2016

Capabilities of Flight Controllers for UAV Group Flight

Brendan Sullivan
Lehigh University

Capabilities of Flight Controllers for UAV Group Flight

by

Brendan Sullivan

A Thesis

Presented to the Graduate and Research Committee

of Lehigh University

in Candidacy for the Degree of

Master of Science

in

Mechanical Engineering

Lehigh University

Department of Mechanical Engineering and Mechanics

May 2016

CERTIFICATE OF APPROVAL

This thesis is accepted and approved in partial fulfillment of the requirements for the Master of Science in Mechanical Engineering.

_____

Date

_____

Thesis Advisor, Dr. Terry J Hart

_____

MEM Department Chair, Dr. Gary Harlow

Acknowledgements

First and foremost, I would like to thank my advisor, Professor Terry Hart, who gave me the opportunity to work on such a project in a new and exciting field even though it was new to my area of expertise. Without his support I would not have been able to explore such interesting and less taught fields as controls and flight optimization. I would like to thank Picatinny Arsenal, specifically Leon Manole, who provided the direction and funding for this project. I also want to thank Professor Terry Hart for bringing on Zachary Rambo (an alumni of Lehigh University with great passion in the aerospace field) to aid in the building and testing of the UAV's. Without his expertise with drones and just hands-on engineering in general this project would not have progressed so far and so smoothly (even though it was still pretty rocky!). He literally taught me how to fly a drone, and was able to make my graduate studies at Lehigh that much more fun.

Last but not least, I have to thank my family for always being there for me and trusting me enough to provide for me at my 5+ years at Lehigh University. My parents, Mike and Cindy Sullivan, have always encourage me to push the limits and I don't think or hope they will stop in the near future.

CONTENTS

Abstract 1

1 Introduction 2

1.1 Project Background 2

1.2 Objectives 3

2 UAV Testbed 5

2.1 Mechanical Design 5

2.2 Flight Controller and Software 9

2.3 Pixhawk Architecture 12

3 UAV Testing and Flying 17

3.1 Mission Planner Data Log Analyzer 17

3.2 Mission Planner “Swarming” 25

4 Kalman Filtering 32

4.1 Kalman Filtering Implementation 32

4.2 Kalman Filtering Tuning . 36

5 Conclusion and Future Work 41

LIST OF TABLES

2.1 Hexacopter Specs 7

4.1 Tuning Parameters 37

LIST OF FIGURES

2.1 Hexacopter UAV 6

2.2 Quadcopter UAV 9

2.3 Pixhawk Connection Schematic for UAV 11

2.4 Person to UAV Block Structure 13

2.5 Ground Station to UAV Block Structure 13

2.6 Flight Stack Block Schematic 15

2.7 Hexacopter “X” Motor Layout 16

3.1 Altitude Analysis 18

3.2 Roll Analysis 18

3.3 Vibrations 19

3.4 Compass/Motor Interference 20

3.5 UAV Compass Mapping 22

3.6 X and Y Velocities 23

3.7 Mission Planner’s “Swarming” Command Setup 27

3.8 Programmed Waypoint UAV Flight 28

3.9 Waypoint Action Item List 28

3.10 Mission Planner Representation of UAV 29

4.1 Earth Velocity Measurements 35

4.2 Earth Position Measurement 35

4.3 Body Magnetic Field Flight Data Logs 38

4.4 Magnetometer In-Flight Noise Levels 39

4.5 GPS Location (Raw) 39

4.6 GPS Location (Filtered) 40

ABSTRACT

With the advancement of single UAV flight control there is a clear understanding of the importance for future group UAV distributed control. This will in turn lead to scenarios of "smart" communication between UAV teams. This hierarchal chain reaction type control of UAV's will provide more enhanced real time flight pattern optimizations without the slow interactions of a UAV to a computer (aka a human). By relaying to just the "Master" from the ground station to switch trajectories the human interaction never needs to go any further to update the flight formations of the rest of the group members. This type of "swarming" with UAV's is only possible with the correct hardware and software improvements. This is especially true when the trend for UAV "groups" are to be higher in number and therefore much smaller (the size of an iPhone).

Apart from the already existing ways to manipulate flight paths – hardware including better GPS locating and new censoring technologies for collision/spatial recognition – software limitations are apparent to be the next large hurdle. To accomplish such interactions between UAV's, optimal flight patterns must be attained before any inter-communication can be implemented. Current designs follow the traditional PD/PID control schematics, but these lack the requirements to correct for real-world disturbances. Kalman filtering control design on the existing architectures of the Ardupilot and Lisa/S flight controllers was implemented to produce the most accurate flight paths of the UAV's.

CHAPTER 1

Introduction

1.1 Project Background

The use for UAV's is becoming prevalent in many different industries and for many different jobs. The drastic reduction in costs needed to build, implement and run UAV's coupled with the replacement for direct human contact makes UAV's a great resource. Other projects at Lehigh University that pertain to the birth of this undertaking deal with decreasing the size and payload capacity of UAV's to be launched quickly and effectively from the ground. In particular the group is trying to make a UAV that can fit inside a 40mm and 60mm tube, launched from said tube (aka endure a lot of force upon launch), open up in flight and start its task. Apart from the mechanical/aerodynamic design needed for creating such a projectile the next steps include actually having the UAV be as effective as possible during flight. The future of such a project is what this thesis set out to explore.

Once optimized UAV's can be launched from any platform and in any situation as quickly as is required. Lack of real time active response among UAV's to changing environments or mission parameters is not optimal. Constant communication between a human(s) and several different UAV's in a formed group can be time detrimental and impractical. The obvious progression of such technology is to create interactions and communication between the UAV's, which can best be described as autonomous flight or

known in the technical world as swarm behavior (*abbreviated throughout also as swarming).*

Significant prior research and many scholarly articles can be found on the dynamics of flight controls. They mostly deal with controlling changing parameters with flight conditions of UAV's. There have been very few approaches on an autonomous level of communication between these vehicles, and these only involve computer interaction to each individual UAV not computer to one UAV and then UAV to UAV interactions.

1.2 Objectives

The long term goal of this project is to create a team of UAV's that are able to be easily deployable in the field on a minutes notice. These UAV's will communicate amongst each other to realize a central mission, designated to the "Master" of the group by a human controlled ground station. This overall plan is estimated on a 4-5 year timeline realized only upon future technological progress of software and hardware. Such advancements include flexible electronics, higher resolution and further developed tech for collision avoidance, etc. Work that can be more solidified by myself in the shorter time period of 1.5 years deals with testing the theory of swarming amongst UAV's by building a UAV testbed of 4 hexacopters.

While mathematically there are plenty of theories and equations governing swarming or particle swarm optimization, not to mention examples in the real world (most notably with birds), there is still a large hurdle to apply this technique to flight programming/controls. This is mainly due to the fact that implementing these governing

equations/algorithms to mimic swarming are highly process intensive. The first step of this project was to test the current systems that had supposed swarming capabilities. After understanding the limitations on the current technology either corrective algorithms would be written or a new process to resemble swarming would be created. This process took roughly .5 to 1 year and ended in changing the focus of this thesis.

As the project progressed it became quite evident that swarming was a long ways away from being implementable in flight patterning. What did become apparent is the lack of a true architecture in place that could support certain more advanced control algorithms for flight path optimization. Current systems use almost a multitude of different hardware/software components to act as a "complete" system. Better put, several different very basic methods act as buffers and redundancies for and to each other (accelerometers coupled with GPS, etc.). The focus of this thesis changed to look into analyzing the control systems behind "how" the hexacopters fly and to improve upon those by applying Kalman filtering. This would be accompanied at the same time by understanding what it takes to create a ~.2 scale of the created hexacopters.

CHAPTER 2

UAV Testbed

2.1 Mechanical Design

The first step in analyzing UAV group flight is to actually build a platform for testing such devices. The UAV testbed was chosen to consist of 4 hexacopters, one of which is pictured below in *Figure 1.1*. The first consideration for building these hexacopters was making them small enough to easily house the components but not be too much to handle while transporting and flying. The hexacopters also needed to model what an actual 60mm tube launched UAV could be down the road, so a roughly 5-scale model was chosen. The symmetric frame design made it so placing the center of mass was easier, there was no need to worry about the frame itself attributing to the mass. The frame shape also allowed for the easy attachment and support structure for 6 props. More specifically, a 6 prop UAV – hexacopter – was chosen to provide redundancies in case of inflight complications. If any of the props broke (which happened quite often) the individual controlling the UAV could compensate with the other props to land the vehicle, and actually still fly if wanted.

Figure 2.1: Hexacopter UAV

The hexacopter has a relatively small air-frame, 260mm class, because it doesn't have to hold much; a small camera, a flight controller, and a battery. The frame of each hexacopter was comprised of two 3D printed plates. The two plates functioned to provide easy placement of all components (and subsequent movement when needed). Another reason for having two plates on top of each other is allowing certain components – the flight controller and radio controller – to be on separate "levels" for the different frequencies to interfere the least. This also allowed for the compact battery fitting on the frame by being attached below the plates by Velcro. Another separate attachment feature (shown projecting out ~4 inches from the top of the hexacopter in figure 1.1) needed to be added later on to deal with the interference from the GPS "puck" signal. 3D printing everything kept costs low and made it very easy to print in different colors, and this was extremely important as many times during flight 2 people would have to watch 4 hexacopters and therefore recognizing each one by bright colors (red, green, white, and

blue). 3D printing made it very easy to change the design of the hexacopter. At one point after some testing we decided to alter the shape of the frame, and it was a very quick, inexpensive fix.

