. [Accessed: 05-Aug-2016].
[24] “3DR X8-M Specifications,” p. 25.
[25] “I2CXL - MaxSonar ® - EZ™ Series,” 2012.
[26] B. Pendleton and M. Goodrich, “Scalable Human Interaction with Robotic Swarms,” *AIAA Infotech@aerosp. Conf.*, pp. 1–13, 2013.
[27] I. D. Couzin, J. Krause, R. James, G. D. Ruxton, and N. R. Franks, “Collective memory and spatial sorting in animal groups.,” *J. Theor. Biol.*, vol. 218, no. 1, pp. 1–11, 2002.

Appendix A

Sonar Script

```
import MissionPlanner

Script.ChangeParam('WPNAV_SPEED',250)              #Initializing with 2.5m/s
NAV speed

sonar = cs.sonarrange                                  # Getting first Sonar
Reading
#print 'Sonar Distance: %s' %sonar

Script.SendRC(6,1380,True)                         #Centering gimbal
Script.SendRC(7,1400,True)
Script.Sleep(500)

Minairspeedms = 0                                  # Setting Min speed to 0
TrimAirspeed = Script.GetParam ('WPNAV_SPEED')     #Getting paramters for NAV
Speed
Maxairspeedms = 5                                  # Setting max speed to 5

#print 'Max Airspeed (m/s): %s' %Maxairspeedms
MaxAirspeed = Maxairspeedms * 100                   #Convert min/max airspeed to
cm/s
#print 'Original Trim Airspeed: %s' %TrimAirspeed
#print 'Minimum Airspeed (m/s): %s' %Minairspeedms
Minairspeed = Minairspeedms * 100

while True:
   sonar = cs.sonarrange

   while (sonar >=4.5):    #Sonar Using Gimbal to Search

      sonar = cs.sonarrange

      if sonar >= 4.5:
         Script.SendRC(6,1330,True)
         Script.SendRC(7,1400,True)
         Script.Sleep(300)
         sonar = cs.sonarrange
      if sonar >= 4.5:
         Script.SendRC(6,1330,True)
         Script.SendRC(7,1500,True)
         Script.Sleep(300)
         sonar = cs.sonarrange
      if sonar >= 4.5:
         Script.SendRC(6,1380,True)
         Script.SendRC(7,1500,True)
         Script.Sleep(300)
         sonar = cs.sonarrange
      if sonar >= 4.5:
         Script.SendRC(6,1400,True)
         Script.SendRC(7,1500,True)
         Script.Sleep(300)
         sonar = cs.sonarrange
      if sonar >= 4.5:
         Script.SendRC(6,1400,True)
         Script.SendRC(7,1400,True)
```

```
      Script.Sleep(300)
      sonar = cs.sonarrange
   if sonar >= 4.5:
      Script.SendRC(6,1380,True)
      Script.SendRC(7,1400,True)
      Script.Sleep(300)
      sonar = cs.sonarrange
   if sonar >= 4.5:
      Script.SendRC(6,1330,True)
      Script.SendRC(7,1400,True)
      Script.Sleep(300)
      sonar = cs.sonarrange
   if sonar >= 4.5:
      Script.SendRC(6,1330,True)
      Script.SendRC(7,1300,True)
      Script.Sleep(300)
      sonar = cs.sonarrange
   if sonar >= 4.5:
      Script.SendRC(6,1380,True)
      Script.SendRC(7,1300,True)
      Script.Sleep(300)
      sonar = cs.sonarrange
   if sonar >= 4.5:
      Script.SendRC(6,1400,True)
      Script.SendRC(7,1300,True)
      Script.Sleep(300)
      sonar = cs.sonarrange
   if sonar >= 4.5:
      Script.SendRC(6,1400,True)
      Script.SendRC(7,1400,True)
      Script.Sleep(300)
      sonar = cs.sonarrange
   if sonar >= 4.5:
      Script.SendRC(6,1380,True)
