UAV FORMATION FLIGHT UTILIZING A LOW COST, OPEN SOURCE CONFIGURATION

A Thesis

Presented to

the Faculty of California Polytechnic State University

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Aerospace Engineering

by

Christian Lopez

June 2013

COMMITTEE MEMBERSHIP

TITLE:	UAV Formation Flight utilizing a Low Cost, Open Source Configuration
AUTHOR:	Christian Lopez
DATE SUBMITTED:	June 2013
COMMITTEE CHAIR:	Eric Mehiel, Ph.D., Associate Professor, Aerospace Engineering Department
COMMITTEE MEMBER:	Robert McDonald, Ph.D., Associate Professor, Aerospace Engineering Department
COMMITTEE MEMBER:	Kira Abercromby, Ph.D., Assistant Professor, Aerospace Engineering Department
COMMITTEE MEMBER:	Charles Birdsong, Ph.D., Associate Professor, Mechanical Engineering Department

ABSTRACT

UAV Formation Flight utilizing a Low Cost, Open Source Configuration

Christian Lopez

The control of multiple unmanned aerial vehicles (UAVs) in a swarm or cooperative team scenario has been a topic of great interest for well over a decade, growing steadily with the advancements in UAV technologies. In the academic community, a majority of the studies conducted rely on simulation to test developed control strategies, with only a few institutions known to have nurtured the infrastructure required to propel multiple UAV control studies beyond simulation and into experimental testing. With the Cal Poly UAV FLOC Project, such an infrastructure was created, paving the way for future experimentation with multiple UAV control systems. The control system architecture presented was built on concepts developed in previous work by Cal Poly faculty and graduate students. An outer-loop formation flight controller based on a virtual waypoint implementation of potential function guidance was developed for use on an embedded microcontroller. A commercially-available autopilot system, designed for fully autonomous waypoint navigation utilizing low cost hardware and open source software, was modified to include the formation flight controller and an inter-UAV communication network. A hardware-in-the-loop (HIL) simulation was set up for multiple UAV testing and was utilized to verify the functionality of the modified autopilot system. HIL simulation results demonstrated leader-follower formation convergence to 15 meters as well as formation flight with three UAVs. Several sets of flight tests were conducted, demonstrating a successful leader-follower formation, but with relative distance convergence only reaching a steady state value of approximately 35 ± 5 meters away from the leader.

ACKNOWLEDGMENTS

This work is dedicated to my grandfather, Harold Condon. At the age of 95, He continues to be an inspiration for me, driving me to follow his example: To work hard towards my goals, and to keep my family close to my heart.

With such an ambitious, multi-faceted project, I am extremely fortunate to have had so much support from faculty, peers, friends, and family. Without so many people willing to donate their time and expertise, I could have only accomplished a fraction of the work that is presented here.

First, I would like to acknowledge Dr. Eric Mehiel who served as my adviser, guided my work, and provided the resources to make my lofty project goals a reality. I would also like to acknowledge the other members of my thesis committee, Dr. Rob McDonald, Dr. Kira Abercromby, and Dr. Charles Birdsong. Their presence and support throughout the various stages of the project was a valuable resource, and I am grateful to have had such an involved committee. I am especially grateful to Dr. McDonald who accommodated me in the Cal Poly UAV Lab and provided a significant amount of guidance throughout the project. I must also make a special acknowledgment of Dr. Abercromby's involvement, especially her persistent encouragement to finish strong towards the end of this odyssey.

Early in the process, the easiest risk to identify in my project goals was the reliance I would have on other students to help me during the testing phases of this work. Thankfully, I have had the steadfast support of a core group of my peers who I am lucky enough to call my friends. I would not have had a chance to do any flight testing if it were not for Cory Suebert, Brian Marchini, and Martin Bialy, who generously served as my pilots throughout the testing. A

special thanks to Cory Seubert for being a part of nearly every flight test, despite working and living four hours away in LA. I would also like to acknowledge Adam Chase, Dan Brown, Alex Gary, Lance Genato, Trevor Goehring, Brad Shab, Chris Satterwhite, Greg Gradwell, and Robby Campbell. Their assistance during the pre-flight preparations and flight testing were instrumental to the successes achieved in this work. In addition to their help during the flight tests, I must acknowledge Adam Chase, Brian Marchini, and Michael Darling for the moral support they provided daily during the long hours spent together in the UAV lab, as well as their assistance reviewing and editing this work. Additionally, a special thanks goes out to Cory Hackett-Robles, Aleksey Pavlov, Scott Sawyer, Derek Goss, and Todor Anguelov. All of these good friends helped me to keep some level of sanity during this long process, and often found their way out to the flight range early in the morning to provide their support both during, and after the testing.

This project commanded a great deal of determination, often testing my mental and physical limits. I am not sure where I would be today without Monica McPherrin, who served as a constant beacon of positive energy and fed me a steady stream of love and encouragement. I am thankful for her patience and understanding, as well as her family's support of my efforts. Especially her mother Tonya, who assisted by editing several sections in this work.

Similarly the love and support provided by my family has enabled me to succeed throughout my college career. I am increasingly grateful for the solid foundation my uncles Ed and Norman provided for my academic success through their Montessori schools. Their value of education and their continuing support of my academic achievement, including the review and editing of this work, has been a large factor in my success. I must also acknowledge the unwavering en-

couragement and well-timed phone calls from my brother, Eddie. No matter how far apart our worlds tend to be, he can always be counted on to provide the guidance and insight that only a brother can have.

Last but not least, I am eternally grateful for the unconditional love and support provided by my father, Emilio, his partner, Troy, and my mother, Margee. The opportunities they have provided for me have allowed me to reach for something only a small percentage of people in this world have achieved. All that I am, I am because of them. My successes are their successes.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 Introduction

1.1 Topic Area

The increased use of small and micro unmanned aerial vehicles (UAVs) by soldiers on the battlefield and, more recently, by various civil institutions, has brought renewed focus on how these technologies can be utilized effectively. Dramatic improvements have been made towards the utility of small UAVs, driven by parallel advancements in subsystem technologies. Yet, small UAV applications will always be limited by reduced payload capability. Many of the applications considered for small UAVs marginalize these payload limitations by proposing the use of multiple systems to accomplish a common goal, capitalizing on the reduced cost and increased redundancy associated with using multiple small UAVs instead of one large UAV.

For many of these proposed applications, multiple UAV control strategies must be developed to enable driving capabilities such as vehicle cooperation, collision avoidance, and task coordination. General UAV control strategy development requires several levels of simulation and testing. For small UAVs, a larger gap exists between the computational power available on the desktop computers used during preliminary simulation and the small embedded control systems used in flight. These computational limitations often generate pushback on the control system design, and place greater emphasis on lower-level simulation, such as hardware-in-the-loop (HIL) simulation in order to accurately characterize performance during the development process. Furthermore, multiple UAV control strategy development requires that much of this simulation process is carried out in parallel for each system, allowing for the control strategy effectiveness to be

evaluated at a global level. It is this multiple UAV control strategy development and evaluation process for small UAVs that serves as the topic area of this paper.

1.2 General Problem

Control strategies for multiple UAV applications have received significant attention in recent years as the utility of small UAVs has increased and cost has decreased. However, the majority of this attention has been directed towards high-level simulation, with very few cases of full implementation and flight test evaluation using embedded control systems. This lack of attention is understandable considering the amount of redundant equipment required to support such evaluations.

An example of such a study could include the development of a control strategy designed to coordinate four UAVs charged with a mission to optimally search for unhealthy radiation levels in a region recently affected by a reactor leak at a nuclear energy plant. In order to fully evaluate the ability of the designed control strategy to accomplish the objectives of such a mission, at least four complete UAVs must be acquired along with their associated support systems and test instrumentation. The cost of four redundant systems alone can prove to be prohibitive, especially when equipment failures and system losses, which are unavoidable during flight testing, continuously add to the cost of sustaining such a study. Beyond cost, the higher combined probability of debilitating component failure substantially elevates the risk associated with performing successful flight tests for the evaluation of the control strategy.

While UAVs are multi-disciplinary by nature, the additional subsystems required to enable relative situational awareness and control for more than one sys-

tem are rooted heavily in engineering disciplines not traditionally associated with UAV control strategy development. Additionally, the majority of previous work demonstrating multiple UAV control systems have all started from scratch, developing custom embedded control systems and testbed aircraft from the ground up, tailoring these systems for the specific application considered. This approach does not provide a legitimate baseline for other projects, nor does it encourage coordinated research on multiple UAV control that is linked together by a common thread.

Lower cost and greater availability of small UAV components has significantly lowered the barriers to establishing the necessary infrastructure for these studies. Additionally, the surge of low-cost hardware and open-source software for UAVs has yielded a launch pad to begin control system research without having to start from scratch and develop the required lower-level autopilot control system. By utilizing open-source software and not developing an entirely new custom architecture, a baseline is set that is adaptable to several projects and supported by a larger community, providing a common thread not only among related projects at Cal Poly, but also among academics and hobbyists around the world.

1.3 Project Goals

Previous work at Cal Poly has focused on the development and high-level simulation of a control strategy for accomplishing formation flight with multiple UAVs, based on concepts such as leader-follower swarm logic, potential function guidance, and virtual waypoint navigation. The scope of this work is to build upon these previous studies and progress the control strategy development and evaluation all the way to flight test demonstration. The goals of this work are summarized as:

- Acquisition of the necessary infrastructure for multiple UAV control studies at Cal Poly
- Adaptation of the developed control strategies for an embedded control system
- Development of a viable communication scheme
- Implementation of a parallel simulation tool able to accommodate multiple UAV control strategies
- Demonstration of formation flight through flight testing

Each of the first four goals listed truly serve as milestones which must be reached in order to accomplish the final goal of formation flight demonstration through flight testing.

Expanding on these goals: first, in order to empower future related studies at Cal Poly in a way that will be both meaningful and sustainable, low cost hardware and open source software will be selected wherever feasible. Second, the formation flight control strategy developed in previous work was developed in Matlab and Simulink and must be adapted for implementation on a commercially-available control system. Third, a communication scheme must be developed to allow the UAVs to exchange state information necessary to accomplish the coordination required for formation flight. Additionally, the communication scheme must be developed to accommodate adequate system monitoring from a portable ground control station for real-time evaluation during simulation and flight testing . Fourth, a HIL simulation tool must be set up to support multiple systems in parallel so that the outer loop multiple UAV control strategy can be tested in simulation and verified before attempting to demonstrate the strategy during flight testing. Lastly, demonstration of the formation flight control strategy can be divided into two stages:

- Leader-Follower convergence to a relative position offset of 15 meters with an offset error tolerance of ± 10 meters
- Flock convergence with two linked Leader-Follower pairs, a total of three UAVs, with each pair achieving the 15 meter offset convergence requirement with the same offset error tolerance of ± 10 meters

These target offset values and error tolerances are heavily influenced by GPS relative position error, discussed in more detail in Section 8. Additionally, controller performance expectation and the testbed's susceptibility to atmospheric disturbances also factor into the derivation of this goal.

The scope of this project does not aim to make vast improvements to the control strategy itself or solve complex problems such as decentralized relative state estimation. Instead, this work will contribute a baseline model for future multiple UAV control studies at Cal Poly, offering viable solutions to many of the challenges associated with propelling multiple UAV control strategy development to a flight test stage. This work will show that it is feasible to use low cost hardware and open source software to demonstrate control strategies for multiple UAVs, such as formation flight, through HIL simulation and flight testing.

1.4 Thesis Layout

This work is divided in to 13 chapters, providing a comprehensive account of the FLOC project. Chapters 1 and 2 cover some background information. Chapters 3 and 4 cover the control system and communication architecture design. Chapter 5 covers the hardware selected for this project. Chapters 6 and 7 discuss the software components of this work. Chapter 8 covers the GPS error characterization performed for this work. Chapters 9 and 10 cover simulation,

and chapters 11 and 12 cover flight testing. The final chapter presents some concluding remarks and suggestions for future work.

Chapter 1 has already introduced the topic area for this work, discussed the general problem that has motivated this study, and has stated the goals for this project. Chapter 2 will provide some background on current and future applications of multiple UAVs with a focus on small UAVs, such as the ones used in this work. Background information is provided for some of the various focuses of this work, such as biomimetic swarm logic, potential function guidance, as well as simulation and flight testing for small UAVs. A brief review of related literature, with an emphasis on other examples of multiple UAV control research is also conducted in this chapter.

Chapter 3 details the traditional components of an autopilot control system architecture and steps through previous work done at Cal Poly to highlight the trajectory of the control architectures developed and how they have influenced the control architecture design for this work. Details of the specific implementation of potential function guidance, virtual waypoint navigation, and swarm organization logic are all presented in this chapter as well. Chapter 4 develops the state-sharing method used to provide each UAV with the other formation members' state information, required to estimate the relative states used for guidance and navigation control.

Chapter 5 lays out the hardware components selected for this project with basic descriptions and justification for their selection. Chapter 6 describes the open source software, ArduPlane, chosen as the backbone of the control system developed in this work. Chapter 7 continues on to discuss the modifications and additions made to the ArduPlane software in order to accommodate formation flight capability and the communication network used for state-sharing.

Chapter 8 details an in-depth experiment conducted to quantify the error expected using GPS for relative position approximations. Chapter 9 discusses the simulation approaches taken and the models utilized for each approach. Chapter 10 discusses the HIL simulation performed in greater detail. In this chapter, the configuration of the simulator is discussed, simulation results are presented, and errors associated with the simulation are discussed.

Chapter 11 focuses on requirements and procedures for multiple UAV flight testing. Here, general requirements are laid out, and the specific flight test set up for formation flight demonstration is discussed. In chapter 12, the flight test results are presented, with a summary of the various attempts as well as analysis and explanation of the errors observed during the testing.

Lastly, chapter 13 provides some concluding remarks, summarizing the contributions made by this work and discussing the overall success of this project. Additionally, several recommendations for future work are provided. For this type of project, this is a very important section, as many of the lessons learned from this work have blazed a trail for similar projects and will allow them to achieve higher levels of success with respect to control studies, without the need to focus so much attention on developing the necessary tools to support a vast interdisciplinary project such as this one.

2 Background

2.5 The Role of UAVs in the World Today

Unmanned aerial vehicles, commonly referred to as "drones" have emerged from the shadows, where engineers have been quietly improving their capabilities over the last few decades, and found their way into the mainstream, becoming a controversial topic that is part of a larger national conversation about the ethics of modern warfare. The Nova documentary, "Rise of the Drones", which aired on PBS, refers to this emergence of UAVs as a new chapter in aviation history that comes straight from science fiction [91]. This documentary and others like it signal a shift in the public appetite to learn more about this newly-controversial technology that is making headlines around the world as more and more of these systems are deployed into high-profile situations.

In general, the American public response towards the increasing capabilities of UAVs and the way in which they are being deployed to wage war remotely has been favorable. However, new questions about executive power and privacy, such as the questions brought forward during the thirteen-hour filibuster on March 6th, 2013 by Kentucky senator Rand Paul, shows that there is beginning to be a shift in this perception, and pushback on "drone" policies have already begun at the congressional and judicial levels of the United States government [44].

One of the first armed UAVs was the MQ-1 Predator, the poster child of "drones", and a symbol of the type of power UAVs now have to conduct warfare in an entirely different manner, providing a smaller footprint, higher precision strike capability, and less direct exposure of military personnel to traditional risks. These capabilities, however, are not always interpreted in such a positive light,

especially by others around the world [31]. The image of an armed MQ-1 Predator has eclipsed other widely-used UAVs, as well as the reality that UAVs come in all shapes and sizes, designed for a wide range of applications. Fully autonomous UAVs have even become common enough to be found in the workshop of an amateur radio-control hobbyist. Additionally, several online communities have emerged in the last few years, dedicated to the development of UAV technologies for recreational, non-military applications, the largest and most famous of these communities being DIY Drones [3].

In fact, with the easing of restrictions on UAVs in national airspace, it is more and more common to find UAVs designed for use in civil institutions. Notable examples often overlooked include: agricultural applications, such as unmanned crop-dusting; environmental monitoring, especially in remote locations; and aerial photography in situations ranging from counting fish to filming extreme sports. More well-known is the growing market for UAVs designed for law enforcement and border security agencies, which holds the promise of greater public safety, but also raises serious privacy concerns [18]. Civil applications are predicted to grow rapidly once regulation is reformed. However, despite the projected market growth, the reality is that small UAVs are not as well developed as their bigger brothers. "For many of these applications to develop maturity, the reliability of [unmanned aerial systems] needs to increase, their capabilities need to be extended further, their ease of use needs to be improved, and their cost must decrease" [22]. The improvements necessary to bring UAVs to the maturity level appropriate for a wider range of civilian applications are likely going to be derived from UAVs designed for military applications.

The "drone" poster child, the MQ-1 Predator, has already been discussed, but this class of UAV is not actually the most common employed by the US

military. "The original vision for UAVs was to replace manned aircraft in the dull, dangerous, and dirty combat roles" [14]. Strike aircraft, such as the armed MQ-1 Predator, is the exception, not the norm for military UAVs. Military applications cover such a large spectrum, it can be useful to discuss them in terms of the class definitions provided in [19]. The Class III and Class II UAVs shown in Figure 2.1 have been more common until, in the last decade, wars in Iraq and Afghanistan have demonstrated the utility of a small, Class I UAV handled by a platoon-sized group.[19].

Figure 2.1: UAV class definitions adapted from reference [19]

2.6 Small, Mini, and Micro UAVs

While the term "small" is used quite often in literature to refer to a distinct class of UAV, presently there is no single, unified definition of what constitutes a small UAV, or under what circumstances a UAV is considered a miniature or

micro UAV, both of which are referred to as MAVs to adding to more confusion. In the previous section the authors of reference [19] chose to classify small UAVs based on the way they were deployed in combat, noting that these are sometimes expendable, often back-packable, and typically require minimal set-up or logistical support. In reference [22] Beard and McLain offer definitions for both small and miniature UAVs. According to their convention, small UAVs

- Are fixed-wing aircraft with wingspans between 5 and 10 feet
- Are usually gas-powered
- Often require a runway for take-off and landing
- Typically designed to operate on the order of 10 - 12 hours
- Intended to carry payloads between 10 - 50 pounds

While in their definition of miniature air vehicles, these MAVs

- Have wingspans less than 5 feet
- Are typically battery powered
- Typically hand-launched and belly-landed
- Operate for periods of time anywhere from 20 minutes to several hours
- Intended to carry payloads ranging from ounces to several pounds

Lastly, in reference [66], the authors define small UAVs in terms of weight and wing span and make a distinction between small and micro UAVs, which they also refer to as MAVs. Figure 2.2 shows regimes defining both small and micro

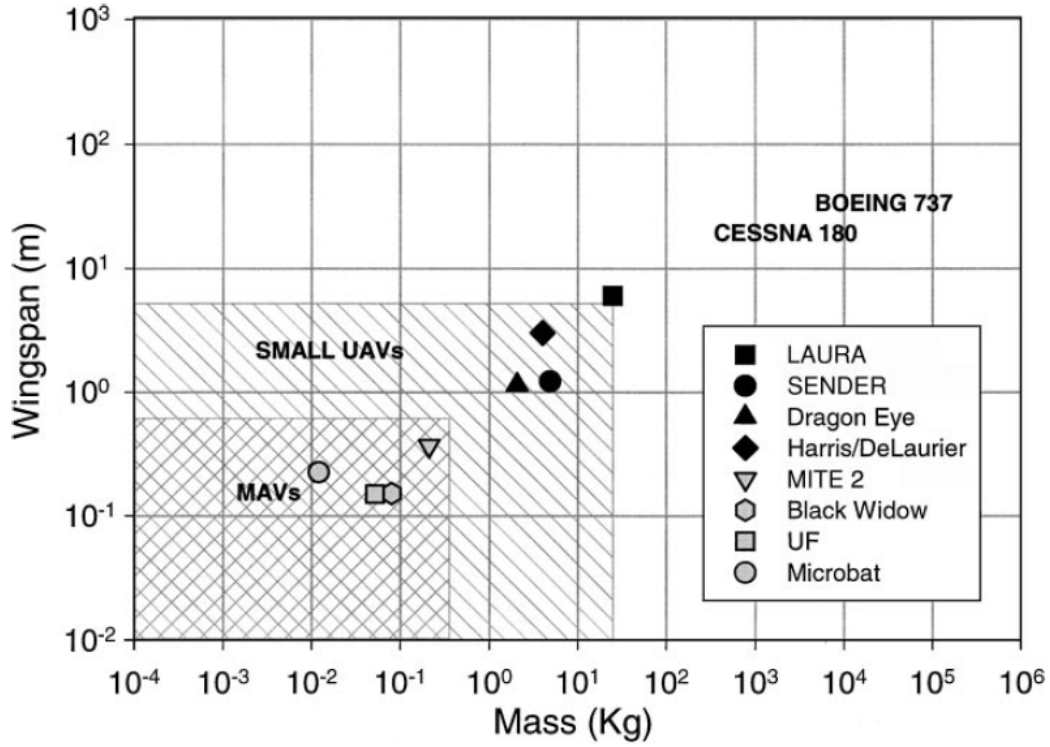


Figure 2.2: Small and Micro UAV classification in terms of wingspan and weight from reference [66]

UAVs with respect to mass and wingspan. Also included in the figure are data points of existing UAVs to provide references that justify their definition.

For this work, the intended definition of a "small UAV" is more consistent with what references [66] and [19] refer to as small or Class I UAVs, but what reference [22] considers a "miniature" air vehicle. This definition will avoid confusion over the acronym MAV from being interpreted as "micro air vehicle", which is the more common usage of MAV in literature, but implies a drastically different class of UAV.

Current applications for small UAVs are heavily tilted towards surveillance and battlefield situational awareness [55]. Additionally, these systems are used to track vehicles, enable communications, capture signals, and detect biological,

chemical, or nuclear materials [19]. Concepts for future applications are poised to take advantage, not only of the relatively lower cost and portability of small UAVs, but also of the implications of multiple UAVs working cooperatively to accomplish a common goal. Low-tech requirements for a single agent in a cooperative behavior concept will allow for mass production, reducing the cost per agent, while increasing the redundancy and therefore, survivability of the system [14]. An often-cited future application is the concept of a mobile ad-hoc network that could be deployed in any environment with a swarm of small, inexpensive UAVs [19]. Another application mentioned often in literature is the concept of cooperative forest fire monitoring. The motivation being that, while satellite imagery provides valuable information to ground personel, the imagery is not provided continuously, as the satellites only pass overhead a few times per day. Low-Altitude, Short Endurance UAVs could provide such continuous coverage, cooperating to identify and monitor new hot spots [82]. In reference [69] these concepts are all mentioned along with a novel idea for parallel distribution of payloads with small UAVs. There are countless other ideas for future applications, and among many of them, there is strong agreement that the applications with the most utility will incorporate small UAVs into teams or swarms to accomplish complex tasks that are not possible with a single UAV, small or large.

2.7 Learning From Nature

Science and technology have a long, rich history of taking cues from the natural world to inspire and guide new developments in every possible field. As technologies shrink, greater attention is being given to parts of the natural world which are sometimes difficult to observe, and whose complexities are being uncovered in parallel with the technologies they inspire.

Aviation in particular was spawned through natural inspiration. Now that technology has enabled aircraft that more similarly represent the scale and performance of the animals that provided the original muse for flight, renewed focus has been directed towards how and why these animals behave the way that they do, and how those capabilities may be translated for applications with utility in modern society. These small UAVs have opened the door for greater understanding of flocking behavior, or even more generally, cooperative behavior, which can be observed, not just by birds, but by a multitude of other examples in nature who make up for their individual size, performance, and utility by working together to solve problems. In *Small Power: The Role of Micro and Small UAVs in the Future* [14], the author notes that

> *"Over the last millennium, some of the world's strongest creatures have become extinct while some of the smallest have flourished. One reason these small creatures have survived is their ability to cooperate as a group to accomplish seemingly impossible tasks. Scientists today are working with the same behavioral concepts for tomorrows UAVs."*

The behaviors that these small creatures exhibit, such as nest construction, foraging, brood sorting, hunting, navigation, and emigration are all local behaviors, with individuals responding to other individuals or their environment [24]. Analyzing these behaviors can provide a guide for what the authors of [24] call an "inverse problem", where individual behavior is designed such that a group can achieve a desired "macroscopic goal".

Specific group behaviors most often considered for translation to multiple UAV applications are flocking and swarming behaviors, observed from birds and insects. Bird formations provide a particularly useful model for several multiple UAV applications, according to the authors of [63], because

1. *"Birds are highly mobile agents capable of flying independently for long distances with small energy.*
2. *The birds only use local neighborhood information to direct their movement within a group.*
3. *There is no specific bird that directs movement of a flock. Yet, overall, the flock moves in a directed manner."*

The hallmark of the behaviors observed in nature is that they are distributed or decentralized behaviors, which is a beneficial control architecture for many small UAV applications which operate under communication or computational constraints. A more in-depth discussion of the benefits of decentralized control architectures for multiple UAV applications, especially formation flight, can be found in [86] and [36].

Algorithms which look to emulate these natural behaviors, sometimes referred to as "biomimetic" algorithms, often seek the most distilled, simplified form to implement these behaviors locally. For the flocking of agents, a simple set of rules are presented in [75], which have been applied to "Boids", often used for computer-generated representations of birds. Simple rules for cohesion, following, homing, dispersion, and alignment are classic techniques for implementing flocking [63]. Biomimetic algorithms span across several fields and disciplines, drawing inspiration, not just from high-level animal group behaviors, but the complexities found in genetic and neural sciences. A good example of how these algorithms can be incorporated into a single application is a project presented in [84], where biologically inspired processes for motion cues, optical flow, pattern recognition, vision and neural control systems, and sensing and communication techniques are all combined with a birds-of-prey inspired search and track algorithm for a UAV envisioned for operation on Mars. Additionally, the investigation of biomimetic algorithms, such as the work presented in [29] translates both ways, providing valuable insight into how and why animals behave as they are observed.

2.8 The Challenge of Situational Awareness

A key component of achieving autonomous formation flight with UAVs is providing situational awareness for the vehicles and the humans monitoring the system. The requirement of situational awareness is not unique to autonomous aircraft, and has been defined in numerous ways, with each definition tuned for the application under review. The most widely-accepted definition of situational awareness is one suggested by Endsley in 1988 which specifies three levels of situational awareness: Level 1) the perception of the elements in the environment within a volume of time and space; Level 2) the comprehension of their meaning; and Level 3) the projection of their status in the near future[35].

With current technologies, meeting these three levels of situational awareness can prove to be quite difficult for an autonomous UAV, and this challenge has been one of the largest barriers to integration of UAVs into civil airspace. Commonly referred to as the "See and Avoid" clause, the Code of Federal Regulations Title 14, Part 91.113 defines right-of-way rules to avoid in-air collisions, emphasizing the operators responsibility to see and avoid other aircraft. Citing safety as their primary concern, the FAA has been reluctant to allow UAVs into civilian airspace, until it can be proven that eminent in-air collisions can be detected and avoided[34]. Although these regulations are sure to change near-term, civil agencies must acquire a certificate of authorization to operate UAVs.

Achieving situational awareness which satisfies all three levels of Endsley's definition lies in the sensor technologies that are available as well as the confidence and accuracy associated with these technologies.

2.9 Potential Function Guidance

Potential Function Guidance (PFG) is one of the main vehicles driving formation flight capability in this work. This guidance method has been a common thread in the previous related work at Cal Poly, discussed in Section 2.12.

With a reputation for relative simplicity and computational efficiency, PFG-related control schemes have a long history of successful employment in unmanned vehicle control and path planning. This guidance strategy was first introduced in [52], published in 1986. The author sought to augment the computationally burdensome and relatively slow high-level path planning algorithms for a robot arm with a lower-level obstacle avoidance strategy that was capable of running real-time. "Artificial Potential Fields" would pull the end-effector of the arm towards the desired position, while also pushing the arm away from obstacles as it traveled.

The concept is analogous to physical phenomena such as the potential fields found in the studies of gravity and electromagnetism, making them somewhat intuitive to visualize and implement. However, the concept is not without faults and limitations, the most famous being the "local minima problem" where the superposition of a global goal potential field and an obstacle potential field result in the guidance of an object to a local minima that is not the global goal. Several studies have been conducted to mitigate these problems, such as those published in references [53] and [79], where analytical analysis or the inclusion of additional concepts attempt to improve upon these shortcomings.

With efficiency gains and simplicity of integration generally outweighing the drawbacks of PFG, this method has been applied to numerous applications which require collision or obstacle avoidance. These include applications concerning

autonomous ground vehicles [80] [87], autonomous underwater vehicles [43] [25] and UAVs [26] [88]. PFG has found a special niche in UAV formation flight applications, which combine the need for supporting evolving goal positions with dynamic collision avoidance requirements. Variants of PFG have been applied in [83] [20] [72] and notably [86], which has inspired the majority of the concepts implemented in this work and is discussed in more detail in Section 2.12. The specific implementation of PFG for this project is discussed in greater detail in Section 3.18

2.10 The Role of Simulation and Flight Testing

2.10.1 Simulation

Flight simulation has come a long way from the 1929 Link Trainer often cited as the first real training simulator. The progress from analog to digital computers in the 1960s introduced the utility of the flight simulator, not only as a training resource, but as a valuable tool for development and testing of new aircraft. Mathematical models with second order effects could be implemented into a "universal" simulator [16]. "As computers became more powerful, the tasks grew more complex" [92]. Modern simulation tools can handle simple linearized decoupled models as well as complicated non-linear, coupled models that vary dramatically in fidelity. The availability of such tools combined with the substantial increases in computing capabilities and rapidly decreasing cost, have made simulation one of the most widely accepted tools in aircraft systems analysis [78]. Driven by the growing complexity of aerospace systems, emphasis on simulation has spanned technical disciplines at every level with a track record of cost reduction and development time compression [74]. For UAVs at the system

level, referred to as UASs, simulation that spans several technical disciplines is a necessity, as a UAS relies more heavily on systems not traditionally considered in the aerospace engineering discipline. For UAVs, simulation not only empowers designers to reduce development risk for new vehicles or traditional subsystems, but also reduces the types of risks commonly associated with automatic control systems and also allows for an evaluation of robustness. For UAVs meant to carry sensors and equipment, this process is invaluable. Tools such as Software-in-the-Loop (SIL) and Hardware-in-the-Loop (HIL) simulation can test the most error-prone components of UASs before the UAV has ever flown, saving expensive electronics from devastating crashes in flight testing. Coupled with accurate models of the UAV, performance requirements can be approximately met through control system tuning, saving time and money required for flight testing.

2.10.2 Limitations of Simulation for Small UAVs

A natural tendency with simulation, according to [76], is to simulate too much detail, instead of too little. The author goes on to contend that a model should always be designed around the goals of the study, and that adding more and more detail to a model is not going to allow the computer to solve all of the engineer's problems. In fact, the truly significant aspects may be lost among the trivial details [76]. While computers and simulation have come a long way since the publishing of [76], the author's warning is still relevant. For the simulation of small UAVs, it is important to keep in mind the goals of simulation, such that the fidelity reflects those goals, and resources are not spent in places where there is minimal value added. With regard to small, low-cost UAVs, many of the traditional bennifits of simulation break down. The argument made in [48] is that the resource trade-off between simulation and flight testing becomes skewed

at this level, and that, while simulation is still an essential tool for keeping the amount of flight testing and risk manageable, the fidelity and implementation method of simulation should be selected with goals calibrated accordingly.

The largest factor for this shift in relative resource requirement between simulation and flight testing is the over-all lower cost of the system. With little redundancy and minimal payloads, the cost of replacement is relatively low. Furthermore, the risk of injury or damage caused by a failure is also much lower [48]. The lower cost and risk, however, is not the only reason for the shift. Small UAVs are also more difficult to model. Many of the assumptions used when modeling traditional fixed-wing aircraft are often violated, such as assuming the aircraft is operating in mostly linear flight regimes. Additionally, low Reynolds numbers and relatively high atmospheric disturbance both introduce other unsteady or non-linear aerodynamic phenomena, which have magnitudes that can not be ignored at this scale [66]. Many studies, such as the one presented in reference [51], rely on traditional tools for developing stability derivatives corresponding to linearized relationships about several points of operation, such as Digital Datcom. However, tools such as these are often developed for and validated by comparison to full-scale aircraft, and should be used with caution when extrapolating their output to small UAVs. Ultimately, achieving the same level of fidelity for a small UAV in comparison to a full-scale aircraft can prove to be more difficult with currently available tools and data-sets. This should be considered when developing simulation goals for a small UAV system.

2.10.3 Flight Testing

In contrast with simulation, flight testing actually becomes more manageable for UAVs at the smaller end of the scale spectrum. UAVs, in general, present

a number of challenges with respect to flight testing. Translating traditional flight test techniques from manned-aircraft to UAV applications can prove to be difficult, not only due to the absence of sensory cues usually experienced by a pilot, but also because the vehicle, although an important component, is only one part of the complex system [89]. For UAVs, the extensive infrastructure traditionally required for manned flight testing is not required due, in part, to the instrumentation that is already included on-board the UAV and streamed down through a telemetry link. However, regulations and safety concerns make a suitable testing facility difficult to find, often requiring project partnerships with government institutions [89].

While some of the challenges of UAV flight testing, such as reaching stabilization or measuring the full aircraft state, are exacerbated when testing small UAVs, fortunately several other challenges are mitigated. Especially those concerning the infrastructure requirements for testing. As mentioned in the previous section, costs and risks associated with flight tests of small UAVs are significantly reduced [48]. Additionally, the size of the required test field shrinks, and in the case of many small UAVs, the requirement of a runway is irrelevant, as many of the aircraft falling in to this category are hand-launched and belly-landed.

2.11 Multiple UAV Control Literature Review

One of the motivations for this work is the investigation in to what makes small UAVs different than large ones, and in what applications their utility can be maximized. This basis for investigating formation flight demonstration with a low cost, open source configuration is supported by the vast collection of related research and publications. Broadly, multiple UAV control studies represent the majority of research conducted with small UAV applications in mind.