A complete list of components and specs of the UAV are listed in *Table 1.1* below. The Multistar motors were the best at the time for the size, the props almost touch and provide the most pull from these motors. All other motors were either too heavy, or too tiny and wouldn't produce enough thrust. 6x3 props were chosen because they provided easier take off and we didn't really care about having a faster speed, or else we would've used a prop with more thrust; like a 4.7x.4.7 prop. The battery was compact enough to easily fit on the air-frame of the hexacopter but strong enough to provide plenty of flight time, which was important at points for testing all hexacopters at once on a planned loop course when leader-follower schema was in place (and therefore it took some time to set up and actually complete the course).

Hexacopter Specs	**Vendor**	**Stats**
Motors (6)	Multistar	2206 2150 KV
ESC's (6)	Multistar	10 amp V2 with BLheli firmware
Props (6)	Generic	6x3 2-Blade
Battery (1)	Zippy	3S 2200mah Li-Po
Radio Controller	Pixhawk	2.4 GHz Spread Spectrum
Radio Transmitter	Spektrum DX6i	2.4 GHz, 6-channel transmitter
Flight Controller	Pixhawk	PX4 w/ GPS & Telemetry
Static Thrust	-	2760 g Total
Flying Weight	-	950 g
Avg. Flight Time	-	10 minutes
Top Speed	-	16 m/s

The second main consideration was making the hexacopters robust enough, through design, so they could survive plenty of crashes. As learning how to fly UAV's was new to myself and takes a "slight" learning curve to get used to, this aspect was important. The motors and props were attached by poplar wood arms. These provided enough rigidity to not flex in flight but offered the desired flexibility when crashing where they would either distribute the forces or take all the damage, splinter, and break off without other parts of the hexacopter being ruined. Not to mention the wooden arms are cheap and easy to replace. 12 screws held the entire hexacopter together except for the motor mounts, where 6 screws were used.

When Lisa/S testing started another UAV was built (*Figure 2.2 below)*, this time it was a simple quadcopter kit that was bought off the shelf. The design of which could easily hold all Pixhawk and Lisa/S components as the original testbed and in this case it was made to hold a larger battery and house larger props. This UAV allowed for longer flight times and quicker speeds. This quadcopter was built to transfer from the Pixhawk to the Lisa/S and to understand how the Lisa/S flight controller works.

Figure 2.2: Quadcopter

While both designs lend well to robustness and adaptability, which was needed several times, in the long run they actually became a hindrance. After learning about all the hardware components and how they interact with each other and affect the flight of the hexacopter and more so understanding the lags/limitations associated with the parts it became apparent that the system as a whole was nowhere near advanced enough. To truly test the effectiveness of varying the flight controller parameters for the flight control architecture much more advanced system hardware would be needed to compliment the software.

2.2 Flight Controller and Software

Choosing to use the Pixhawk PX4 flight controller was one of the first and foremost decisions that was made when building this testbed. The Pixhawk PX4 is an

inexpensive all in one unit; housing FMU (Flight Management Unit) and IO (Input/Output module) aspects. This flight controller was chosen because it is widely used and already has autonomous flight capabilities with a reasonably proven "beta" swarming setting. It was quickly learnt that when Pixhawk advertises "swarming" all they really mean is that you can control multiple UAV's for extended autonomous flight, no actual inter-communication but still multiple computer-drone interactions. The Pixhawk is also very easy to "plug and play," as you can see in *Figure 2.2* below. It is already set up to work directly with "ground station" simulating computer programs such as Mission Planner, where you control the autonomous flight features and paths for the UAV's.

The Pixhawk also has a solid architecture for connecting with any of the other hardware components that were required; radio controller, GPS, etc. The Pixhawk turned out to be a perfect fit for the testbed in understanding how the system works. Down the road the Lisa/S was swapped in because of its size and the future projects it can be utilized in, compared to the larger, bulky and user friendly designed Pixhawk.

The Lisa/S flight controller is optimal for very small scale operations, it is 20x20x5 (mm) and only weighs 2.8g. The Lisa/S while being smaller still has the capabilities to sustain autonomous flight (the company advertises its autonomous feature heavily) and other than not having as much processing power and architecture for easy adaptability as the Pixhawk is very similar to it. The point of research to look into the Lisa/S was to examine whether or not it had the capabilities to add components to its flight control architecture.

Luckily in the new and rapidly growing field of at home UAV's most all of the source codes of these components are open source. Both the Pixhawk and Lisa/S are open source and this helped greatly in working with the code. The companies of both flight controllers willingly lend the code for their consumers, and sometimes contributors can even end up providing code updates and fixes for them. Throughout the whole project the multitude of questions that needed to be answered were done by peers through forums and support channels.

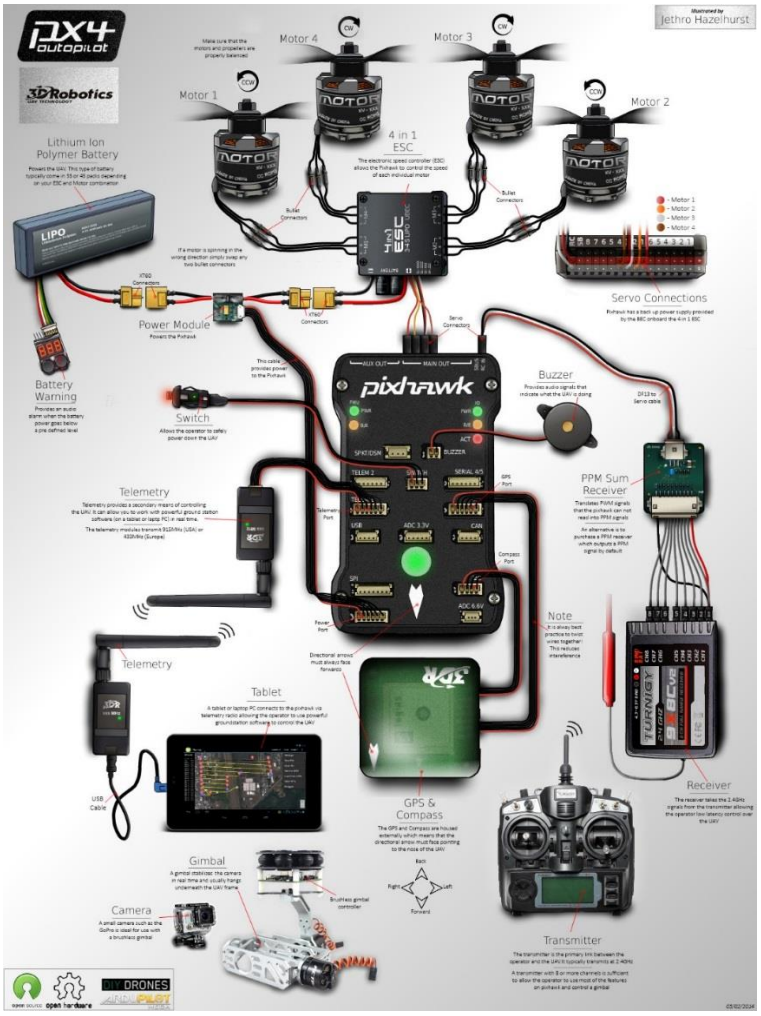

Figure 2.3: Pixhawk Connection Schematic For UAV

The flight controller is the medium between the UAV flying and inputs from the user's handheld transmitter. In the case of autonomous flight the flight controller communicates with a ground station (running on a laptop) and for this research Mission Planner was chosen. Another very similar program that could have been used is QGroundControl. Mission Planner is essentially the ground station to communicate with the UAV. The setup and design is very straightforward and set up to be extremely user friendly. The main aspects of MP that made it the perfect program are:

- It is made to work with the Pixhawk flight controller
- Can quickly setup, configure, and tune the parameters of your UAV for performance
- Can plan, save and load autonomous flight paths with waypoints on a GPS controlled map into the UAV
- Allows you to download and analyze all of the logs created from autopilot
- Allows you to monitor all of the vehicle's status while in autopilot mode

Mission Planner ended up being very helpful in seeing what types of latencies were inherent in the systems' components and allowed us to better analyze changes to the flight controls later on.

2.3 Pixhawk Architecture

The flight controller is the medium between the UAV flying and inputs from the user's handheld transmitter. In the case of autonomous flight the flight controller communicates with a ground station. The high-level software architecture is set up very simply to exchange blocks quickly and easily. Below are the two different scenarios for

I/O structures in the Pixhawk (*Figures 2.4 and 2.5)*. Each of the blocks below is self-contained in terms of code and dependencies, connected to other blocks by the arrows through I/O or publish/subscribe calls.