      Script.SendRC(7,1400,True)
      Script.Sleep(300)
      sonar = cs.sonarrange

#*******Sonar within range. Take avaerage of 10 readings over .45 seconds
if sonar < 4.5:
   sonar1 = cs.sonarrange
   Script.Sleep(50)
   sonar2 = cs.sonarrange
   Script.Sleep(50)
   sonar3 = cs.sonarrange
   Script.Sleep(50)
   sonar4 = cs.sonarrange
   Script.Sleep(50)
   sonar5 = cs.sonarrange
   Script.Sleep(50)
   sonar6 = cs.sonarrange
   Script.Sleep(50)
   sonar7 = cs.sonarrange
   Script.Sleep(50)
   sonar8 = cs.sonarrange
   Script.Sleep(50)
   sonar9 = cs.sonarrange
   Script.Sleep(50)
   sonar10 = cs.sonarrange

   sonar = (sonar1 + sonar2 + sonar3 + sonar4 + sonar5 + sonar6 + sonar7 +
```

```
sonar8 + sonar9 + sonar10)/10  # Average Sonar reading over 0.25 seconds
      print 'Sonar Average Distance: %s' %sonar

   if sonar >= 3.25:
      currenttrimairspeed = Script.GetParam ('WPNAV_SPEED')
      newtrimairspeed = currenttrimairspeed + 100
      #print 'New Airspeed: %s' %newtrimairspeed
      if newtrimairspeed >= MaxAirspeed:
         print 'Trim Airspeed is at Maximum'
      if newtrimairspeed < MaxAirspeed:
         Script.ChangeParam('WPNAV_SPEED',newtrimairspeed)
         TrimAirspeed = Script.GetParam ('WPNAV_SPEED')
         print 'New Assigned Trim Airspeed: %s' %TrimAirspeed

   if 3.05<= sonar < 3.25:
      currenttrimairspeed = Script.GetParam ('WPNAV_SPEED')
      newtrimairspeed = currenttrimairspeed + 50
      #print 'New Airspeed: %s' %newtrimairspeed
      if newtrimairspeed >= MaxAirspeed:
         print 'Trim Airspeed is at Maximum'
      if newtrimairspeed < MaxAirspeed:
         Script.ChangeParam('WPNAV_SPEED',newtrimairspeed)
         TrimAirspeed = Script.GetParam ('WPNAV_SPEED')
         print 'New Assigned Trim Airspeed: %s' %TrimAirspeed

   if 2.95<= sonar < 3.05:
      currenttrimairspeed = Script.GetParam ('WPNAV_SPEED')
      newtrimairspeed = currenttrimairspeed
      print 'New Airspeed: %s' %newtrimairspeed
      Script.ChangeParam('WPNAV_SPEED',newtrimairspeed)
      TrimAirspeed = Script.GetParam ('WPNAV_SPEED')
      print 'Nav Speed Unchanged: %s' %TrimAirspeed

   if 2.75 <= sonar < 2.95:
      currenttrimairspeed = Script.GetParam ('WPNAV_SPEED')
      newtrimairspeed = currenttrimairspeed - 50
      #print 'New Airspeed: %s' %newtrimairspeed
      if newtrimairspeed <= Minairspeed:
         print 'Trim Airspeed is at Minimum'
      if newtrimairspeed > Minairspeed:
         Script.ChangeParam('WPNAV_SPEED',newtrimairspeed)
         TrimAirspeed = Script.GetParam ('WPNAV_SPEED')
         print 'New Assigned Trim Airspeed: %s' %TrimAirspeed

   if sonar < 2.75:
      currenttrimairspeed = Script.GetParam ('WPNAV_SPEED')
      newtrimairspeed = currenttrimairspeed - 100
      #print 'New Airspeed: %s' %newtrimairspeed
      if newtrimairspeed <= Minairspeed:
         print 'Trim Airspeed is at Minimum'
      if newtrimairspeed > Minairspeed:
         Script.ChangeParam('WPNAV_SPEED',newtrimairspeed)
         TrimAirspeed = Script.GetParam ('WPNAV_SPEED')
         print 'New Assigned Trim Airspeed: %s' %TrimAirspeed
```