The most extensive set of approaches to various multiple UAV control applications can be found in the large body of simulation-based studies, which have more room to explore new concepts without being grounded by the difficulties of hardware implementation. A brief overview of some of these works is presented here to give this study some context. An overview of emerging results in cooperative UAV control is presented in [77]. In this paper, the author discusses several studies which have addressed one of the three predominant cooperative UAV control subsets: 1) Aerial surveillance and tracking, 2) collision and obstacle avoidance, and 3) Formation configuration. Additionally, in [60], a method of cooperative control for simultaneous arrival is presented and demonstrated through simulation. The study uses a decentralized decomposition strategy in order to find the team-optimal solution to the task. In [82], the authors present a study for cooperatively monitoring forest fires using a role-dependent decision making strategy to monitor existing hot-spots as well as seek out new hot-spots as they emerge. Threat navigation for a swarm of UAVs using a bio-inspired concept of digital pheromones is presented in [71]. Reference [50] demonstrates, through simulation, multi-vehicle path planning that guarantees complete coverage over a region of interest using a spanning tree algorithm.

More specific to the type of multiple UAV control strategy presented here, there are several simulation-based works that investigate formation flight. In [65], the author presents simulations of a leader-follower formation flight with an adaptive output feedback control technique. Formation flight of three quadrotors is presented in [39], where the authors use a pre-assigned leader-follower configuration with a proportional-integral (PI) controller. Close formation flight of multiple UAVs is presented in [38] using LQR control for tracking and a decentralized, Dijkstra-based formation management algorithm. In [70], point mass

simulations of multiple UAVs are conducted, achieving formation with "Boids"-like swarm intelligence concepts. Kim et al. present an interesting approach to rotary UAV formation flight in [54], where a behavior-based decentralized approach is undertaken with a controller design using a feedback linearization rule with a diffeomorphic transfer map, derived for a three degree of freedom (3DOF) point mass model. Lastly, in [17], simulations for accomplishing surveillance with UAV formations is presented. In this study, the authors use an asymmetric formation control structure and a PI controller for flock-center tracking.

Not as numerous are the number of works approaching multiple UAV control strategies with an emphasis on hardware implementation and demonstration. Recently, the number of these publications and the programs supporting them have begun to swell, some even attracting fame outside of the academic community.

The most notable of these programs is the GRASP laboratory at University of Pennsylvania (UPenn). Videos of their small quadcopters performing astonishing cooperative maneuvers have gone viral on the internet. Similarly, MIT has produced highly publicized work out of their ACL program. One of the secrets behind their success is the infrastructure in place to conduct research on multiple UAV control. The GRASP testbed at UPenn, described in [62], and the RAVEN testbed at MIT, described in [47], both support these studies with a complex motion-capture system, Vicon. This infrastructure allows for studies to be conducted indoors with a high level of situational awareness accuracy, utilized by the test vehicles for coordination purposes. While exciting, ground-breaking concepts can be demonstrated in this environment, the infrastructure requirement and lack of application to real-world problems makes these studies somewhat limited.

There are, however, several examples of programs conducting multiple UAV control studies that have looked at more realistic near-term applications, demon-

strating their control schemes outdoors, where these situations are most likely to occur.

Brigham Young University (BYU) has conducted several studies relating to cooperative control, such as [23], published in 2006. In this work, a cooperative surveillance control scheme was demonstrated with three UAVs, persistently capturing images of a target at fixed intervals of time. These experiments were conducted with a testbed designed in-house including a commercial-off-the-shelf (COTS) small, foam airframe, a custom autopilot, as well as COTS communication and radio-control (RC) electronics.

The APL program at John Hopkins has supported several studies like [19], published in 2006. Here, several approaches were demonstrated including a consensus variables approach for a dynamic surveillance network and a stigmeric potential field approach for cooperative search. The testbeds used in [19] were developed by third party manufacturers, utilizing Cloud Cap Piccolo autopilots for low-level vehicle control as well as COTS communication and RC equipment.

The Georgia Institute of Technology (Georgia Tech) supports several multiple UAV control studies, described in [49]. This work is conducted out of the UAVRF, headed by Eric Johnson. UPenn, while better known for their quadrotor work, also supports fixed-wing studies out of their GRASP lab, like the one presented in [21]. Similarly, MIT supports several fixed-wing studies through the ACL program that reach beyond their RAVEN testbed, which is described in [46].

Additionally, several programs outside the United States have gained momentum. A good example is the Australian Centre for Field Robotics at the University of Sydney, which supported the work presented in [28], published in 2010. In this work, information gathering missions are addressed, studying decentralized

approaches with different levels of communication: No communication, limited coordination, and full cooperation. The testbed used for the work includes a custom RC-scale airframe, custom autopilot, differential GPS, full inertial sensor suite, color and infra-red cameras, as well as COTS communication and RC electronics.

While it is important to look at how different projects have approached multiple UAV control problems, most applicable to this work are studies which demonstrated some variant of formation flight. In [59] and [32], which were supported by the Air Force Institute of Technology and the Chung-Shan Institute of Science and Technology, respectively, distinct formation flight control schemes and implementation strategies were presented, but were, ultimately, only demonstrated with a single UAV and a virtual leader simulated from the ground. In [59], an ARF Rascal 110 RC airframe was used with a Cloud Cap Piccolo II autopilot and COTS communication and RC components. Based on telemetry data sent to the ground control station from both UAVs, new waypoints were generated on the ground and fed back up to the trailing aircraft. Airspeed of the trailing aircraft was dictated by a ground-based ad-hoc controller. In [32] a testbed was developed with a custom airframe and autopilot system. Fuzzy logic control was used for the control system, with relative state information transmitted to the follower from the leader. To mitigate the vulnerability of this scheme to data-link loss, relative states are also estimated with an extended kalman filter, which kicks in when the link is lost, and is corrected once the link is established again.

An overview of the first phase of the NASA Dryden Flight Center Autonomous Formation Flight Project is presented in [42]. This full project objective was to bring autonomous formation flight to a technology readiness level high enough to attract commercial cargo operators and military interests. In this phase, leader-

follower formation was achieved within a pre-determined position error budget using two F/A-18 aircraft. The original flight control computers were used with modifications to the following aircraft for lateral and vertical commands based on a PID controller output. State information was sent from the leader to the follower aircraft, and the pilot in the trailing aircraft was responsible for throttle control, which dictated spacing.

Studies [33] and [90] conducted at the University of Singapore, and [68] at Georgia Tech focused on formation flight applications which included the use of rotary vehicles. At Stanford, studies [56] and [57] were conducted to look in to accomplishing formation flight with relative state information provided by a vision system.

Lastly, the work presented in [41], supported by West Virginia University, represents the best example of successful formation flight demonstration with at least three UAVs. The scope and time-line of this project makes it difficult to compare to the work presented here, but none-the-less, there are many similarities in the approach taken to solve the formation flight problem. Custom-developed airframes with features similar to the YF-22 make up the backbone of the testbeds used in this project. A custom avionics system was integrated, complete with a high-end flight computer and full sensor suite, including a 20Hz GPS receiver. Additionally, COTS RC components were integrated for manually flying the aircraft. This project was quite extensive, with over 100 flights were conducted over the span of three flying seasons between 2002 and 2004. Ultimately, successful two-aircraft (Leader-Follower) formations were demonstrated four times, and there was one successful demonstration of a three-aircraft formation.

2.12 Previous Work at Cal Poly

The work presented in this study is intended to be an extension of previous work conducted by Cal Poly faculty and graduate students over the last decade, defining its scope based on the trajectory of this previous work. The body of previous work is made up of four different projects that have all contributed, in some fashion, to the work presented here. While specific contributions will be highlighted throughout the discussion, it is important to provide some context by giving a brief overview of each of these previous works.

Inspired by the observation of flocking geese, Eric Mehiel, now the Aerospace Engineering Department Chair at Cal Poly San Luis Obispo, authored a paper with Mark Balas titled *A Rule Based Algorithm That Produces An Exponentially Stable Formation of Autonomous Agents*[61]. Published in 2002, while Dr. Mehiel was a graduate student at The University of Colorado, Boulder, this paper provided the foundation for an interest in formation flight that has guided several projects at Cal Poly. In this paper, a simplified two-dimensional model is used to simulate exponentially stable formation flight using a discrete-time, non-linear logic based control algorithm. The main contributions made by this paper to the future works at Cal Poly include distinct near-field and far-field control regions, the use of a multiple leader-follower pair architecture, and the logic necessary to produce a structured 'V' formation.

In 2008, Masamitsu Tsuruta presented a thesis titled *An Integrated Formation Flight Algorithm Via Potential Function Guidance and Biomimetics*[86]. Advised by Dr. Mehiel, Tsuruta integrated many of the concepts from [61] with a more realistic 6DOF linearized aircraft model and a PFG algorithm. Using a standard PID control scheme for low-level lateral and longitudinal controllers,

Tsuruta simulated results in a Matlab/Simulink environment, demonstrating the functionality of the PFG algorithm and the decentralized formation controller. While Tsuruta noted one of the benefits of the decentralized technique was that it could be accomplished with limited communication, this was not addressed in his simulation, with each formation member having perfect knowledge of the other aircrafts' relative position. Furthermore, simulations were run for an ideal condition where the formation leader's body frame was aligned with the inertial frame, with the leader simply proceeding along a straight trajectory in the X-Y plane. Ultimately, Tsuruta showed the utility and relative simplicity of using a PFG algorithm for formation control as well as collision avoidance with multiple agents participating in the formation, linked together in leader-follower pairs.

A thesis titled *Development of a Small and Inexpensive Terrain Avoidance System for an Unmanned Aerial Vehicle via Potential Function Guidance Algorithm* was presented by Shane Wallace in 2010[88]. While this work deviated from the concept of formation flight, it retained the use of a PFG algorithm, similar to what was developed in [86], and began a push to apply these guidance concepts to hardware. While the full system was never completed, Wallace set out to construct the terrain detection sensor using cheap COTS hardware and intended to implement this sensor with a custom algorithm to map terrain obstacles and feed a commercially-available autopilot new waypoints to direct the UAV around the obstacle. In the end, Wallace was not able to fully integrate the terrain avoidance system with the guidance algorithm, but did demonstrate the guidance algorithm through HIL simulation. This HIL simulation was conducted with a Cloud Cap Piccolo autopilot and the software provided with the system. Based on information from a simulated terrain detection sensor, the PFG algorithm provided commands for the placement of new waypoints which were then added to the

waypoint queue used by the Piccolo for navigation.

Although not yet completed, in 2010, Michael Manuel began work on a thesis tentatively titled *Development and Implementation of a Potential Function Guidance and Virtual Waypoint Algorithm UAV Navigation and Collision Avoidance*[58]. While Manuel's focus also deviates from the initial theme of formation flight, his methodology utilized the PFG algorithms presented in [86] and [88] and like Wallace, focused more attention to hardware integration. While Wallace made use of the relatively high-end Piccolo autopilot system, Manuel, instead, shifted his focus to the low cost, open source autopilot system, ArduPilot. Additionally, instead of adding waypoints to a pre-existing queue, Manuel made use of a concept he dubbed "Virtual Waypoints", where the waypoint used for navigation purposes would constantly be moving, assigned based on the output of the PFG algorithm. The obstacles used in his study were artificial representations of obstructions or terrain, used in order to limit the scope of his work to not include obstacle detection. Manuel's latest draft shows that he was able to successfully demonstrate his method through both simulation and flight testing.

2.13 Unique Nomenclature Utilized

As expected of any technical report, several acronyms are utilized throughout this paper, some of which have been presented already. In general, acronyms are presented first with their full description before stand-alone use in the text. Acronyms such as UAV, for unmanned aerial vehicle, are not unique to this paper, and should not require much explanation. Some acronyms, such as PFG, used for potential function guidance, VWP, for virtual waypoint, and GCS, for ground control station, are used heavily in this work, but are not necessarily unique to this paper either.

In several places, this work is referred to as the FLOC project, which stands for Formation flight demonstration utilizing a Low cost, Open source Configuration, the title of this work (With the selected acronym letters admittedly cherry-picked). Additionally, the FCOM network is referred to often as well. This is not so much an acronym as a title given to the inter-UAV network that stands for the Formation Communication Network. Lastly, a unique reference frame, the LNAV or localized navigation frame, is presented in this work. This reference frame is simply the NED frame, rotated about the z, or down, axis in an amount equal to the UAV's heading. This frame is especially convenient for describing formation distance offsets.

3 Autopilot Control System Architecture

To the average person, references to autopilots often bring to mind the latest science fiction movie, with a single button push and an intelligent-sounding voice with a prickly personality. Because the cutting edge of automation is what often gets the headlines, this perception is somewhat justified. However, to qualify as an autopilot, a system does not need a cool accent.

Historically, autopilots have been implemented to augment a pilot's control, not replace it. In military aircraft designed to push the envelope of possibility, autopilots can turn an inherently unstable aircraft into something manageable for the pilot without losing the maneuverability the aircraft was designed for. Stability augmentation systems like these are also often used to make inherently stable aircraft simply easier to fly, requiring less pilot workload. Higher level autopilots, often found in commercial airliners, control velocity, altitude, and heading, further reducing the workload of pilots, as well as squeezing every possible fuel savings by programming the autopilots to follow a schedule that will maximize efficiency. Automatic landing and takeoff autopilots are also used extensively in the commercial airliner world to assist in poor atmospheric conditions.

Up until the relatively recent widespread use of UAVs, fully autonomous aircraft were rare and limited. The higher levels of path planning and decision making require a greater ability to detect and interact with the world and therefore have typically been reserved for human operators. It is important to understand each level of autonomy so that function and dependencies of each level can be put into context.

3.14 Example Control System Architectures

In [22], Beard and McLain detail a system architecture for use throughout their text. Figure 3.1 shows the block diagram used to describe their system, with each block differentiated by function and required inputs. It is important to note that the authors refer to the more basic form of an autopilot in their "autopilot" block. This is the form in which there is an inner loop controller which controls the roll and pitch angles as well an outer loop controller for airspeed, altitude, and course heading. This system architecture will serve well to describe the approach taken in previous works as well as the approach presented here. In [61], where

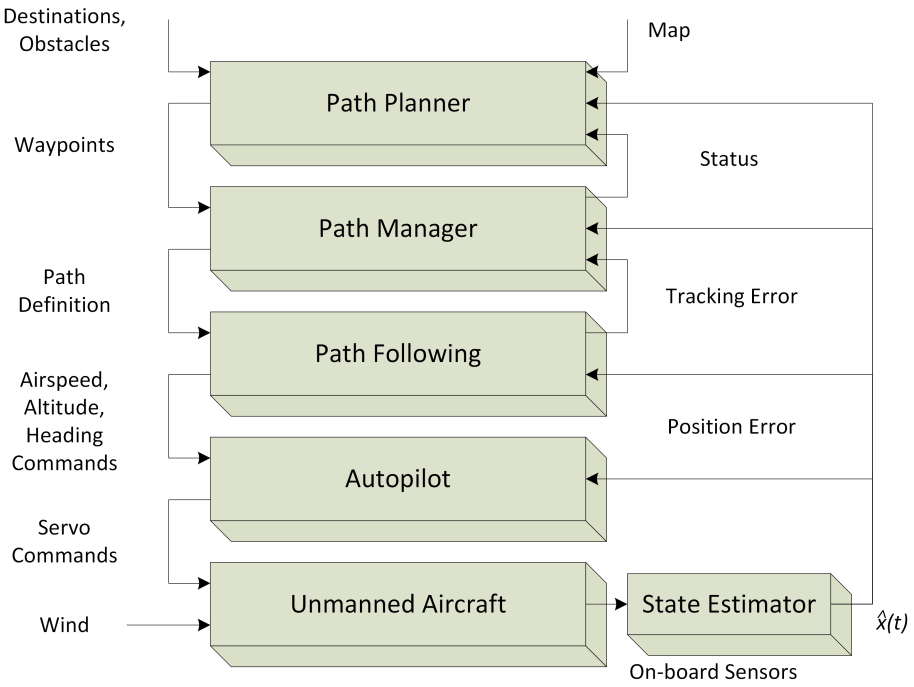


Figure 3.1: A baseline representation of a UAV control system architecture.

the author investigates the algorithms and associated stability of a formation flight control scheme to emulate migrating geese, he utilized a simplified system architecture reflecting his assumptions.

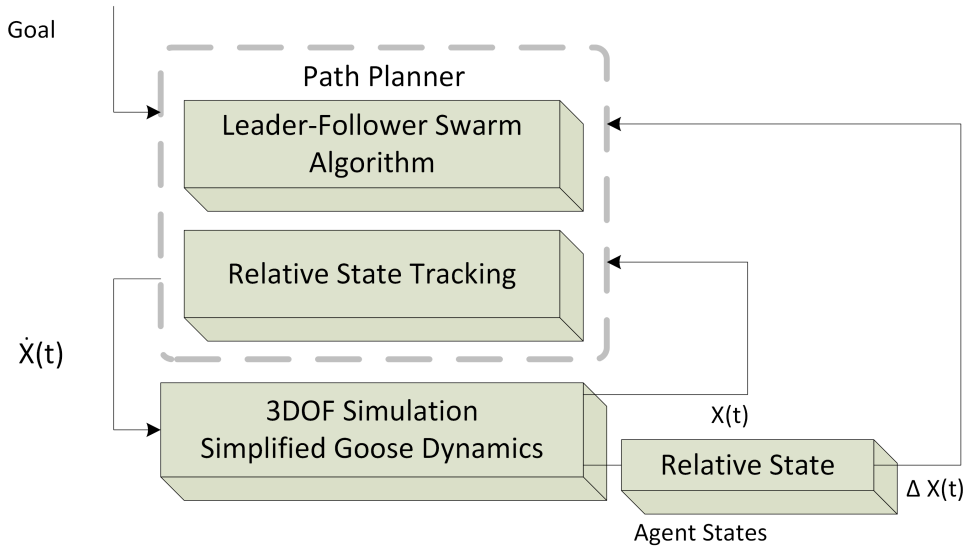


Figure 3.2: A system architecture for formation flight using a rule-based algorithm and simplified dynamics, presented in [61].

Figure 3.2 shows a representation of his system architecture so that it may be compared with the architecture in Figure 3.1. The author's architecture is best described by sub-dividing the path planning block into two distinct sub-blocks: one for implementing leader-follower swarm organization and one for generating commands based on relative distances. With the author's assumptions and simplifications, the commanded values are tracked perfectly, removing several blocks from the architecture. Furthermore, with the assumptions of perfect state estimation and simplification of the dynamics, the only other block in the author's system architecture is the flock member dynamics using two degrees of freedom. This architecture is appropriate for high level algorithm development, but must be expanded upon in order to simulate this control scheme in a real world application.

In [86], the author's goal was to integrate the concepts from reference [61] with a simulated aircraft model, expanding upon those concepts to demonstrate how formation flight may be accomplished with a real aircraft. The addition of a lin-

earized six degree of freedom aircraft model and limited actuator models required a more complex system architecture to control each member of the formation. Figure 3.3 shows a representation of the system architecture utilized in reference [86]. As was the case in the system architecture presented in reference [61], it is useful to discuss the path planning block as a combination of two sub-blocks, separated in the same fashion. With the leader-follower swarm organization block integrated relatively unmodified, the only change here is the potential function guidance block, responsible for generating the autopilot commands. The autopilot block is included in the author's system architecture to account for the altitude, velocity and heading controllers necessary to carry out the respective commands generated by the potential function guidance algorithm. Lastly, the UAV block represents the 6DOF linearized plant model and constrained actuator models for a well documented business jet included in the simulation.

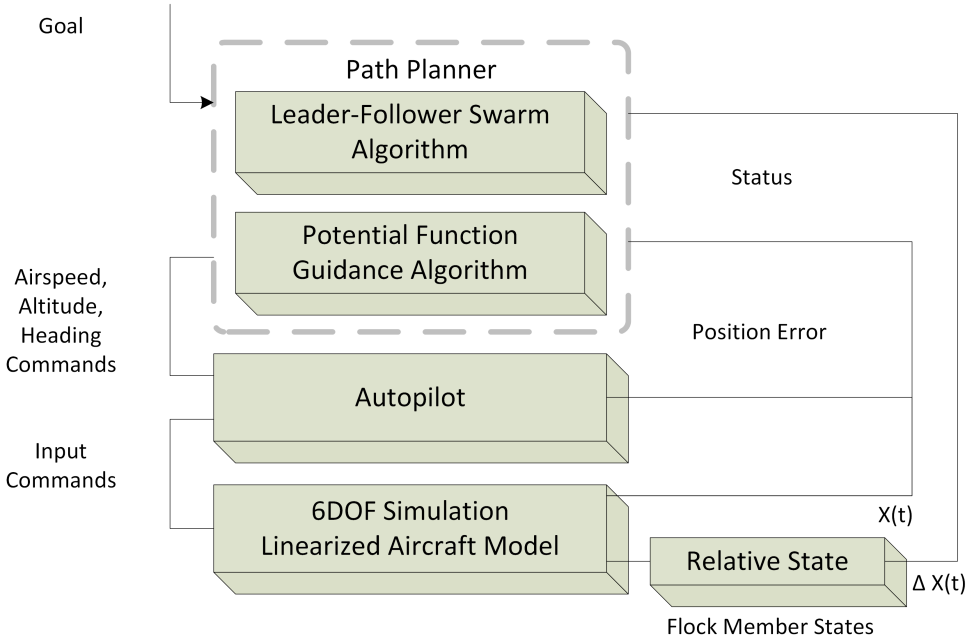
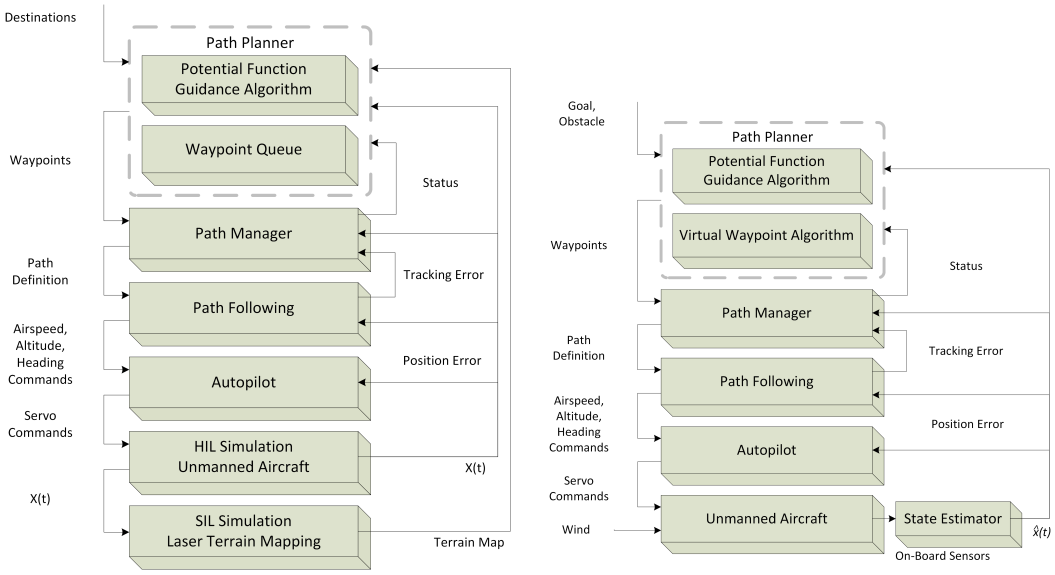


Figure 3.3: A system architecture for formation flight simulation using a 6DOF aircraft model, presented in [86].

With a focus on translating the algorithm from reference [61] and not necessarily representing some of the constraints that would be applied in a real-world flight demonstration, the author makes some simplifications, reducing the complexity of their system architecture when compared with the standard presented in [22].

Shane Wallace and Mike Manuel, authors of [88] and [58] respectively, took the concept of potential function guidance in a different direction. With a focus on navigation and terrain or object collision avoidance, they used a system architecture more representative of the architecture presented by Beard and McLain. Figures 3.4a and 3.4b represent the architectures utilized by these studies, presented side by side to highlight the slight differences between the two approaches.



(a) A system architecture for terrain avoidance, presented in [88].

(b) A system architecture for obstacle avoidance, presented in [58].

Figure 3.4: System architectures presented in previous works utilizing a common thread of PFG for terrain and obstacle avoidance

In [88], the author used a control architecture centered around the use of a commercially-available waypoint navigation based autopilot system, wrapped by a custom terrain avoidance system. While he was never able to fully demonstrate the developed hardware, his control algorithm operated by adding waypoints into a pre-existing queue that would theoretically navigate the vehicle around a terrain obstacle. The potential function guidance algorithm served as the path planner, and was used to place these additional waypoints. The Piccolo autopilot, represented by the next three blocks in the system architecture, simply followed the resulting path as if it was a pre-planned waypoint mission. To sense the terrain, Wallace experimented with a gimballed laser range finder, which is included as a sub-block in the state-estimation block, with the HIL simulated sensor suite making up the second sub-block of the state estimation block.

In [58], the author took on a different approach focusing more on demonstration, and less on the detection hardware. While his final configuration has not been detailed, his original approach utilized a RC aircraft modeled after Aerovironment's Raven, represented by the UAV block, and ArduPlane, an open-source autopilot software that is designed to operate on the ArduPilot Mega (APM) hardware, represented by the lower level system blocks. While using the APM sensor suite for state estimation, the author injected an artificial object, represented by the simulated detector sub-block, in order to demonstrate the control system's ability to successfully navigate the aircraft around the obstruction autonomously. Instead of adding waypoints to a growing queue, the author implemented a "virtual" waypoint sub-block in the path planning system block, where a new waypoint is generated during each cycle, determined by the output of the PFG sub-block directly above. The virtual waypoint provides a clever way to avoid some of the lag and uncertainty in the decision process which was

encountered by the author of reference [88] during his HIL simulations. While this strategy allows for use of the core functionality of a commercially-available waypoint-based autopilot system, it is slightly more invasive than simply adding waypoints to the queue. This extra complexity is necessary, especially when obstacles in motion, like other members of the formation, drive down the control system's tolerance of lag.

The largest difference between a "virtual" waypoint and a conventional waypoint is that, in theory, the aircraft is never allowed to reach the virtual waypoint. All of the waypoint path-following algorithm are used for guidance, which are usually built into a commercially-available autopilot system, but the waypoint itself is being redefined at a relatively high frequency, placing the new waypoint a predefined distance in front of the aircraft with a bearing defined by the potential function guidance algorithm.

Combining requirements derived from the above previous works, and considering real world demonstration of formation flight with a commercially-available autopilot system, the necessary control system architecture components can be identified as shown in . At the highest level, in the path-planning block, some form of swarm logic must be present to organize the members. The other two sub-blocks in the path planning system block include a block to generate commands based on relative states of members, and a block to translate those commands into something usable by the highest level of a commercially-available autopilot system. The path manager, path following and autopilot blocks will be furnished by the chosen commercially-available autopilot system, with a slight addition to the path following block that will include a velocity matching sub-block. Because of the decentralized approach, the control system architecture only includes one UAV system block, not a block for each UAV system in the formation. However,

the state estimation block for a formation flight controller must be subdivided into two blocks: one to estimate the state of the UAV with respect to inertial space, and one to estimate the state of the UAV with respect to the other UAV formation members.

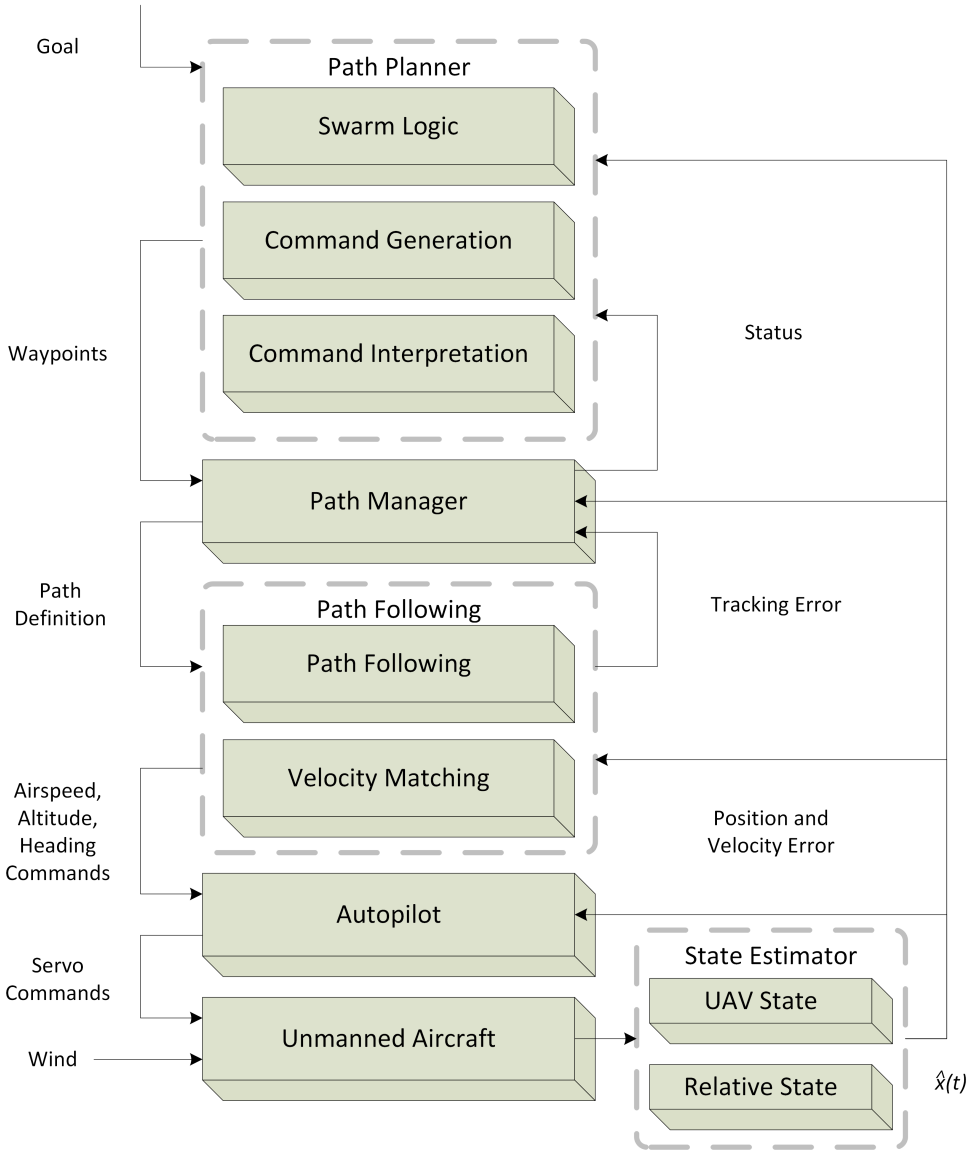


Figure 3.5: A general control system architecture for formation flight demonstration.

3.15 Control System Architecture Presented

The specific control system architecture presented in this study was constructed based upon a few assumptions which heavily influenced its final form. First, in an effort to maintain continuity with previous work, the backbone of the presented architecture was dictated by some of the decisions made in those works. The goal was to not diverge from the methods presented in the previous works, except in matters that concerned feasibility during flight testing. Second, as seen in references [88] and [58], using a commercially-available autopilot system was a top priority to manage the scope of the project and encourage focus at a higher level of control instead of getting bogged down in the development of the lower level autopilot systems. Autopilot development from scratch is the subject of entire senior projects or theses, depending on the strategy chosen.

With the use of commercially-available autopilot systems, a modular approach encourages adaptability and preserves software that has been rigorously debugged and tested, making troubleshooting more manageable during the final stages of the project. Limiting the number of variables during testing is always a positive goal. ArduPlane, which was the initial autopilot system selection in reference [58], is capable of being modified heavily because of its open-source software. However, another goal set during the implementation of formation flight capability into the ArduPlane code was to limit modifications to the existing code as much as possible. Keeping the other modes fully functional despite the addition of the optional formation flight mode would make gain tuning and debugging more feasible during testing, which was highly desirable. Accomplishing this desired modularity often resulted in a more complicated library development process, requiring agreement between established conventions and structure. However,

this lead to faster error identification and more efficient debugging, as well as the ability to use tools such as HIL simulation interfaces that had already been developed to work with the ArduPlane conventions. The last assumption that was made when designing the control system architecture was that the state information for all of the UAVs participating in the formation will be available to each UAV. This information could be furnished in a variety of ways, which will be discussed in more detail later, but the assumed formation member knowledge is key in the architecture development.

Figure 3.6 shows the detailed control system architecture presented for this project. The architecture presented meets all the requirements set forth for a formation flight control system presented in the previous sections, and borrows specific implementation strategies from the related previous work.

3.16 Near-Field and Far-Field Considerations

As discussed in references [61] and [86] it is also beneficial to design the formation flight controller for use in two distinct "regimes" of the formation process. When the UAV is far away from its local leader, it is considered to be in the far-field regime. As the UAV moves closer to its local leader, and the relative distance becomes less than the regime threshold, χ, the UAV enters the near-field regime. The specific method of determining χ will be discussed in section 3.20, but the minimum threshold value is limited by the UAV's manueverability. In general, when the algorithms are developed to operate in the far-field regime, the focus is finding the other formation members and meeting up with the formation. In the near-field regime, the focus shifts to accomplishing both relative position and relative velocity tracking, encouraging leader-follower convergence while avoiding collisions with other formation members.