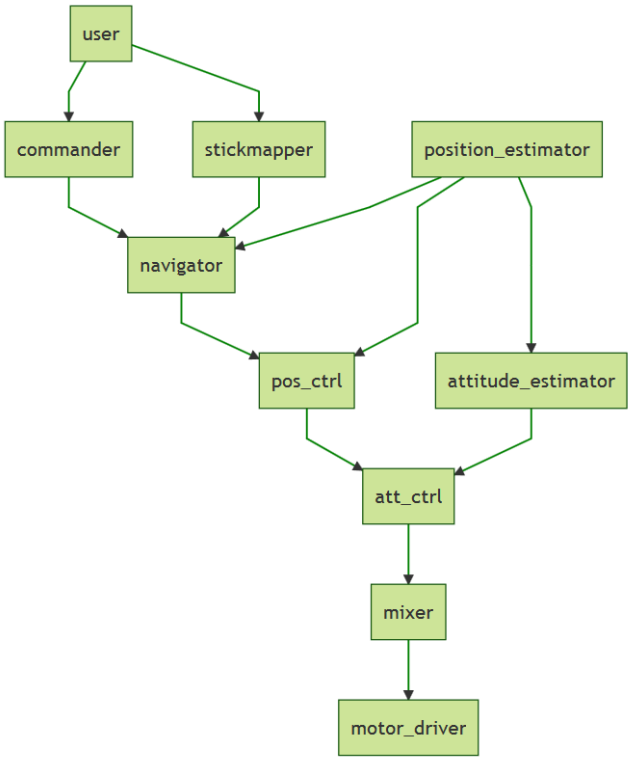

Figure 2.4: Person to UAV Block Structure

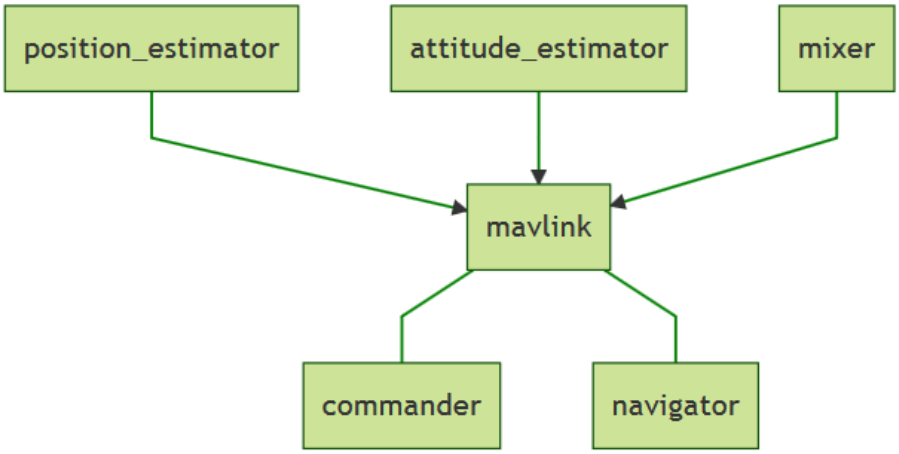

Figure 2.5: Ground Station to UAV Block Structure

While the architecture for transmitter to UAV communication is straightforward the UAV to ground station has several extra blocks that delegate inputs. These interactions are controlled by "business logic" applications including the commander (general command & control, e.g. arming), the navigator (accepts missions and turns them into lower-level navigation primitives) and the MAVlink application (creates the publish/subscribe data structures and consumes sensor data and state estimates).

The PX4 flight stack was chosen over the APM ardupilot because it is the newer updated model; with a modern-32 bit processor it can handle such aspects as if a motor fails the system automatically can adapt to turn it from a hexacopter to just a quadcopter. While the APM 2.5+ is more well documented because it is older the PX4 is still all open source so really no issue for getting used to it. The PX4 already includes controllers in its flight stack for multirotor airframes (where going from quad- to hexa-copter was no issue at all). The flight stack itself is a collection of guidance, navigation, and control algorithms and estimators for attitude and positon for the autonomous flight being tested. Below is an example of the implementation of these blocks. Another great aspect of the PX4 is the fact that it already has simulation software for running the autopilot. This came in handy at many times when testing different applications without having to actually fly the UAV's.

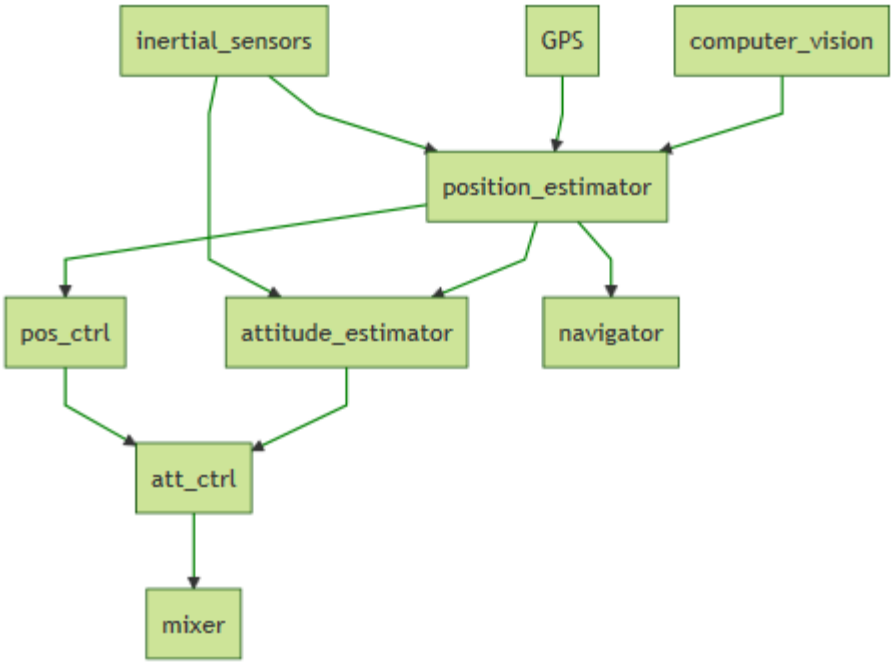

Figure 2.6: Flight Stack Block Schematic

The above "mixer" block is basically what defines the flight output parameters for the UAV, <turn right> command is defined as actuator commands controlling the motors and servos. In the case of multirotor airframes the mixer combines 4 control inputs (roll, pitch, yaw, and thrust) into actuator outputs driving the motor speeds in relation to each other to get the desired output, <turn right>. This is where the ESC's come into play, which are simply electronic speed controllers for each motor that take those desired output commands. The motor map becomes designated in the hexacopter "X" layout as the numbers in the below *Figure 2.7* and can be see implemented on the actual UAV in *Figure 2.1*. The main difference between another common "+" layout is the offset rotors from the point of heading.

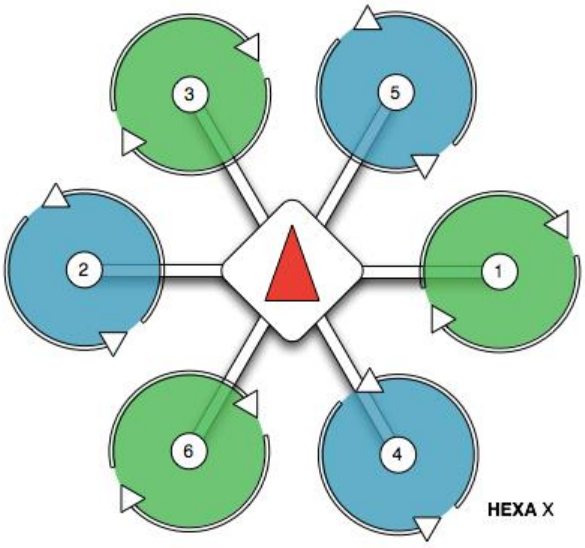

Figure 2.7: Hexacopter "X" Motor Layout

The whole code is able to be seen, edited and uploaded back onto the UAV's flight controller through Github (links located in the Appendix).

CHAPTER 3

UAV Testing and Flying

Creating the test bed and accomplishing autonomous flight with the ground station let us initially look at the effectiveness of these systems. Before adding the strain of the formation and flight pattern algorithms the inherent problems (latencies, lagging, etc.) in the system needed to be understood and documented so they could be accounted for in the long run. The starting point was using Mission Planner's built in data analyzing aspects to optimize the flight of each individual UAV. Then more group testing was needed to understand how the "leader" during "swarming" interacts with the ground station and then how that is transferred to the other 3 "follower" UAV's. Mission Planner turned out to be perfect for creating different flight paths with waypoints and being able to log the data for later studies. The program also easily allowed for the user to control, edit and create new commands/parameters beforehand and some of them in real time.

3.1 Mission Planner Data Log Analyzer

Mission Planner has a built in data log analyzer for comparing the real to the ideal flight data. For example, in *Figure 3.1* below you can see that the planned altitude is in green and the actual altitude is in red (relative in blue). In areas where the red line is further off the planned "course" you can speculate that most likely wind disturbed the system to the point that an immediate reaction by the UAV mixer wasn't possible and it

needed time to get back on the path, and in some cases overshoots or take its time to recalculate (right around 1 min mark).

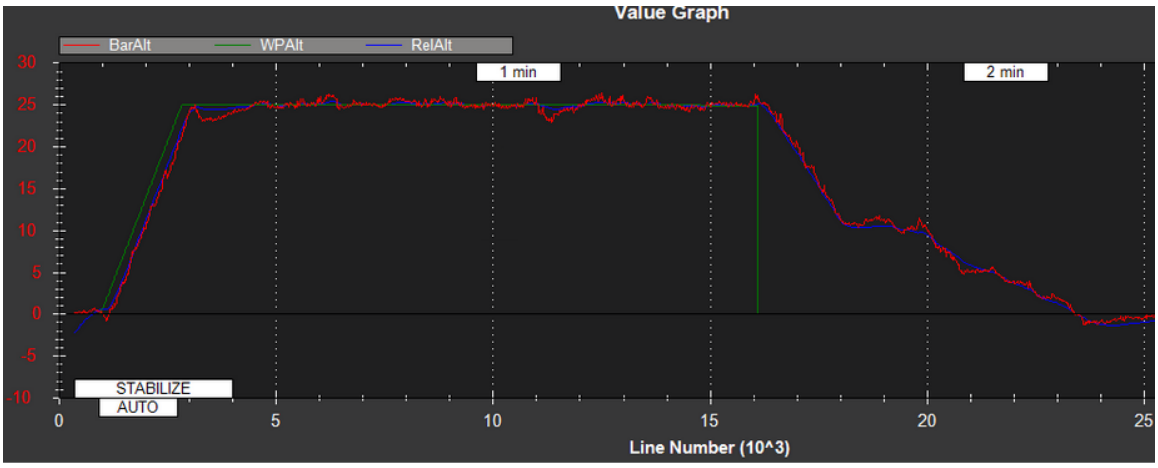

Figure 3.1: Altitude Analysis

Mission Planner's data logs allowed the assessment of any type of failure and to most accurately describe it as mechanical, vibrational, compass interference, GPS glitches, power problems or unknown. Below are examples of each type of failure that we experienced and were able to contribute to a specific problem.