Appendix B

Follower Script

```
#FlockingModeFollower Capt Robert McClanahan(Modified from Gray 2015 to Use
Drone Kit)
#   Gets location of leader vehicle and sets waypoints to make follower vehicle
follow at
#   a fixed offset distance
#
#   Prerequisits:
#       Mavproxy Running to split data to Drone Kit and Mission Planner
#
#   Notes:
#       for best results, update system time

import socket
import sys
import math
import time
from datetime import datetime
import re
from numpy import matrix
import numpy as np
from LLA_ECEF_Convert import LLA_ECEF_Convert
from multi_vehicle_toolbox import follower_pos
from dronekit import connect, VehicleMode, LocationGlobalRelative, Command,
mavutil

'''INIT PARAMS'''
#Follower offset parameters (relative to leader's body frame)
off_l1_s=1      #L1 lead time constant [s] for forward offset waypoint
off_r = 2       #radial distance [m] away from leader
off_theta = 0   #angle (deg) from -x axis (out of tail), CCW is (+) rotation
alt_agl_cmd=11  #alt agl [m] to be commanded, used in guided_pos

#timing prarameters
t_freq=8.0       #control loop frequency, must be faster than leader and float
(0.0)
freq_store=2.0    #frequency of storage of data to disk and must be float (0.0)
freq_print=1    #frequency of print statements (try to reduce this)

#other
msg_size=128    #size of msg to be passed

'''DRONEKIT INT'''
v_follower = connect('127.0.0.1:14550', wait_ready=True)# Connect to follower
vehicle in Dronekit via MavProxy Split

print "Follower Vehicle Object Created"

'''DATA FILE INIT'''
timestr = time.strftime("%m-%d-%Y_%H-%M-%S")    #date-time for file name
file_name='follower_gcs_tel_' + timestr         #file name appended with date
time
data_file = open(file_name, 'a')    #create txt doc to append to
msg_data='%s %s %s' %(off_r,off_theta,off_l1_s)
data_file.write(msg_data + '\n')
print 'telemetry file open'
```

```
'''CONNECTION INIT'''
#Setup TCP link with leader_server
Port = 50005    # Port to TX/RX to/from leader_server
IP = '127.0.0.1'      #Local Host IP
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
print 'socket created'
s.bind((IP,Port))    # Connect socket
print 'Bound to port ' + str(Port)

'''Main Loop'''
rc_ch=v_follower.channels         #Get Channel 5 Position
t_write=0   #forces first write to occure on start
t_print=0   #forces first print to occure on start
#Current location is pos0, next location is pos1
cmds = v_follower.commands
cmds.download()
cmds.wait_ready()

print 'starting control loop'
while True:
    #time.sleep(.1)

    try:
        #get current time for sleep...
        t1=time.time()

        if rc_ch['5'] > 1200:   #MANUAL MODE FAIL SAFE, will not store data
            v_follower.mode = VehicleMode("STABILIZE")
            print 'Channel 5: %s' %rc_ch['5']

            if time.time() - t_print > 1/freq_print:
                print "Follower Mode Set to Manual" +
str(datetime.now().time())

                t_print=time.time()

            time.sleep(0.01)

        else:
            print 'getting leader telem'
            #read leader tel from udp port
            tel_leader = s.recv(msg_size)   #get "lat(deg) lon(deg) alt(m)
gc(rad) v(m/s)"

            #manipulate leader tel to parse out lat,lon,alt,heading/gc,velocity
            pattern = re.compile("[ ]")         #Data patern (data seperated by
[ ] i.e space)
            param = pattern.split(tel_leader)   #split data based on data
patern

            pos_leader = np.array([float(param[0]),
float(param[1]),float(param[2])])
                                    #leader pos [lat(deg) lon(deg) alt(m)]
            heading_l = np.rad2deg(float(param[3]))     #leader ground course
(rad)
            v_l = float(param[4])                       #leader velocity (m/s)

            #calculate desired position