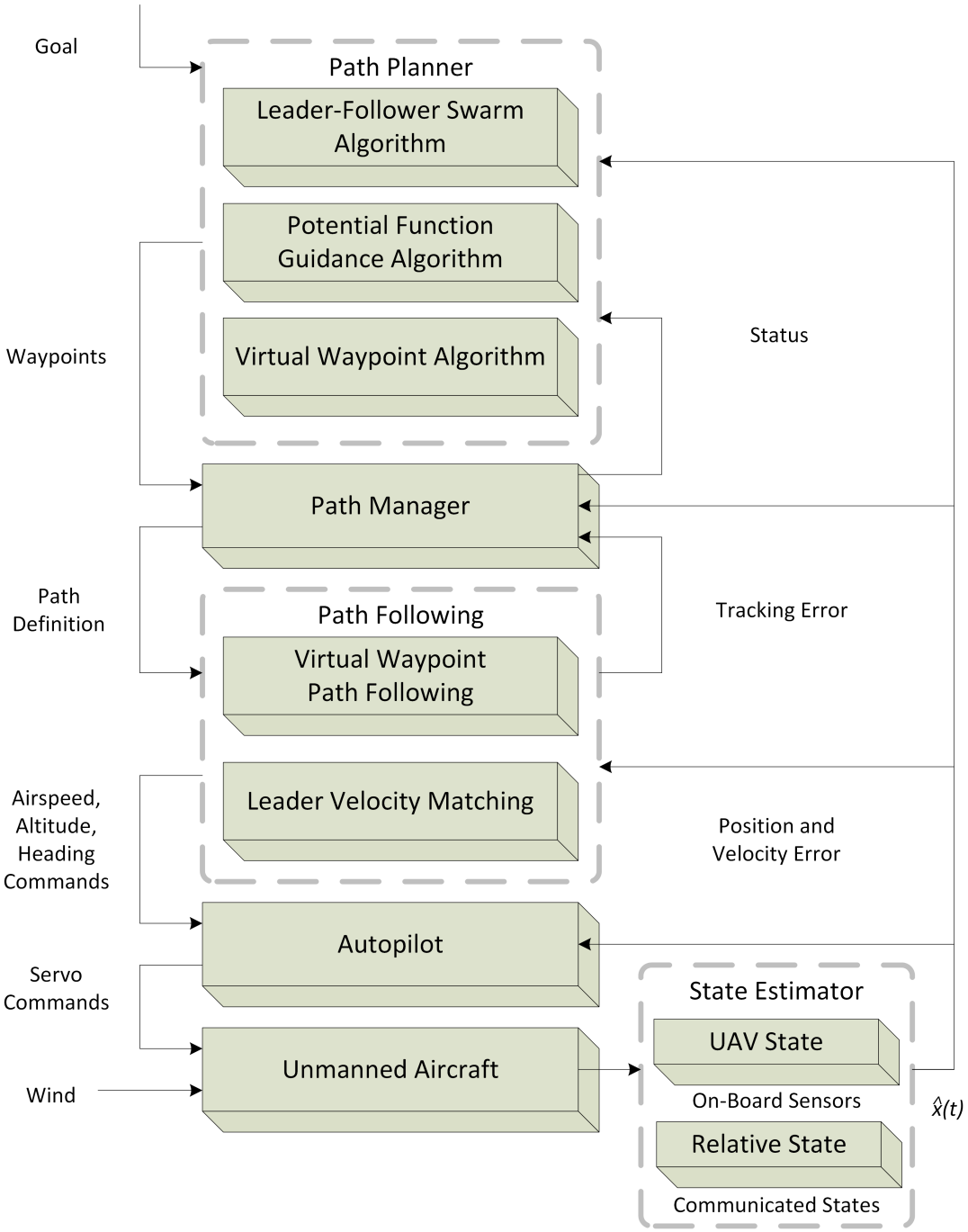


Figure 3.6: The system architecture presented in this work.

3.17 Swarm Organization Algorithm

One of the common threads of the previous work in references [61] and [86] was the adherence to decentralized control as much as possible. To adequately

represent the original muse for these studies, migrating geese, there should be no central controller. Instead, each UAV should be making their own decision based on observable information such as relative position, both between other formation members and a global landmark, and relative velocity.

Figure 3.7: The determination of global versus local leadership

The swarm organization logic proceeds as follows:

1. Identify who is participating in the formation and evaluate leadership

   Case 1: If no other member is in front, the UAV becomes the global leader

   Case 2: If there are members in front, pick the closest one, that is the local leader

2. Choose the side on which to follow the local leader

   Case 1: If there is more than one other member in the formation, the side chosen will be opposite of the flock center

   Case 2: If there is only one other member (the global leader), the side will be chosen based upon the side in which the following UAV is approaching.

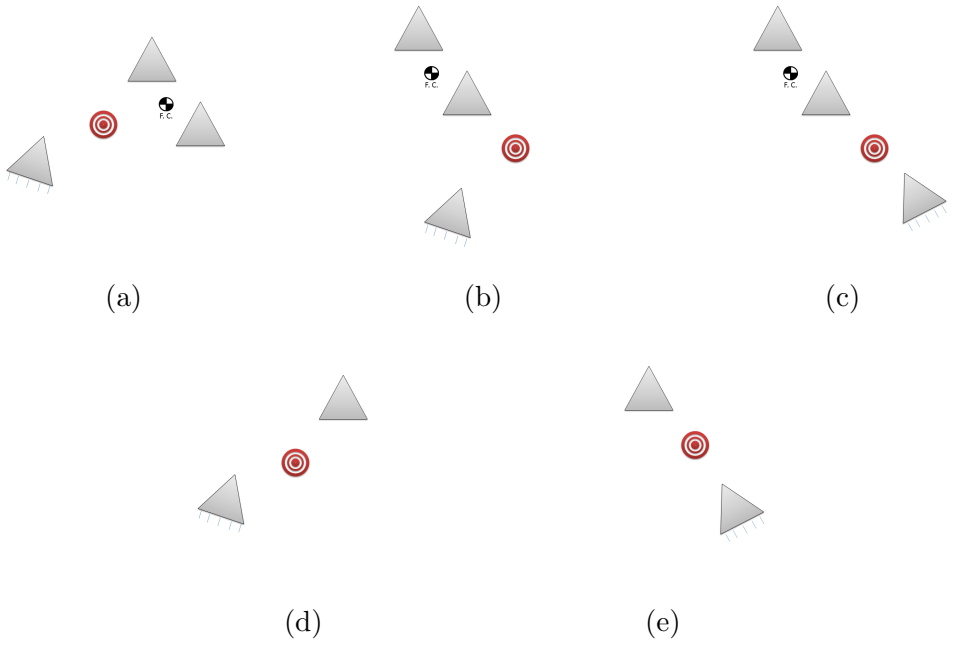


Figure 3.8: Examples of side determination for swarm organization logic

While the algorithm itself is the same whether the UAV is in the far-field or near-field regime, it is not evaluated once the UAV has determined its local leader. Because small fluctuations in relative distance is expected as each formation member battles long-period modes to maintain their commanded relative state, leadership changes based on those fluctuations should be avoided. Instead, the algorithm is only evaluated if the UAV is currently the global leader, or if contact is lost with the UAV's local leader. This strategy allows the expected fluctuations to occur without adding unpredictable leadership changes, while still maintaining the desired dynamic response to changes in formation membership.

This swarm logic is based heavily on the logic developed by Mehiel in reference [61] who shows that the emergent behavior of this strategy results in a V, or half-V formation when combined with a controller maintaining relative distances between the leader-follower pair.

One major difference in the swarm logic developed for this work is the addition of a reference system for global leadership. The assumption made in references [61] and [86] is that there exists a global reference to define the concept of one agent "in front" of another. The obvious solution is to use a cardinal direction, as birds do when they migrate. However, for the purpose of flight testing, it is not feasible to simply have the test aircraft head North. Instead, the formation members will converge while the global leader follows a large,circular loiter orbit around the test area at the Cal Poly Experimental Flight Range (EFR). This will ensure that the aircraft have enough time to achieve formation without the risk of the UAVs flying beyond the line of sight of the RC pilots who must constantly monitor the UAVs during the flight tests in order to manually recover the aircraft if that were to become necessary. Unfortunately, this solution removes the meaningfulness of a traditional global reference, like North. A circle neither has a beginning nor an end, so the concept of being "in front" while flying in a circular orbit is undefined. Therefore, the concept of a "spooled" global goal was implemented to reconcile the swarm logic already developed with the use of a circular loiter for the global leader's flight path during flock convergence. To visualize this concept, consider a spool of thread. The thread represents the path to be traveled, with one end representing a reference location and the opposite end serving as the goal. As the thread is wound around the spool, the distance to the goal (along the thread) remains the same, but the center of the spool moves closer to the reference location for each new winding. This spooled global goal can be described mathematically by

$$\Delta_{global\ goal} = \begin{cases} \Delta_0 + \Delta X_{goal\ wp}, & \Delta X_{goal\ wp} > r_{goal\ loiter} \\ \Delta_0 + r_{goal\ loiter}(1 - \int \Delta\Psi dt), & \Delta X_{goal\ wp} \leq r_{goal\ loiter} \end{cases} \tag{3.17.1}$$

Where r is a radial distance and ΔX is the magnitude of a relative distance vector. Integrating the change in the heading angle, Ψ, during a loiter allows for the approximation of arc length when multiplied by the loiter radius. The initial spooled reference value, Δ_0, is included so that the integrated arc-length can be subtracted instead of accumulated. The global reference value, $\Delta_{global\ goal}$, is ultimately what determines the order of leadership, with the member broadcasting the lowest reference value becoming the global leader.

3.18 Potential Function Guidance Algorithm

The potential function algorithm presented in this project was based primarily on the functions used by Tsuruta in [86] and Wallace in [88] in order to maintain continuity between similar studies as well as limit the scope of the project.

3.18.4 Attractive Potential Functions

A slightly different approach towards the attractive potential function was taken in this study compared to the work done by Tsuruta. First, the calculation of the local goal location is more involved when the lead UAV is flying in a circular loiter pattern instead of in a straight direction. Relative position information is calculated in the NED frame, but the formation offset must be applied with respect to the local leader. Therefore, a transformation of the relative position vector from NED to the local leader's LNAV frame is required to incorporate the offset. Once the local goal location is adjusted for both offset distance and commanded side, the updated relative position vector must be transformed back into the NED frame to be used in the PFG algorithm. Second, instead of a single quadratic potential function serving as the goal function for the entire domain of

relative distances between a leader-follower pair, two different forms of potential functions were included.

In the far-field regime, a linear attractive potential function serves well because the gradient magnitude is constant. This provides saturation of the attractive potential function and ensures that there are no large gradient magnitudes that will eclipse the repulsive potential functions' gradient values, something that is necessary to provide collision avoidance capability even when the goal is far away. The linear attractive potential function is described by

$$\Phi_{att}(\bar{x}, \bar{x}_{goal}) = \left|\underline{\lambda}(\bar{x} - \bar{x}_{goal})\right|. \tag{3.18.2}$$

where $\underline{\lambda}$ is a diagonal matrix of weighting parameters which will be discussed later.

In the near-field regime, a quadratic attractive potential function is used. A quadratic potential function has a gradient magnitude that approaches zero smoothly, which is a desirable quality when considering the output of the function as controller input command. This type of potential function is described by

$$\Phi_{att}(\bar{x}, \bar{x}_{goal}) = (\bar{x} - \bar{x}_{goal})^T \underline{\lambda}(\bar{x} - \bar{x}_{goal}) \tag{3.18.3}$$

In order to match the two functions at the regime threshold, the linear potential function must be adjusted by multiplying it by a constant equal to the quadratic function gradient magnitude evaluated at the regime threshold, χ, and then offset by the quadratic function magnitude at χ. Therefore the attractive potential function in the far-field regime can be described by

$$\Phi_{att}(\bar{x}, \bar{x}_{goal}) = 2\chi \left|\underline{\lambda}^T \underline{\lambda}(\bar{x} - \bar{x}_{goal})\right|. - 2\underline{\lambda}\chi^2 \tag{3.18.4}$$

Figure 3.9 shows an example of a two-dimensional attractive potential field that combines both the linear and the quadratic functions, transitioning from one to the other at χ.

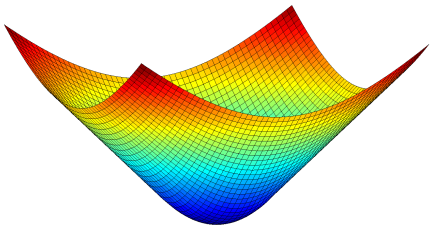


Figure 3.9: A 2D representation of an attracitve potential field which combines a linear and quadratic potential function

3.18.5 Repulsive Potential Functions

The Gaussian repulsive potential functions used in this study were also modeled after those used by Tsurutu in [86] and can be expressed by

$$\tau_j = \tau_{0,j} \exp^{[\frac{-1}{\sigma_j}(\bar{x}-\bar{x}_j)^T \mathbf{I}(\bar{x}-\bar{x}_j)]} \tag{3.18.5}$$

where $\tau_{0,j}$ is a repulsive strength parameter and σ is a sizing parameter, both of which will be described in section 3.20.

The total potential field in either regime can be described by the superposition of the attractive and repulsive potential fields, such that

$$\Phi_{total} = \Phi_{att} + \sum_{j=1}^{N} \tau_j \tag{3.18.6}$$

3.18.6 PFG commands

The potential functions themselves are actually quite useless from a command generation perspective. They provide a bridge for real-world analogies and can generate a visual representation of the total potential field, but the do not serve a purpose when it comes to issuing meaningful commands to a control system. To generate such commands, the gradient equations are used instead of the scalar

function equations. The gradient vector,

$$\nabla\Phi_{total} = \langle \Phi_x, \Phi_y, \Phi_z \rangle \tag{3.18.7}$$

is generated directly using the set of equations

$$\left\{ \begin{array}{l} \Phi_x = 2\chi \frac{\lambda_x (x - x_{goal})}{|\underline{\lambda}(\bar{x} - \bar{x}_{goal})|} + \sum_{j=1}^{N} \frac{-2\tau_{0,j}(x - x_j)}{\sigma_j} exp^{[\frac{-1}{\sigma_j}(\bar{x} - \bar{x}_j)^T \mathbf{I} (\bar{x} - \bar{x}_j)]} \\ \\ \Phi_y = 2\chi \frac{\lambda_y (y - y_{goal})}{|\underline{\lambda}(\bar{x} - \bar{x}_{goal})|} + \sum_{j=1}^{N} \frac{-2\tau_{0,j}(y - y_j)}{\sigma_j} exp^{[\frac{-1}{\sigma_j}(\bar{x} - \bar{x}_j)^T \mathbf{I} (\bar{x} - \bar{x}_j)]} \\ \\ \Phi_z = 2\chi \frac{\lambda_z (z - z_{goal})}{|\underline{\lambda}(\bar{x} - \bar{x}_{goal})|} + \sum_{j=1}^{N} \frac{-2\tau_{0,j}(z - z_j)}{\sigma_j} exp^{[\frac{-1}{\sigma_j}(\bar{x} - \bar{x}_j)^T \mathbf{I} (\bar{x} - \bar{x}_j)]} \end{array} \right\} \tag{3.18.8}$$

$$\left\{ \begin{array}{l} \Phi_x = 2\lambda_x (x - x_{goal}) + \sum_{j=1}^{N} \frac{-2\tau_{0,j}(x - x_j)}{\sigma_j} exp^{[\frac{-1}{\sigma_j}(\bar{x} - \bar{x}_j)^T \mathbf{I} (\bar{x} - \bar{x}_j)]} \\ \\ \Phi_y = 2\lambda_y (y - y_{goal}) + \sum_{j=1}^{N} \frac{-2\tau_{0,j}(y - y_j)}{\sigma_j} exp^{[\frac{-1}{\sigma_j}(\bar{x} - \bar{x}_j)^T \mathbf{I} (\bar{x} - \bar{x}_j)]} \\ \\ \Phi_z = 2\lambda_z (z - z_{goal}) + \sum_{j=1}^{N} \frac{-2\tau_{0,j}(z - z_j)}{\sigma_j} exp^{[\frac{-1}{\sigma_j}(\bar{x} - \bar{x}_j)^T \mathbf{I} (\bar{x} - \bar{x}_j)]} \end{array} \right\} \tag{3.18.9}$$

Where equation 3.18.8 is used while the formation member is considered in the far-field regime and equation 3.18.9 is used once the formation member enters the near-field regime.

An additional correction to the total potential function is applied to the near-field attractive potential function gradient vector in order to reconcile the virtual waypoint implementation, detailed in section 3.19, with abrupt sign changes in the attractive potential function gradient values as the UAV oscillates around the goal location. Inspired by the simple rules presented in reference [75], heading matching between the leader and follower pair is fused with goal location tracking and the attractive potential function gradient magnitude to ensure a directional command that encourages convergence towards the goal, is sympathetic to the

direction of travel of the leader, and still allows for the repulsive potential function gradient to overtake the attractive, steering the follower away from a collision with other formation members. As depicted in figure 3.10, this is accomplished, first,

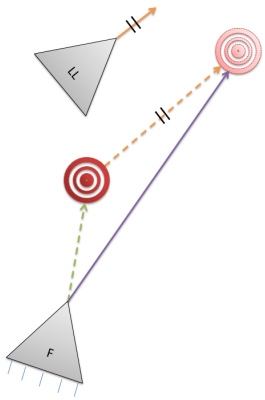


Figure 3.10: The process used for the near-field correction of the attractive potential function gradient values

by defining an extrapolated goal location, offset by a constant value in a direction parallel to the heading of the leader. Adding the vectors pointing to the original goal and this extrapolated goal produces a vector which points in the desired heading command defined by

$$\Psi_c = \tan^{-1}(\frac{\bar{x}_{c_N}}{\bar{x}_{c_E}}) \tag{3.18.10}$$

To ensure that the repulsive potential function is not eclipsed by this correction, the original magnitude of the attractive potential function gradient is used in conjunction with the extrapolated heading command direction to calculate a new corrected attractive potential function gradient vector

$$\nabla\Phi_{att_c} = \langle|\nabla\Phi_{att}|sin(\Psi_c), |\nabla\Phi_{att}|cos(\Psi_c), 0\rangle \tag{3.18.11}$$

Because this corrected attractive potential function gradient vector is used only for the generation of the North and East offsets of the next virtual waypoint, and not the altitude command, the "Down" component of this vector is simply set to

zero. Both the corrected and uncorrected attractive potential function gradient vectors are retained and added independently to the summation of repulsive potential function gradient vectors, producing two distinct total potential function gradient vectors

$$\nabla\Phi = \nabla\Phi_{att} + \sum \nabla\tau_j$$
$$\nabla\Phi_c = \nabla\Phi_{att_c} + \sum \nabla\tau_j \tag{3.18.12}$$

In order to transform these into useful command vectors, the total potential function gradient is normalized, producing a unit vector pointing towards the potential field minima,

$$\hat{U} = \frac{\nabla\Phi}{|\nabla\Phi|}$$
$$\hat{U}_c = \frac{\nabla\Phi_c}{|\nabla\Phi_c|} \tag{3.18.13}$$

guiding the UAV into formation, while preventing collision with other members.

3.18.7 PFG Algorithm Parameters

In order to fine-tune the potential function guidance algorithm, several parameters are included in the various equations:

$\bar{\lambda}$ is a diagonal matrix,

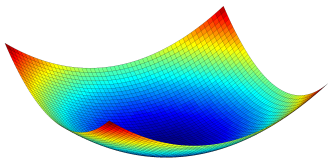
$$\begin{bmatrix} \lambda_x & 0 & 0 \\ 0 & \lambda_y & 0 \\ 0 & 0 & \lambda_z \end{bmatrix}$$

comprised of weighting parameters for the attractive potential function. This weighting is used to control the relative strength of the attractive potential function in each axis of the NED coordinate frame. The best example of when weight-

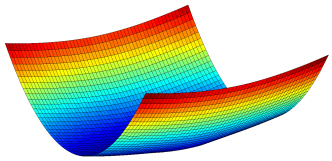
ing could prove beneficial is found when taking into account a fixed-wing aircraft's ability to maneuver. Tracking the velocity and altitude of a leader is much more difficult to accomplish than tracking the heading of a leader, due to a constant battle between the controller and the low-frequency modes of the UAV dynamics. Therefore, it may make sense to weight the altitude attractive potential function relatively higher, to compensate for the limitations in maneuverability. It is worth noting that reference [86] states that $\bar{\lambda}$ must be positive definite for the attractive algorithm to function properly, meaning that each weighting parameter, λ should be positive. This makes sense, considering the physical interpretation of this parameter.

For the repulsive function, there are two explicit parameters and one derived parameter that must be determined. σ is the repulsive function shaping parameter, directly relating the region for which the repulsive function influences the total artificial potential field. A larger σ will cause the repulsive function to be "seen" from a larger relative distance between two formation members, which would increase the amount of time the UAVs have to avoid a collision, but could negatively affect the UAV's ability to settle in the desired formation structure. τ_0 is the repulsive potential strength constant, determining the maximum value of the repulsive potential, when $\bar{x} - \bar{x}_j = 0$. This value plays an important role in the superposition of the attractive and potential functions, but is mostly tuned based on the derived parameter, $\frac{\tau_0}{\sigma}$. While τ_0 is the parameter which sizes the repulsive potential function, $\frac{\tau_0}{\sigma}$ is what ultimately sizes the gradient of the repulsive potential function. Because, it is the gradient of the total potential field that translates into the controller command, this ratio becomes crucial for the ability of the UAV to avoid other formation members when the attractive potential function component of the gradient is large (when the UAV is far from

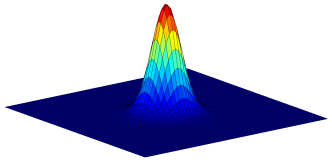
the goal). While attractive potential gradient saturation is applied in the far-field case, which was discussed in more detail earlier in this section, the derived parameter $\frac{\tau_0}{\sigma}$ must be sized so that the repulsive component of the potential function gradient exceeds the maximum attractive potential function gradient value at a reasonable relative distance between two formation members. To better visualize



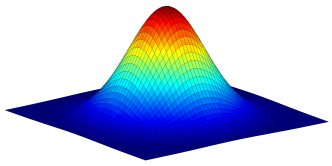
(a) Attractive potential field with $\lambda_x = \lambda_y$



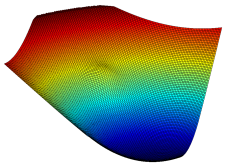
(b) Attractive potential field with $\lambda_x \neq \lambda_y$



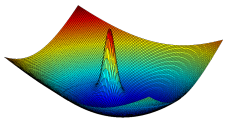
(c) Repulsive potential field with small σ



(d) Repulsive potential field with large σ



(e) Combined attractive and repulsive potential fields with a small τ_0 for the repulsive field



(f) Combined attractive and repulsive potential fields with a large τ_0 for the repulsive field

Figure 3.11: Examples of the effect of various potential function parameters

these trade-offs, Figure 3.11 shows two dimensional representations of several parameter configurations and their effect on the individual potential function, the potential field, and the gradient of that field.

3.19 Virtual Waypoint Placement and Airspeed Commands

As discussed in the previous section, taking the normalized gradient of the total potential field provides a useful unit vector, pointing in the direction of the potential field minima. With so many inputs and outputs of the presented control system architecture based in the NED coordinate frame, it was convenient to evaluate the potential function gradients in the NED coordinate frame as well. Generally, the virtual waypoint was placed by multiplying a pre-determined constants, Δ_{VWP} and Δ_{VWP_Z}, by the unit vector components to come up with the desired offsets,

$$\left\{ \begin{array}{c} \Delta North = \Delta_{vwp}\hat{U}_N \\ \Delta East = \Delta_{vwp}\hat{U}_E \\ \Delta Altitude = -\Delta_{vwp_z}\hat{U}_D \end{array} \right\} \tag{3.19.14}$$

determining the coordinates of the virtual waypoint by extrapolating the reported position by these offsets.

For the application presented in reference [58], which inspired the selection of this virtual waypoint strategy, there was no need for distinction between far-field and near-field regimes; the goal was always treated as if it were far away from the UAV. However, when applied to formation flight, the near-field regime requires some modification of the strategy in order to prevent erratic behavior. Tracking both airspeed and altitude while combating wind disturbances and sensor error tend to induce low frequency oscillations about the goal, which are difficult to damp out. Often times, the unit vector meant to place the next waypoint will

point in a direction that is behind the UAV, presenting a logic problem for virtual waypoint strategy, directing the UAV to "turn around" which is accomplished by the commanding the maximum bank angle allowed. The necessary correction to the potential function guidance algorithm was discussed in the previous section, and translates into the virtual waypoint algorithm such that

$$\left\{ \begin{array}{c} \Delta North = \Delta_{vwp} \hat{U}_{c_N} \\ \Delta East = \Delta_{vwp} \hat{U}_{c_E} \end{array} \right\} \tag{3.19.15}$$

Additionally, the altitude command is no longer generated using the command unit vector, but instead by the direct altitude error between the leader and follower.

In the far field regime, the airspeed command is simply set to the maximum, steady-level airspeed which is limited by the maximum allowed throttle. However, in the near field regime, airspeed commands are dictated by a velocity matching controller and the original potential field gradient, before it is corrected for the near-field virtual waypoint algorithm. The velocity matching is a product of the swarm logic which is necessary for formation flight, as discussed in [61]. By driving the relative airspeed error between the leader-follower pair of UAVs to zero, the formation will move at a uniform airspeed, dictated by the global leader. Because the virtual waypoint algorithm is limited to control in the y-z plane of the LNAV frame, the second component of the airspeed command is generated by the x component of the original potential function gradient in the LNAV frame. These two command reference values are combined into a unified airspeed command using what could be interpreted as a proportional-integral controller, such that

$$\Delta V_{aspd_{com}} = K_{\Delta V} \Delta V + K_{\Phi} \Phi_x \tag{3.19.16}$$

Care must be taken to make sure the maximum overshoot of the airspeed con-

troller is small enough to avoid coming too close to the local leader, prompting the repulsive potential function to overtake the attractive potential function and steer the UAV away from an eminent collision.

3.20 Algorithm Parameter Selection

3.20.8 Near-Field and Far-Field Regime Threshold

The point at which the control scheme switches from the far field to the near field regime is an important parameter because it heavily influences both the performance of the UAVs as they aim to reach the formation from relatively large distances as well as the convergence of the UAVs as they track the commanded relative distance from their local leader and maintain formation. To determine this factor in three dimensional space, it is best to discuss the relative energy height of the leader-follower pair

$$\Delta Z = \Delta h g_c + \frac{1}{2}\Delta V^2 \tag{3.20.17}$$

as well as the distance required to dissipate the relative energy height. At the assumed formation conditions, the time and therefore the distance required to bring the aircraft into formation can be approximated by

$$\Delta t_{dis} \approx \frac{\Delta Z}{\frac{d}{dt}(\Delta Z)_{max}} \tag{3.20.18}$$

$$\chi \approx (\frac{V_{aspd,max} + V_{aspd,form}}{2})\Delta t_{dis} \tag{3.20.19}$$

where $\frac{d}{dt}(\Delta Z)_{max}$ is directly related to the maximum amount of drag the UAV can generate at the formation flight condition.

3.20.9 Potential Field Parameters

In reference [86], Tsuruta does not present a methodology for assigning the various parameters included in the potential functions he outlines, but does speak to some of the trades between the various parameters. In reference [88], Wallace develops several methods to solve for the parameters based on approximate performance limitations. Continuing along the lines of what Wallace suggested, equations were derived to translate meaningful metrics into the potential function parameters discussed in section 3.18

Three new metrics provide a proxy for the potential function guidance algorithm parameters, allowing for the selection of more intuitive values. These metrics include: The threshold radius, X_{tr}, the repulsive peak radius, $X_{\nabla_{rep_{max}}}$, and the repulsive magnitude maximum, $\nabla_{rep_{max}}$.

The metric X_{tr} serves to tie in the UAV's minimum turning radius, taking in to account the UAV's ability to avoid collision with a planar maneuver. $X_{\nabla_{rep_{max}}}$ is term that is necessary due to the properties of a Gaussian repulsive potential function. With a Gaussian function, the gradient of that function has a maximum that is not at the center of the function, meaning that the maximum repulsive command will be issued at some distance away from the center of the function, with that command weakening as the gradient is evaluated closer to the center of the function. An aircraft is not a point-mass, and therefore, the strongest repulsive command should not be at the center of gravity of the aircraft, but rather around the perimeter of the aircraft. Therefore, $X_{\nabla_{rep_{max}}}$ is set as the representative radius around the aircraft, promising the highest repulsive command near the point of physical collision. $\nabla_{rep_{max}}$ is a somewhat arbitrary number that simply scales the range of values used in the algorithm process. For

an embedded system, where this algorithm is intended to run, the use of small data types and integer scaling could factor into the selection of a lower number for this parameter.

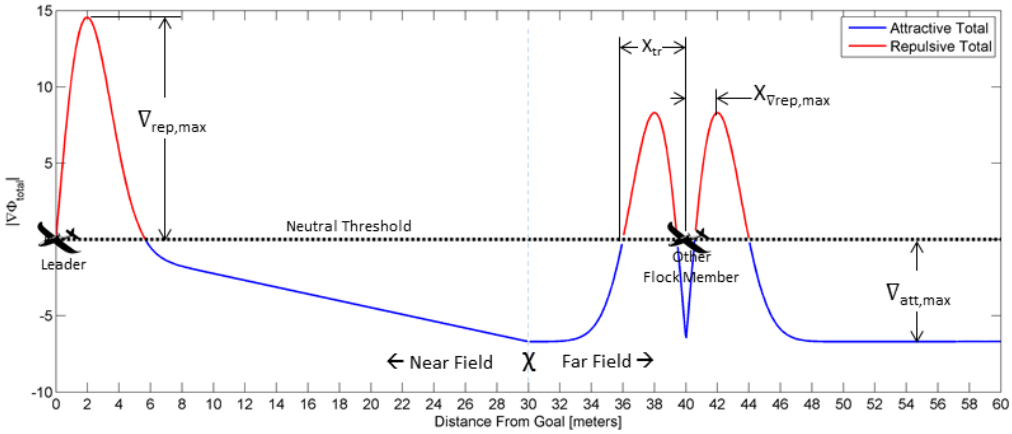


Figure 3.12: The physical interpretation of the new potential function guidance algorithm sizing metrics

Figure 3.12 shows how these parameters relate to total gradient magnitude with respect to the UAV's distance behind the leader. The figure gives an example of the extreme case of another formation member in the far-field regime, where the attractive potential function gradient is at its maximum.

The first potential function guidance algorithm parameter calculated is for the repulsive function's "influence" parameter, σ, where

$$\sigma = 2X^2_{\nabla_{repmax}} \tag{3.20.20}$$

The next parameter to be calculated is the total repulsive function magnitude, τ_0, such that

$$\tau_0 = \nabla_{rep_{max}} X_{\nabla_{rep_{max}}} e^{\frac{1}{2}} \tag{3.20.21}$$

With both of the parameters calculated for the repulsive potential function equation, the next step is to determine the values for λ_x, λ_y, and λ_z. This process

begins by interpreting the physical limitations of the UAV flying at the conditions expected during formation flight. Substituting an equation for a steady turn radius, a conservative approximation for the minimum turn radius required as a function of relative velocity and maximum roll angle, Φ_{max}. This relationship is described to be

$$X_{tr} = \frac{\Delta V^2}{g_c tan\Phi_{max}} \tag{3.20.22}$$

In order to ensure that the collision avoidance algorithm functions in far-field regime, where the attractive potential function gradient magnitude reaches its maximum, a method was derived to set that maximum gradient magnitude directly. By setting $\nabla_{att_{max}}$, the attractive potential function gradient is guaranteed not to eclipse the repulsive potential function gradient inside of the threshold radius, X_{tr}. This maximum is set by

$$\nabla_{att_{max}} = \frac{2\tau_0 X_{tr}}{\sigma} e^{\frac{-X_{tr}^2}{\sigma}} \tag{3.20.23}$$

With a set maximum desired value for the attractive potential function gradient, the corresponding sizing parameter for the attractive potential function, λ can be calculated by

$$max(\lambda_x, \lambda_y, \lambda_z) = \frac{\nabla_{att_{max}}}{2\chi} \tag{3.20.24}$$

It is worth noting that this equation only constrains the maximum lambda. Potential function weighting can still be accomplished using different values of λ, so long as they do not exceed the calculated maximum.

4 Achieving Relative Situational Awareness

The difficulties of achieving situational awareness with UAVs was introduced in section 2.8. While achieving absolute situational awareness is necessary for many applications, and is necessary for any realistic implementation of formation flight, the form of situational awareness that successful formation flight is most dependent on is relative situational awareness. Achieving this relative situational awareness is not a trivial problem, and many studies are focused solely on methods which could be used for this purpose.

Considering purely decentralized approaches, the most promising technique involves the use of computer vision. Natural inspiration shows that vision is often a key sense employed by animals of all shapes, sizes, and complexity, used to interpret their surroundings, and establish some sense of relative situational awareness. Although making significant progress in recent years, vision systems are still too complex and have not reached a level of maturity that could provide a cheap, robust way for a small UAV to achieve relative situational awareness. Nature also demonstrates that several other techniques can be employed to interpret the environment an animal is operating in. Active localized sensing like echolocation provides inspiration for using sonar, radar, or laser range sensors to develop a map of the environment and its inhabitants in order to achieve some sense of situational awareness. While many sonar sensors are available in a cheap form factor, complicated algorithms are required to map surroundings, and limitations of range and accuracy make them difficult to employ over a wide range of possible distances. Radar and laser technologies have developed significantly, but are still too costly and often too heavy to integrate in to a small UAV. Additionally, even radar and laser mapping require complicated algorithms to process

detected terrain and obstacles.

Moving away from purely decentralized approaches, there are also a few options for distributed or centralized methods that could provide some form of relative situational awareness. Independent observation is the common denominator for most of these methods, which relies on some form of communication of the observations to the UAVs. Favored by many academic institutions for its accuracy and robust performance, motion capture systems have been employed for many multiple UAV applications to provide relative state information to the vehicles, such as the systems described in section 2.11. These systems, however, are limited to use indoors. When considering applications which must be observed outdoors, the most likely candidate to provide situational awareness is GPS.

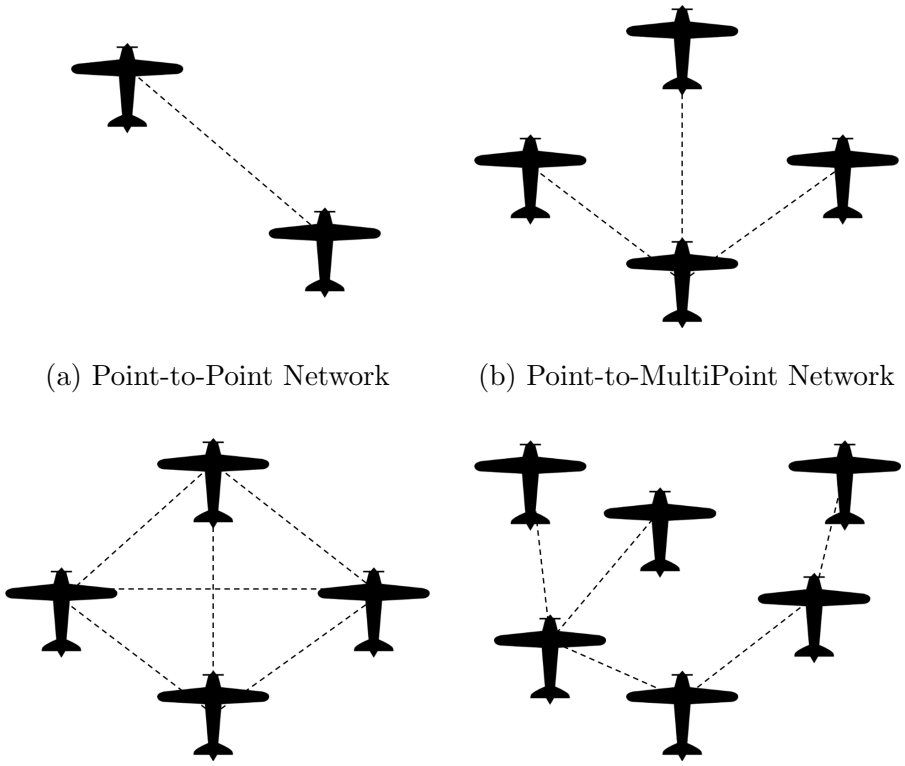
Ultimately, this project proceeded in a direction that acknowledged that decentralized formation member observation was not within the scope of this project. Therefore, an independent observer strategy was desired to achieve relative situational awareness. Accordingly, to relay the independently observed observations among all of the UAVs, a strategy for communicating each formation member's observed state was developed.