Mechanical Failures

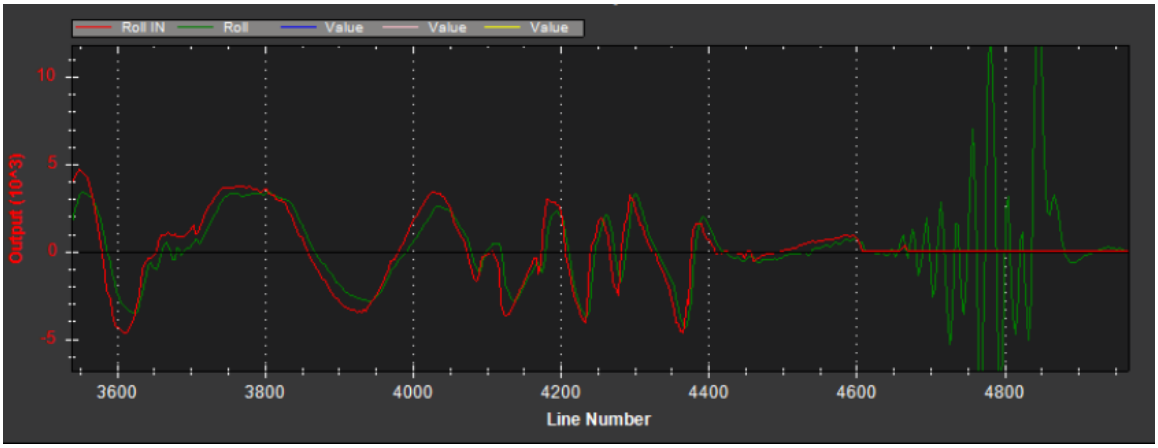

Figure 3.2: Roll Analysis

- The huge divergent oscillations in roll indicates a mechanical failure; motor, prop, or ESC failure. This can also appear from a pitch graph. This wasn't as useful since you can clearly see when this happens in flight and usually can clearly speculate on the mechanical failure before needing to analyze the logs.

Vibrational Excitations

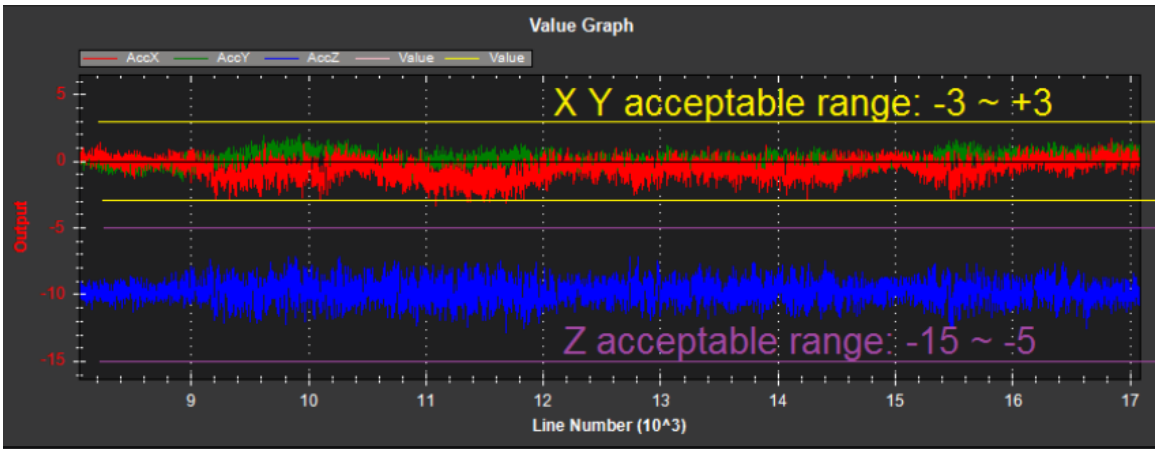

Figure 3.3: Vibrations

- Vibrations can be the direct cause of problems with the altitude hold and loiter commands. As seen above vibrations are most easily understood through graphing the accelerometer values AccX, AccY, and AccZ values. AccX and AccY are primarily used for the horizontal "x-y" positon control and AccZ is vertical "z" position control (with acceptable ranges are in the above *Figure 3.3*). If the UAV falls within these ranges when hovering then you are safe to assume during flight any momentary outlier is probably just due to the movement of the UAV.
- To reduce this vibration the flight controller was attached to the airframe by sticky rubbery/foam pads (3M foam from 3DRobotics), this reduced high/medium

frequencies while still allowing the low frequency board movement from flying movements. Some other interesting and simple ways to accomplish this found on forums from other users was simply platforming the flight controller on a double bed frame held together with o-rings or even earplugs.

Compass and Magnetic Interference

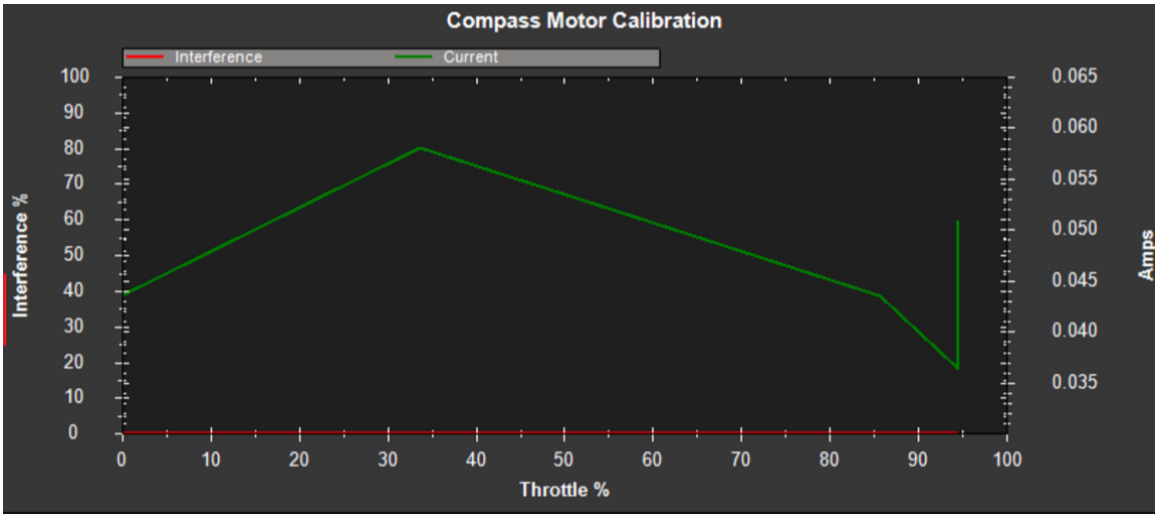

Figure 3.4: Compass/Motor Interference

- This was an extremely important factor in dealing with troubleshooting problems. Compass interference is when any of the various electrical components on the airframe (motors, ESC's, battery, PD board, etc.) throws off the compass heading. The process to calibrate magnetic interferences works only if you have a battery current monitor, where the magnetic interference is linear with the current drawn (which is due to how much throttle output). The process was simple, you secure the UAV to the ground and with Mission Planner's "Compass/Motor Calibration" open you variably increase and decrease the throttle to introduce into the system the

magnetic interferences. An ideal UAV that has no interferences from current changes is above in *Figure 3*.5 (given this never actually happens so perfectly).

- Problems with magnetic and compass interference can create catastrophic and very annoying problems with the UAV's, where "toilet-bowling" or just flying off in the completely wrong direction can occur (which happened quite a bit). We found that an acceptable amount of magnetic interference is ~10-20% but a flyable amount is more in the 5-10%. At one point when the testbed was first built the magnetic interference was around 1000%.
- Once calibration is performed the system knows more of how to compensate out, as much as possible, these interferences by switching how it routes power to the different ESC's and motors. Further ways to reduce interference include moving around the components on the airframe (the GPS+compass puck up and away on a mast as mentioned previously), making all the connecting wires as short as possible, replacing the ESC's with a 4-in-1 ESC, and add aluminum shielding to certain components and around wires. Other ways to work with decreasing the magnetic field is trying to increase the voltage as much as possible to decrease the current draw. Interference readings of <25% were usually what was sought out, anything higher and the autopilot mode would not function correctly.

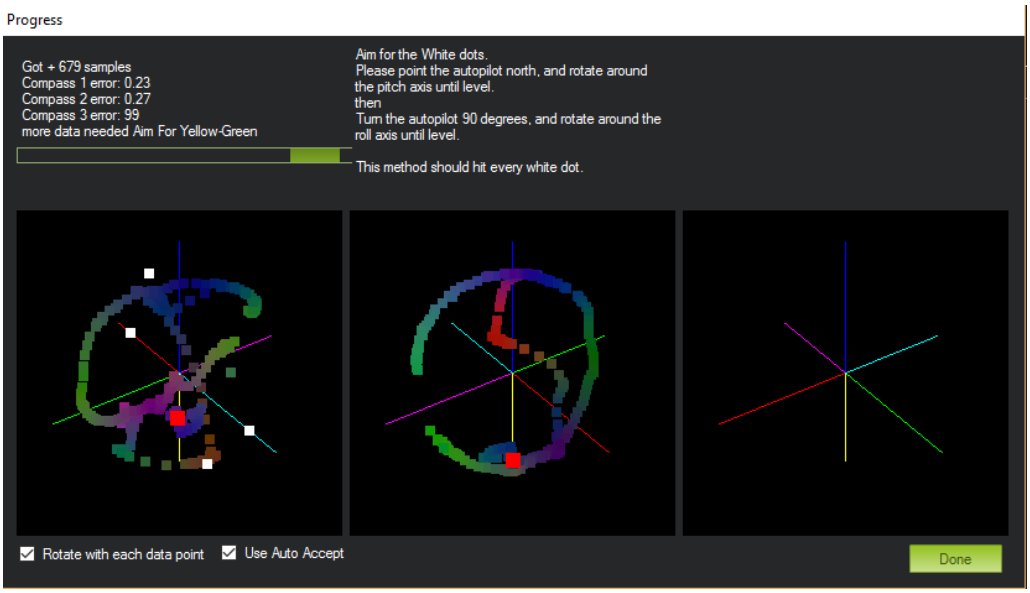

Figure 1.5: UAV Compass Mapping

- Accurately setting up and calibrating the compass is extremely important as it provides the heading for the UAV. Without this heading the autopilot mode is practically impossible. With an internal compass the UAV has the most trouble with interferences, it gets better with an external compass and a UAV can be programmed to take up to 3 on board compasses. At the startup of every UAV connection with the ground station you can perform a live compass calibration/mapping (*Figure 3.10* below), essentially spinning the UAV in circles slowly to get its orientation. This calibration also lets you know if there is any magnetic interference present. The trail from the rotating UAV is colored, where yellow and red indicate medium and high levels of magnetic interference.