            off_l1=off_l1_s*v_l     #forward offset dist. ( [m] = [s] * [m/s])
            pos1_f=follower_pos(off_r,off_theta,off_l1,
```

```
                                pos_leader,heading_l)   #pos1_f = [lat(deg)
lon(deg) alt(m)]

            #Set new follower guided point
            guided_pos= LocationGlobalRelative(pos1_f[0],pos1_f[1]-
360,alt_agl_cmd)   #pos1_f[1]-360
            #print 'guided position: %s' %guided_pos
            if v_follower.mode != "GUIDED":     #if not already in guided...go
guided
                v_follower.mode = VehicleMode("GUIDED")

            v_follower.simple_goto(guided_pos) #send guided point
            cmds.upload()

            #get telemetry information for storage
            lat=str(v_follower.location.global_relative_frame.lat)
#latitude (9 bytes CHECK)
            lon=str(v_follower.location.global_relative_frame.lon)
#longitude (9 bytes CHECK)
            alt_asl = str(v_follower.location.global_relative_frame.alt)
#altitude above sea level (6 bytes CHECK)
            sonar = v_follower.rangefinder.distance         #Sonar reading
            wpnav = v_follower.parameters['WPNAV_SPEED']    #WP Nav Speed
            p=float(np.deg2rad(v_follower.attitude.pitch))  #pitch (rad) of
vehicle relative to NEU frame
            r=float(np.deg2rad(v_follower.attitude.roll))   #roll (rad) of
vehicle relative to NEU frame
            y=float(np.deg2rad(v_follower.attitude.yaw))    #yaw (rad) of
vehicle relative to NEU frame
            v_b= v_follower.velocity                #velocity in x dir relative
to body (CHECK)

            #determine gc relative to vehicle frame (NEU)
            v_b=np.array([[v_b[0], v_b[1], v_b[2]]])
            c_r=np.cos(r);  s_r=np.sin(r)
            c_p=np.cos(p);  s_p=np.sin(p)
            c_y=np.cos(y);  s_y=np.sin(y)
            R_v_b=np.array([    [c_p*c_y,               c_p*s_y,
-s_p    ],
                                [s_r*s_p*c_y-c_r*s_y,   s_r*s_p*s_y+c_r*c_y,
s_r*c_p ],
                                [c_r*s_p*c_y+s_r*s_y,   c_r*s_p*s_y-s_r*c_y,
c_r*c_p ]   ])
                                #rotation transform from vehicle to body
            v_v=np.dot(R_v_b.T,v_b.T)  #velocity vector relative to vehicle
(NEU) frame
            gc=np.arctan2( v_v[1],v_v[0] )  #ground course relative to NEU
(CHECK)
            v=np.linalg.norm(v_v)       #velocity of leader, used to calc L1
            t_tel=time.time()

            #if V is too slow use yaw as ground course
            if v < 1:
                gc= y
            cmds.upload()

            #build telemetry data str
            tel_msg_raw ='%s , %s , %s , %s , %s , %s , %s , %s  , %s ,  %s'
%(lat,lon,alt_asl,str(float(gc)),str(v),

str(pos1_f[0]),str(pos1_f[1]-360),str(pos1_f[2]), sonar, wpnav)
```

```
            #append data with unix time on a new line of data txt file
            if time.time() - t_write > 1/freq_store:
                msg_data='%s , %s' %(t_tel,tel_msg_raw)
                data_file.write(msg_data + '\n')
                t_write=time.time()

            #print update message
            if time.time() - t_print > 1/freq_print:
                print 'cmd sent & telemetry stored: ' +
str(datetime.now().time())
                t_print=time.time()

            #determine sleep time
            t2=time.time()
            t_remaining= ( 1/t_freq ) - ( t2 - t1 )
            if t_remaining > 0:     #sleep for remainder of this control cycle
                time.sleep(t_remaining)
            else:                   #the operations in the while loop took too
long
                print 't_freq is too high'

    except KeyboardInterrupt:   #only way to stop the ride
        data_file.close()
        break

    except:
        print "Unexpected error:", sys.exc_info()[0]
        data_file.close()
        break


# exit
s.close()
print 'End of Script'
```