4.21 Communication of Observed States

4.21.10 Network Topologies

While network topologies are often protocol dependent, understanding and selecting the right topology to share formation member states was an important factor in deciding the final communication strategy. In this discussion, network topology refers to the way in which one formation member communicates to

another, ignoring the distinction between different protocol layers that enable that form of communication. With that in mind, the available wireless network topologies include the following: Point-to-Point, Point-to-Multipoint, Peer-to-Peer, and Ad-Hoc Mesh.



(a) Point-to-Point Network (b) Point-to-MultiPoint Network

(c) Peer-to-Peer Network (d) Ad-hoc Network

Figure 4.1: Examples of various communication network configuration options for a state-sharing strategy to achieve relative situational awareness

Illustrated in Figure 4.1a, the simplest form which can be implemented is a point-to-point topology configured in either full duplex or half duplex, depending on whether or not each node waits for the other to stop "talking" before it begins. Many telemetry links for monitoring UAVs use this topology because it is simple to set up and many radios come pre-configured for this use. Figure 4.1b shows the point-to-multipoint topology, sometimes referred to as a star topology. Many

half duplex radios that support point-to-point also support point-to-multipoint configuration. This topology allows multiple nodes to communicate with each other, but all communication is routed through a central coordinator. Figure 4.1c shows the Peer-to-Peer topology, where each node is able to communicate with all the nodes directly, without a central coordinator. However, by removing the coordinator, complexity is added, and the number of network connections follows the n^2 approximate proportionality model. Figure 4.1d depicts the Ad-Hoc Mesh topology which uses the peer-to-peer concept, but enables each node to route communication intended for another node as well as the ability for nodes to join, leave, and rejoin the network without causing destabilization[27].

4.21.11 Requirements for Formation Flight Demonstration

From the standpoint of network topology, the communication requirement of formation flight utilizing the state-sharing method discussed is relatively simple: Each UAV must be able to communicate with every other UAV in the formation. For many practical applications, including demonstration, it is also necessary to monitor each UAV and send commands. Therefore, an additional requirement can be added: Each UAV must be able to communicate with the ground station. With the decentralized control strategy developed, a decentralized communication network should be put in place to mirror the control strategy and remove communication dependencies from a central coordinating node. This is accomplished by adding the requirement: The network must not be dependent on a central router for communication. Lastly, one of the major benefits attributed to decentralized control of multiple UAVs is the inherent tolerance to the loss of a formation member, even the global leader. Therefore, a network that is able to accommodate the same fluctuation of formation members has the requirement:

Any potential formation member may join or leave the communication network freely. While not required for this project, considering larger scale implementations leads to two more desirable network capabilities: 1) Any UAV in range of at least one other UAV should be able to communicate with all of the UAVs in formation, and 2) Network communication links should be developed intelligently to avoid saturating the network when a large number of nodes are present.

4.22 Detailed State-Sharing Strategy

4.22.12 States based on GPS

For this project, in order to limit scope and reduce the number of variables introduced into demonstration, a GPS-based approach to understanding relative states was adopted. Accordingly, each UAV will simply broadcast their GPS-reported state to the other UAVs participating in the formation, providing all of the UAVs with the information required to determine their state relative to each of the other formation members. Specifically, each UAV will broadcast

- UTC Time
- Longitude and Latitude
- Absolute Altitude and Relative Altitude, determined with the on-board altitude sensor
- Velocities V_X,V_Y, and V_Z, in the NED coordinate frame
- Heading

While GPS filtering could be implemented to offset some of the error associated with this method, it was not included in the scope of this work.

4.22.13 Ad-Hoc Mesh Network

To meet all of the requirements listed in the previous section, truly embrace the decentralized approach, and accommodate the desirable capabilities for future scalability, the best choice of network topology is an ad-hoc mesh. Ad-hoc networks allow for

- All formation members to communicate freely with no central ground station required for routing
- Potential formation members to join or leave the network at any time
- Communication through "hops" removing the requirement of direct connections between UAVs
  - At a larger scale, UAVs could still communicate with all formation members even if some are out of its range
  - With many more UAVs, intelligent routing through hops could help to keep a large communication network manageable

Despite the ability of Ad-Hoc networks to meet all of the requirements *and* desirable capabilities, early hardware tests demonstrated that the complexity, required configuration, and number of variables to adjust made this network topology inconvenient for demonstration at this scale. Therefore, the desirable capabilities for scalability were abandoned in favor of a less complicated topology that still met the requirements.

4.22.14 Peer-to-Peer Network

With the peer-to-peer network topology, all UAVs are able to communicate directly without being routed through a central node, they are still able to communicate with the ground station, and they may still come and go from the network as they please. Hardware testing demonstrated that setup of this network

was substantially easier, with fewer variables to adjust while trouble-shooting errors. With only three UAVs being used in this demonstration, it is expected that all UAVs will be in range of each other and the ground station, and the network will not be overloaded with too many connections.

4.23 Limitations of Strategy

While the proposed state-communication does satisfy the requirements for accomplishing formation flight, there are several limitations associated with the strategy. One of the most problematic limitations lies in that all of the relative state information that is calculated is derived from the GPS data in one way or another. As discussed earlier, GPS error can be significant, especially with respect to the scale of the test-bed aircraft. This reliance on GPS for state communication makes the relative states calculated only as accurate as the combined GPS accuracy of the formation members. Another drawback to the state-communication strategy proposed is the reliance on a network to achieve that communication. Any network failures will cause members to disappear from simulated "view", disconnecting the UAV's perceived reality from the physical world. The last major limitation of this state-sharing strategy is that it is not possible to form a completely decentralized control system architecture, free of dependencies on other UAVs to determine its own desired state. This form of control strategy reaches its full utility when there are no observable transmissions and no dependencies on any other external systems. Only a full vision system could make that dream a reality. While the state-sharing strategy does not allow the full control system to be decentralized, the way in which the strategy has been implemented serves as a place-holder, should an appropriate vision system be developed in the future.

5 Hardware

5.24 Selection Methodology

Embedded system design is not a trivial task, with entire studies conducted to evaluate emerging trends in the field and suggest improvements to the process. Conventional wisdom suggests that hardware selection be conducted only after a full, detailed prototype of the software has been completed, insuring that the selected hardware will meet the requirements of the software.

In the relatively new era of cheap, ubiquitous prototyping boards such as the Arduino Mega, Beagle Board, or Rasberry Pi, to name only a few, there is an alternative to the traditional design process for embedded systems, ideal for fast-paced research applications where the embedded system is just a means to an end, and not the focus of the work itself.

With a basic understanding of the software requirements, gleaned from a high-level prototype, such as a Matlab or Simulink implementation, a platform can be selected which roughly fits the requirements of a system. The drawback to this approach is that without an in-depth understanding of the software, as it must be realized for an embedded system, there is always a chance of running up against a limitation of the platform chosen, and end up having the hardware push back on the software requirements. For research where the objective is not to use a specific form of an algorithm or technique, this hardware push-back can usually be accommodated.

Best described as a "hacker" methodology, this process can greatly accelerate research where an embedded system is required to accomplish the study's objective, but does not constitute the main focus of the project.

5.25 Requirements

The requirements derived for the FLOC project hardware selection were heavily influenced by the objectives of the project, laid out in Section 1.3. One of the largest driving requirements for hardware was the selection of low cost components that would be easy to obtain, repair and replace. This requirement spans the entire system from the RC aircraft test-bed to the communication hardware to the autopilot. Another driving requirement is the flexibility and customization of the hardware. The hope is that this hardware will support, not only the FLOC project, but future multiple UAV control work as well. Selecting flexible hardware that can be altered for various uses will help maintain continuity between studies and minimize the learning curve associated with applying this hardware in multiple UAV control studies.

Figure 5.2 shows the requirements flow-down for hardware selection. Many of these, such as the RC aircraft requirements, are somewhat arbitrary, considering the number of solutions that satisfy the requirements. Some requirements to highlight are the stable platform and efficient, long endurance requirement. Stability is important to narrow the scope of this problem: It is unnecessary to have to consider stabilizing the vehicle in addition to accomplishing formation flight. Long endurance is a requirement due to the nature of multiple UAV flight testing, and the need to allow time to launch and recover the UAVs safely. "Long" is intended to be relative to traditional RC flight times, referring to flight times on the order of 15-30 minutes as opposed to 5-10 minutes, which is common among high-performance RC aircraft. The communication requirements are derived from the state-sharing method of achieving relative situational awareness, discussed in Section 4.22, as well as from telemetry logging and parameter

alteration support required for flight testing. Additionally, range requirements reflect the test range used and the regulations requiring RC aircraft to fly below 400 feet. Figure 5.1 shows the approximate dimensions of the test field area to be utilized. Combined with the 400 foot altitude limitation, a worse-case range requirement can be found to be $\approx$ 1900 feet, or about 0.36 miles. Another impor-

Figure 5.1: The approximate dimensions of the test area expected to be utilized for flight testing

tant requirement is the communication hardware's ability to accommodate the necessary data rates. With no video feedback requirements, the data throughput required is relatively light. However, for high fidelity logs, telemetry data must be sent at a high rate.

The autopilot hardware requirements are heavily driven by the stated goal to use low-cost commercially available autopilots that support open-source software. The type of autopilots that can meet those goals must be customizable, well documented, readily available, and cheap. The selected autopilot must also be capable of supporting the software required to enable formation flight.

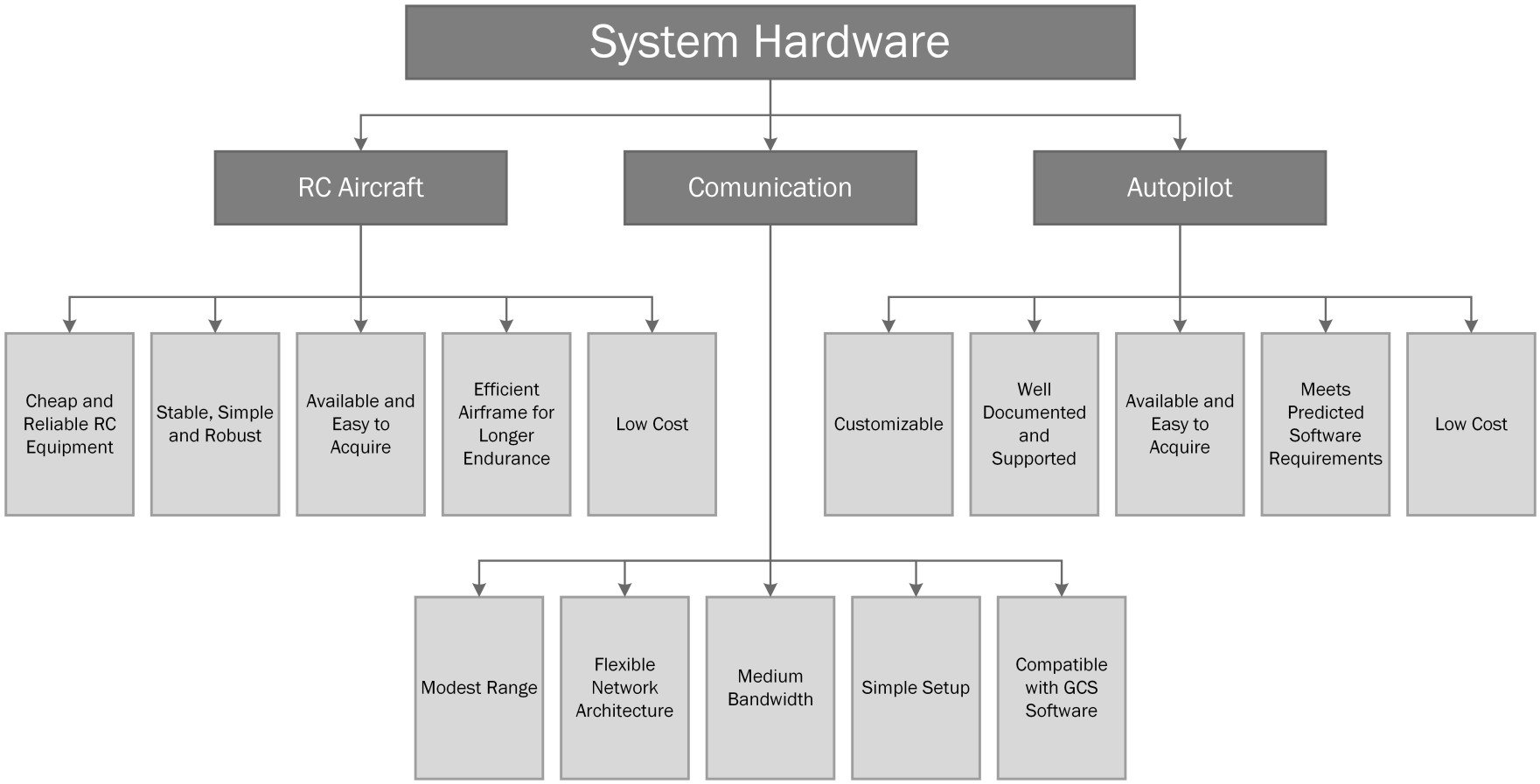


Figure 5.2: Flow-down for system hardware requirements

5.26 Communication Options

As discussed earlier, the development and miniaturization of wireless technologies have made wireless communication ubiquitous in the span of two decades. With so many wireless devices, requirements can range from short distance, low data rate for applications such as a smart home thermostat, to long range, high data rate for applications such as real-time surveillance video from a UAV. While investigating appropriate strategies for this application, it was found that most wireless devices are classified by the standard or protocol for which they are designed. Without considering options intended for more extreme applications, the options for communicating each UAV's observed state to other members of the formation include the following standards:

- GSM/UMTS/LTE (Cellular 2G/3G/4G)
- IEEE 802.11 (WiFi)
- Bluetooth
- IEEE 802.15.4 (ZigBee, MiWi)

Figure 5.3 shows how devices which are designed for each standard generally compare with respect to range and data rates. Cellular device technologies have all but replaced wired telecommunications, encouraged by feasible infrastructure requirements due to its long range. With each generation, data transfer rates have increased and have now reached a level rivaling other technologies designed for internet-scale data rates [15]. Despite the rise of 3G and 4G mobile devices, WiFi has long been the face of wireless internet access due to its high data throughput. It has moved beyond businesses and residences through the spread of "hot spots", enabled by cheap wireless routers, which compensate for the limited range [45]. Generally, Bluetooth devices are designed for applications such as

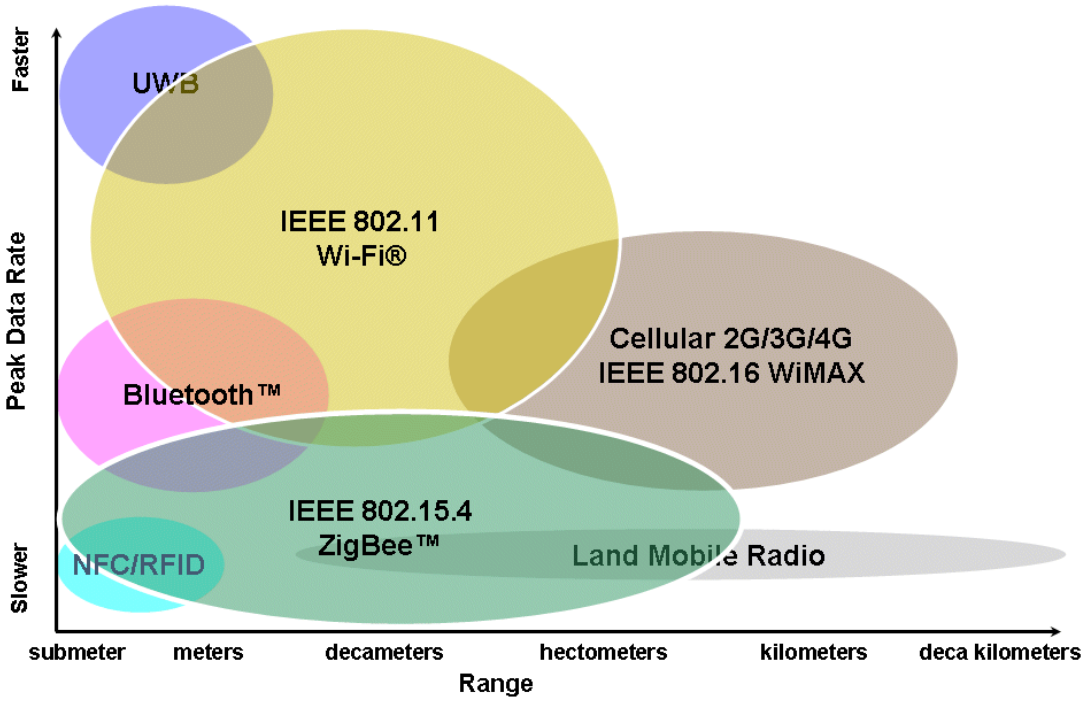


Figure 5.3: Typical range and data rates for various wireless technologies

personal computer interface accessories, cellular phone hands-free accessories, and many other applications where the required range is relatively small [85]. Less well known are the devices that use IEEE 802.15.4 based protocol such as ZigBee devices. Often found in low power applications, these devices have found a niche in sensor networks of all kinds. Also, because of the relatively low cost and simplicity of the devices, hobby robotics applications often turn to this solution for wireless communication. While not capable of internet-scale data throughput, these devices are ideal for moderately distanced applications exchanging sensor data on a power budget [85].

5.27 Selected Hardware

5.27.15 Airframe Selection

The airframe serving as the test-bed for this project is the Sky Surfer 1400, produced by X-UAV and distributed in the United States by ReadyMadeRC as an ARF kit that includes integrated servos, an ESC, and a brushless motor. Derived from the classic 3 channel Multiplex EasyStar, the Sky Surfer is one

Figure 5.4: The Sky Surfer 1400

of several 4 channel clones that comes highly recommended for basic UAV and FPV platforms by the DIY Drones community and other RC enthusiast message boards. The specifications for the Sky Surfer 1400 airframe are shown in table 5.1. This airframe meets all of the requirements listed in the previous section, making it a good candidate for use not only in this work, but in future multiple UAV studies.

Intended to be flown by beginner pilots, this airframe is a very stable platform. Constructed out of EPO foam, the aircraft is durable enough to take a hard

Table 5.1: Sky Surfer Specifications

Material	EPO
Wing Span	54 in
Wing Area	410.5 in^2
Empty Weight	22 oz

landing, and in the event of a crash, can be repaired easily. Hand-launched and belly-landed, the aircraft provides flexibility in test facilities, as it does not require a runway. At approximately $50 for the airframe alone, the Sky Surfer is cheap enough to replace in case of a total loss. The Sky Surfer's configuration also resembles that of a motor-glider, making it an efficient airplane, capable of flying at low speeds with relatively long endurance. Payload capacity and internal volume is suitable for the electronics required for this project, with enough space and capacity left over to accommodate future studies that require additional, similarly sized hardware. Lastly, the popularity of this airframe as well as the many other EasyStar clones out there, such as the Hobby King Bixler, guarantee that the Sky Surfer or something very similar will be available well into the future.

5.27.16 RC Equipment Selection

A portion of hardware included in the test-bed are not unique to an autonomous UAV, but necessary for all RC aircraft to operate. Many of these components were supplied in the ARF kit, while others were selected with consideration for supporting the autopilot hardware, described in the next section, as well as consideration for the applicable hardware requirements outlined in Section 5.25, such as cost and reliability. Suggestions from the DIY Drones community

and RC enthusiasts at Cal Poly were taken into consideration during the selection process as well. Table 5.2 lists all of the traditional RC components acquired for

Table 5.2: RC Equipment Selected

Transmitter (Tx)	2.4 GHz Spektrum Dx6i
Receiver (Rx)	6 Channel Spektrum AR26210
Motor*	1900 Kv 2620 Brushless Outrunner
ESC*	20A
Servos*	9g
Battery	2200 mAh Turnigy 3S 25C LiPo
BEC	3DR Power Module

*Included in the Sky Surfer ARF kit

this project. The Spektrum Dx6i provides a balance between cost, reliability, and utility. Six channels are the minimum number required to control a 4-channel airplane as well as support the capacity to select several autopilot flight modes during flight. The receiver was included with the transmitter, but one of the selling points of Spektrum products is that receivers can be quickly interchanged with a simple binding procedure. Furthermore, the Dx6i supports the storage of the multiple receiver profiles and transmitter configurations within its on-board memory. This is a nice feature to have, especially if there are multiple projects running in parallel, where the relatively expensive transmitter must be shared.

A larger capacity battery was selected, despite the recommendation of the manufacturer, to increase the endurance of the aircraft. Any extra time in the air is quite valuable, especially when considering the time it takes to launch and recover three aircraft safely. To account for the increased battery size in the weight and balance of the aircraft, several metal washers serving as counterweights were

removed from the nose of the aircraft.

Lastly, an external Battery Eliminator Circuit (BEC) is included in the test-bed, selected based on the autopilot selection discussed in the next section. These components serve to remove the requirement of a second battery pack, dedicated to powering the receiver and servos on the aircraft, and are now typically built in to the ESC. However, the power coming from the BEC that is integrated in to the cheap ESC included in the Sky Surfer ARF kit, rarely produces clean power, with wide variations in voltage supplied when under different loading conditions. While these fluctuations are usually within the tolerance of the relatively robust receiver and servos, more sensitive hardware, such as an embedded control system, require a cleaner, more reliable source of power. The BEC provides this, along with some extra redundancy in the event that the cheap ESC were to fail during flight.

5.27.17 AutoPilot Selection

The autopilot selected for use in this project is the ArduPilot Mega 2.0, often simply referred to as the APM 2.0. During the course of this project, improvements were made to the design of the APM 2.0 solely for manufacturing purposes, which left all of the components more or less the same, but changed the over-all layout of the board. The new board with the modified layout was named APM 2.5, and production of APM 2.0 was ceased. Therefore, a mixture of APM 2.0 and APM 2.5 boards are used in this project. This autopilot was selected based on recommendations from previous work done in reference [58] as well as high praise from the DIY Drones community, which has heavily shaped the development of this hardware since its inception.

The APM 2 meets all of the requirements set forth in the previous section for

Table 5.3: Specifications of the APM 2

μC	ATMega 2560 (Processing) ATMega 32U-2 (USB functions)
6-axis Accel/Gyro	Invensense MPU-6000
3-axis Digitial Compass	Honeywell HMC5883L-TR
High Resolution Altimeter	MEAS MS5611-01BA03
GPS Receiver	MediaTek MT3329
Airspeed Sensor*	3DR Pitot Tube Freescale MPXV7002DP
Data Storage	4MB MicroSD (APM 2.0) 4MB Dataflash (APM 2.5)

*Optional Sensors

a suitable autopilot for multiple UAV control research at Cal Poly. With a price tag that is almost 1% of the reported cost of the Cloud Cap Piccolo autopilot, which is a high-end autopilot used in previous work at Cal Poly, cost represents a dominant factor in the selection of this autopilot. While it is well understood that the Piccolo retains its high price tag for a reason, vastly improved low-end sensors produced for consumer products have begun to close the gap between high-end products like the Piccolo, and these relatively new APM autopilots. Additionally, it is not feasible to purchase and sustain the number of autopilots required for a successful multiple UAV control program if high-end autopilots like the Piccolo are selected over the cheaper alternatives like the APM 2.

The APM 2 is highly customizable, both in the sense of hardware and software. While the software is discussed in more detail in the next section, the open source APM printed-circuit-board (PCB) designs and publicly available compo-

nent and sensor details, mean that any custom software can be written to utilize the various sensors included on the board. In the sense of hardware, the APM 2.0 and 2.5 both have numerous unused digital I/O pins, as well as SPI and I^2C ports and an unused UART serial port, which allows for the addition of a number of extra components and sensors. Because the PCB schematic and layout files are open source, the board design can even be altered relatively easily to accommodate a custom piece of hardware, and then printed from a third-party PCB manufacturer.

Due to the fact that the board design and modifications have been born out of the DIY Drones community, the APM 2 has extensive documentation and support between a project wiki and the community forum, where even representatives of the manufacturer routinely answer questions from users. While there has been so much demand for this autopilot, and availability has been limited, it seems that the manufacturer has worked out the kinks in the supply chain, and availability is much improved. Furthermore, the publicly-available PCB design files insure that the board can continue to be produced by a third-party manufacturer if 3D Robotics halts its production of the board.

Lastly, using the "hacker" model discussed earlier, the most difficult aspect to evaluate at the early stages of hardware selection is whether or not the system meets the predicted software requirements. The ATMega 2560, the brain of the APM 2, is an 8 bit, 16 MHz microprocessor that has 256 KB of flash storage and 8 KB of SRAM (memory) [12]. While these specifications do not lavish the APM with heaps of untapped performance, there is enough of a reserve of storage and capability to be considered a good fit for the prototyping needs of this project, with software performance requirements that are not precisely known, but can be ball-parked with some certainty.

5.27.18 Communications

The communication hardware selected is split into two distinct sets of radios: For the FCOM inter-UAV network, 2.4 GHz XBee Pro 802.15.4 (S1) radios were selected, and for the dedicated telemetry link, the 900 MHz 3DR Radios were selected.

Table 5.4: Specifications for the selected communication hardware

	3DR Radio	XBee Pro 802.15.4
Frequency Range	915 MHz	2.4 GHz
Range	∼1 Mile	∼1 Mile
Power Usage	100 mW	60 mW
Data Throughput	Up to 250 kbps	Up to 250 kbps
Network Topology	Point-to-Point	Point-to-Point Point-to-MultiPoint Peer-to-Peer

The XBee modules selected meet all of the requirements set forth for communication, set forth in Section 5.25. With a range up to 1 mile, they meet the specific requirements derived for flight tests at the Cal Poly EFR as well as the throughput requirements for the FCOM inter-UAV network. Capable of configuration for point-to-point, point-to-mulitpoint, and peer-to-peer, these modules are well suited for an inter-UAV network that requires a distributed peer-to-peer topology. Most importantly, the XBee modules are easy to configure and are well documented. While these modules offered compatibility to the ground control station software in a point-to-point, transparent mode configuration, the peer-

to-peer network, necessary for the FCOM network to function, requires that the data intended to be transmitted be integrated into an XBee-defined API packet. Details about the XBee API protocol can be found in the XBee documentation [13].

The standard ground control station software, which will be discussed in the next section, offers a streamlined data logging and in-flight parameter adjustment interface, but unfortunately does not support reading the XBee API packet, nor can it handle multiple UAV telemetry transmission through a single radio at this time. Therefore, the vital telemetry monitoring and logging, as well as the in-flight parameter and waypoint functionality is, instead, enabled through the use of a second RF module, the 900 MHz 3DR Radio. This point-to-point radio offers the same range and data throughput as the XBees, and is also relatively easy to set up.

6 ArduPlane

6.28 The ArduPlane project

When the first ArduPilot board was made available for purchase on Sparkfun in 2009, There was no distinction between hardware and software, with both referred to simply as "ArduPilot". During the explosive development over the last four years, both the hardware and software have come so far that they hardly resemble their humble origins. One exception is the root of their name, Ardu, which pays homage to the Arduino platform they grew from (and, ironically, have recently outgrown).

Over the course of development, the hardware, now called ArduPilot Mega, has become a universal autopilot, capable of controlling any vehicle, given that the appropriate software is loaded. New software developed for several different classes of vehicles, coupled with the growing confusion from referring to both hardware and software by the same name, the ArduPilot Mega software for fixed wing applications was renamed ArduPlane. Similarly, software for multi-copters and land vehicles were renamed ArduCopter and ArduRover respectively. Since August, 2011, when ArduPlane branched off of ArduPilot, 25 versions of the software have been released by the development group. Almost every single release has added new functionality, improved performance, or both.

The version of ArduPlane serving as the foundation for this work is ArduPlane v2.66, released on October 30th, 2012. Since then, 7 new versions of the software have been released. The rapid development and dramatic increase in autopilot performance is a testament to what community-driven, open source projects can achieve.

In addition to the ArduPlane software, developers in the DIY Drones community have also released several tools to support testing and simulating with ArduPlane and the ArduPilot Mega. These tools include various Arduino test sketches, SIL simulation support through FlightGear, and HIL simulation support through both FlightGear and X-Plane.

6.29 Standard Functionality

To give the reader perspective on the modification made to ArduPlane for formation flight functionality, a rough summary of the standard v2.66 code is presented in this section. For a more detailed explanation of the standard code, the ArduPlane project wiki [1] is well documented, and regularly updated to reflect changes in the software.

The flight modes supported in ArduPlane v2.66 include

- Stabilize
- Fly-By-Wire A/B
- Automatic Take off and Landing
- Autonomous Waypoint Mission
- Guided
- Loiter
- Return-to-Launch

ArduPlane also supports telemetry communication based on the MAVLink protocol [6], which can be used to display pertinent data, change on-board parameters, or even directly control the aircraft movement in guided mode, all from a ground control station. Datalogging of sensor, performance, command, and many other values is supported as well, both on-board and through the ground control station.

The Micro Air Vehicle communication protocol, MAVLink, "is a very lightweight, header-only message marshaling library for micro air vehicles" [6], and is the communication protocol of choice for many open source autopilot projects. The protocol details can be found on the MAVLink project wiki [6], but some elements of the MAVLink project are worth mentioning in this discussion of ArduPlane, as it ties in to some of the software modifications discussed in the next section.

The MAVLink project distributes a library generator that takes an input KML file, with a formatted list of message definitions, and generates all of the necessary C++ header and source files to fully integrate the protocol into software. ArduPlane v2.66 uses MAVLink v1.0, which is a standardized build that supports all of the message types for ArduPlane to communicate telemetry data to the ground control station, as well as receive parameter changes and waypoint commands, the format for which is also defined by the MAVLink protocol.

The generated library also supplies "convenience functions", which ArduPlane uses to encode and decode telemetry payload packets, as well as prepare the packet header and calculate the packet checksum prior to sending. It should be noted, however, that the MAVLink library does not provide a direct interface to the ArduPlane software. This is achieved through through ArduPlane's `GCS` library and the `GCS_MAVLink` Arduino Sketch, which is responsible for the preparation and the handling of the the MAVLink message payload.

6.30 Software Process Flow

Figure 6.1 is a high-level graphical description of how the ArduPlane v2.66 code is organized. There are four primary loops: The fast loop, medium loop, slow loop, and one-second loop, running at 50Hz, 10Hz, $3\frac{1}{2}$Hz, and 1Hz respectively.

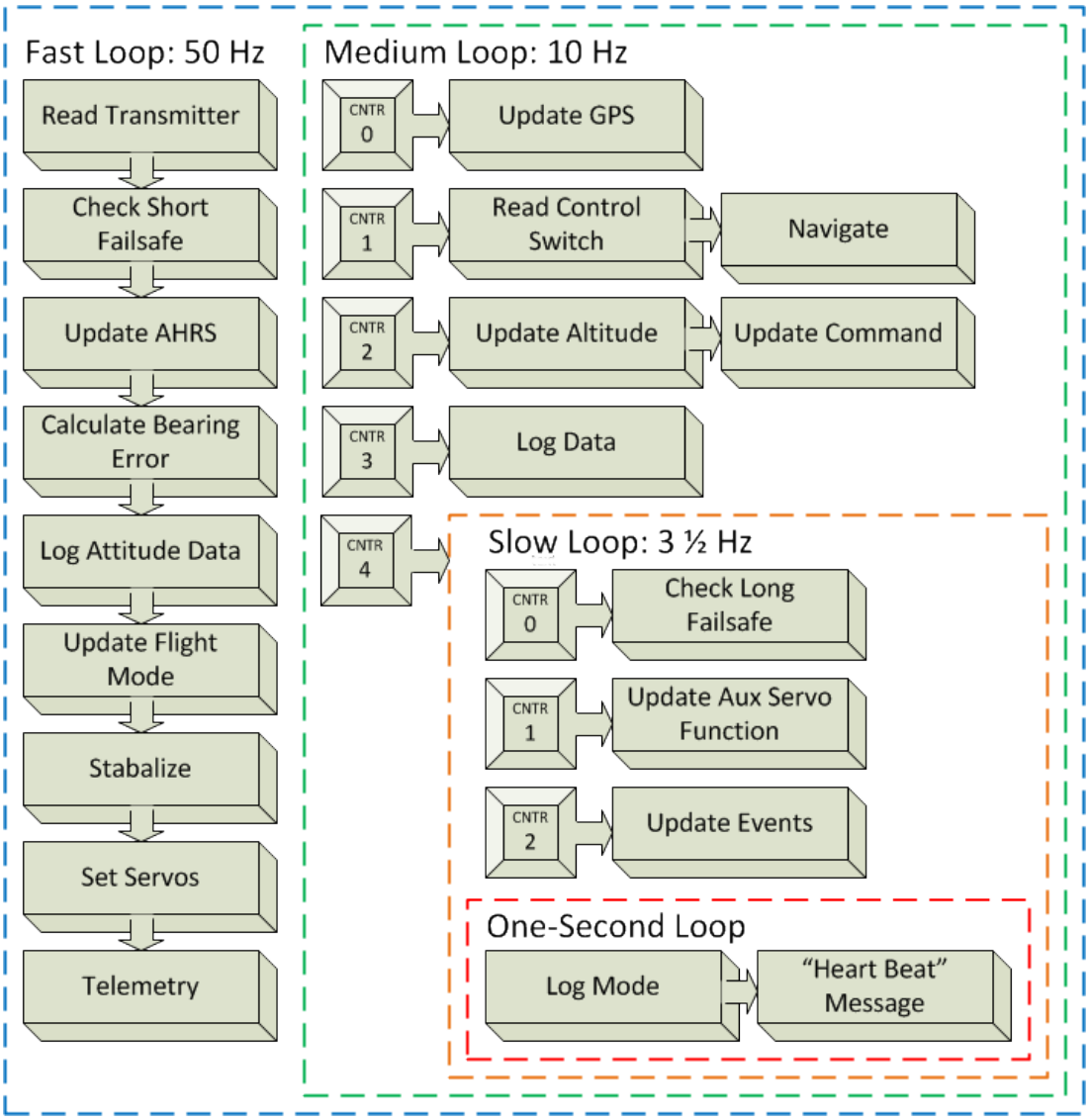


Figure 6.1: The loop-oriented process flow for ArduPlane v2.66

In general, functions are organized into loops according to bandwidth requirements. An example in the fast loop is the functions reading the transmitter and setting the servos. These functions should be called at the highest rate to emulate hardware-like throughput of commands while in `Manual` mode. Another example in the fast loop is the inner-most control loop function `stabilize`. Because it is the most inner-loop controller, the rest of the control loops are bandwidth-limited

by the rate it is executed. Attitude sensors are updated in this loop along with the Direction Cosine Matrix and various other state estimates and corrections.