Other Analyzable Data

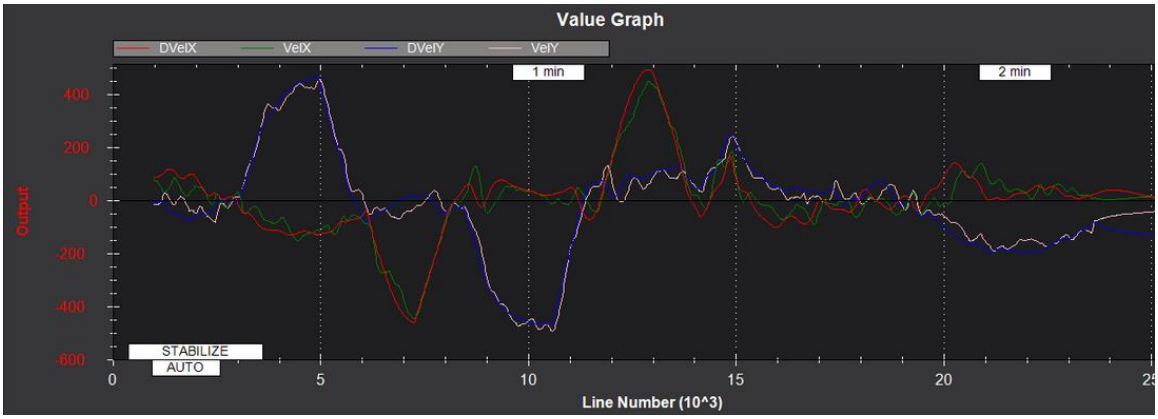

Figure 3.6: X and Y Velocities

- Velocities were good to analyze to see how well the ground station/flight controller were handling controlling the flight of the UAV's. Optimally the velocities would follow a perfect line, not bumpy, but that is not practical. This allowed us to see how quick the response time was to any fluctuations to the system (wind pushing it off course and having to speed up to relocate) and to in general improve on the oscillatory nature in the system (working with the motors and ESC's to reduce the size of these ups and downs that the flight controller allowed). In the end this wasn't a large concern except for in certain areas you can see where the proposed speeds were changing quickly and the UAV overshot a good amount such that is almost misses the next speed change (around horizontal tick mark 8-10).

The data logs were helpful in reading how far off the real value was from the intended value, as seen in the red and green lines of the various graphs above. This was extremely helpful in tuning the gains of the UAV's (pitch, roll, etc.). While an experienced MAV controller can usually do this by feel it wasn't practical to do this at the start of every flight with so many UAV's and such little battery lives. So with the help of these graphs we were able to a good underlying "start" point for all the gains. We

looked to optimize such that it would allow for a slight overshoot (since that is not a large issue) but the quickest reaction time to get back on the right path and least oscillations afterward.

One factor during flight that was pretty much impossible to correct for through the software was GPS glitches. After all of the flight tests it was obvious that the UAV's were most easily affected by loss in GPS, once compass interferences were left out. Lehigh's mountaintop campus had ~11 satellite locks and that was nowhere near good enough to be foolproof. This created a cascading effect if the leader's GPS lock was lost. In one such flight test one of the follower UAV's lost GPS signal and went off path, so it wasn't in its correct X-Y position coordinate in "swarming" mode relative to the leader. Once the UAV regained signal it took too long to return to its path before slamming into another of the following UAV's. While this could also be attributed to the flight path it took to get back on its correct path, to solve that problem would require to implement inter-communication between the drones or a more complicated algorithm to assess the correct flight path immediately from its new incorrect point to not overtake any other UAVs' flight path. This is definitely doable and is a future step consideration (definitely an important aspect to note).

Another large factor is just the loss in connection between the transmitters of the ground station to the receivers of the UAV. The quickest way to get around this was trying to fly in open areas with nothing that could block the frequencies, aka trees. You could fix these issues with hardware updates or using a land based GPS external locating station to add better, more consistent GPS, but those are expensive and not practical for these purposes.

In the end all of the analyses from Mission Planner helped to stabilize the UAV's, work out the bugs and optimize the hardware as much as possible. In most cases it meant changing the firmware/software but in some cases it helped to make changes on the airframe itself. As mentioned before, this is where it became apparent to move the GPS puck 4 inches above everything else because there were too many GPS glitches and compass interferences. We also had to switch the location of the radio receiver for interference purposes with the motors. We ran diagnostics to change the frequencies for the ESC's and motors so they wouldn't interfere. Once we moved on from updating and optimizing each individual UAV we were able to focus more on the swarming feature of Mission Planner and group flight.

3.2 Mission Planner "Swarming"

The first several individual flight tests took months – with the main issue that kept arising being troubleshooting – so it definitely made transitioning to group flight that much easier. It did take various tests to get the correct setup for connecting the UAV's with the computer and making sure that every single one was designated as the correct unit, leader or follower. The general rundown for setting up a swarming procedure is as follows:

- Connect each UAV to the ground station to upload any coding changes and connect with the flight controller. During this stage you can turn on the UAV and make sure that GPS locating occurs correctly.
- Once all UAV's are connected each one of their radio transmitters are hooked up to the ground station.

- Once a flight path has been created and uploaded into the program and the UAV's have been set in the field you are ready for takeoff.
- A secondary chart window pops up that provides the "swarming" data. On this screen you can toggle between two 2D coordinate maps to create the 3D positioning of the UAV's in relation to the "leader" you have designated. In particular *Figure 3.7* below shows the X-Y coordinates of a UAV (abbreviated MAV sometimes) in relation to the leader, designated as the center point. The other 2D map shows the Z axis. At the same time you can see the positions of all UAV's in relation to the earth in the main Mission Planner window.
- Once you press start the UAV's will assume the correct positions and hence start the planned flight path. The leader will start and the other UAV's will follow, while trying to stay in the correct positions the whole time.
- At any point you can toggle a designated switch on the handheld transmitter to switch the UAV autopilot to user controlled to manually take over if anything goes wrong.

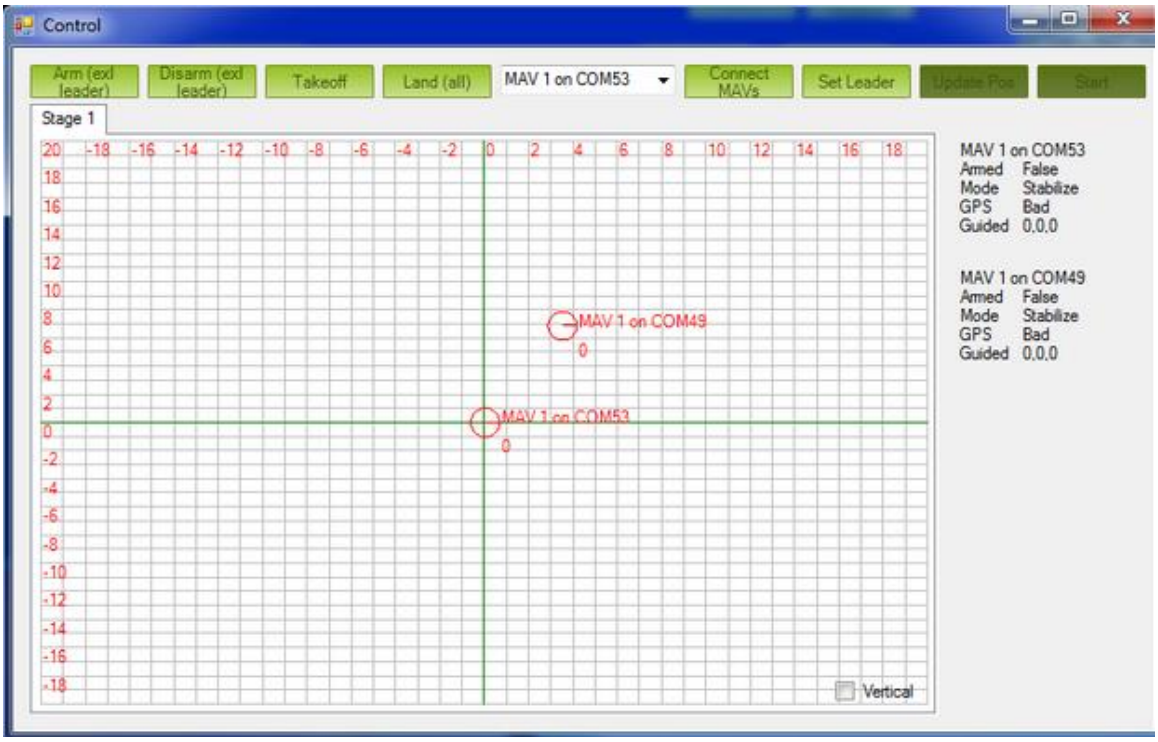

Figure 3.7: Mission Planner's "Swarming" Command Setup

An example of a planned route, conducted on Lehigh's University's Goodman Campus is in *Figure 3.8* below. Each point represents a waypoint or the takeoff/landing points. There is a way to alter the commands at every single point, which is the screenshot (*Figure 3.9)* of the command list from Mission Planner. For example, you can make the UAV's hover or loiter for a certain amount of time, change altitudes by design, and a good amount of other commands. When loitering you can plan for the UAV to go in x number of circular loops around that waypoint before proceeding on the flight path, which obviously for reconnaissance or search and rescue is important. This is also important a lot of time for agriculture, where designated paths can be set and certain tasks be performed at certain points. Apart from at the specific locations themselves you can command the UAV's to fly certain ways when transitioning through the waypoints. So

for instance if you want more of a smooth flight path you can spline the point, or you can go through the waypoint and plan for an overshoot, or just the quickest route in general.