Appendix C

Leader Script

```
#FlockingModeLeader Capt Robert McClanahan(Modified from Gray 2015 to use Drone
Kit)
#   Gets location request from follower and gives the leaders location and
heading
#
#   Prerequisits:
#       Drone Kit Open
#
#   Notes:
#       for best results, update system time

import socket
import sys
from droneapi.lib import VehicleMode
from droneapi.lib import Command
from droneapi.lib import mavutil
import numpy as np
import math
import time
from datetime import datetime
from LLA_ECEF_Convert import LLA_ECEF_Convert
from multi_vehicle_toolbox import follower_pos
from dronekit import connect, VehicleMode, Command, mavutil

'''INIT PARAMS'''
freq_control=4.0    #frequency of control loop, must be < follower, must be
float (0.0)
freq_store=2.0      #frequency of data storage, must be float (0.0)
freq_print=1.0      #frequency of printed updates, must be float (0.0)
msg_size=128        #size of msg to be passed

'''DRONEKIT INIT'''
v_leader = connect('com14', wait_ready=True, baud=57600)# Connect to lead
vehicle

print "Leader Vehicle Object Created"

'''DATA FILE INIT'''
timestr = time.strftime("%m-%d-%Y_%H-%M-%S")    #date-time for file name
file_name='leader_gcs_tel_' + timestr         #file name appended with date
time
data_file = open(file_name, 'a')    #create txt doc to append to
print 'telemetry file open'

'''CONNECTION INIT'''
#Setup UDP link with leader_server
Port = 50005    # Port to TX/RX to/from follower_client
IP = '127.0.0.1'      #Local Host IP
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)   # Create TCP socket
object
print 'socket created'
address=(IP,Port)


'''Main LOOP'''
```

```
t_write=0   #forces first write to occure on start
t_print=0
print 'starting control loop'
#Current location is loc0, next location is loc1

while True:
    try:
        #get current time for sleep...
        t1=time.time()

        #get telemetry information
        lat=str(v_leader.location.global_relative_frame.lat)
#latitude (deg)
        lon=str(v_leader.location.global_relative_frame.lon)
#longitude (deg)
        alt_asl = str(v_leader.location.global_relative_frame.alt)
#altitude above sea level (m)
        p=float(np.deg2rad(v_leader.attitude.pitch))    #pitch (rad) of vehicle
relative to NEU frame
        r=float(np.deg2rad(v_leader.attitude.roll))     #roll (rad) of vehicle
relative to NEU frame
        y=float(np.deg2rad(v_leader.attitude.yaw))      #yaw (rad) of vehicle
relative to NEU frame
        v_b=v_leader.velocity                           #velocity vectory (m/s)
relative to body
        t_tel=time.time()                               #time telemetry was
recieved

        #flush data to leader
        v_leader.flush()

        #determine gc relative to vehicle frame (NED)
        v_b=np.array([[v_b[0], v_b[1], v_b[2]]])
        c_p=np.cos(p);  s_p=np.sin(p)
        c_r=np.cos(r);  s_r=np.sin(r)
        c_y=np.cos(y);  s_y=np.sin(y)
        R_v_b=np.array([    [c_p*c_y,               c_p*s_y,                -
s_p    ],
                            [s_r*s_p*c_y-c_r*s_y,   s_r*s_p*s_y+c_r*c_y,
s_r*c_p ],
                            [c_r*s_p*c_y+s_r*s_y,   c_r*s_p*s_y-s_r*c_y,
c_r*c_p ]   ])
                            #rotation transform from vehicle to body
        v_v=np.dot(R_v_b.T,v_b.T)   #velocity vector relative to vehicle (NED)
frame
        gc=np.arctan2( v_v[1],v_v[0] )    #ground course (rad) relative to
North
        v=np.linalg.norm(v_v)                   #velocity of leader, used to
calc L1

        #if V is too slow use yaw (rad) as ground course
        if v < 0.25:
            gc= y

        #build telemetry msg to be a known length (msg_size)
        tel_msg_raw ='%s %s %s %s %s' %(lat,lon,alt_asl,str(float(gc)),str(v))
#build msg
        tel_msg=msg_size*' '
        if len(tel_msg_raw) < len(tel_msg):         #set msg size to known
length
            n_spaces=len(tel_msg)-len(tel_msg_raw)
```

```
            tel_msg=tel_msg_raw + n_spaces * ' '
        else:
            print 'err: udp message exceeds length. Increase msg_size'
            break

        #send leader telemetry to follower over UDP
        s.sendto(str(tel_msg),address)

        #append data w/ unix time on new line of data txt file, if 1/freq_store
has pased
        if time.time() - t_write > 1/freq_store:
            tel_msg_rawc = '%s , %s , %s , %s , %s'
%(lat,lon,alt_asl,str(float(gc)),str(v))
            msg_data='%s , %s' %(t_tel,tel_msg_rawc)
            data_file.write(msg_data + '\n')
            t_write=time.time()

        #print update message
        if time.time() - t_print > 1/freq_print:
            print 'telemetry sent & stored: ' + str(datetime.now().time())
            t_print=time.time()

        #determine sleep time
        t2=time.time()
        t_remaining= ( 1/freq_control ) - ( t2 - t1 )
        if t_remaining > 0:     #sleep for remainder of this control cycle
            time.sleep(t_remaining)
        else:                   #the operations in the while loop took too long
            print 'freq_control is too high'

    except KeyboardInterrupt: #only way to stop the ride
        data_file.close()
        break

    except:
        print "Unexpected error:", sys.exc_info()[0]
        data_file.close()
        break

# exit
s.close()
print 'End of Script'
```