The medium loop houses most of what is required for the outer-loop control of the UAV, such as the Navigation PID controller and the Attitude PID controller. Autonomous missions are also processed in this loop, using GPS updates to evaluate when waypoints have been reached. The GPS and airspeed sensor are updated in this loop, and a majority of the data logging occurs in this loop as well.

The slow loop and the one-second loop contain functions that are meant for redundancy or only used in specialized cases which do not require high-rate execution. An example of this is the `HEARTBEAT` MAVLink message that is transmitted from the one-second loop, and serves as part of MAVLink's convention to verify that a system is still connected.

6.31 Controller Implementation

The control architecture nested inside the various loops can be best described as a cascading PID control system. Figure 6.2 shows how each tier interacts with the others. Different levels of autonomy use select levels of the controller: `STABILIZE` only uses the Attitude PID controller, converting it into a regulator by setting the command inputs to zero. `FBW`, or Fly-By-Wire mode, Also only uses the Attitude PID controller, but the command inputs are set on the transmitter stick deflections. Only in the fully autonomous modes, such as `AUTO` and `GUIDED`, do commands proceed through the Navigation PID controller, which in turn feeds commands into the Attitude PID controller.

More recent versions of ArduPlane have included optional alternative con-

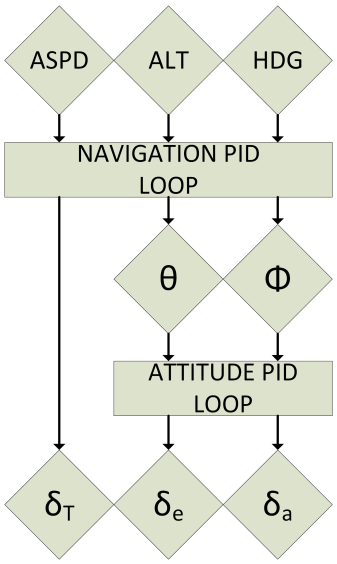


Figure 6.2: The PID control structure used in ArduPlane v2.66

troller schemes, moving away from the simple cascading PID. Early reports from developers and community members have indicated that controller performance has increased with these additions, but inclusion of these new control schemes is the subject of future work.

6.32 Compatible Ground Control Stations

The APM Mission Planner is ground control station software that serves both as a traditional ground control station and as a configuration utility for the APM and associated firmwares. The software was written by a member of the DIY Drones community and is primarily intended for use with the APM system. Because of its primary focus on the APM ecosystem, it provides the most straight-forward and intuitive interaction with ArduPlane. Included in the software is the ability to load new firmware, update parameters, and access a command-line interface for more advanced functionality. As the name implies, fully autonomous waypoint missions can be designed and uploaded to the APM

hardware. When connected to an RF link, telemetry data can be displayed during flight, as well as logged for later analysis. Furthermore, the APM Mission Planner acts as a bridge between the APM hardware and supported simulation environments, providing a straight-forward means to perform HIL simulation.

From the standpoint of Multiple UAV control, there is one crucial element lacking from the current version of the Mission Planner software: The ability to connect to more than one UAV in the same instance of the the executable. While this summary provides some point of reference on the Mission Planner software, a more detailed description can be found on the ArduPlane project wiki [1].

Because ArduPlane utilizes the MAVLink protocol, which has become a standard in the open source autopilot community, there are several other ground control stations that are compatible with ArduPlane, including

- HappyKillmore GCS
- QGroundControl
- MAVProxy

The drawback of these ground control station options is that due to their ability to accommodate many different systems, the interface is not fine-tuned for ArduPlane, making them less intuitive for use with the software.

7 Modification to ArduPlane for Formation Flight

One of the main objectives for the software development portion of this project is to build on top of, not replace, existing open-source autopilot software. By leaving the core of the standard ArduPlane v2.66 software intact, a common thread is established for future UAV work using a similar system architecture, not necessarily confined to a formation flight application. To accomplish this, the modular nature of the original software was mirrored when creating new libraries and Arduino sketch files. Object-oriented programming is modular by nature, making this process convenient for new libraries that can reference and inherit classes from the standard ArduPlane libraries to maintain conventions and increase compatibility.

Besides the convenience of having building blocks in place and maintaining a common thread that is suitable for general UAV work, another benefit of building on top of the existing software is that several versions of this software are flown by hundreds of amateur hobbyist, and have been rigorously tested and debugged by developers and members of the DIY Drones community. This helps to limit the possibility of bugs popping up outside of new or modified libraries, making debugging more manageable overall in the later stages of the project.

There is a downside to using the source code for a relatively complex system: the architecture and interface conventions must be thoroughly understood in order to make meaningful additions to the software. Furthermore, making modifications to existing libraries requires a certain level of understanding about the dependencies of that library in order to avoid introducing bugs or causing compiling errors. Many of the interfacing problems can be addressed by respecting and mirroring the conventions already put in place wherever it is possible,

maintaining conventions such as:

- Function name conventions
- Variable name conventions
- Access to variables
- Data structures
- Variable scope
- Pointers and references
- Macros and pre-processors
- Integer representation of decimals and significant digits

Starting a project from scratch can give more freedom to the programmer to put in place conventions they find more intuitive, removing the need to reverse-engineer some of the more subtle conventions and their intended purpose. However, as discussed before, having debugged, flight-tested code to begin with drastically outweighs the steep learning curve associated with adopting a foreign set of conventions in the long run.

7.33 New Classes

With the tiered structure of ArduPlane, most of the code written to interface hardware suites, sensors, and optional functionality can be found in the included libraries in the form of C++ classes . This is intended to promote maintainability and limit the impact of changes to the hardware or supported functionality on the main arteries of the code: The Arduino sketches that tie everything together.

During initialization, many class objects are instantiated to represent physical sensors, such as `gps` and `airspeed`, which are instances of the `AP_GPS` and `AP_AIRSPEED` classes respectively. In some cases, class objects are instantiated to

represent more abstract elements that often inherit methods from sensor classes to interface with the physical world, but also include methods intended to perform corrections, conversions, rotations, and approximations to the data collected by sensors to generate values that are not directly measurable. A good example of this type of class is the `AP_AHRS` class. The new classes introduced in this work are similar to the `AP_AHRS` example in that they do not represent a direct interface to a physical hardware entity.

The `flock_member` class is an addition to ArduPlane that allows for the local representation of each formation member. The methods included in the `flock_member` class are intended to comply with existing ArduPlane conventions, providing access to set and retrieve protected variables. In that manner, the information relevant to formation flight for each member is stored within their instance of the `flock_member` class. Such information includes

- State information
  - Latitude
  - Longitude
  - Altitude
  - Velocity
  - Heading
- Local leadership
- Global reference value
- Last communication

The `local_member` class is a more specialized form of the `flock_member` class, intended to represent the UAV itself. `local_member` inherits the entirety of the `flock_member` class, which allows for the convenient storage of the same type of information required to represent the other formation members. Additionally, `local_member` contains methods for the management of its local interpretation of

the flock architecture, as well as methods to perform the relative state estimation required for the presented formation flight control scheme.

Flock management methods include methods that add or remove other members from the local member's relative state calculations. When called externally, usually by the FCOM message handling function, which is discussed in next section, the member ID, as well as a pointer to the local `flock_member` instance representing that member, are both added to arrays referenced when performing relative state estimation. These arrays are kept up to date, adding or removing member information, based on external interpretation of whether or not those members are participating in the formation.

The relative state estimation methods use the state information stored in the relevant `flock_member` objects, the UAV's state stored in the `local_member` object, and the available leadership information to calculate relative positions and velocities with respect to the other formation members and the UAV's local leader.

Because position state information is stored in the form of latitude, longitude, and altitude, great circle approximations, already built in to the ArduPlane libraries, were used to obtain relative position values. Furthermore, to avoid numerous rotations, requiring computationally expensive trigonometry, relative state calculations are all carried out in the NED reference frame.

The `pf_field` class provides ArduPlane with the ability to employ the potential function guidance algorithm with virtual waypoint implementation. An instance of the `pf_field` class, when given access to the `local_member` object, uses the relative state information stored inside the `local_member` object to generate virtual waypoint and airspeed commands. Methods included in the `pf_field`

class serve to update the potential function guidance algorithm, retrieve a new virtual waypoint command, retrieve a new airspeed command, and retrieve values that give transparency to the algorithm for use in monitoring and debugging. All of the potential function guidance calculations are performed in the NED frame, with rotations to and from the LNAV frame when goal offsets are applied. Functions included in existing ArduPlane libraries are used to extrapolate the new virtual waypoint coordinate from the current location coordinate and calculated North and East offsets. Similar functions are employed to perform the near-field region correction discussed in section 3.18.6.

7.34 New Arduino Sketches

In line with the discussion of code structure in the previous sections, two new Arduino sketches are included in order to weave the new formation flight functionality together, and fold it in to the main ArduPlane code structure.

Included in the `formation_flight` sketch, functions are implemented for formation management, swarm organization logic, and command updates. At this level in the code structure, these functions have top-level scope, and greater access to variables used directly by the PID controllers. It is in this sketch that

- The local state is updated
- The global reference value is re-evaluated
- The presence of other formation members is monitored
- Global leadership, or a suitable local leader is established through the developed swarm organization logic
- Navigation commands,directing towards either a virtual waypoint or predefined global goal, are set, depending on the UAV's leadership role

The FCOM_MAVLink sketch is modeled after the standard GCS_MAVLink sketch, and includes functions to prepare, send, and handle all of the messages intended for the FCOM inter-UAV network. In the interest of maintaining convention, the MAVLink protocol discussed in section 6.29 is used for the FCOM network as well. While two standard MAVLink messages, the HEARTBEAT and the GLOBAL_POSITION_INT messages, are used to establish membership presence and exchange state information, A third, custom message type, FLOCK_STATUS, was developed to share necessary formation status information, such as each member's global reference value and their local leader's ID. This custom message type, and others discussed in section 7.37 were realized with the MAVLink library generator, briefly discussed in section 6.29. The new MAVLink library generated serves as a replacement to the standard library, providing encoding, decoding, and packaging support for the new message types along with all of the standard message types as well.

The functions responsible for sending the FCOM messages resemble those defined in the GCS_MAVLink sketch. Message deferment and MAVLink packaging process are almost identical, but unlike telemetry communication, messages intended for the FCOM network must accommodate XBee API requirements. The XBee library available for use on Arduino platforms requires some modification to the serial interface in order to work with the ArduPlane software. With these modifications in place, the revised library, dubbed AP_XBee, is used to instantiate an object to interface with the local XBee module. Methods incorporated into this AP_XBee class include: address specification, broadcasting options, and payload packaging to meet the API protocol requirements. The FCOM network uses the available Serial2 port, initialized during the ArduPlane setup for the FCOM network instead of a telemetry link, to communicate with the XBee module.

Message handling occurs in a similar fashion to the message handling defined in `GCS_MAVLink`. The `fcom` object is an instance of the `GCS_MAVLink` class, assigned the `MAVLINK_COMM_1` channel, and therefore, message receiving from the serial buffer occurs in exactly the same fashion that it would for messages received from a ground control station. After receiving, the difference between the two lies in a modification of the standard `GCS` library. Messages coming from any system ID other than the ID assigned to the ground control station are intercepted and redirected to be handled by a function in the `FCOM_MAVLink` sketch. The different messages received can be handled to update member states, request that a new member be added to the formation, and update information used for leadership determination.

7.35 New Configuration Files and "common" Header Files

The ArduPlane convention uses two formats to give variables global scope: variables are either included in the global variable structure or defined using a pre-processor macro. Variables that require the ability to change during run-time, like parameters, are usually included in the global variable structure. Variables that change fundamental aspects of the way ArduPlane is compiled or runs, such as the HIL configuration variable, are defined as pre-processor macros. Additionally, variables that do not change value during run-time are also defined as pre-processor macros.

A drawback to using pre-processor macros, and global variables in general, is that the variable could be used anywhere in the code, and therefore could introduce bugs by using unintended values that were assigned for an unrelated purpose. Additionally, defining a pre-processor macro in more than one place will cause errors during compiling. To address this problem, the ArduPlane

convention is to group these pre-processor macros into configuration files and "common" or "defines" header files.

Adding formation flight functionality to ArduPlane involves the addition of several new global variables in the form of pre-processor macros. Mirroring the conventions in place, variables created to alter the way that the new functionality operates during run-time are grouped into two configuration files: `APM_config` and `APM_config_formation_flight`.

In general, variables in these files serve to enable the formation flight functionality, define the formation member, define FCOM and new log parameters, define potential function guidance algorithm parameters, and define the global goal. Additionally, new global variables that do not fundamentally alter the process of the code, or new interface data types can be found in the `formation_common` header file.

One specific configuration worth highlighting is the selection of the formation member for which the compiled firmware is intended. Several aspects of the control scheme depend on an assumption that the participating systems are homogeneous. A single line of code separates the firmware loaded on one autopilot from the others, enforcing this homogeneous assumption, and making firmware updates more manageable.

7.36 New Control Modes

Included in `APM_config_formation_flight` is the addition of two new mode macros: Formation Flight, and Manual in Formation. Each of these modes are defined with macros corresponding to a numeric value, consistent with the convention used for modes already defined for the standard version of ArduPlane.

7.36.19 Formation Flight

The majority of new capability added to ArduPlane is executed while the autopilot is in Formation Flight mode. In this mode, each flock member is communicating on the FCOM network, broadcasting their state and updating the locally-stored states of other flock members based on information sent over the network. Also while in this state, the swarm organization algorithm, potential function guidance algorithm, and virtual waypoint algorithm are all engaged. Lastly, the "formation health" is monitored in this mode, tracking the heartbeat communications from other members and deciding when they are no longer participating in the formation. This mode is a fully-autonomous mode where the UAV uses prescribed logic to decide to either go to the global goal or follow a local leader.

7.36.20 Manual in Formation

This mode is included as a way to accomplish interesting experiments with the formation "on-the-fly". In this mode, the UAV maintains its current leadership role (global, local, or none) and continues to broadcast its state, but is under manual control. This allows for observing what happens when a formation member diverges from the global goal and how their leadership role alters the response of the rest of the formation members.

7.37 New Additions to Assist in Software Debugging, Test Transparency, and Post-Test Analysis

Early simulation and testing revealed the need for greater transparency into the code, real-time feedback of algorithm performance, and more data specific to

the new formation flight functionality. Accordingly, several additions are included in the modified ArduPlane to accommodate this need, such as

- Real-time debugging mode for HIL simulation
- Additional custom MAVLink messages for transmission to the ground control station
- Additional logs specific to the new formation flight algorithms

Microsoft Visual Studio 2010 was used throughout this project as the primary IDE for code development. With the VisualMicro plug-in [7], this IDE combines helpful utilities, such as intelisense, with support for the Arduino language and associated hardware. Furthermore, the Visual Micro plug-in supports a convenient debugging mode that allows trace points to be added to the software that is loaded on the hardware, which enable streaming variable values and debugging messages real-time through an open serial port. This type of debug support offers improved flexibility that is easier to implement and requires less memory than a traditional debugging scheme for embedded systems. In order to stream the debugger messages back from the APM board, a serial port is required to interface with Visual Studio on the monitoring PC. In the developed configuration, the often-unused serial port, `Serial2`, is occupied by the XBee module serving as the transmitter and receiver for information on the FCOM network during HIL simulation. Therefore, new functionality is included in the initialization process to re-calibrate the port typically used for interfacing with the GPS receiver, `Serial1`, which is not required for HIL simulation due to simulated GPS data streaming from the simulator through the standard simulation interface connected to `Serial0`.

The new custom MAVLink messages

- `FF_PF_FIELD`

- FF_VWP
- FF_REL_STATE

were designed with greater transparency during flight testing in mind. Values for the potential field gradient vectors, coordinates for the virtual waypoint, and values for the relative state vector used in calculations are included in these messages. These new messages are included in the custom MAVLink library generated, supporting their use in the ArduPlane code.

Unlike the other messages intended for the FCOM network, these messages are addressed to be sent directly to the ground control station XBee module, included to serve as a ground-based monitor for the FCOM network. Support for these new messages on the ground requires a custom version of QGroundControl software, with the source code edited to include the custom MAVLink library, and then re-compiled for use on a PC. An additional benefit associated with using QGroundControl to monitor the formation during flight testing is the support of multiple UAVs in its interface. Therefore, not only can the custom MAVLink messages be interpreted and monitored in their MAVLink widget, but markers for all three UAVs appear on the main map, giving a bird's-eye-view to evaluate formation convergence during testing.

Lastly, new functions to log data relevant to the formation flight algorithms are defined and incorporated into the medium loop of the ArduPlane code. These logs generally parallel the data included in the custom MAVLink messages sent to the ground control station, making it easier to analyze the performance of the various new algorithms introduced after testing, and determine improvements for future iterations of the software.

7.38 Implementation in to the ArduPlane v2.66 Process Flow

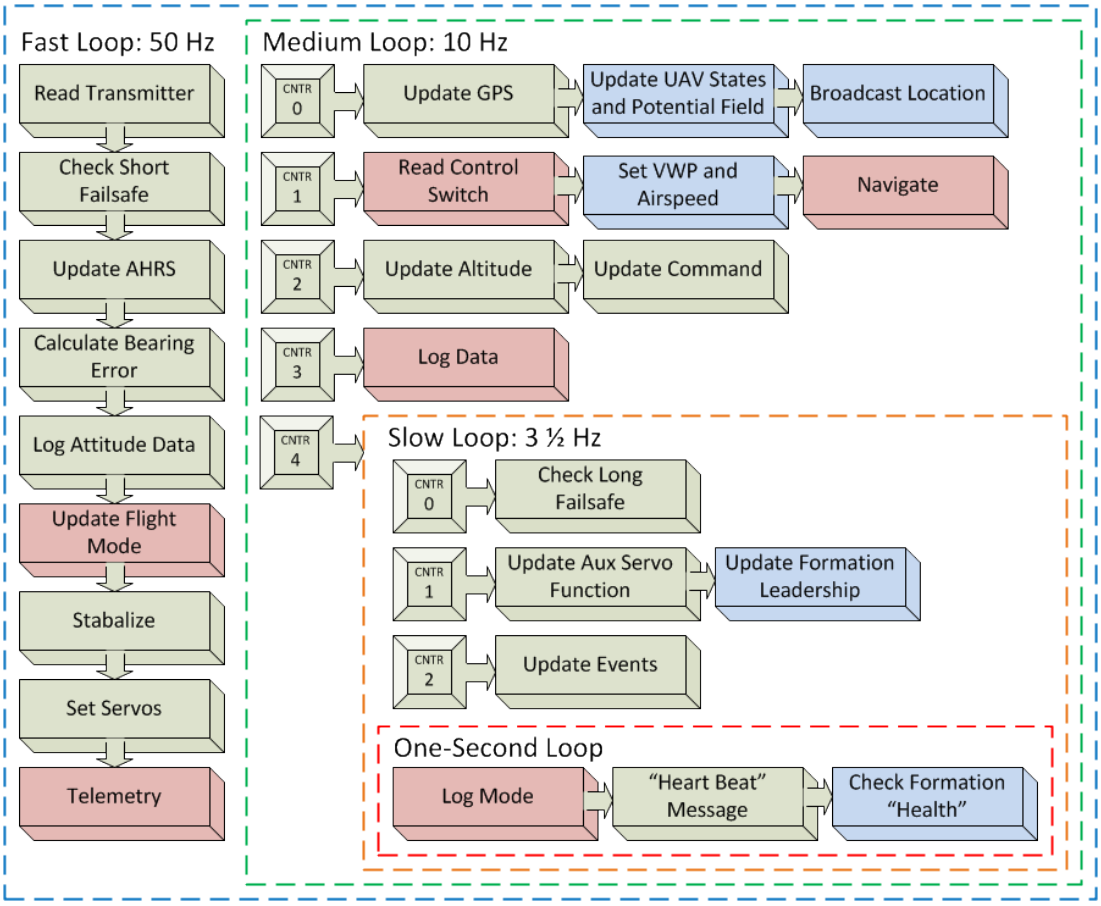


Figure 7.1: The Augmented process flow for ArduPlane v2.66 FLOC Edition

Figure 7.1 shows where the various changes and additions to the standard ArduPlane code are implemented in the custom, FLOC Project version. Process blocks highlighted in red represent areas of the original ArduPlane v2.66 source code that were modified to support formation flight functionality. Process blocks highlighted in blue represent new functionality added to ArduPlane, enabling formation flight control. The placement of these new processes in the flow were determined by examining existing ArduPlane conventions for communication support, navigation, and event determination.

8 GPS 2D Relative Error Testing and Evaluation

A significant number of the concepts and schemes presented in this work rely on the assumption that there is some level of relative situational awareness available to each UAV. As discussed in section 4, the method chosen for this implementation scheme is a state-sharing approach, based on broadcasting GPS data for the other formation members to use in their relative state estimate. Furthermore, GPS provides the only feasible way to observe the UAVs during flight testing in a way that can be quantitatively analyzed to evaluate leader-follower convergence and formation cohesion. Such high dependency from both a control scheme and observational perspective makes the GPS the linchpin of this whole project.

The importance of understanding and quantifying the error expected from GPS-based relative distance calculations was recognized early in the development of the presented system architecture, as this error trickles down through the entire control scheme, having a large impact on the upper-bound limit of convergence expectation of the formation flight controller.

8.39 Source of GPS Error

Documentation[11] for the GPS receivers used in this work, the MediaTek MT3329, cites a 2D RMS absolute position accuracy of 3 meters. However, absolute position is not very relevant to a formation flight problem, which generally deals more with relative positions. What is relevant is the relationship between this absolute position error and the error associated with GPS-based relative distance calculations. GPS error is broken down in to several categories, based on the source of that error[22]. These error sources include

- Ephemeris Data
- Satellite Clock
- Ionosphere
- Troposphere
- Mutipath Reception
- Receiver Measurement

Several of these error sources are rooted in variables such as satellite visibility, atmospheric interference, and hardware precision. With that in mind, it could be conceived that two identical GPS receivers, placed in relatively close proximity, would benefit with a reduction of relative error.

8.40 GPS 2D Relative Error Experiment

To understand the expected error that would be associated with relative distance calculations, an experiment was designed to quantify this error expectation using the same hardware and firmware used in flight testing.

Based on the information stated in the previous section, a hypothesis was developed for the results of this experiment: If the 2D RMS error associated with a receiver's reported absolute position is 3 meters, then a lower 2D RMS error for relative distance measurements can be demonstrated by using identical receivers in close proximity.

To test this hypothesis, static GPS tests were performed at the Cal Poly EFR. Reference distances were established by taping markers, spaced 2 feet $\pm$ $\frac{1}{4}$ inch apart. Placing 19 markers, the set up supports relative distance measurements between 2 feet and 36 feet. Two full sets of hardware were utilized, complete with an APM 2, MediaTek GPS receiver, Tx/Rx combination, XBee RF module,

3S LiPo, and BEC. Using flight-ready custom ArduPlane firmware, the relative distance measurement process that is performed during flight was replicated on the ground.

Figure 8.1 shows the set up of the experiment out at the Cal Poly EFR.

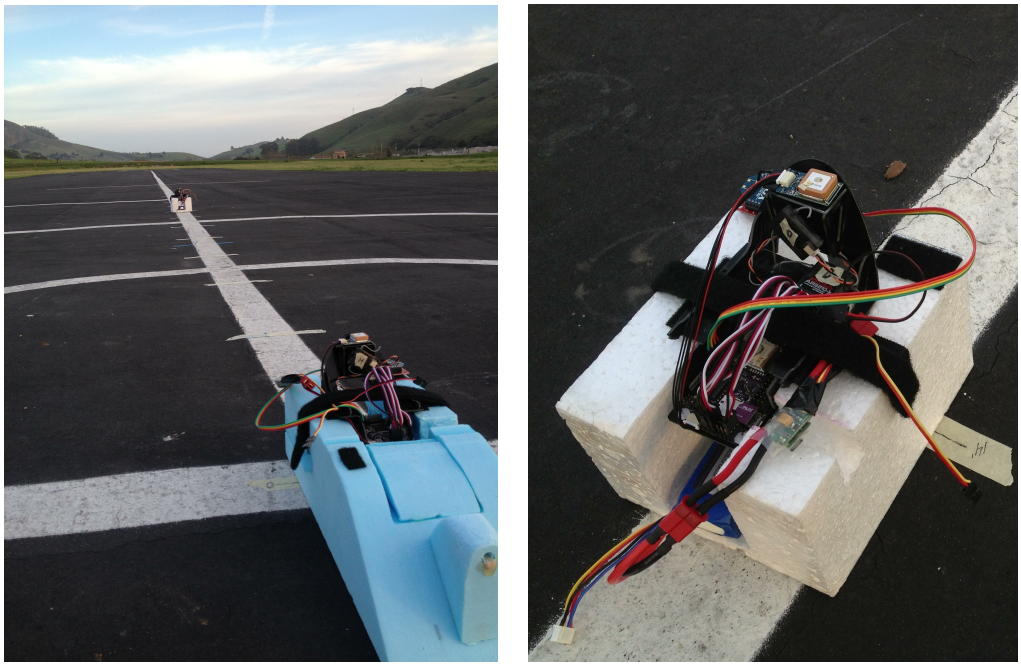


Figure 8.1: The GPS relative distance error evaluation test set-up at the Cal Poly EFR

The following procedure was conducted during the experiment:

1. Both systems should be powered on, with the Tx mode set to `Manual`
2. The "leader" system should be placed at the center of the runway at the "0 ft" marker and the "follower" system should be placed down the runway at the last marker, labeled "36 ft"
3. Allow for both systems to establish a 3D GPS lock, verifying their lock at the ground control station

4. Toggle both systems into `Formation` mode simultaneously by changing the mode switches on their respective Txs. Leave in `Formation` mode for ∼60 seconds

5. Monitor the broadcast location and relative state messages from both systems at the ground control station. Also using the output from the GPS receiver connected to the ground control station, record the satellite IDs, HDOP value, and a reference UTC time for each trial

6. After each trial, toggle the systems in to `Manual` mode simultaneously, so that data sets are differentiable later during log parsing

7. Move the "follower" system closer to the "leader" by placing it at the next marker

8. Repeat steps 4-7 18 times, which will log the relative distance values calculated by the "follower" for actual relative distances between 36 feet and 2 feet.

While in `Formation` mode, the follower calculates the relative distance in the same fashion it would during flight: Evaluating the difference between its own GPS-reported position and the GPS position information received from the leader over the FCOM network. That value is then logged as part of the new `REL` log, integrated in to the custom version of ArduPlane developed for this work. Obtaining a "measured" relative distance this way guarantees that the error expectation developed from the analysis of this experiment will be directly applicable to the error expected for relative distance calculations in flight.

To provide a reference point for analysis, standard error propagation techniques were used to find the most probable error of the GPS-based relative dis-

tance using the 2D-RMS error for the GPS's absolute position measurements, providing a baseline 2D RMS relative distance error of $3\sqrt{2}$ meters.

Because GPS error is often discussed in terms of RMS value, this convention was used to analyze the data for each trial point, using

$$2DRMS\Delta d_{rel} = \sqrt{\frac{\sum_{i=1}^{N}(d_{rel_i} - d_{rel_{mrk}})^2}{N}} \tag{8.40.1}$$

Figure 8.2 shows the calculated RMS value for each trial point as the follower is moved closer to the leader.

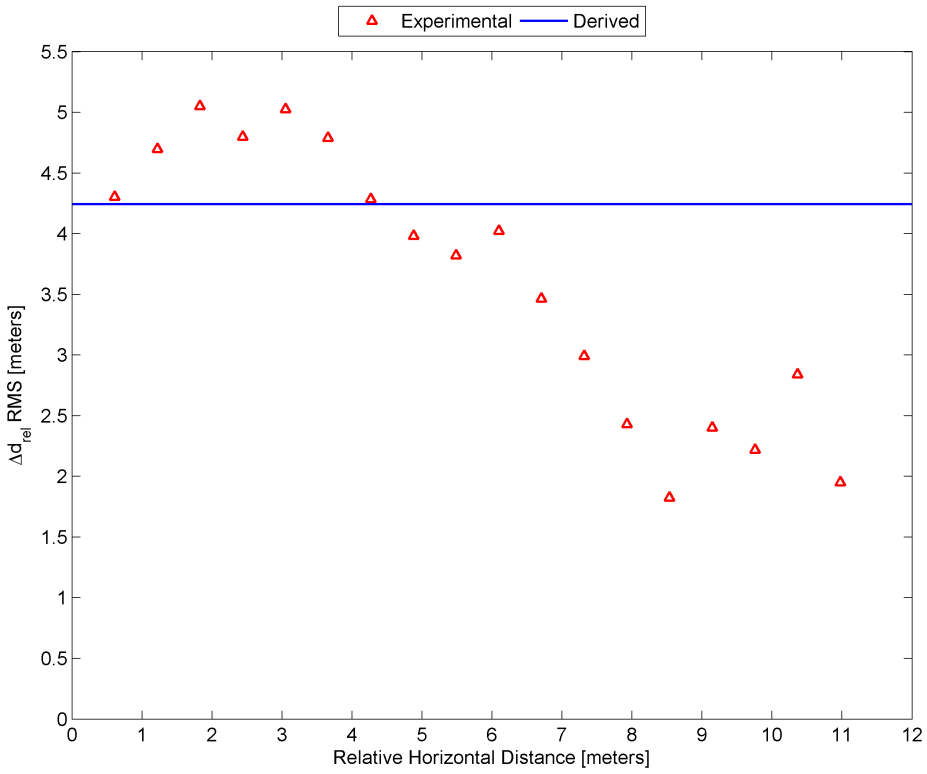


Figure 8.2: The calculated 2D RMS relative distance error value compared to the derived 2D RMS relative distance error value.

As Figure 8.2 shows, there is a trend showing that as the systems get closer together, the 2D RMS error increases. Unfortunately this trend does not show

decreases in RMS error across the board, as was hypothesized. Upon further research, it was found that with WAAS corrections now embedded into the GPS signal, there is little actual correlation of errors between receivers, making the original hypothesis il-posed. Trying to accomplish differential GPS (DGPS) with two GPS receivers that work with pre-filtered data has been given the misnomer "poor man's DGPS" because, although it resembles a cheaper implementation of a type of relative GPS technique, in order to really reap the accuracy benefits of DGPS measurements, a higher-end GPS receiver that can work with the raw psuedo range values must be used.

Despite the failure to demonstrate increased accuracy of relative GPS measurements using the selected GPS receivers, this experiment was able to provide a characterization of the relative position error expected during flight testing. From the perspective of flight testing data analysis, Figure 8.2 does provide valuable information about the error associated with each measurement, which must be taken in to account when interpreting the flight test results. However, for control scheme purposes, Figure 8.2 does not provide a straight-forward means to evaluate how close two UAVs can be commanded to fly together without risking a collision due to GPS-related error. Instead, a better way to look at this problem is through a lens of probability. Now the question is re-framed to be: If the UAVs are flying at some distance apart, what is the probability that their GPS error is larger than that distance, risking a collision? Based on the distribution of GPS data[64], the probabilty equation can be written as

$$Probability(Error \leq Distance) = 1 - e^{-(\frac{Distance}{RMS_{Error}})^2} \tag{8.40.2}$$

This information is more useful because it's talking about probable error, not an error metric like RMS, which has several cases that lie outside its bounds. Therefore, a certain level of risk can be quantified when commanding the UAVs to

fly in formation a specific distance apart. Figure 8.3 shows the probability curves

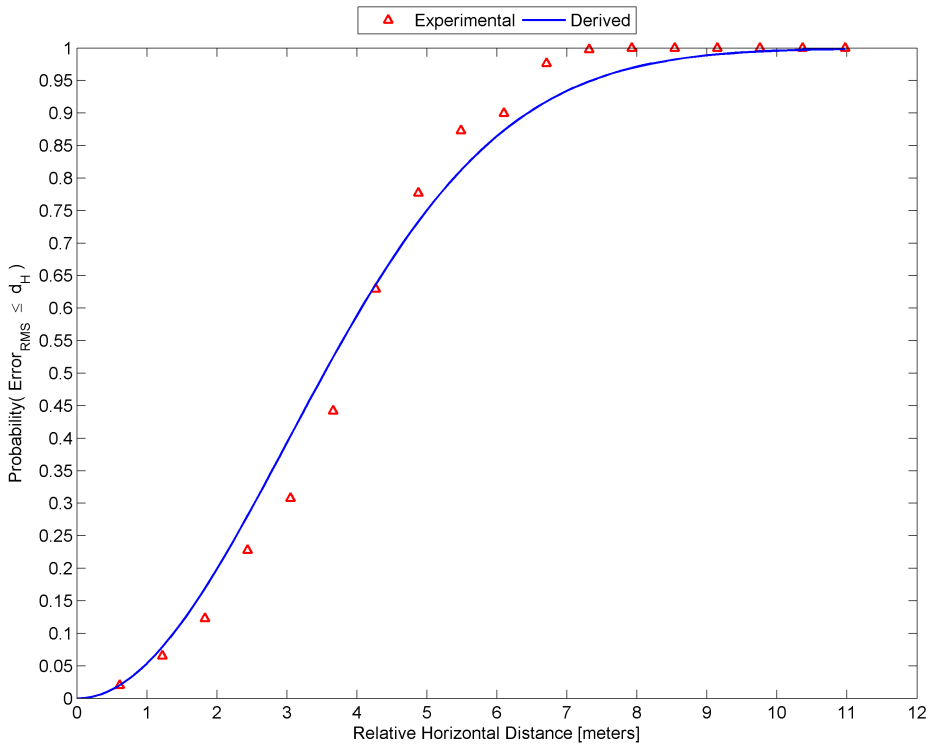


Figure 8.3: The calculated probability curve for relative distance error that is less than the relative distance itself

based on the derived RMS error and the experimental RMS values. While the convergence goal error budget, discussed in section 1.3, is comprised of multiple components, the GPS error contribution is the most significant. The error budget presented in section 1.3 includes a GPS-based distance value that represents the desire to avoid the possibility of mid-air collisions by chosing a high probability for the GPS relative distance error to be less than the relative distance value itself.