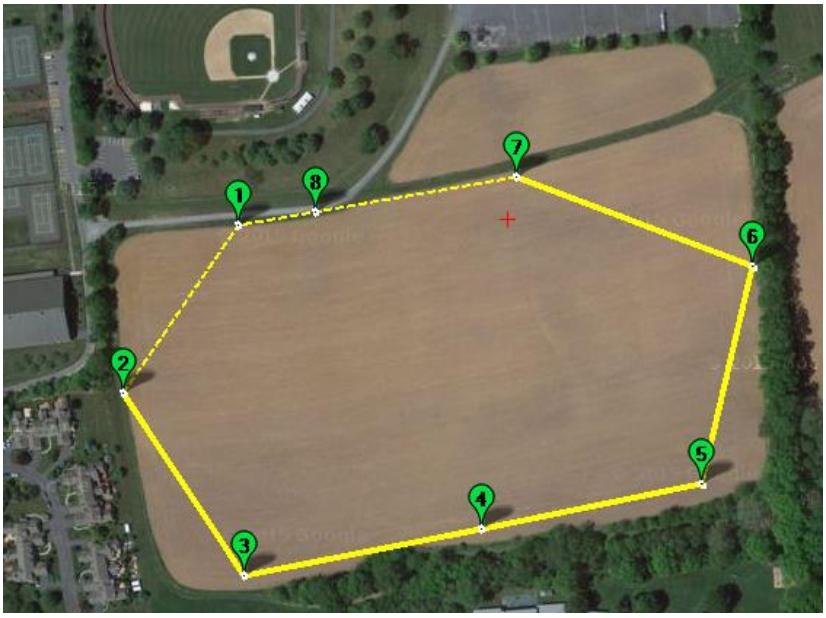

Figure 3.8: Programmed Waypoint UAV Flight

Waypoints

WP Radius 2 | Loiter Radius 45 | Default Alt 100 | Verify Height | Add Below | Alt Warn 0 | Spline

	Command								Delete	Up	Down	Grad %	Angle	Dist	AZ
1	WAYPOINT	0	0	0	0	40.5817825	-75.3578317	20	X			0.0	0.0	8776785.0	280
2	WAYPOINT	0	0	0	0	40.5808454	-75.3586793	25	X			4.0	2.3	126.4	214
3	WAYPOINT	0	0	0	0	40.5798269	-75.3577888	30	X			3.7	2.1	135.9	146
4	LOITER_TIME	0	0	0	0	40.5800876	-75.3560400	35	X			3.3	1.9	150.5	79
5	WAYPOINT	0	0	0	0	40.5803402	-75.3544199	35	X			0.0	0.0	139.7	78
6	WAYPOINT	0	0	0	0	40.5815543	-75.3540444	25	X			-7.2	-4.1	138.7	13
7	WAYPOINT	0	0	0	0	40.5820514	-75.3557825	25	X			0.0	0.0	156.8	291
8	RETURN_TO_LAUNCH	0	0	0	0	40.5818558	-75.3572631	20	X			0.0	0.0	126.9	260

Figure 3.9: Waypoint Action Item List

Examples of planned individual and group missions (failed, catastrophic, and successful) that were flown were logged and saved, they can be located from the link in the Appendix. By downloading and installing the free program Mission Planner one can upload any flight and play it from start to finish, while also being able to analyze the data. Essentially you are able to see all aspects of flying the UAV's. The UAV is represented

as a simple quadcopter (*Figure 3.10)* and has 3 leading lines extending from it. The black line the GPS track and the red line is the current/actual heading. The orange line is designate as "Direct to Current WP," which is the current directional vector to the next waypoint. This is most useful to understand what the UAV is doing transitioning between waypoints.

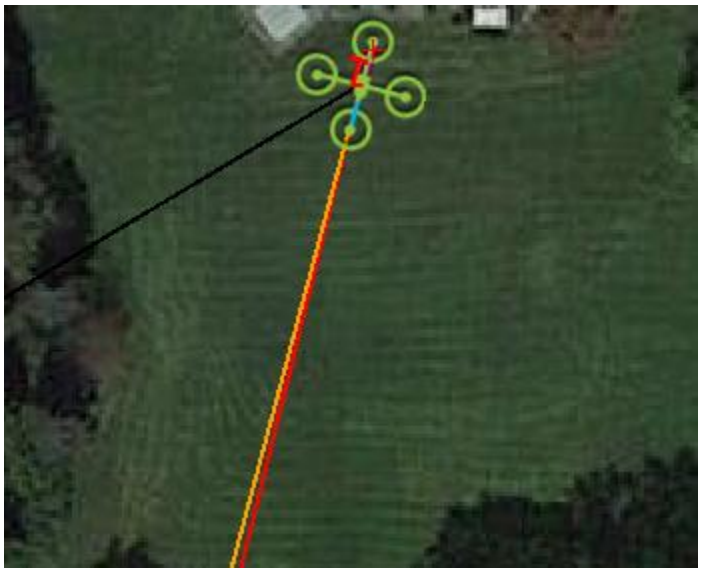

Figure 3.10: Mission Planner Representation of UAV

What became most apparent in the first several flight tests is how the "follower" UAV's communicate with the "leader." Each individual UAV communicates with the ground station (Mission Planner on a laptop) and all inputs and outputs go through the computer. When a "mission" starts the GPS location of each UAV is shown and the followers rely only on the GPS location that Mission Planner is receiving from the leader UAV. Of course the limitations of this type of communication architecture is simply more areas for signals to be lost or slow. In the future some type of inter-communication sub-structure on the flight controller can easily be created (and there is a ton of research

being done currently on this with everything from cameras, lasers, simple radio, rf readers, etc.).

Another aspect of Mission Planner that was used is to create and implement user defined commands. One of the first things we noticed is that Mission Planner does not allow you to switch the drone into Autonomous Mode and start the mission without the UAV already in flight (which is strange because it has a built in command "Take-Off"). We were able to implement in the command structure MAV_CMD_COPTER, not technically a command but as an Action item parameter. This allowed the UAV to be toggled into Autonomous Mode while on the ground, disarmed and start the mission; ie. take-off to the preset altitude from the ground without the user. In this example it became very easy with the user friendly command parameters interface to edit parameters, commands and action items to control the UAV's flight.

At the same time you can set it up to make certain commands/actions _DO_ commands, such so they are only executed if another parameter is met. This made it very easy that if a certain command structure was ignored or failed then a failsafe command would be executed (usually return to some path or return to home). Other types of uses for this is with editing the yaw of the vehicles. Interestingly the yaw of each UAV is not pre-programmed in the autopilot. So you can either have the user control it himself from a transmitter during flight or create a command/action item MAV_CMD_CONDTION _YAW to point the vehicle in the specified heading for a certain amount of time. If the user does change the yaw it does not affect the flight path. We did implement some protocol to show how the yaw can be pre-programmed and it is very useful for anything like reconnaissance or for search and rescue (applying maximum coverage area to more

important sectors, etc.). To go along with the yaw you can also pre-program commands to control autonomous on-board camera functions, like take a picture once you reach X waypoint, hover for Y seconds, and turn to Z yaw.

The end result of working with Mission Planner is that you can manipulate the UAV's with plenty of commands that allow you to get a good feel for how and why the UAV's do certain things in flight (apart from the "mission"). This gave a really good base, or control state, that later on analyzing flight control algorithms was a lot easier.

CHAPTER 4

Kalman Filtering

4.1 Kalman Filtering Implementation

As said previously the Pixhawk PX4 flight controller's code is all located on Github as an open source code for anyone to download, edit and upload to their own UAV for testing (instructions to do this can also be found at http://dev.px4.io/tutorial-hello-sky.html). The process is extremely simple and straightforward to apply any new application/commands/parameters to the code.

Kalman filtering is basically an optimal estimator, it takes in data noises and "filters" out those noises. The beauty of such a filter is that it can react in real-time to correct a flight path to unexpected and un-programmed changes in an environment. Kalman filtering was chosen to explore as an option to better "tune" the flight controller because it is fairly straightforward and easy to implement (much of the math around Kalman filtering is readily found for application uses or study). The most significant factor of using a Kalman filter is that it cannot only correct for otherwise problematic inputs on the UAV but can change the state estimates to reflect these new measurements.