Appendix D

Multi-Vehicle Script

```
'''
multi_vehicle_toolbox.py
    Calculations required for multi-vehicle operations
        1) flocking follower pos calculation
        2) comm relay relay vehicle midpoint pos calc
'''

import numpy as np
from LLA_ECEF_Convert import LLA_ECEF_Convert

def follower_pos(off_r,off_theta,off_l1,loc0_l,heading_l):
    #function   description:    determines the next desired location of the
follower
    #                           vehicle in lat lon alt (LLA)
    #Inputs:    off_r:          radial distance away from leader [m]
    #           off_theta:      angle (deg) from -x axis (out of tail), CCW is
(+) rotation
    #           off_l1:         distance the guided point is placed forward of
the desired
    #                           follower location
    #           loc0_l:         location of the leader at current incriment of
time
    #           heading_l:      heading of the leader at current incriment of
time
    #Outputs:   loc1_f:         current desired location of the follower


    #Follower loc1 relative to leader body frame
    off_theta+=180   #add 270 deg to make offset relative to east (+X axis for
math)
    off_theta=np.deg2rad(off_theta)
    loc1_f= off_r*np.array([np.cos(off_theta),np.sin(off_theta),0]) +
off_l1*np.array([1,0,0])


    #Follower loc1 relative to Local Level Frame (L, North-East-Down) frame
            #Heading is negative because +rotation of pix is -rotation in NED
frame
    cos_h=np.cos(np.deg2rad(heading_l))
    sin_h=np.sin(np.deg2rad(heading_l))
    R_BtoL=np.array([   [cos_h,     sin_h,  0],
                        [-sin_h,    cos_h,  0],
                        [0,         0,      1]  ])  #Rotation from body to
local
    loc1_f=np.dot(loc1_f,R_BtoL)
    loc1_f=np.array([loc1_f[1], loc1_f[0], -loc1_f[2]])   #NED to ENU

    #Follower loc1 from Local Level Frame (L, East-North-UP) to ECEF (E)
    phi= np.deg2rad(loc0_l[0])      #latitude of leader
    la= np.deg2rad(loc0_l[1])      #londitude of leader
    sin_la= np.sin(la)
    cos_la=np.cos(la)
    sin_phi= np.sin(phi)
    cos_phi=np.cos(phi)
    R_LtoE=np.array([   [-sin_la,  -sin_phi*cos_la,    cos_phi*cos_la   ],
```

```
                        [cos_la,   -sin_phi*sin_la,    cos_phi*sin_la   ],
                        [0,        cos_phi,            sin_phi          ]   ])
#Rotation from local to ecef


T_LtoE=LLA_ECEF_Convert(np.rad2deg(phi),np.rad2deg(la),loc0_l[2],'LLAtoECEF')
    loc1_f= np.dot(R_LtoE,loc1_f) + T_LtoE.T


    #Follower Location ECEF to lat lon alt (LLA)
    loc1_f= LLA_ECEF_Convert(loc1_f[0], loc1_f[1], loc1_f[2], 'ECEFtoLLA')

    return loc1_f

def relay_pos(pos_gcs_llh,pos_rem_llh):
    #function   description:    Calculates the midpoint between the GCS and
remote vehicle
    #                           to send the relay vehicle
    #