9 Simulation

9.41 Requirements for Formation Flight Simulation

To aid in the simulation system design process and configuration selection, several requirements were developed. These requirements varied from general requirements for small UAV simulation to requirements specific to formation flight. Figure 9.1 shows the requirement flow-down for simulation, grouped accordingly.

9.42 Simulation Approaches

9.42.21 Theoretical simulation with Matlab and Simulink

Matlab and Simulink provide a powerful set of tools for control system design and prototyping at multiple levels of fidelity. In [86], the proposed control system and formation flight simulation script was all built inside a Matlab and Simulink environment. This work closely follows the implementation strategy of the PFG algorithm presented in [86], therefore preliminary simulation was conducted for this work based on a variant of the simulation tool developed by Tsuruta in [86]. The objectives of this preliminary simulation work were somewhat limited, due to the distinct differences between the complete control system architecture presented in this work and that which is presented in [86]. However, by integrating a new aircraft model, valuable insight into the sensitivities of the PFG algorithm was gained. Specifically, parameter changes to due the flight regime of the aircraft or its maneuverability developed a better understanding of the algorithm.

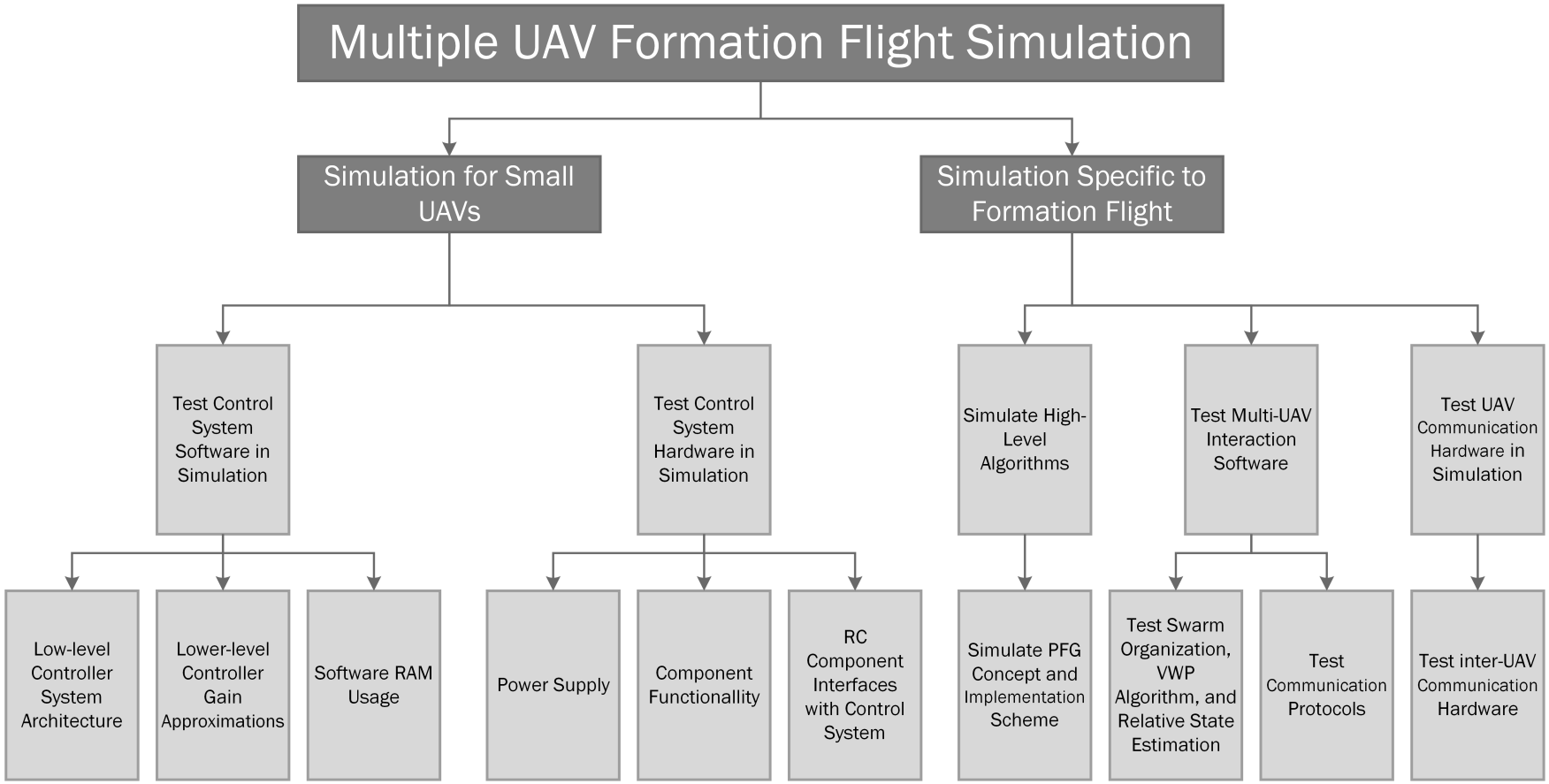


Figure 9.1: Flow-down for multiple UAV simulation requirements

9.42.22 HIL simulation with X-Plane

With the focus of this project heavily weighted towards hardware integration and flight testing, less simulation was devoted to high-level algorithm development and control system prototyping. Because of this, HIL simulation was more appropriate than theoretical simulation throughout most stages of this project. Accordingly, the objectives for HIL simulation were much more extensive than those for Matlab and Simulink simulation. HIL simulation, as employed in this work, has the capability to test $\approx 75\%$ of the embedded control system software, running real-time during the simulation. The goals of HIL simulation revolve around the ability to test the customized version of the embedded control system software in the environment in which it is meant to run. These goals include the testing of code functionality and performance on the hardware during real-time operation. Another benefit of HIL simulation is that the additional hardware components are included as part of the set-up. The functionality of the communication modules, as well as the RC receiver and transmitter can be verified, to some extent, using HIL simulation. The last goal is to provide a tool for gain tuning and serve as a predictor for flight-test performance. This goal must be interpreted loosely. The scope of this project does not include a proper validation of simulation, and the simulation and flight test process discussed here resembles more of an open-loop relationship, not the type of closed-loop relationship which would improve simulation based on flight test results and build more confidence in the simulation output. From this perspective, it is not the goal of HIL simulation to match flight test results, but it is expected that the fidelity of the simulation will provide results that resemble the results of flight tests. At least enough to make relevant conclusions regarding system functionality using simulation output alone.

9.43 Aircraft Model

Simulation results are only meaningful if they are interpreted within the context of the system model fidelity. Constructing an accurate model of the system is a difficult task that often becomes iterative as some system behaviors are often unknown until discovered during testing. However, a lower fidelity model does not render the simulation useless as long as the model deficiencies are well understood and the underlying purpose of the simulation does not hinge on a high fidelity representation of the system. The Matlab/Simulink simulation conducted during the early stages of the project utilized an aircraft model in the linearized, decoupled state-space form described by the equations shown in Figure 9.2.

$$\begin{bmatrix} \Delta\dot{u} \\ \Delta\dot{\alpha} \\ \Delta\dot{q} \\ \Delta\dot{\theta} \end{bmatrix} = \begin{bmatrix} X_u & X_w U_0 & 0 & -g \\ \frac{Z_u}{U_0} & \frac{Z_\alpha}{U_0} & 1 & 0 \\ M_u + M_w Z_u & M_\alpha + M_{\dot{\alpha}} \frac{Z_\alpha}{U_0} & M_q + M_{\dot{\alpha}} & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \Delta u \\ \Delta\alpha \\ \Delta q \\ \Delta\theta \end{bmatrix} + \begin{bmatrix} X_{\delta_E} & X_{\delta_T} \\ \frac{Z_{\delta_E}}{U_0} & \frac{Z_{\delta_T}}{U_0} \\ M_{\delta_E} + M_{\dot{w}} Z_{\delta_E} & M_{\delta_T} + M_{\dot{w}} Z_{\delta_T} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta\delta_E \\ \Delta\delta_T \end{bmatrix}$$

$$\begin{bmatrix} \Delta\dot{\beta} \\ \Delta\dot{p} \\ \Delta\dot{r} \\ \Delta\dot{\phi} \\ \Delta\dot{\varphi} \end{bmatrix} = \begin{bmatrix} Y_v & 0 & -1 & \frac{g}{U_0} & 0 \\ L_\beta & L_p & L_r & 0 & 0 \\ N_\beta & N_p & N_r & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta\beta \\ \Delta p \\ \Delta r \\ \Delta\phi \\ \Delta\varphi \end{bmatrix} + \begin{bmatrix} 0 & Y_{\delta_R} \\ L_{\delta_A} & L_{\delta_R} \\ N_{\delta_A} & N_{\delta_R} \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta\delta_A \\ \Delta\delta_R \end{bmatrix}$$

Figure 9.2: State Space Equations for a Linearized Aircraft Model

These systems of equations are commonly used to represent an aircraft model during analysis, especially as the "plant" for control system design. The coefficients in the equations are often referred to as stability derivatives, and are a way to represent the dynamics of the system at a specific condition, often a trim point for the aircraft. These stability derivatives are well understood for aircraft models with a long history of use in academic examples or validation cases, such as a Cessna 172. However, for a new design, or an aircraft that has not yet been modeled, these derivatives must be generated. There are many ways to go

about obtaining these values. References [67] and [81] present several analytical methods which rely on certain assumptions about configuration and use degenerated representations of the geometry to approximate these values; AVL[2] and XFLR5[9] are capable of outputing many of these derivatives with reasonable accuracy based on a vortex-lattice approach. A program called Datcom+[37] uses the Digital Datcom software to combine analytical approximations with corrections based on flight testing conducted by the United States Air Force. In some cases, coefficients can be approximated based on similar systems that have been well documented and modeled. Ultimately, the best source of these derivatives come from detailed testing, either in a wind tunnel or through flight test.

9.43.23 Sky Surfer Model

For the Sky Surfer model used in this project, as described in section 5, many of these sources ended up being utilized to approximate the stability derivatives of the system, with the exceptions being Datcom+, wind tunnel testing, and flight testing. According to documentation[37], Datcom methods begin to break down at Reynold's numbers lower than 600,000 where the Sky Surfer has a Reynold's number on the order of 150,000 at cruise conditions. Wind tunnel testing is well beyond the scope of this work, but could be the subject of a future project, especially if the Sky Surfer continues to serve as the test-bed for multiple UAV projects at Cal Poly. Flight test data is difficult to distill into representative stability derivatives at the RC aircraft scale because obtaining decent approximations during flight testing relies on reaching a steady trim condition; something very difficult to accomplish with a small, light RC aircraft. Published works have shown that algorithms can be employed to deal with this problem, but this process is also well outside the scope of this work. The majority of the stability deriva-

tives for the Sky Surfer were approximated using XFLR5, which relies on AVL for most of its calculations, but also includes approximations from XFOIL[10] for airfoil section drag approximation, something AVL does not account for. XFLR5 is well documented in [30] which shows that there is reasonable agreement between its approximations and validation cases. Figure 9.3 shows the model generated

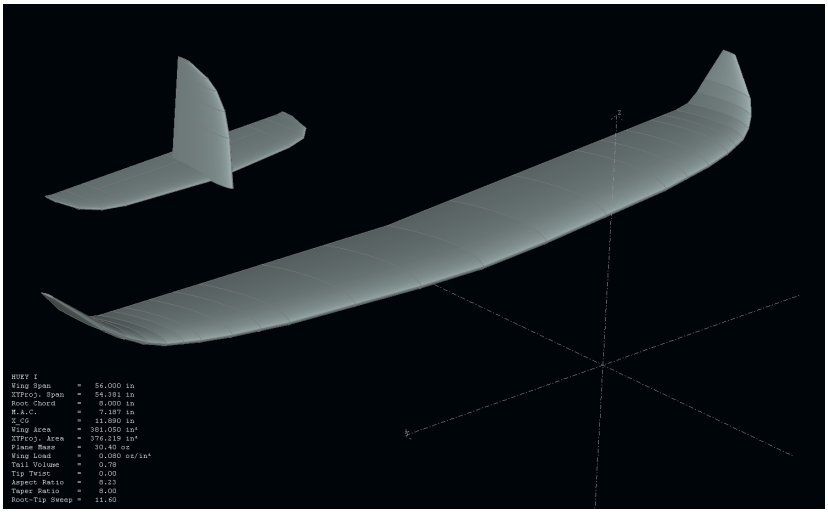


Figure 9.3: The Sky Surfer model constructed using the XFLR5 interface, used to calculate stability derivatives

through the XFLR5 interface to represent the aerodynamically-dominant components of the Sky Surfer: the wing and the empenage. In addition to airfoil cross-section and surface geometry, mass properties were also required in order to approximate the stability derivatives through XFLR5. The process to obtain the moments of inertia empirically from a physical model is described in [40].

While these "swing tests" were conducted for the model, shown in Figure 9.4, the calculated moments of inertia were not reasonable. It was decided that at this scale, and without a system for measuring the oscillation period precisely, this method is too susceptible to measurement errors, making it difficult to produce an accurate approximation. Instead, a component build-up was used, calculating the moments of inertia for each component, approximated by simple geometries,

of which the moment of inertia equations are well known, and then applying the parallel axis theorem to approximate the total moments of inertia for the Sky Surfer. Fortunately the Sky Surfer comes disassembled, making this process relatively straight forward. This inertia matrix was input into XFLR5, and the stability derivatives were generated for the expected cruise velocity of 10 m/s. While the interface for XFLR5 supports an intuitive way to include elevator deflections, generating reasonable coefficient values for other control surface deflections proved to be more difficult. Therefore, analytical approaches described in [67] were used to calculate the control surface derivatives for the Sky Surfer. Lastly, one of the most difficult aspects of an RC sized aircraft to model is the propulsion system. It was decided that replicating the propulsion model from a similar aircraft, documented in [22], was a more efficient route.

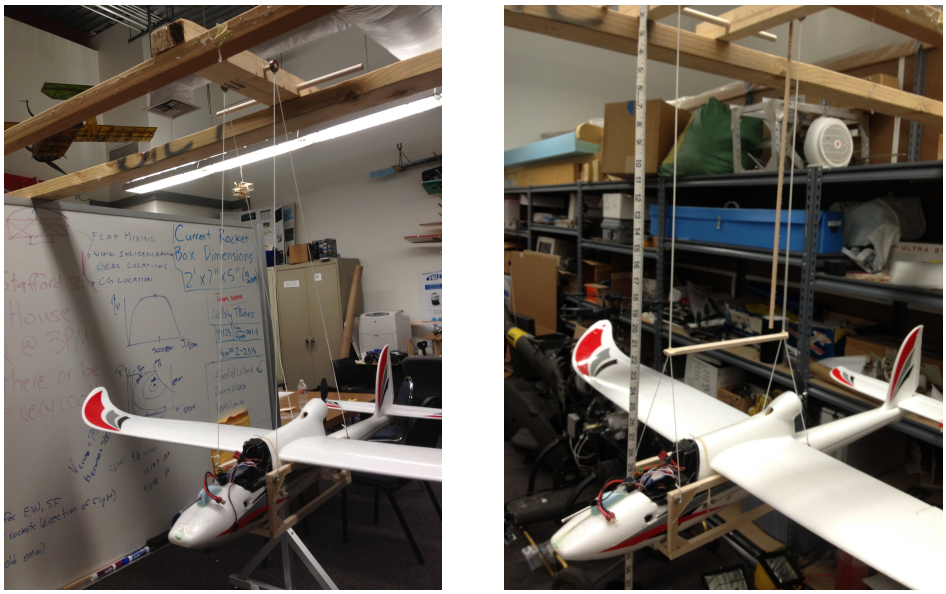


Figure 9.4: "Swing Test" performed to determine inertial properties of the SkySurfer

9.43.24 Extending the Sky Surfer Model for use in HIL Simulation

As stated before, the goals of theoretical Matlab simulation in this project were limited, and developing a model that could be integrated into a HIL simulation proved to be much more useful. Briefly, Flight Gear [4] and JSBSim [5] were considered for use during HIL simulation, providing an intuitive transition from the Sky Surfer model developed in Matlab into a form usable by JSBSim. JSBSim uses look-up tables populated with stability derivatives and relevant coefficients, evaluated at various flight conditions. The same approach used to construct the Matlab-based model of the Sky Surfer was simply repeated at several flight conditions and the resulting stability derivatives and coefficients were placed in the KML input file for JSBSim.

As the project progressed, it was discovered that X-Plane [8] would provide a more suitable interface, with built in support for multiple aircraft visualization and more documentation on the HIL set-up for ArduPlane and APM hardware. X-Plane, however, uses a completely different approach to simulating the aerodynamic forces on the system, computing the solution at each time-step based on aircraft geometry and Blade Element Theory. The details of this approach can be found in documentation for [**?**], but what is important to distinguish is that instead of using previously calculated coefficient and stability derivative look-up tables, capturing the geometry of the system is what will define the accuracy of the model.

9.43.25 HilStar17f

Fortunately, the DIY Drones community forum lead to a well-supported model for the Hobbyking Bixler, a close cousin to the Sky Surfer, developed by Mike

Pursifull [73]. While the model is still under development, and not all aspects of its performance have been fine-tuned to reflect real flight-test results, great care has been taken to compare simulation data against flight test data in order to improve agreement between the X-Plane model and the actual system. More importantly, these comparisons have lead to relatively well-documented deficiencies in the model, which is extremely valuable when attempting to extrapolate simulation results and predict flight test results.

The model, titled the HiLStar 17F, gets part of its name from the scaling that was introduced to accommodate the limitations of X-Plane's implementation of Blade Element Theory. According to the model documentation in [73], the light weight of the aircraft is what causes computational problems with X-Plane. Therefore, to increase to weight of the aircraft, but still maintain accurate flight performance, the geometry of the aircraft was scaled by a factor of 1.733, and the propulsion system was also scaled accordingly. Many of the model characteristics were tweaked in X-Plane's Planemaker to show better agreement with empirical testing of the system, with specific emphasis on wing aerodynamics, propulsion, and cruise performance. Thankfully, the documentation does point out that the model lacks consideration of matching the control surfaces and that the real aircraft cruises at a higher velocity for the same throttle setting. Additionally, the documentation highlights that the lag introduced into the system with the HIL interface, causes discrepancies between the optimal controller gain values between simulation and flight testing. Therefore, at this point, the gains tuned during HIL simulation should not be trusted for use in flight tests. While there is plenty of room for improvement, the HiLStar 17F provides a valuable, well documented, and continuously supported model for simulating the Sky Surfer in X-Plane, especially for the HIL simulation objectives outlined in section 9.42.22.

10 HIL Simulation

The HIL set-up with X-Plane allows for testing of $\approx 75\%$ of the software, compiled and running real-time on the actual flight hardware. What is not compiled are the libraries and processes that support the various sensors, translating their analog or digital signals into meaningful variable assignments to be used by the higher-level logic of the control system. The HIL set-up utilized in this work relies on an interpretation of the simulated physics and environment in order to set those variables directly, bypassing the libraries and processes typically used.

10.44 HIL Simulation Set-up

The standard configuration of the HIL simulation, designed for simulation of one UAV, can be discussed in terms of three distinct sets of components and the interfaces that are put in place to allow these components to interact. Figure 10.1 shows a diagram of the software configuration for a single UAV and Figure 10.2 shows diagrams of the hardware configuration.

The aircraft itself is simulated in X-Plane with the HiLStar 17f model. Although X-Plane is typically used as an "all-in-one" flight simulator, it can easily be used for developing external flight software. X-Plane can be set up to take in control surface deflection and throttle commands and output relevant state information such as forces and moments, GPS position, and airspeed.

The APM Mission Planner serves as the bridge between the hardware and software. With the data it collects from X-Plane, The APM Mission Planner generates MAVLink packets containing the relevant information to be decoded by the software on the APM autopilot board. MAVLink serves well as the pro-

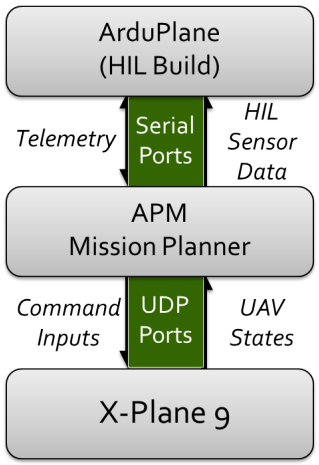


Figure 10.1: HIL simulation software setup for a single UAV using XPlane v9, APM Mission Planner v1.2.32, and ArduPlane v2.66

tocol standard because it is already used to transmit telemetry data, and the library contains messages designed to enable a HIL simulation environment. The APM Mission Planner also constructs and decodes packets intended for X-Plane, based on the X-Plane custom protocol. Furthermore, the APM Mission Planner maintains unit consistency between X-Plane and the APM sensor libraries. X-Plane and the APM Mission Planner can be running on a single computer, or on multiple PCs. In the configuration used throughout this project, a single desktop computer was used to run an instance of APM Mission Planner and X-Plane. The two programs communicate through UDP ports, assigned in the configuration settings for each program.

The third set of components are made up of all the ground-testable hardware: The APM board, RC reciever, and RC transmitter. All the hardware components are hooked up as they would be during flight, but the APM board is connected to the desktop PC via USB. This USB connection serves as the link to transmit the simulated sensor outputs from the APM Mission Planner software, but also takes the place of the stand-alone telemetry link. All of the communication to the APM

board is framed in MAVLink packets. With the board receiving inputs from the simulated physics, as well as inputs from the RC transmitter, the software runs in much the same way as it would while it is flying. The outputs from the board, usually servo rotation commands or ESC throttle commands, are translated and forwarded to X-Plane by the APM Mission Planner.

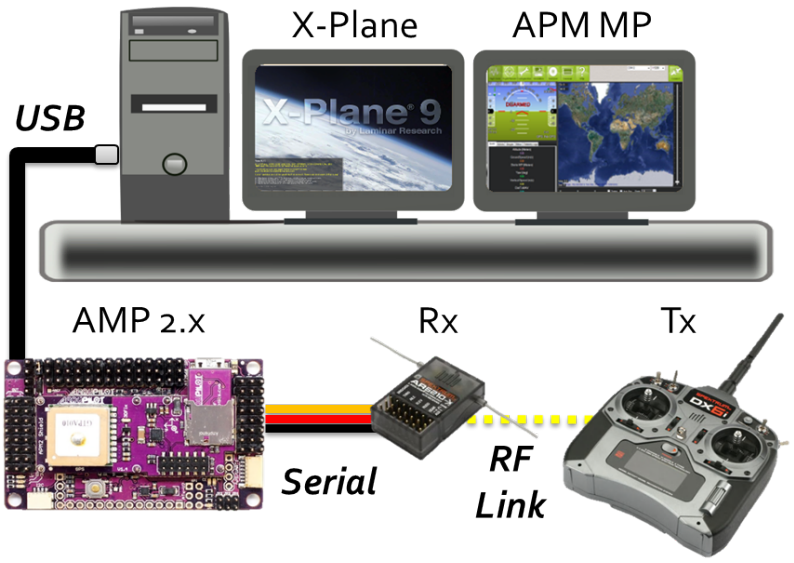


Figure 10.2: HIL simulation hardware setup for a single UAV

While requiring a significant amount of additional hardware, extrapolating the HIL set-up from one to three UAVs was more or less straight forward. As Figure 10.3 shows, the hardware requirements were effectively multiplied by three: three desktop PCs, three APM boards, and three RC Tx/Rx pairs. Added to the system for formation flight capability and observation were three XBee modules, one for each APM board, as well as an additional module, connected to a fourth computer: the laptop which serves as a ground control station during testing. A beneficial feature of X-Plane is its support of a "multi-player" mode over a LAN connection. This allowed for the other aircraft to appear in the visualization of each HIL set-up, providing useful visual feedback during the simulation process.

The set-up allowed for the integration of the inter-UAV network in the same way as during flight-testing, but the dedicated telemetry links, included in the second iteration of the communication system design, were replaced by the USB link to the desktop PC. In this fashion, each UAV was simulated independently, sharing their state information as they would during flight. By having all three UAVs interacting in the same environment, both digitally and visually, this multiple-UAV HIL simulation tool served as a cornerstone for meaningful debugging and evaluation of the custom software implemented into the system.

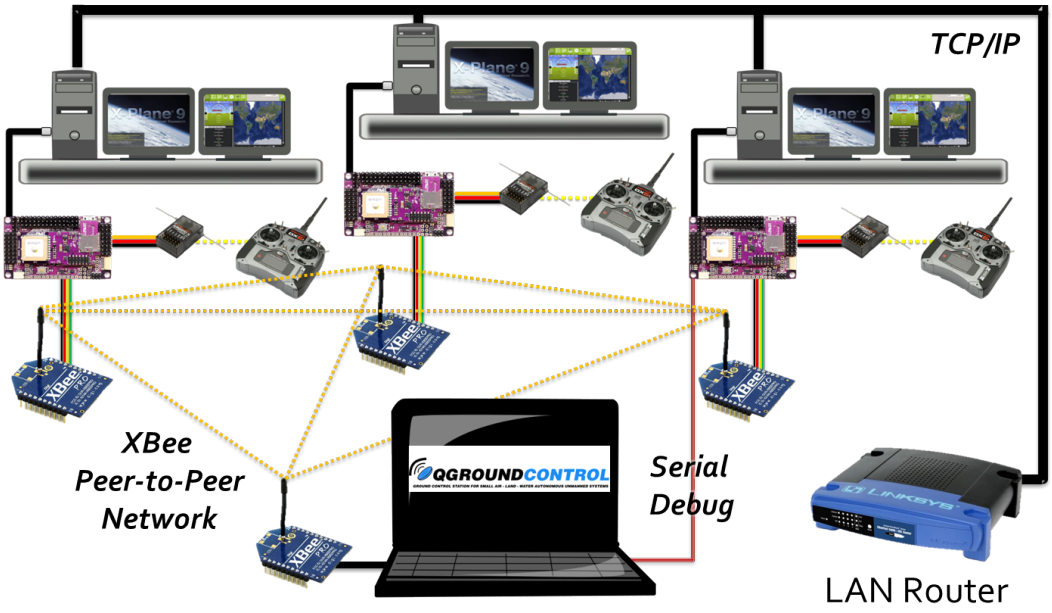


Figure 10.3: HIL simulation hardware setup for 3 UAVs

10.45 HIL Simulation Procedures

Running HIL simulation on three HIL stations simultaneously while monitoring FCOM and debugger output proves to be somewhat confusing for a single operator, making the establishment of a set of HIL simulation procedures helpful to streamline the process and ensure that run cases are performed as desired.

As shown in Figure 10.3, there are five separate connections that must be in place for simulation data to stream properly:

1. HIL Station to HIL Station over LAN
2. XPlane to APM Mission Planner over UDP
3. APM Mission Planner to APM over USB Serial
4. APM to APM over XBee API 2.4 GHz
5. APM to Tx thru Rx 2.4 GHz

Before initiating a test, these connections are tested to make sure the simulation results are produced under the right circumstances, as the leading cause of HIL simulation malfunction found using this implementation is a bad connection. A major benefit of the simulator is the pause button: it is much easier for one operator to run these simulation cases when the simulator can be paused and unpaused at will. With all three HIL stations connected, each instance of XPlane is initialized with a pre-set scenario, placing the HilStar17f model at the Cal Poly EFR at an altitude of 130 meters, removing the need to take off and fly each simulator manually for each run. Once loaded in position, the simulator is paused while the following checks are performed:

1. The Tx to Rx connection is verified visually by observing that the Rx has a steady orange LED, signaling a healthy connection with the Tx to which it is bound.
2. The APM to APM connection is verified visually by toggling the mode switch on all three Txs to `Formation` mode, and observing that the XBee adapter boards flash both mounted LEDs for the Tx and Rx lines at a rate of $\approx$10Hz.
3. The APM to APM Mission Planner connection is verified by toggling the mode switch to various modes on the Tx and observing that the HUD displays the changes in flight mode correctly

4. The APM Mission Planner to XPlane connection can be verified by unpausing the simulator and flying the model manually, observing that the stick inputs correlate to the expected attitude changes for the model.
5. The HIL station to HIL station LAN connection can be verified visually by panning the camera around the model and ensuring that all three models are present in each HIL station visualization of the environment.

Once these checks have been performed, each simulation test point can be run without concern for connection functionality.

10.45.26 HIL Simulation Run Matrix

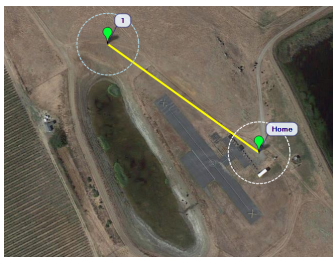
Several test points were conceived in order to test different aspects of the custom software functionality and observe the behavior of the algorithm in various situations and configurations. Table 10.1 lists each test as well as their objective.

Table 10.1: Matrix of simulation run cases and intended purpose

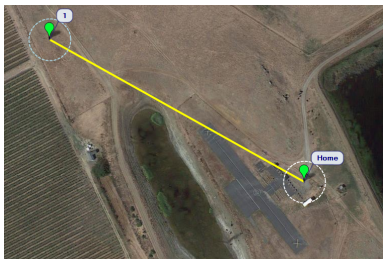
Simulation Run Cases		Leader-Follower Convergence	Flock Convergence	Swarm Organization	Collision Avoidance	Flock Recoverability	Leader Tracking
2 Member Simulations	*Different Start Locations*	X					
3 Member Simulations	*Different Start Locations*		X	X			X
	Leader Disappears		X			X	
	Manual Leader						X
	Collision Attempts				X	X	

10.45.27 Start Locations and Goal Location

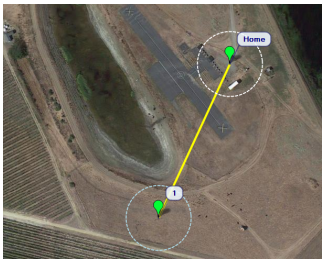
In the hope relating simulation results back to flight testing, a set of standardized start locations were developed to be used, both in HIL simulation and flight testing. Although the UAVs would never start in precisely the same location, these distinct start locations make each run case more repeatable and much easier to differentiate. In addition to standardized start locations, a standardized



(a) Start Location A



(b) Start Location B



(c) Start Location C

goal location was developed to provide repeatability to HIL simulations as well as provide a way to directly compare simulation and flight test results. This goal location and loiter radius, shown in Figure 10.5, is the same goal location and loiter radius used during flight tests.

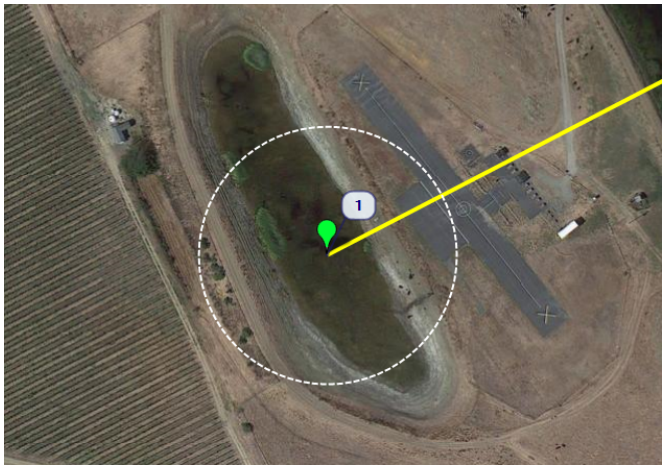


Figure 10.5: Goal location for leader loiter flight path

10.45.28 Algorithm Parameters

For the simulation run cases, inner-loop controller gains were tuned through trial and error, with variation in gain values between HIL stations, despite the use of an identical model. The reasons for different gain sets will be discussed in the next section, however it is worth noting here that they are not identical among all the HIL stations, or even among all the run cases.

Remaining constant, however, were a set of outer-loop gains and parameter values which had minimal dependency on the variations in HIL station performance. These gains and parameter values are summarized in table 10.2.

Table 10.2: Algorithm parameter values used during HIL simulation cases

Parameter	**Description**	**Value**
χ	Near field/Far field Threshold	30 meters
τ	Repulsive Sizing Parameter	50
σ	Repulsive Influence Parameter	8
$\lambda_{x,y,z}$	Attractive Weighting Parameters	.2
ΔX_{goal}	LNAV X-axis Goal Offset	10 meters
ΔY_{goal}	LNAV Y-axis Goal Offset	10 meters
Δ_{VWP}	VWP 2D Distance Offset	60 meters
Δ_{VWP_Z}	VWP Altitude Offset	2 meters
$Tr_{formation}$	Throttle Setting for Formation Leader	60%

10.46 HIL Simulation Results

The first HIL simulation run cases performed aimed to verify basic leader-follower convergence and global leadership determination based on start location. Figure 10.6 shows the convergence of Huey and Dewey in a leader-follower configuration, with Dewey assuming the global leadership goal. With Huey starting at location B, shown in Figure 10.4b, and Dewey starting at location A, shown in Figure 10.4a, global leadership determination was logical, and Figure 10.6 shows that Huey does indeed converge to the desired distance trailing Dewey. Something interesting to note is the convergence rate as Huey crosses the near-

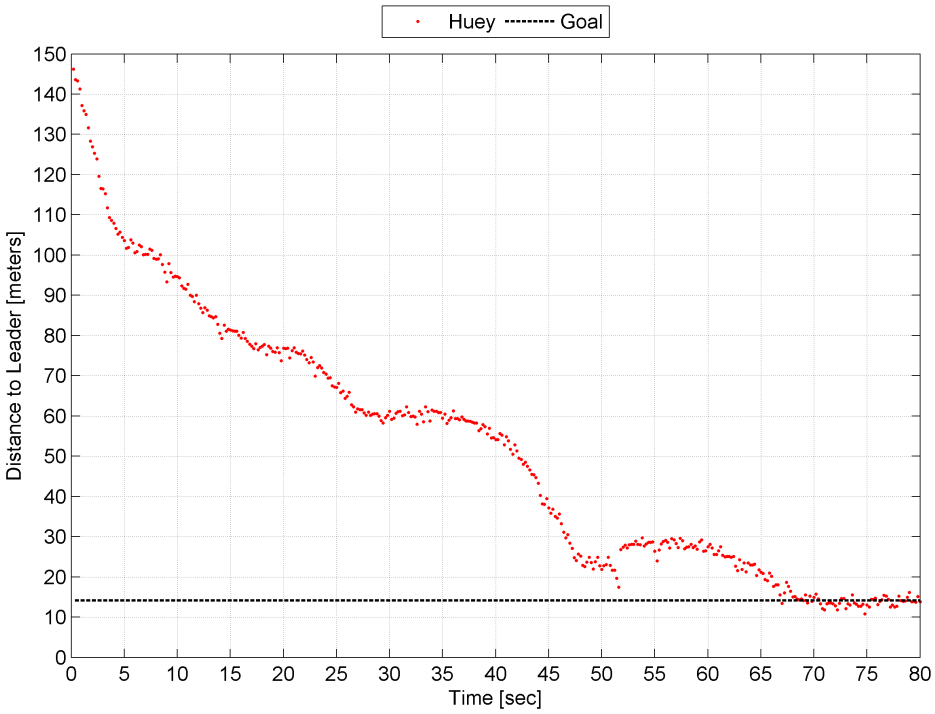


Figure 10.6: HIL Simulation: Convergence of Huey and Dewey in leader-follower formation, with Dewey leading

field/far-field threshold, χ, at 30 meters. At this point, the near-field control

laws dictate throttle commands in addition to heading. As this throttle control kicks in to gear, an initial divergence is caused, but ultimately Huey converges as a more controlled rate, avoiding an overshoot of the goal distance. While the short period of divergence is not necessarily desirable, it is better than a large overshoot of the goal which, during flight testing, could result in a disastrous collision.