While the default tuned parameters of the UAV can allow it to fly, when you move to autopilot the reaction times of the user are decreased greatly to the reaction times of the ground station. In the end of all the flight tests we tuned the UAV's as well as possible but GPS was still a huge issue. The Pixhawk's current attitude estimator is DCM

(direct cosine matrix). Instead a Kalman Filter was applied to the system, and fairly quickly we moved to just implementing an Extended Kalman Filter (EKF). This extended kalman filter algorithm uses accelerometer, compass, GPS, gyroscope, airspeed and barometric pressure to estimate the position, velocity and angular orientation of the flight vehicle. You can implement the EKF estimate as a compliment to the DCM to detect excessive errors or you can just run the EKF solely as the attitude estimator (which is what was attempted). The main reason for the EKF is for highly more accurate handling of GPS data loss, which is the largest issue if you were to update all the hardware/firmware (*Figures 4.3 and 4.4).*

The following is a basic description of how the filter works (author of the original code/directions is Paul Riseborough, code in Appendix A):

1. Inertial Measurement Unit, IMU, angular rates are integrated to calculate the angular position
2. IMU accelerations are converted using the angular position from body X,Y,Z to earth axes and corrected for gravity
3. Accelerations are integrated to calculate the velocity
4. Velocity is integrated to calculate the position

   This process from 1) to 4) is referred to as 'State Prediction'. A 'state' is a variables we are trying to estimate like roll, pitch yaw, height, wind speed, etc. The filter has other states besides position, velocity and angles that are assumed to change slowly. These include gyro biases, Z accelerometer bias, wind velocities, compass biases and the earth's magnetic field. These other states aren't modified

directly by the ‘State Prediction’ step but can be modified by measurements a described later.

5. Estimated gyro and accelerometer noise (EKF_GYRO_NOISE and EKF_ACC_NOISE) are used to estimate the growth in error in the angles, velocities, and position. Making these parameters larger causes the filters error estimate to grow faster. If no corrections are made using other measurements this error estimate will continue to grow. These estimated errors are captured in a large matrix called the ‘State Covariance Matrix’.

   Steps 1) to 5) are repeated every time we get new IMU data until a new measurement from another sensor is available.

   If we had a perfect initial estimate, perfect IMU measurements and perfect calculations, then we could keep repeating 1) to 4) throughout the flight with no other calculations required. However, errors in the initial values, errors in the IMU measurements and rounding errors in our calculations mean that we can only go for a few seconds before the velocity and position errors become too large.

   The Extended Kalman Filter algorithm provides us with a way of combining data from the IMU, GPS, compass, airspeed, barometer and other sensors to calculate a more accurate and reliable estimate of our position, velocity and angular orientation.

An example of the above steps being utilized is:

6. When a GPS measurement arrives, the filter calculates the difference between the predicted position from 4) and the position from the GPS. This difference is

known as an ‘Innovation’. An example of the Innovations that are recorded are in *Figure 4.1* below. They show the innovations for the N, E, D GPS velocity measurements, which show the health of the navigation filter. If you have good quality IMU and GPS data it should look like the image below, with small to zero measurements. These measurements would help in real flights by running the diagnostics to see if while not moving the vehicle had any value for Earth velocities, and if so you can offset and make the starting point or “noise” level for the offset that amount. Same concept but with position is shown for the x-y coordinate in the next *Figure 4.2.*

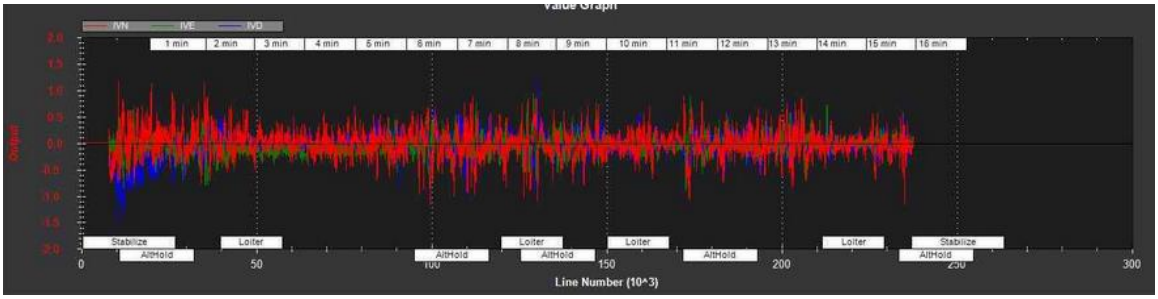

Figure 4.1: Earth Velocity Measurements

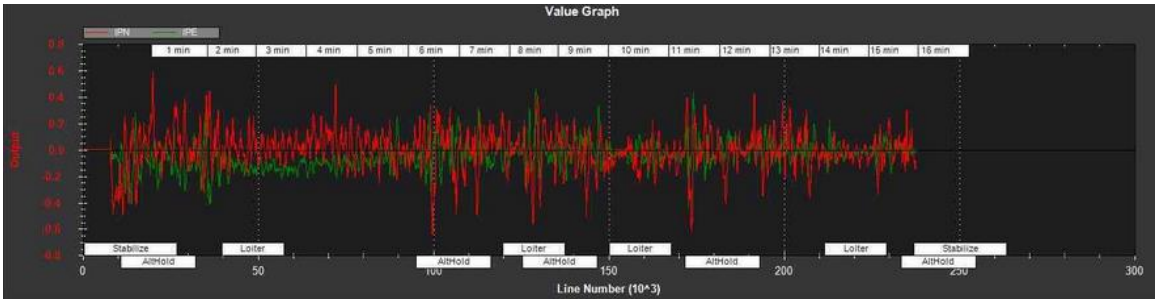

Figure 4.2: Earth Position Measurement

7. The ‘Innovation,’ ‘State Covariance Matrix,’ and the GPS measurement error specified by EKF_POSNE_NOISE are combined to calculate a correction to each of the filter states. This is referred to as a ‘State Correction’.

This is where the Kalman Filter is important, it can correct measurements other than the ones being measured. For example GPS position measurements can also correct errors in position, velocity, and angles. The amount of correction is controlled by the assumed ratio of the error in the states to the error in the measurements. If the filter thinks its own calculated position is more accurate than the GPS measurement, then the correction from the GPS measurement will be smaller. The accuracy of the GPS measurement is controlled by the EKF_POSNE_NOISE, making this parameter larger causes less accuracy.

8. Because we have now taken a measurement, the amount of uncertainty in each of the states that have been updated is reduced. The filter calculates the reduction in uncertainty due to the 'State Correction', updates the 'State Covariance Matrix' and returns to step 1)

4.2 Kalman Filtering Tuning

Once the EKF is setup to run on the UAV you still need to conduct tuning on ~25 parameters to get the best baseline for the navigation filter to work off of, the parameters are in *Table 4.1* below. Essentially tuning these parameters allows you to pick which measurements you want the filter to "trust" or consider more than others. For example, if you think the accelerometer is cheap and not working great you can tune the value such that the filter does not recognize changes in the sensor much (there would need to be a large spike or something for the filter to take it into account). Of course these tuning parameters can change due to location and even from UAV to UAV, since the motor layout, wind, interferences, etc. can easily be different.

Designation	Function
EKF_ABIAS_PNOISE	Vertical accelerometer bias state error
EKF_ACC_PNOISE	Accelerometer measurement errors
EKF_ALT_NOISE	RMS value of noise in altitude measurements
EKF_EAS_GATE	Airspeed measurement consistency check
EKF_EAS_NOISE	RMS value of noise in compass measurements
EKF_GBIAS_PNOISE	Speed and noise amounts of gyro bias state error
EKF_GLITCH_ACCEL	Maximum allowed difference horizontal acceleration between predicted filter value and GPS measured value
EKF_GLITCH_RAD	Maximum allowed difference horizontal position between predicted filter value and GPS measured value
EKF_GPS_TYPE	Whether or not to use GPS velocity measurements
EKF_GYRO_PNOISE	Estimated error from gyro measurement errors (excludes bias)
EKF_MAGB_PNOISE	Body magnetic field state errors
EKF_MAGE_PNOISE	Earth magnetic field state errors
EKF_MAG_CAL	Active learning during flight of needed magnetometer offsets
EKF_MAG_GATE	Magnetometer measurement consistency check
EKF_MAG_NOISE	RMS value of noise in magnetometer measurements
EKF_POS_DELAY	msec that GPS position measurements lag being inertial measurements
EKF_POSNE_NOISE	RMS value of noise in GPS horizontal position measurements
EKF_POS_GATE	GPS position measurement consistency check
EKF_VELD_NOISE	RMS values of noise in vertical GPS velocity measurement
EKF_VELNE_NOISE	RMS values of noise in North/East GPS velocity measurement
EKF_VEL_DELAY	msec that GPS velocity measurements lag behind inertial measurements
EKF_VEL_GATE	GPS velocity measurement consistency check
EKF_WIND_PNOISE	Noise controlling growth of wind state error estimates

EKF_WIND_PSCALE	Changes rapidness of wind states adapting to changing altitude

Figure 2.1: Tuning Parameters

Some examples of analyzing data to set certain of the parameters above are with EKF_ALT_NOISE, EAS_NOISE described in *Figure 4.1* above. Another very easy to understand example of the filter in action is in *Figure 4.3* below. Graphed are the body magnetic fields biases and by flying at a low speed for a 15 minutes you can see how the lines slowly change. This is basically the filter 'learning' the earth's magnetic field. Afterwards you can know that the magnetic body field in the X coordinate stabilizes at a value of 35, which means you would want to set the value for the compass offset in the X coordinate at -35. The same type of analysis can be done for the magnetometer biases, where we learned about small variances in differences between axes, misalignments, and just varying magnetic fields produced by the electrical components. This led us to set the default value to .05 (indicating a noise level of 50 in the sensor units). In *Figure 4.4* below you can see an example of what happens with no calibration. The graph represents a slow speed copter flight with a bad magnetometer calibration (set at just 0). As the vehicle changes headings the noise levels spike above the appropriate +-50 range.