    #Inputs:    pos_gcs_llh:    pos of GCS in lat lon hae
    #           pos_rem_llh:    pos of remote vehicle in lat lon hae
    #
    #Outputs:   pos_rel_lla:    calculated pos of relay vehicle
    #
    #Notation:
    #           Remote vehicle: rem
    #           Relay vehicle:  rel
    #           Ground Control: GCS

    #convert pos of rem & gcs from llh to ecef
    pos_rem_ecef=LLA_ECEF_Convert( pos_rem_llh[0],pos_rem_llh[1],
                                   pos_rem_llh[2],'LLAtoECEF')

    pos_gcs_ecef=LLA_ECEF_Convert( pos_gcs_llh[0],pos_gcs_llh[1],
                                   pos_gcs_llh[2],'LLAtoECEF' )

    #calculate midpoint in ecef
    pos_rel_ecef=pos_gcs_ecef + 0.5*(pos_rem_ecef-pos_gcs_ecef)

    #convert pos of rel from ecef to llh
    pos_rel_lla=LLA_ECEF_Convert( pos_rel_ecef[0],pos_rel_ecef[1],
                                  pos_rel_ecef[2],'ECEFtoLLA')

    return pos_rel_lla
```

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From – To)*
23-03-2017	Master's Thesis	August 2015 – March 2017

TITLE AND SUBTITLE

Improving Unmanned Aerial Vehicle Formation Flight and Swarm Cohesion by Using Commercial Off the Shelf Sonar Sensors

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

McClanahan, Robert L., Captain, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)
Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way, Building 640
WPAFB OH 45433-8865

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENV-MS-17-M-202

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AFRL AEROSPACE SYSTEMS DIRECTORATE (RQ)
2130 Eighth Street
Wright Patterson Air Force Base, Ohio 45433-7765
(937) 938-4805 paul.fleitz@us.af.mil
ATTN: Paul Fleitz

10. SPONSOR/MONITOR'S ACRONYM(S)

AFRL/RQ

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

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

13. SUPPLEMENTARY NOTES
This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

14. ABSTRACT

Unmanned Aerial Vehicle (UAV) formation flight and swarming are active areas of research within the Department of Defense (DoD). The current use of low cost commercial off the shelf (COTS) components to architect UAV formation flights results in insufficient position accuracy of the UAVs in the formation. This research aims to demonstrate the cohesiveness of formation flights increases by using onboard sonar sensors to accurately measure the distance the follower UAV is from the leader UAV. This research effort reduced the RMSD by 37.3% and the average position error by 70.9% when compared to previous flight test.

15. SUBJECT TERMS
UAV, Swarming, formation flight

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 Jacques, AFIT/ENV
U	U	U	UU	98	19b. TELEPHONE NUMBER *(Include area code)* (937) 785-3355, ext 3329 (david.jacques@afit.edu)

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39-18