The ground track for Huey and Dewey during the later stages of their convergence is shown in Figure 10.7.

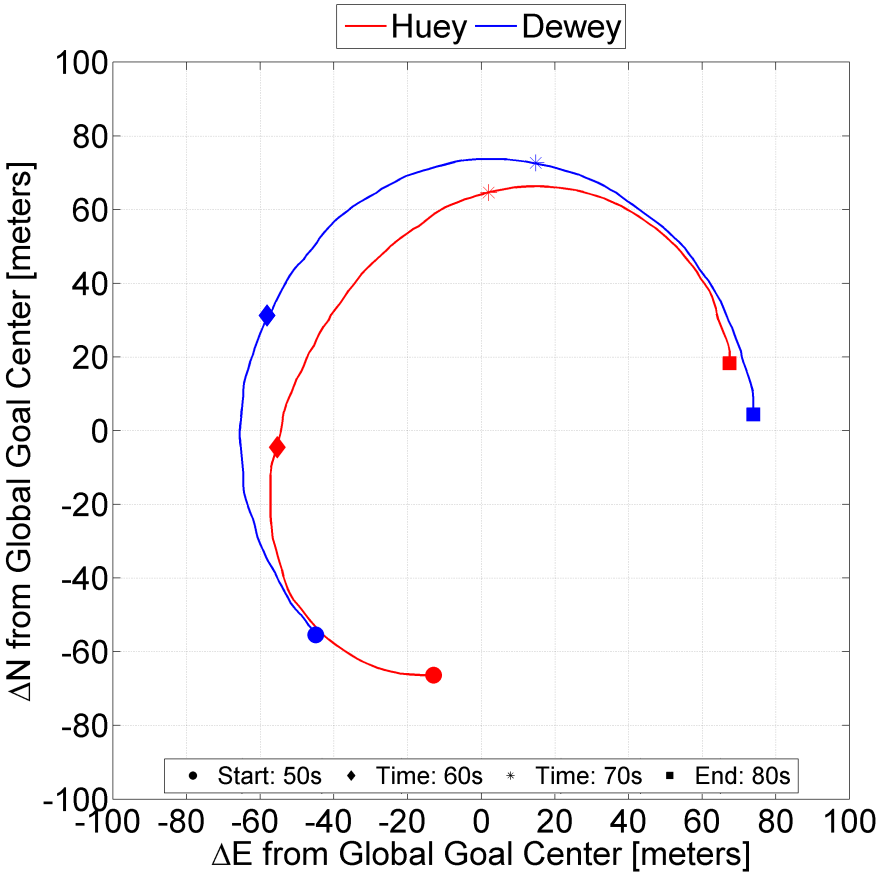


Figure 10.7: HIL Simulation: Ground track for Huey and Dewey in leader-follower formation, with Dewey leading

To provide a better concept of scale and relative distance, the ground track is plotted in a reference frame that is centered at the global goal loiter coordi-

nate, depicted in Figure 10.5. The axes of this reference frame are aligned with the North and East directions, with the listed values representing relative deviations, in meters, from global goal loiter coordinate in those directions. This convention is used to present the ground track data throughout the rest of this discussion instead of the traditional latitude and longitude coordinate values. Two-dimensional ground tracks were selected over three-dimensional trajectory paths in order to present the results in a more digestible manner, and with minimal deviations in altitude, very little value is lost in the reduction of the presented dimensionality.

While performing different variations of leader-follower simulations, an interesting trend was uncovered, which will help to explain some of the difficulties in obtaining perfect algorithm and controller performance in simulation, despite the luxury of idealized sensor inputs. Figure 10.8 shows convergence plots for the same class of simulation, leader-follower convergence verification, but configured such that the "follower" for each of the represented simulations is supported by a different HIL station PC. While there are a number of variables which could affect the overall convergence for two separate simulation runs, it is difficult to account for the offsets such as the one shown in Figure 10.8 without considering the influence of the HIL station itself. The most probable cause for the poor performance is controller tuning. In theory, if two identical control systems are used to control two identical models in relatively identical simulated environments, the controller gain set for the two systems should also be identical. However, it was discovered during the course of this project that HIL latency has a significant impact on the controller performance, and that this latency not only varies between HIL stations, but can also vary depending on uncontrolled computational load, affected by things such as scheduled virus scans or background applications

running during the simulation. These observations are corroborated by the observations noted in the HiLStar17f documentation[73]. Therefore, even controller gains tuned for a specific HIL station could exhibit poor performance somewhat sporadically, making it difficult to evaluate the simulation results in an absolute sense. A solution to this problem lies in a more robust HIL implementation scheme, which is not in the scope of this work, but is discussed in slightly more detail in section 13.57.

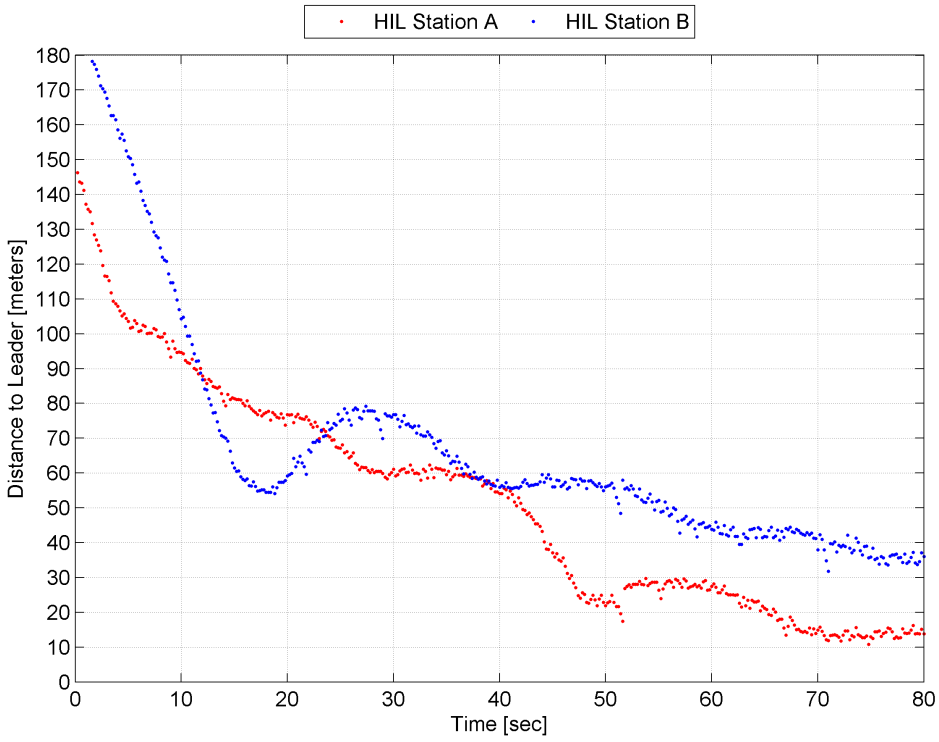


Figure 10.8: HIL Simulation: The effect of HIL station differences on convergence of Huey and Dewey in leader-follower formation

Moving beyond simple leader-follower formations, HIL simulations were conducted with all three formation members: Huey, Dewey, and Louie. Several simulation cases were run to evaluate the leadership determination logic and

total formation convergence when the members' start locations were varied. Figure 10.9 shows a convergence plot for a simulation run where Huey, Dewey, and Louie started at locations A, B, and C respectively. As expected, based on start location proximity to the global goal, Louie assumes global leadership, and Figure 10.9 shows the convergence data for Huey and Dewey. There are some interesting anomalies that can be seen in the figure that are worth highlighting.

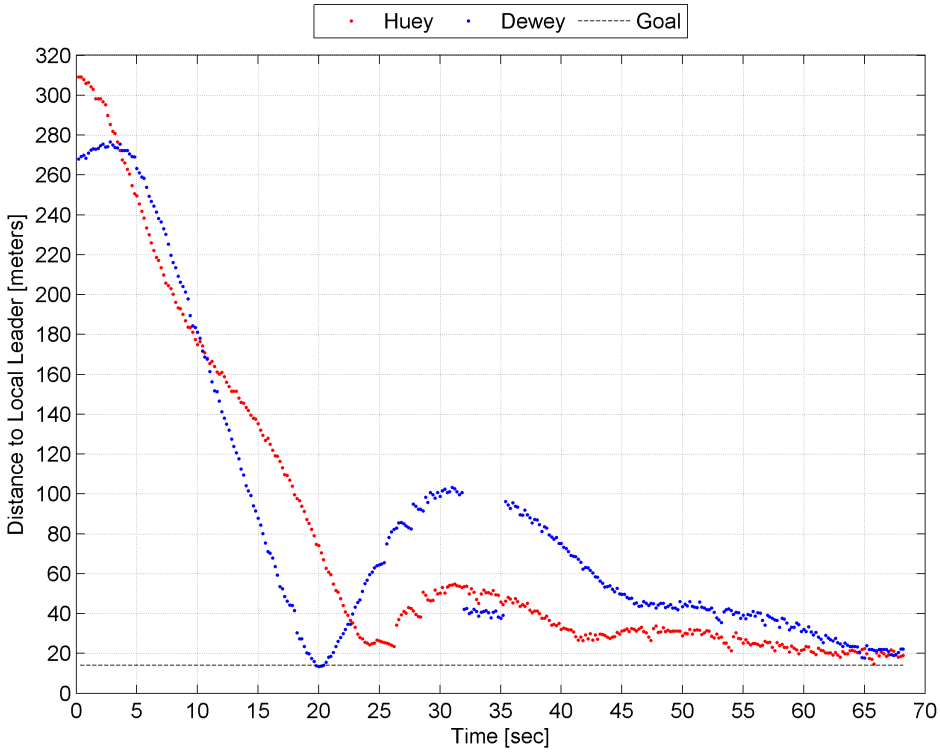


Figure 10.9: HIL Simulation: Formation convergence of Huey, Dewey, and Louie with leader-follower relationships dictated by the decentralized swarming algorithm

In order to enable dynamic leadership changes to support the event of losing one or more of the formation members, part of the swarm organization algorithm includes the ability to "forget" one's current local leader if they have not received any communication from them for a pre-determined amount of time. If this

occurs, a new leader is chosen based on a comparison of global reference values, and if there is no suitable leader, global leadership is assumed. Between the time frame of 33-66 seconds into the simulation run, the convergence data shows a discontinuous portion of relative distance values, which corresponds to a brief change of leader following from Louie to Huey and then back to Louie, supported by the local leader ID data logged by Dewey. The most logical explanation for this leadership change is a brief communication failure, more likely due to corrupted data or dropped packets than range issues, given the proximity of the XBee modules during HIL simulation runs.

Additionally, it is important to note the convergence performance gap between Dewey and Huey. Huey, which was supported on HIL Station A, consistently performs better than both Dewey and Louie during simulation runs, noted in Figure 10.9 by his ability to adjust to the near-field control laws better than Dewey once he crosses the relative distance threshold of 30 meters. HIL station A is actually a higher performance PC than station B or C's PC towers, which helps to explain why latency issues would be less problematic for Huey's control system than for Dewey's or Louie's.

While the plot in Figure 10.9 shows how each leader-follower pair converges independently, it does not provide an intuitive indication of what the formation distribution looks like throughout this process. The concept of formation "cohesion" is presented here, intended to measure how tight or loose the formation is at any one point in time. This formation cohesion value is determined by

$$C_F = \frac{\sum_{i=1}^{N} |\bar{X}_i - \bar{X}_{FC}|}{N} \tag{10.46.1}$$

Where $\bar{X}_{FC}$ is the flock center coordinate, and N is the number of formation members. This cohesion value, C_F, is really just an expression of the average

distance the members are from the centroid of the formation, or "flock center". For an ideal "V" formation with identical relative offset values of 10 meters, the cohesion value would be equal to 8.89 meters. This value is shown as the ideal reference value on Figure 10.10, which plots the formation cohesion value as a function of time for this three member simulation case.

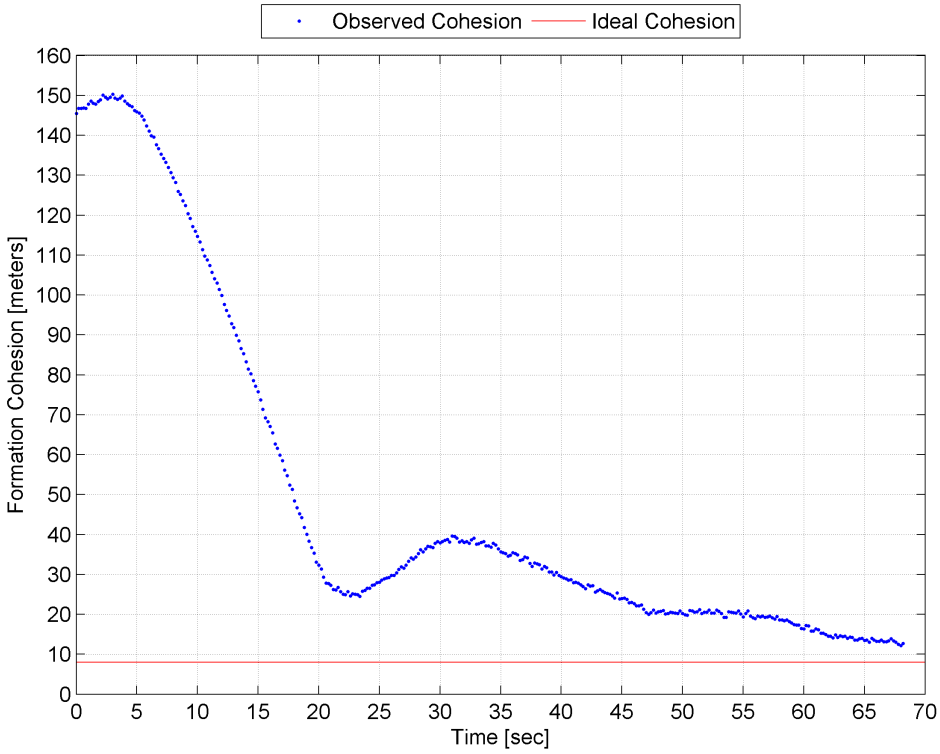


Figure 10.10: HIL Simulation: Formation cohesion of Huey, Dewey, and Louie

While the formation cohesion improves, as expected, as the members converge, it never reaches the ideal cohesion. This is due to less-than-desirable convergence of the following members, as well as the fact that converging to a V formation while the leader is maintaining a steady loiter is somewhat difficult, especially for the member attempting to converge towards the outer track in the loiter

formation.

The ground track for this simulation case is shown in Figure 10.11, again in a reference frame that is relative to the global goal center. This ground track plot further exemplifies the difficulty of reaching the outer track position relative to the leader's loiter path, which is what Dewey is attempting to do throughout the simulation.

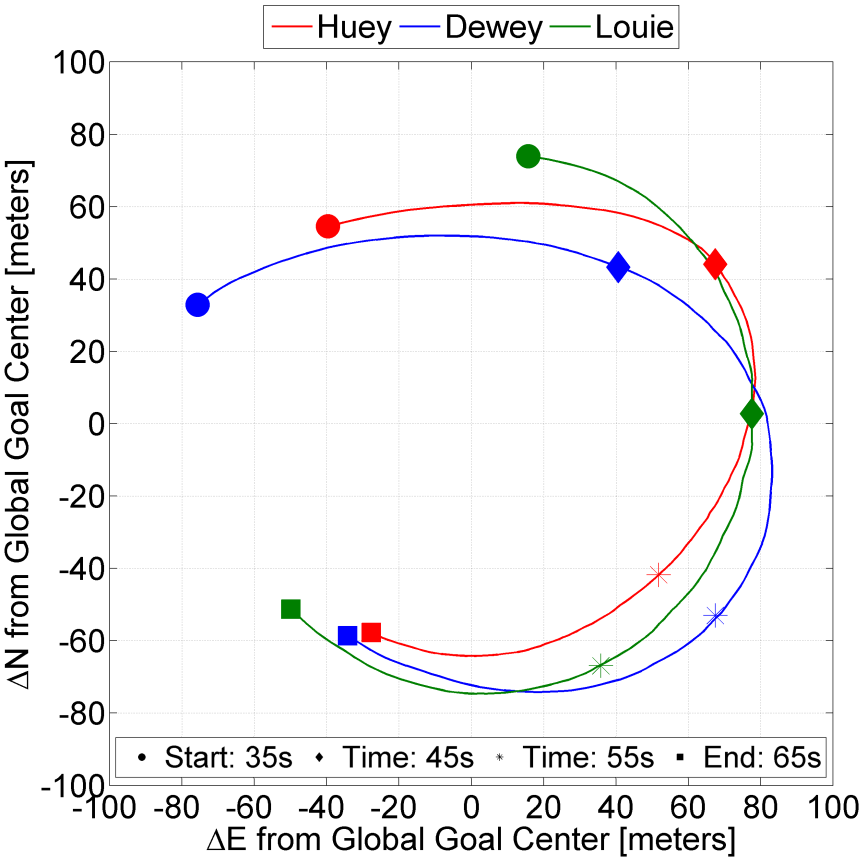


Figure 10.11: HIL Simulation: Ground Track for Huey, Dewey, and Louie three-member formation test

After some analysis, it was determined that this issue is caused by a combination of the constant heading changes necessary to follow a loiter path and the leader throttle setting. Admittedly, a redesign of the experiment could mitigate

some of these issues, such as a leader path that included longer straightaways. However, these problems were fully understood at a later stage in the project, and could not be corrected prior to the scheduled flight testing. Despite some of the unfortunate byproducts of the experimental design, Figure 10.11 does verify basic formation flight functionality, with the algorithms performing their desired duties, albeit constricted by the flawed experimental design. Leadership determination, based on relative positions and the global reference value system, functioned as expected, and multiple leader-follower pairs converged simultaneously. The leadership flexibility built into the control scheme also has an inherent susceptibility to communication failure, which was demonstrated briefly in this simulation.

This leadership flexibility was built into the formation flight control scheme for a reason: to take advantage of a decentralized control scheme's ability to repair itself. When a formation member legitimately disappears, for whatever reason, it is desired that the formation can repair itself, re-establish the global leadership role, and continue on towards the global goal. To test this functionality, a simulation run was conducted where communication was intentionally cut from the global formation leader. After a certain level of convergence, the global leader was switched from the standard `Formation` mode to the `Manual` mode, which does not allow the broadcast of state information to the other formation members, effectively causing the global leader to "disappear". The ground track of this simulation is plotted in Figure 10.12 and shows that the formation fully recovers from the loss of the global leader, who in this case is Dewey. Surprisingly, Louie assumes global leadership instead of Huey, who was actually closer to Dewey at the time of his departure from the global leader role, according to the simulation data. This behavior can be explained, however, by the details of

the "spooled" global goal concept, detailed in section 3.17. The global reference value, calculated by equation 3.17.1, is a combination of both proximity to the global goal center and integrated arc-length of the loiter path already followed by the formation member. Therefore, even though Huey was physically closer to Dewey, Louie had been following Dewey around the loiter path for a longer period of time, accumulating a greater sum of loiter distance traveled, and effectively placing him second-in-line for global leadership. This behavior was not necessarily anticipated or intended, but the simulation shows that the algorithm is robust enough to handle this oddity, recovering the formation, while only inconveniencing Huey by asking him to double back and fall in line behind Louie's global leadership.

To test the collision avoidance aspect of the guidance algorithm, an experiment was designed to force a near-collision between two formation members to see how they respond. A three member simulation was run, with Dewey assuming global leadership, followed by Huey and Louie. After a steady state was reached, Louie was toggled into `Manual in Formation` mode so that he could be manually guided on a collision course, but still seen by the other formation members, allowing them to react to his proximity. Figure 10.13 shows the ground track of this simulation after Louie was switched in to `Manual in Formation` mode, with the heading commands issued by Huey's PFG controller superimposed on to his ground track.

The 82s marker shows the first collision attempt, where Louie was flown to approach Huey directly from behind. As the plotted command vectors show, Huey makes minimal effort to avoid the collision, only generating a diverging command vector when Louie drifts to one side. This response actually makes sense taking in to account one of the assumptions of the PFG-based collision

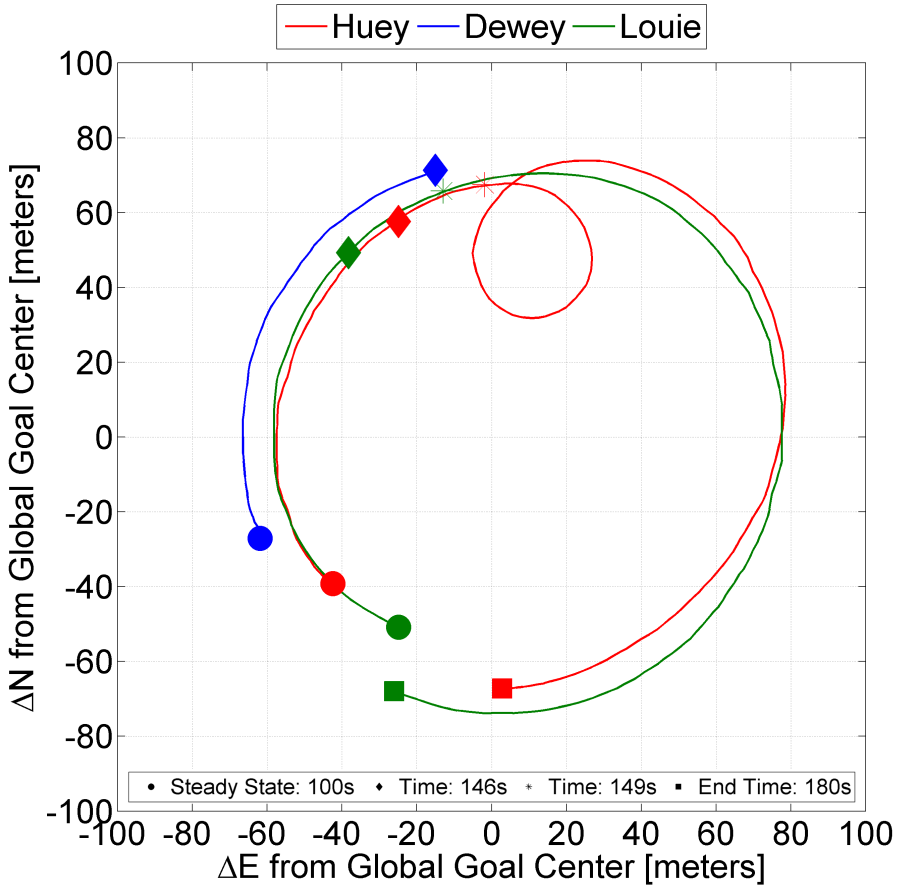


Figure 10.12: HIL Simulation: Ground Track for Huey, Dewey, and Louie during formation leadership recovery test

avoidance. For this algorithm, the burden of avoiding collision is primarily placed on the follower, not the leader. In the the far-field region, the PFG commands are only interpreted as heading commands, having no impact on the throttle setting, which is set to the maximum allowed while the formation member attempts to "catch up" to his local leader. While Louie is approaching from directly behind Huey, the heading command pointing towards the goal behind Dewey is simply made stronger, depriving Huey of any means to avoid Louie's approach. Because the assumption of collision avoidance being the trailing member's responsibility, this behavior is entirely acceptable. In the second collision attempt, noted on Figure 10.13 with the marker for Time = 88s, Louie cuts Huey off, having sped

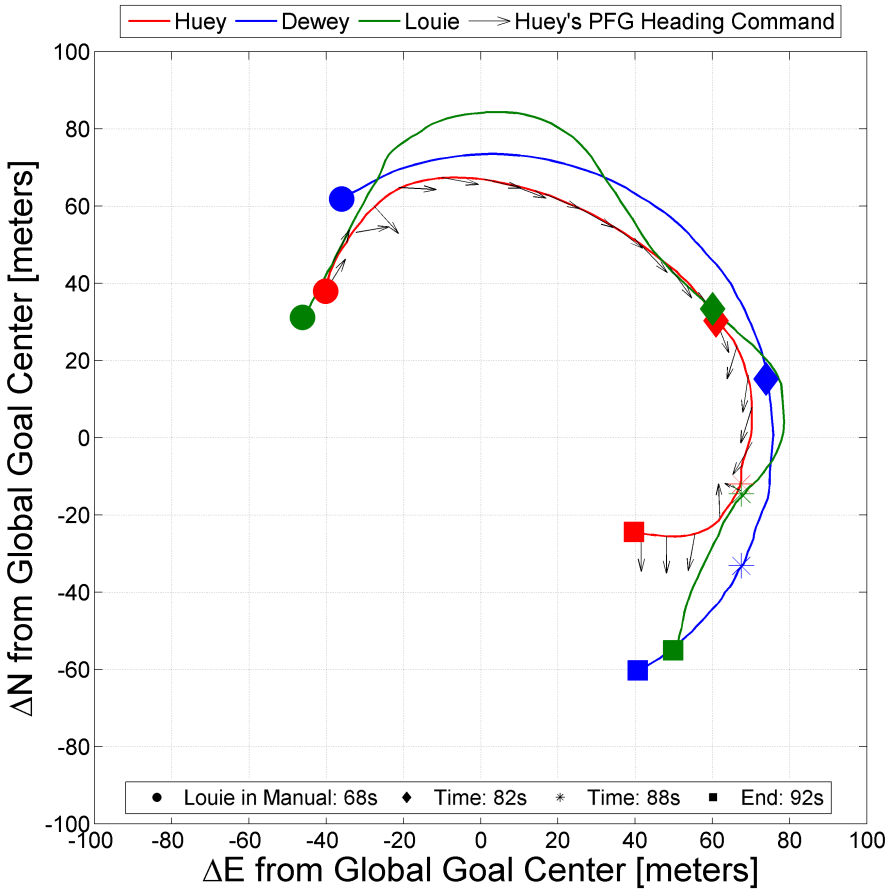


Figure 10.13: HIL Simulation: Ground Track for Huey, Dewey, and Louie during formation collision avoidance test

up and approached from the the side. Huey's commanded heading can be seen to change dramatically, going to the extent of commanding Huey to turn around when Louie is at the closest proximity. Once the collision is successfully avoided, the command vectors plotted in Figure 10.13 show that Huey is directed to rejoin the formation. This simulation shows that the collision avoidance algorithm does indeed command the formation members to avoid the other members in the event that they drift too close together.

10.47 Data Collection and Reduction Techniques

Data for HIL simulation is collected in the same manner as the data collected during flight testing: the data is logged on-board during the simulation, and downloaded later to be parsed and analyzed. The parsing and analysis process makes use of the FLOC APM Log File Parser GUI, shown in Figure 10.14, and is comprised of basic log parsing scripts, written in Matlab by two Cal Poly Graduate Students, Michael Darling and Adam Chase, as well modifications and additional Matlab scripts to support the needs of formation flight data analysis.

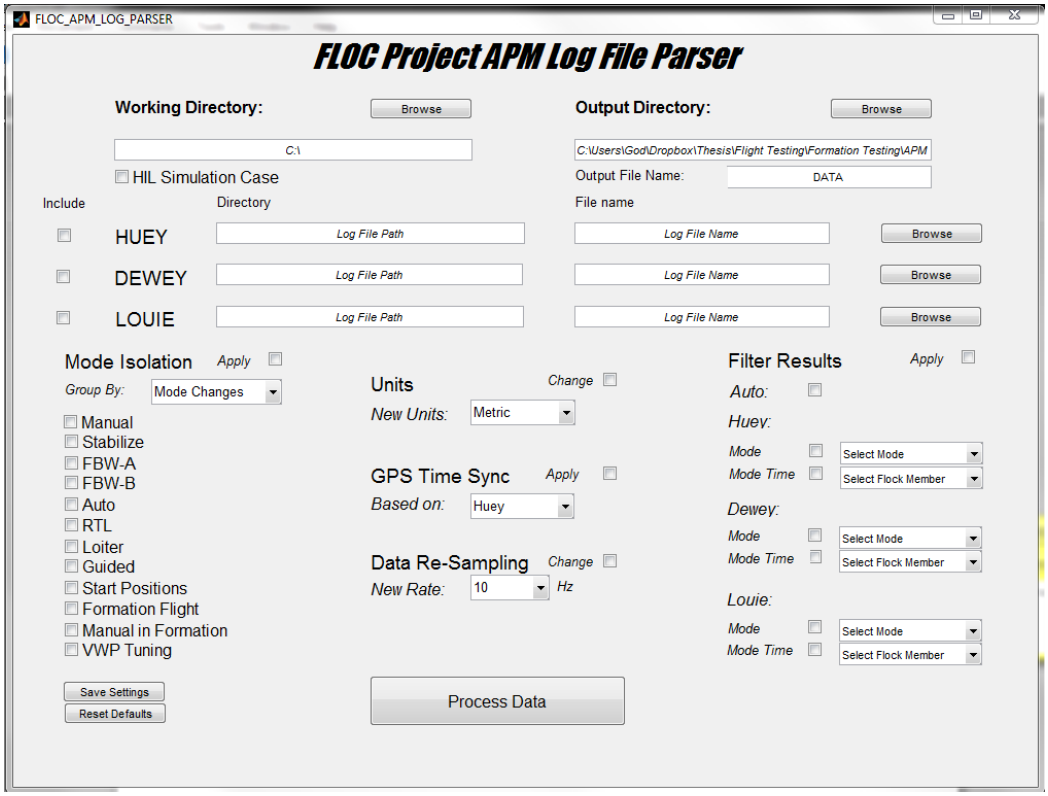


Figure 10.14: FLOC APM Log Parser Tool

The nature of the data contained in the log files made these data sets ideal candidates for conversion in to Matlab Timeseries objects, which provide built-in support for data re-sampling, synchronization, isolation, statistical analysis, and

more. During the process of simulation data analysis, three bugs were discovered which ultimately complicated the data synchronization and analysis process. 1)The GPS UTC timestamp, which is used as the primary means to synchronize data between the three UAVs, is improperly converted during transfer from X-Plane to the APM board. Fortunately, the issue was discovered to be a variable overflow problem, which can be corrected upon reading in the logged value. 2)The GPS UTC times are not fully synchronized between each HIL station running separate instances of X-Plane. Therefore, although visually in the same place at the same time, there is an offset in the time recorded in the logfiles. This problem caused a larger headache, requiring each data set to be manually adjusted so that the offset is eliminated. Thankfully, switching all three members into `Formation` mode at essentially the same time made these offset values easier to identify. 3)While running in real-time with the flight-ready firmware, the timing of the software loops is provided by the digital sensors, which sample data at 200Hz and trigger the 50Hz loop after every fourth sample. Unfortunately, during HIL, the emulated sensor data provided by X-Plane does not stream in at a constant rate, causing log file entries that are sampled irregularly. This irregular sample rate is corrected with the robust re-sampling method built into the Timeseries objects, using linear interpolation or zero-order holds, depending on the type of data, to re-sample the data so that is represented at the expected rates. With these corrections in place, the HIL simulation data very closely resembles the type and format of data collected during flight testing. For the purpose of analysis and plot generation, the HIL simulation data is re-sampled to provide all of the values at a rate equivalent to 5Hz. Accordingly, error is introduced into the presented data in the form of uncertainty attributed to the data corrections, filtering, reduction, and extrapolation techniques.

11 Flight Testing

Besides the challenge of working with hardware and embedded systems, which generally apply implementation limitations and can be prone to failure, simulation is often preferred over flight test demonstration in academia due to the amount of risk involved in these tests and the organization required for success.

11.48 Requirements

Flight test demonstration of not just one but multiple UAVs must have clear test requirements defined in order to achieve success. References [49] and [89] provide valuable insight and suggestions for flight testing requirements and procedures, which have been incorporated in to this work. Many of the requirements shown in Figure 11.1 are straight-forward. However, in some cases, a brief discussion will provide greater clarity with respect to their necessity.

In terms of hardware requirements, the need for data collection and in-flight monitoring is self-evident. However, as suggested by reference [49], data should be logged in two forms: On-board and through a telemetry link. This adds redundancy to the data collection process, ensuring data is protected against corruption or loss. Required support equipment is also listed under hardware, and could include a power supply for the ground control station and battery charger, as well as any equipment needed to take relevant measurements, such as a weight scale, multi-meter, etc. The software requirements are driven by a need for an interface to monitor multiple UAVs all at once and be able to modify parameters in flight for each UAV. Additionally, for flight tests which utilize the concept of "virtual leadership", a portable HIL simulator is required.