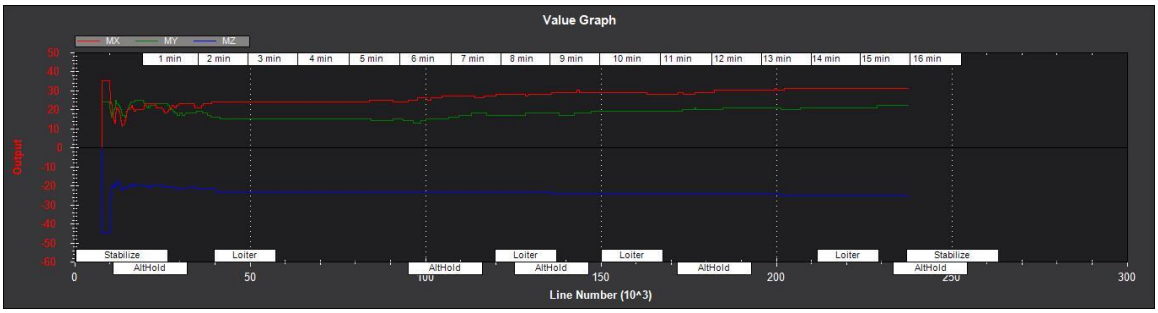

Figure 4.3: Body Magnetic Field Flight Data Logs

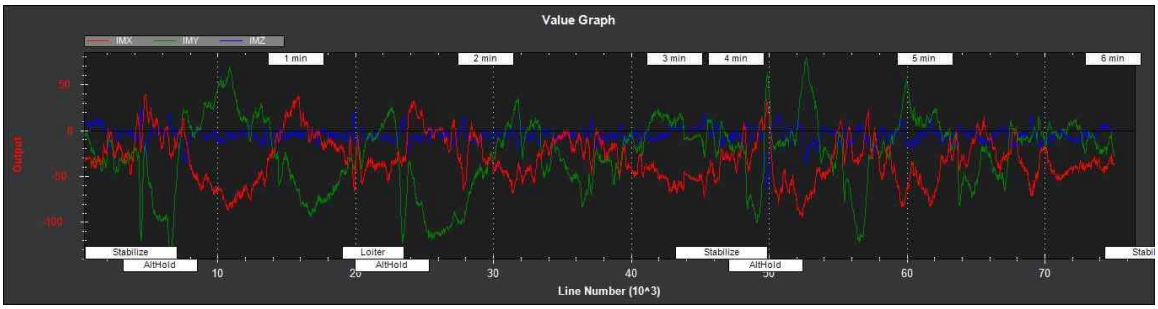

Figure 4.4: Magnetometer In-Flight Noise Levels

We quickly learned at the beginning that the EKF filters in flight were not logged correctly because we found out later on that you had to enable a certain data log feature, AHRS, to be able to log the "flash logs," which are the type of data logs that the EKF information is stored. But once was all said and done we were able to have a good starting point with all of the tuned parameters. We ran a typical simulation setup in Mission Planner's simulation software (Appendix B for codes/direction on how to use in Mission Planner), to see correlation showing the difference in the applied Kalman filter to the raw GPS data. The best results for the simulation are shown in *Figures 4.5* and *4.6* below. Further real world testing should be conducted to analyze how the EKF might be able to be used to implement it on the leader-follower schematic once inter-communication is applied and becomes susceptible to signal disconnections.

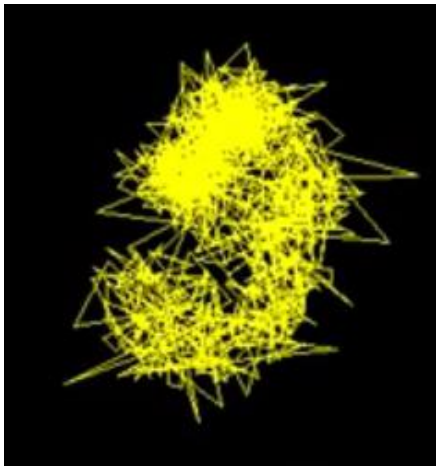

Figure 4.5: GPS Location (Raw)

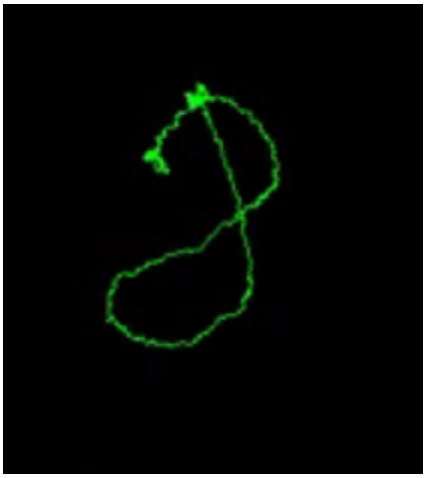

Figure 4.6: GPS Location (Filtered)

After changing out the Pixhawk with the Lisa/S we accomplished flight but could not establish a successful autopilot. The Lisa/S is a little more complicated to work with, more so just less well documented and less people contributing to the open source code editing/improvements. The limited time left was put towards trying to implement a Kalman filter on the Lisa/S. It was soon found out that the previously used extended Kalman filter could not be used on the limited processing power of the flight controller. A simplified version of the Kalman filter was cut to try and just use for certain aspects of flight but could not be thoroughly tested. The obvious next steps would be to analyze the most basic version of a Kalman filter that could, if so, be used on the Lisa/S and if not start to figure out a way to add a sub-system architecture to house Kalman filtering components and link it with Lisa/S.

Chapter 5

Conclusion and Future Work

As this project progressed the scope changed accordingly to the new limitations learned from building and testing the UAV testbed. New understandings gained from diving into the existent control systems and "genetic" makeup of these UAV's shifted the focus of this thesis to a long ways before a point where swarm behavior could be implemented. In the end the flight control algorithms through the Pixhawk architecture were used, analyzed and worked to be improved. The future steps include implementing such EKF algorithms on a future smaller scale UAV using the Lisa/S flight controller.

There was a lot of testing to get to the stages of starting to implement new flight control systems to optimize the flight patterns of these UAV's. At the end of this research we explored using Kalman filtering in the flight controllers and saw the viability and effectiveness of it in the Pixhawk. Next steps would be to take the newly implemented Lisa-S UAV and build the optimized flight control design features on top of that architecture. Once accomplished more hurdles like collision avoidance, accurate positioning, etc. should just come down to hardware updates. The progression for this testbed would be in the end to have in place an autonomous inter-communication among the group for a designated flight path "mission."

Introducing a leader-follower method of communication implies getting as close as possible to AI type responses by individual UAV's in a "group". Limiting the human

interaction increases efficiencies and overall responsiveness of the UAV's. Improving the flight control systems of UAV's lends to a wide range of possibilities for implementation in real world applications. It became obvious why UAV's are becoming more and more widely used in fields such as agriculture and police/fire departments. The multitude of tasks or commands that you can program the UAV to do without the need of constant human interaction is impressive. Other potential outcomes of this research include honed ballistics testing, projectile path optimization and flight formation patterns. Real time dynamic flying creates more accurate flight trajectories, dynamic vehicle control in relation to each other, further recognition and ways to react to the surrounding environment, consistent area coverage control amongst the "team."

APPENDIX A

Mission Planner Log Files

https://www.dropbox.com/sh/0l2lr7uwv2542mh/AADD6hw0VcQWcOinD82aRfswa?dl=0

APPENDIX B

Pixhawk PX4 Source Codes

PX4 Firmware: https://github.com/PX4/Firmware.git

Extended Kalman Filter Code:

https://github.com/priseborough/InertialNav/blob/master/derivations/GenerateEquations22states.m

Simulation Software: https://github.com/px4/jMAVSim

3D Simulation Software for Swarming, Autonomous Flight, etc.: http://gazebosim.org/

All other basic coding: https://github.com/PX4/DevGuide

WORKS CITED

[1] ArduPilot Autopilot Suite. Online. http://ardupilot.org/ardupilot/index.html

[2] 3DRobotics. Online. https://store.3dr.com/products/3dr-pixhawk

[4] Paparazzi. *The Free Autopilot*. Online. http://wiki.paparazziuav.org/wiki/Lisa/S

[5] "Fundamentals of Kalman Filtering: A Practical Approach" Third Edition

[3] Pixhawk PX4 Development Guide. Online. http://dev.px4.io/index.html

[] Balazs Gati Paparazzi Community, *Open Source Autopilot for Academic Research the Paparazzi System.* In *Proceeding of the American Control Conference 2013,* Washington, USA, Pages 17-19. June 2013.

[4] How, Jonathan, *Flight Demonstrations of Cooperative Control for UAV Teams*, Aerospace Controls Lab at MIT, 2004.

[5] B.D.W. Remes, *Lisa-S 2.8g Autopilot for GPS Based Flight of MAV's*, Delft University of Technology at Delft.

[6] Brisset P., A. Drouin, M. Gorraz, P.-S. Huard, J. Tyler. The Paparazzi Solution. http://www.recherche.enac.fr/paparazzi/papers2006/mav06paparazzi.pdf, 2006.

[10] Mao, Guoqiang. *Design of a Extended Kalman Filter for UAV Localization*. School of Electrical and Information Engineering. University of Sydney.

[11] B.D.O. Anderson and J.B. Moore. *Optimal Filtering*. Prentice Hall, 1979.

[12] Jazwinski, A.H. *Stochastic Processes and Filtering Theory*. New York, Academic Press. 1970

VITA

Brendan Sullivan, son of Cindy and Mike Sullivan, is originally from Chicopee, Massachusetts and was born on January 2, 1992. He attended and received a dual degree from Lehigh University in the IBE Honors Program (Integrated Business & Engineering). Brendan's degrees included a Bachelor of Science in Mechanical Engineering in May 2014, and a Bachelor of Science in Integrated Business and Engineering in September 2014. Brendan always had an interest in aerospace engineering and took that a step further by working towards his MS. degree with Professor T. Hart in the Mechanical Engineering Department on UAV design and systems controls.