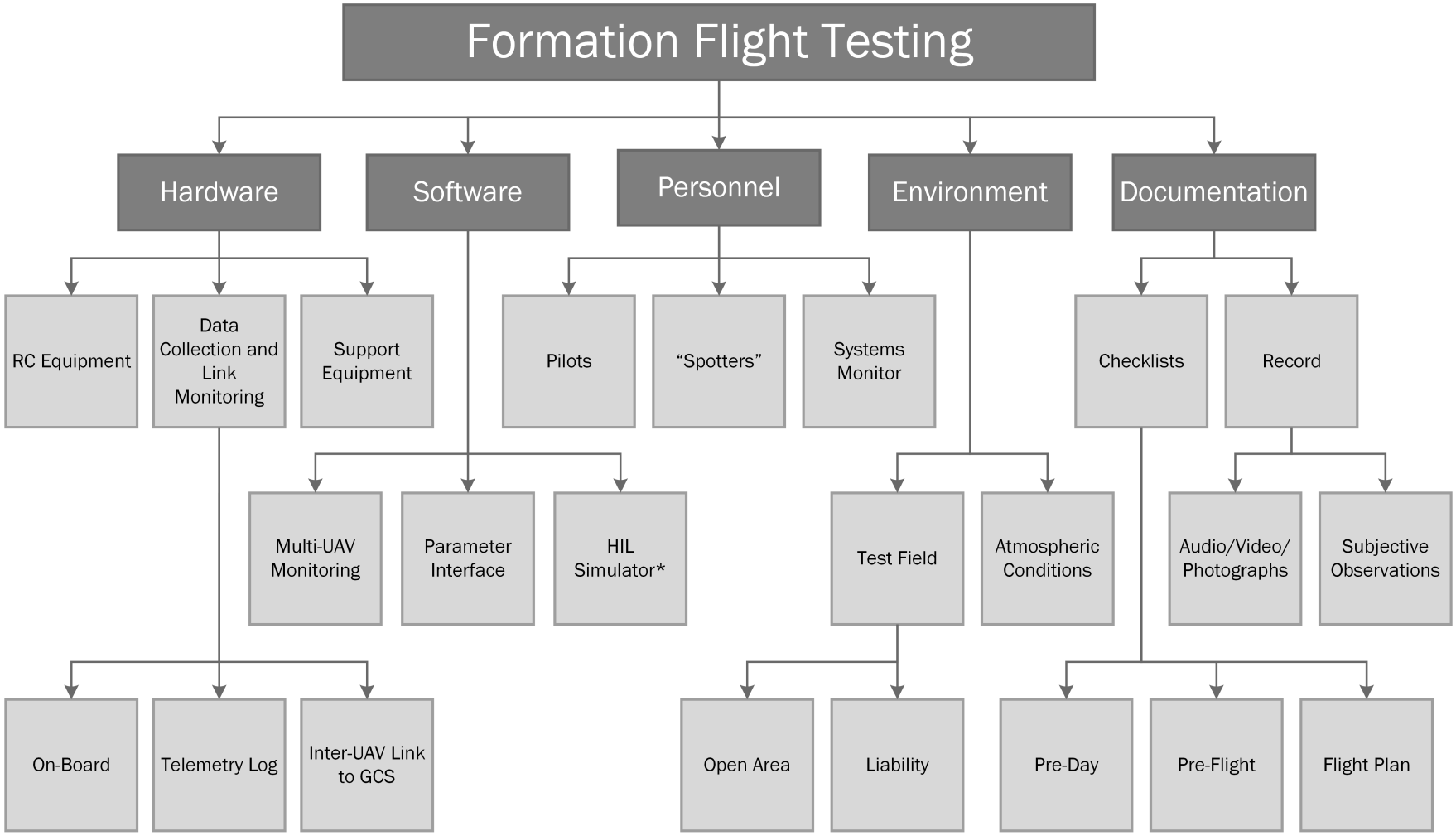


Figure 11.1: Flow-down for formation flight testing requirements

One of the most significant differences between flight test demonstration and simulation demonstration, from an operational standpoint, is the personnel requirement for flight testing. For testing N UAVs, a minimum of $2N + 1$ people are required to support the test: N pilots, N support members, and a test manager. To fully document the test with photos and video, more support personnel is required as well. Another concern that does not surface in simulation-based studies is the requirement for a suitable test environment, which includes test field location, openness, and liability coverage. Good weather is another important requirement due to the disproportionate effect that atmospheric disturbances have on small UAVs compared to larger aircraft.

Lastly, documentation requirements ensure the proper controls are in place, maximizing the opportunity for success. Other aspects of documentation include the observations, audio recordings, photographs, and video that gives some context to the analysis and presentation of results.

11.49 Test Stages

One of the recommendations implemented in this work from [89] is the organization of flight tests into a modular format, incrementally introducing new functionality, therefore minimizing the size of the risk matrix representing the combination of unknowns. It would not be beneficial to use an untested autopilot communication over an untested airframe. Therefore, flight tests were performed in stages. Table 11.1 shows the various stages of flight tests and the specific tests that make up that stage.

Airframe testing was performed to evaluate the quality of the kit construction as well as test the included RC components. Inner-loop and outer-loop controller

Table 11.1: Various stages of flight testing

Preliminary Testing	Sub-System Testing	Control Strategy Testing
Airframe	Telemetry and FCOM Network Range	Virtual Leader
Inner-Loop Gain Tuning	Inflight Power and Memory Load for APM	Leader-Follower
Outer-Loop Gain Tuning		Formation

tuning was accomplished with the standard autopilot software, ArduPlane v2.66, and the standard modes. The `Stabilize` and `FBW` modes were used for for inner-loop gain tuning, while a prescribed waypoint mission was used to tune the outer-loop controller gains. Figure 11.2 shows the ground-track of an outer-loop tuning mission.

Figure 11.2: The ground track of a mission flown to tune the outer-loop navigation PID gains

Sub-system tests were conducted to ensure functionality of individual components before integrating them all together. Communication functionality and range was tested for both the dedicated telemetry link as well as the FCOM net-

work. Autopilot hardware performance testing was introduced after learning a valuable lesson the hard way: through a crash. Power supply and memory usage were tested through an entire mission profile for the board without any actuators connected. This process helped to replicate the conditions of flight testing that may compromise the performance of the autopilot, but without putting the rest of the UAV at risk of a board failure-related crash.

Three different flight tests were developed to test the implemented control strategy, all with different objectives and different levels of risk. 1) The virtual leader test, while ultimately unsuccessful due to significant differences in modeled and actual performance, provided a way to test the control strategy with only one UAV in the air, removing the risk of collision. 2) Leader-follower flight tests allowed for the testing of just the convergence aspects of the formation flight control strategy without the added complexity of swarm organization logic. 3) Complete formation flight testing with three UAVs tested the control strategy from the highest level possible, adding in the final component of swarm logic, used to establish leadership, as well as peripheral formation member collision avoidance.

11.50 Formation Flight Test Set-up

Formation flight testing is performed at the Cal Poly EFR, which is ideal for small UAV testing as it is a large open space that is owned by Cal Poly, and already supports flying small-to-medium sized RC aircraft. A minimum of seven people are present during testing: three pilots, three "spotters" supporting the pilots, and the test manager. Typically, however, more volunteers are present, assisting with support tasks, checklists, photography and video.

The ground control station consists of a standard laptop PC with three 3DR Radio modules to support the dedicated telemetry links and an XBee module to support monitoring the FCOM network. On the ground control station laptop, three instances of APM Mission Planner v1.2.32 are run in parallel to support three independent telemetry links and provide HUD visualization as well as telemetry logging. Additionally, a modified version of QGroundControl v1.0.4, compiled from source with the custom MAVLink library included, is run simultaneously with the instances of APM Mission Planner, providing support for visualizing the locations of all three UAVs and monitoring the custom messages broadcast over the FCOM network. Power for the ground control station and battery charging station is provided by a standard car battery and AC inverter.

11.51 Standardized Checklists

In order to mitigate errors, standardize the process, and properly document the flight tests, various checklists are used to ensure proper hardware integration, software configuration, and pre-flight procedure.

Hardware integration checklists, complete with picture examples, provide a stream-lined method to ensure that all of the subsystems are properly connected and placed where they belong on the airframe. These checklists also serve as instructions for volunteers who are less familiar with the hardware. The most recent version of this checklist can be found in Appendix A.

Software configuration checklists help to verify that the proper firmware is loaded onto the hardware, with configurations set for flight testing. Because HIL simulation requires several changes to the code, this system helps to make sure the various settings are changed to their proper values for flight testing.

Additionally, hard-coded values, such as the global goal location and the home location should be verified. An example of the software configuration checklist used is included in Appendix A.

Lastly, pre-flight checklists are a common necessity for all aircraft, but have some clear distinctions when designed for a small UAV as opposed to a manned aircraft. The purpose of the pre-flight checklist is to verify that all of the subsystems function well together, but also to ensure that the aircraft is safe to fly, whether or not the autopilot is engaged. Furthermore, these checklists serve as documentation to track the exact configuration that is put up in the air, should that information be needed later for troubleshooting purposes. The pre-flight checklist used during the last formation flight attempt is included in Appendix A.

11.52 General Flight Procedures

Specific procedures were developed prior to each flight test, and examples of some of these detailed procedures can be found in Appendix A.

In general, formation flight testing proceeded in the following fashion:

1. All checklists should be completed and verified for each aircraft, ensuring proper configuration as well as healthy telemetry and FCOM links.

2. Each aircraft is launched by the "spotter", or support member, with the pilot flying the UAV manually until achieving sufficient altitude.

3. Pilots toggle the UAVs into `Start Positions` mode, which guides the UAVs autonomously to one of three pre-defined start position waypoints,

where the UAV will loiter until all of the formation members are in place at their respective start positions.

4. Once all formation members are in place, pilots are instructed to toggle the UAVs into `Formation` mode.

5. Pilots observe as the UAVs execute the formation flight control strategy, keeping a close eye on the aircraft in order to ensure there are no malfunctions in the software or sensors that could result in loss of the aircraft.

6. "Spotters" also help keep an eye on the UAVs and note observations about the control system performance.

7. Location, telemetry, and FCOM network communication for all three UAVs are monitored by the flight test manager from the ground control station, keeping a close eye on the command values produced from the PFG algorithm, as well as the swarm organization logic.

8. Once the test is completed, each UAV is landed manually, and the log files are retrieved from the on-board data storage.

12 Flight Test Results

Through the course of this project, four separate attempts were made to demonstrate the formation flight control strategy presented in this work through flight testing. Only in the fourth, final attempt was satisfactory leader-follower convergence achieved, and unfortunately, due to hardware malfunction during flight testing, no successful formation flight attempts were able to be demonstrated with all three UAVs. Table 12.1 summarizes the best results observed in each formation flight attempt along with the types of issues uncovered either during the flight testing or later through analysis of the data logs.

Table 12.1: Summary of results and observed issues for the flight test demonstration attempts

Attempt	Best Result	Functioning UAVs	Software Issues	Hardware Issues
1st	L-F Tracking	☑ Huey ☑ Dewey ☒ Louie	• Fundamental algorithm errors • Data type bugs	• GPS lock problems • FCOM saturation
2nd	Crash	☒ Huey ☒ Dewey ☒ Louie	• Unsustainable RAM requirement	• µC crash/reset
3rd	Limited L-F Convergence	☒ Huey ☒ Dewey ☑ Louie	• Parameter input file units bug	• None Observed
4th	L-F Convergence	☑ Huey ☑ Dewey ☒ Louie	• Mode recognition bug	• Persistent Xbee communication failures

12.53 Initial Demonstration Attempts

During the first formation flight attempt, fundamental errors in the algorithm implementation were identified, leading to drastic changes between the first and second iteration of the control system software. GPS lock problems also led to a change in the configuration for the UAVs, disabling the board-mounted GPS

module and replacing it with an external module, flashed with newer firmware, and mounted in a way that it would have greater exposure to the open sky with less interference from other electrical components. The communication setup in the first iteration relied on a single peer-to-peer XBee network for both FCOM communications and telemetry, which ultimately saturated the network. This discovery led to the use of a dedicated telemetry link in future iterations. Despite these issues, leader-follower tracking was observed. Louie was incapacitated by an inability to achieve a good GPS lock for himself, but Huey and Dewey managed to track eachother's movements, albeit with an altitude offset caused by a bad altitude measurement during the initial GPS lock. Figure 12.1 shows a photograph of Dewey tracking Huey with the previously-mentioned altitude offset.

The problems uncovered during the first round of flight testing led to a second iteration of the software, which added a substantial amount of code to make corrections to the algorithms and provide safeguards for more robust formation flight control. The control software was verified with HIL simulation, but a crucial detail was overlooked: firmware intended for HIL simulation does not include the sensor libraries when compiled, reducing the memory requirements of the software. However, when the flight-ready firmware is loaded, and these additional libraries are compiled and included, the RAM available is not enough to meet the requirements of the software. This results in a series of autopilot resets mid-flight, which unfortunately cuts off manual control from the pilot as the board re-initializes. Huey's crash within the first 10 minutes of the 2nd planned series of flight tests inspired another iteration of software, with greater attention payed towards memory allocation and the requirements of the firmware when compiled for flight.

Figure 12.1: Photograph of Huey and Dewey in a leader-follower formation with Dewey tracking Huey

The third attempt, intended to be the last for this project, was performed with firmware verified extensively through HIL simulation, and pre-tested under flight test conditions before connecting the airframe actuators to the autopilot output. Additionally, standardized checklists were developed to mitigate hardware issues, and provide the best opportunity for success. Ultimately, flight testing did not proceed as expected, with Huey and Dewey seemingly ignoring the navigation commands, which were verified from the ground control station, for all fully-autonomous modes, including the formation flight mode. Leader-follower convergence was briefly achieved, with Louie, the only UAV cooperating in fully autonomous formation flight mode, following Huey, who was flown manually, but still broadcasting his state. Figure 12.2 shows Huey and Louie's leader-follower

formation during the process of convergence with no unexpected offsets like the altitude offset observed during the first demonstration attempt. During the anal-

Figure 12.2: Photograph of Huey and Louie in a leader-follower formation with Louie tracking Huey

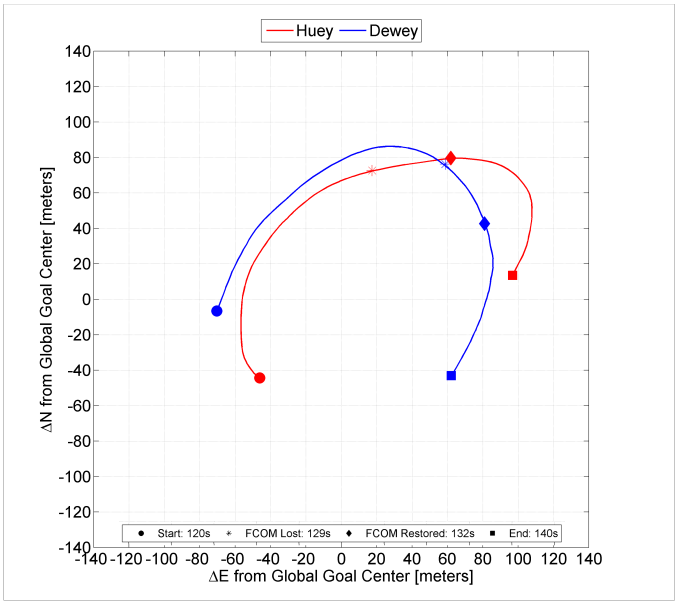
ysis of the data intended to be included in the final results for this work, a small bug was discovered in the input parameter files for both Huey and Dewey, but not Louie. Because separate input parameter files must be used for HIL and flight testing, the bug did not surface during the extensive HIL simulation run cases. The bug: a parameter which limits the commanded roll angle, meant to be set in units of centidegrees, was accidentally set in degrees. Accordingly, roll commands for both Huey and Dewey were limited to 0.3° instead of the intended 30° limitation, which explains why they seemed to "ignore" navigation commands, unable to bank and turn. Such a basic fix inspired a last-minute additional formation

flight attempt.

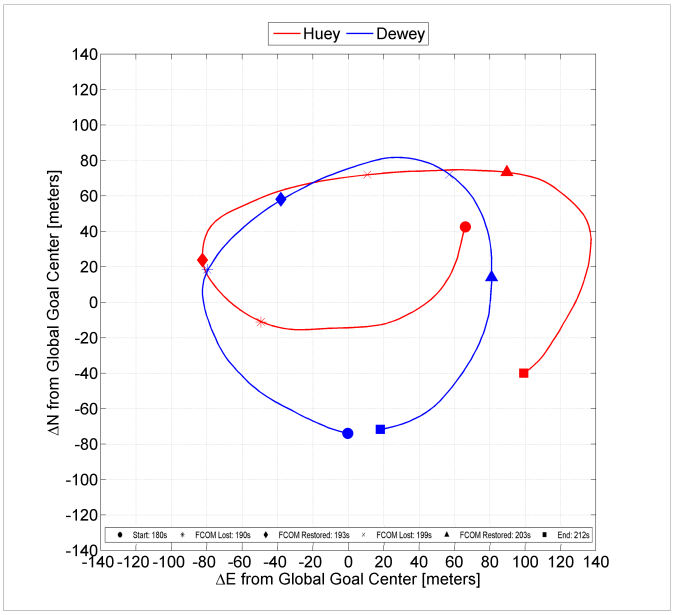
12.54 4th Demonstration Attempt

During the first flight of the fourth attempt, the FCOM network was not functioning for any of the UAVs in the air. After bringing all three down for troubleshooting, Huey and Dewey began to cooperate, but Louie's FCOM broadcast refused to engage, knocking him out of commission for formation flight testing because each UAV is reliant on a good FCOM network connection to receive the state information of the other UAVs in the formation. Even with the FCOM network functioning for Huey and Dewey on the ground, communication problems plagued the entire day of flight testing.

Figures 12.3a and 12.3b show how persistent communication losses prevented Huey and Dewey from achieving fully-autonomous leader-follower convergence. The formation flight specific data logged on each UAV was analyzed to find the cause behind the follower's inability to track the leader without completely overshooting the leader's loiter trajectory before banking to correct its heading. It was discovered that communication disruptions effectively froze the location of the leader as far as the follower was concerned, leading to a commanded heading directed towards where the goal behind the leader was at the time of the communication failure. These times were identified in the log files and mapped on to the ground tracks shown in figures 12.3a and 12.3b. These markers show that each overshoot did indeed corresponded with a loss in communication, evident by the follower's trajectory aimed almost perfectly at where the ground track shows the leader at the time of communication loss. Exacerbating the problem was poor design of the experiment, already noted in section 10.46. With the leader pursuing a loiter path, constantly changing their heading, there is minimal for-



(a)



(b)

Figure 12.3: Ground Track plots for fully-autonomous formation flight test attempt with Huey and Dewey in a leader-follower formation

giveness for issues such as poor controller tuning or the observed communication failures. If the leader had longer, uninterrupted straightaways, short disruptions in communication would not be so detrimental to leader-follower convergence.

The ground tracks presented in figures 12.3a and 12.3b and all the subsequent ground track plots presented in this section are plotted with respect to the global goal center instead of latitude and longitude, keeping with the convention established in section 10.46. With regard to error, these plots are generated with GPS data received at 5Hz and logged at 10Hz. Accordingly, the error associated with the ground track plotted in these figures and all of subsequent ground track plots in this section is the same error associated with the absolute GPS position accuracy, which is ± 3 meters. Additionally, the data used to identify the time when communication failures occur is logged at a rate of 1Hz, leaving room for some uncertainty in the precise position where communication failures occurred. During the last flight of the fourth attempt, a more forgiving leader flight path was executed by the global leader being flown in `Manual-in-Formation` mode. A racetrack-like flight path was flown with long straightaways, allowing for satisfactory leader-follower convergence, despite persistent communication disruptions. Figure 12.4 shows the leader-follower convergence of Huey and Dewey, with Dewey leading in `Manual-in-Formation` mode.

While never reaching a steady-state convergence value at the goal, it is clear that convergence is indeed occurring. The significant spike around the 70 second marker can, once again, be attributed to a communication failure that leads Huey astray during one of the turns. Convergence data is recorded locally, and is calculated using the magnitude of the relative distance vector of the follower with respect to its leader. Accordingly, the most probable error associated with this relative distance magnitude is derived from the errors associated with each rela-

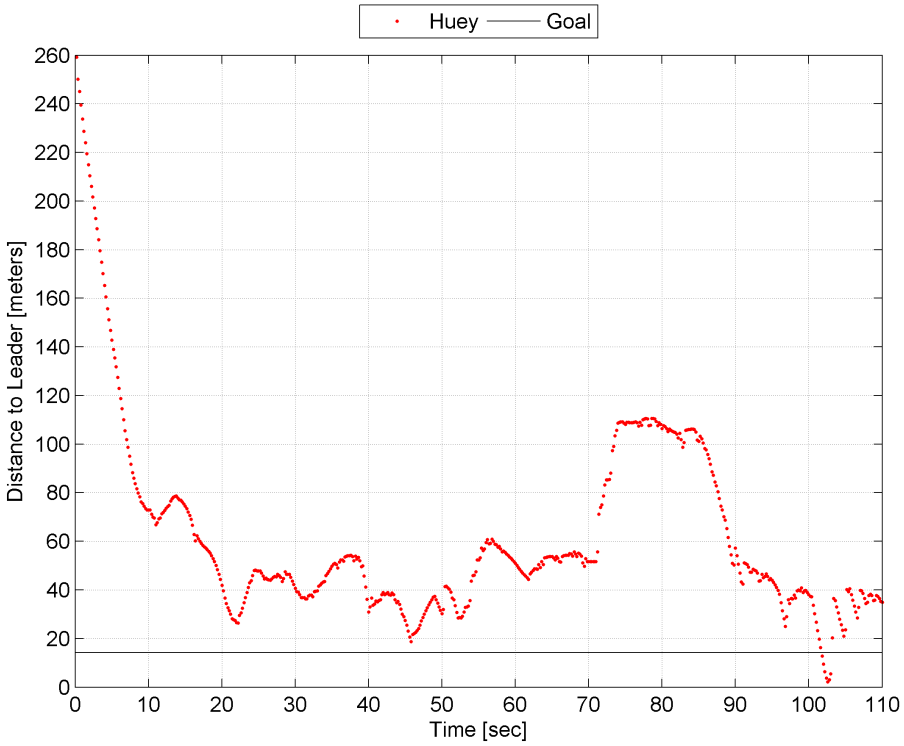


Figure 12.4: Convergence of Huey and Dewey in a leader-follower formation with Dewey in the lead, flown manually

tive distance measurement. The two dimensional relative distance error is based on the relative GPS error discussed in section 8. Including the error associated with altitude measurements, the most probable error of the relative distance values reported is ± 5.2 meters. This error is relevant to the values reported in Figure 12.4 as well as all of the subsequent convergence plots which report relative distance magnitudes.

Figure 12.5 shows the ground track for the majority of this flight, noting where there is evidence of communication loss 68 seconds in to the test and restoration of communication 5 seconds later. Something obvious from this ground track plot is that Huey weaves behind Dewey, a response typically observed when the

proportional roll gain is too high, causing excessive oscillation about a command. Had there been another opportunity to flight test after analyzing the data presented here, this gain could have been modified to determine if this was indeed a contributing factor.

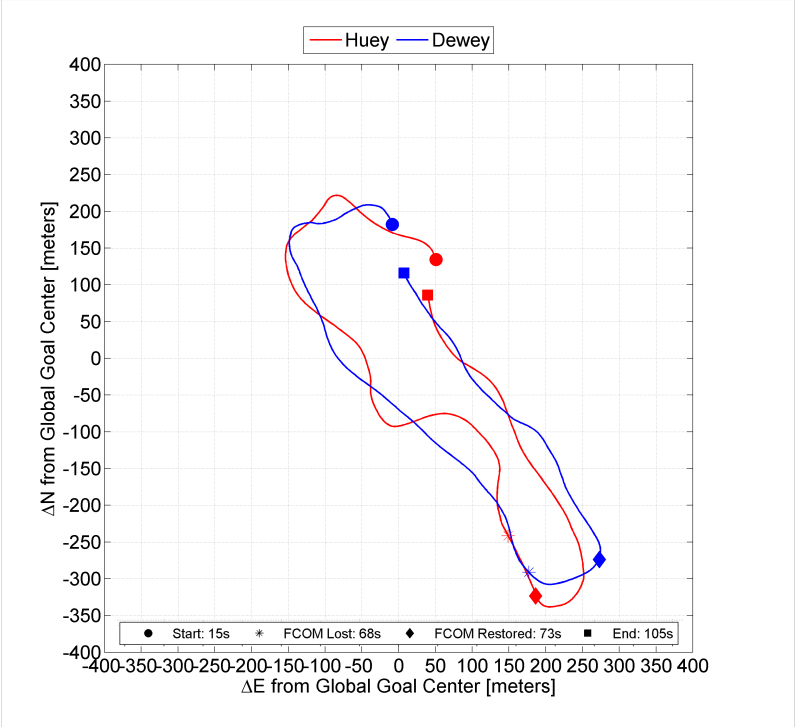


Figure 12.5: Ground track for Huey and Dewey in a leader-follower formation with Dewey in the lead, flown manually

From Figure 12.4, it is clear that there are two distinct regions of convergence during this flight, separated by a communication failure. The region with the best example of convergence, shown towards the end of the flight test, was isolated for further analysis and is presented in Figure 12.6. This figure shows that convergence does occur but fails to meet the stated goal of steady-state convergence of 15 meters $\pm$ 10 meters from the leader. There are several valid explanations for this failure to reach the desired goal, with most centering around

poor controller tuning and non-ideal test conditions. Despite not reaching the goal, this level of convergence was deemed satisfactory considering the obstacles overcome to achieve any form of convergence at all.

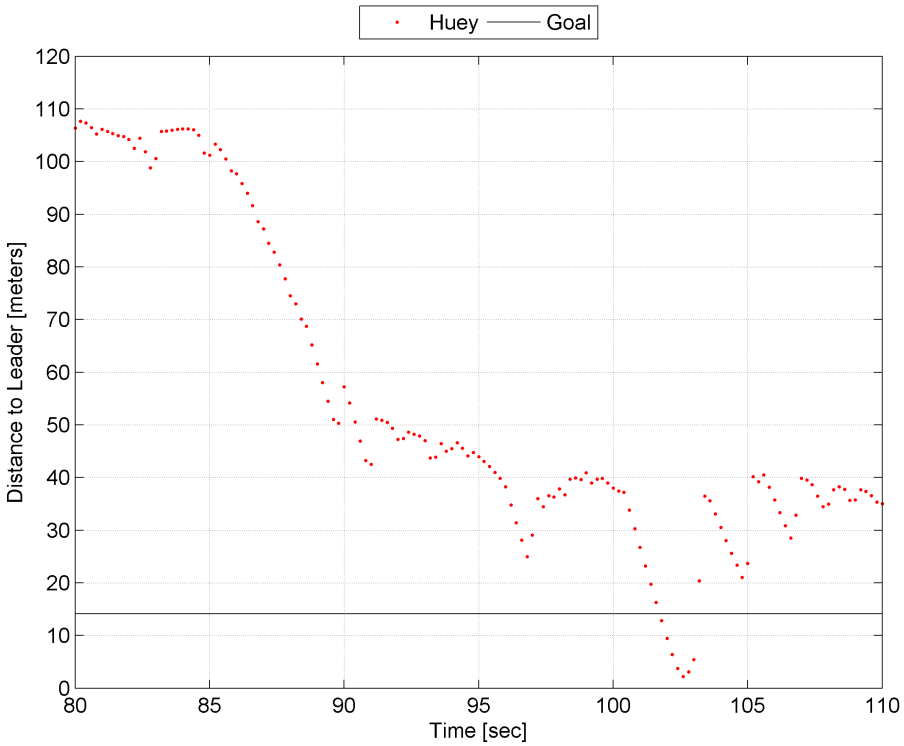


Figure 12.6: Best example of convergence for Huey and Dewey in a leader-follower formation with Dewey in the lead, flown manually

Figure 12.7 is the ground track for this isolated portion of leader-follower convergence, showing the weaving behavior of Huey as he follows Dewey.

Figure 12.8 shows a photograph of Huey following Dewey while Dewey is flown manually from the ground. Further exemplifying the various causes of the convergence of observed, Figure 12.9 breaks down the relative distances between Huey and Dewey by each axis of Dewey's LNAV reference frame. Figure 12.9 shows that convergence in the LNAV z-axis, or "down axis", is relatively good,

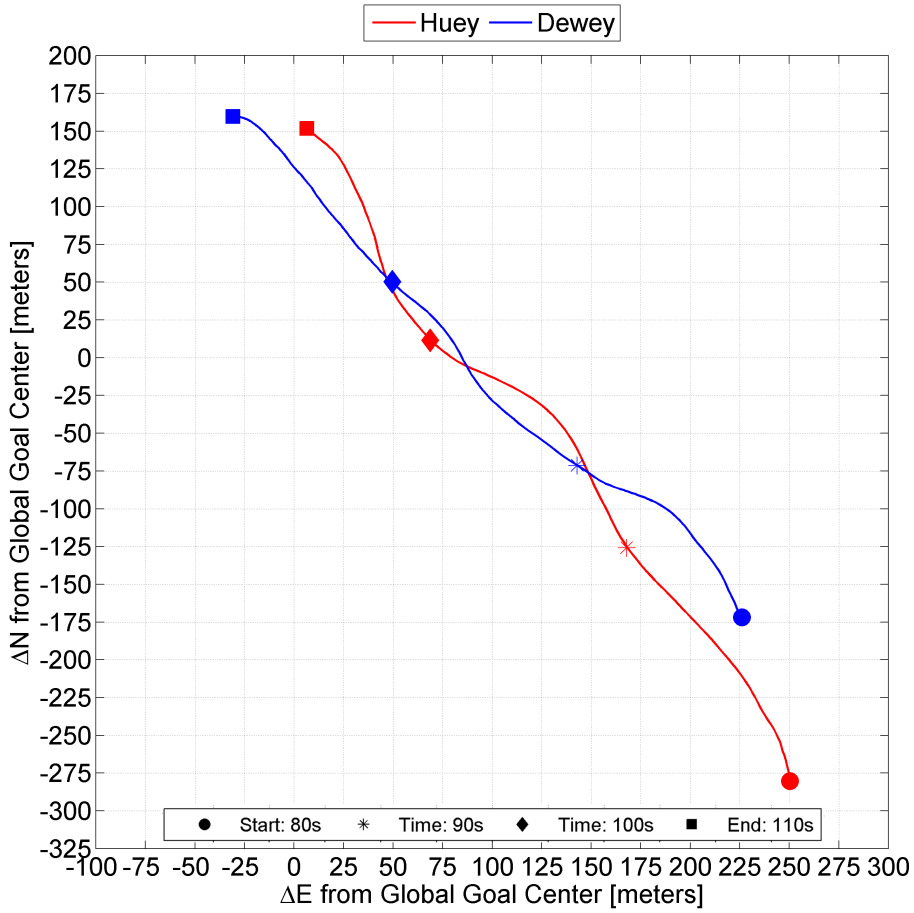


Figure 12.7: Ground track for the best example of convergence for Huey and Dewey in a leader-follower formation with Dewey in the lead, flown manually

and not a primary contributor to the high steady state convergence error. The changes in relative distance along the y-axis further support the evidence of a high proportional roll gain, showing that Huey oscillates significantly about the goal Δy offset value of 10 meters. Another telling aspect of Figure 12.9 is the relative distance values along the x-axis, which corresponds more with throttle control and the ability to match velocity. This axis shows the highest steady offset, signaling that Huey struggled to keep up with Dewey. A better performing throttle controller could possibly mitigate this offset. Another solution would

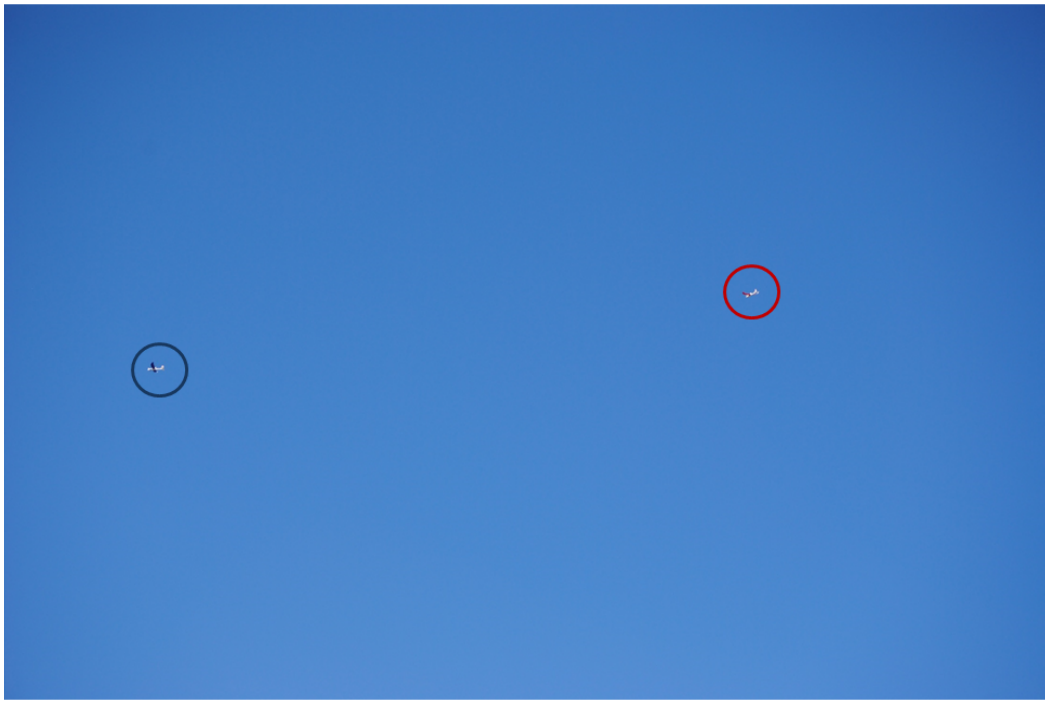


Figure 12.8: Photograph of Huey and Dewey in a leader-follower configuration, with Dewey leading while being flown manually from the ground

be to revert to a control scheme used more often in larger scale UAVs, where pitch commands are determined by velocity errors and throttle commands are determined by rate of climb error. For a UAV at this scale, this control scheme can result in higher oscillations holding altitude, but may also produce more precise Δx convergence. Rotations of relative distances into the LNAV reference frame utilize a heading angle that has its own associated error. Therefore, the error associated with the reported relative positions rotated into the LNAV frame is $\pm$ 5%, which is a relative error instead of absolute, as the relative distance error introduced by angular measurement uncertainty grows proportional to the relative distance itself.

Something worth noting, which is apparent in both figures 12.6 and Fig-

ure 12.9 is a brief moment around the 103 second marker, where convergence approaches zero very quickly, and then spikes significantly. This occurrence inadvertently provides the means to evaluate the functionallity of the PFG-based collision avoidance algorithm during flight, something that would have been very difficult to coordinate on purpose without risking a mid-air collision. As shown

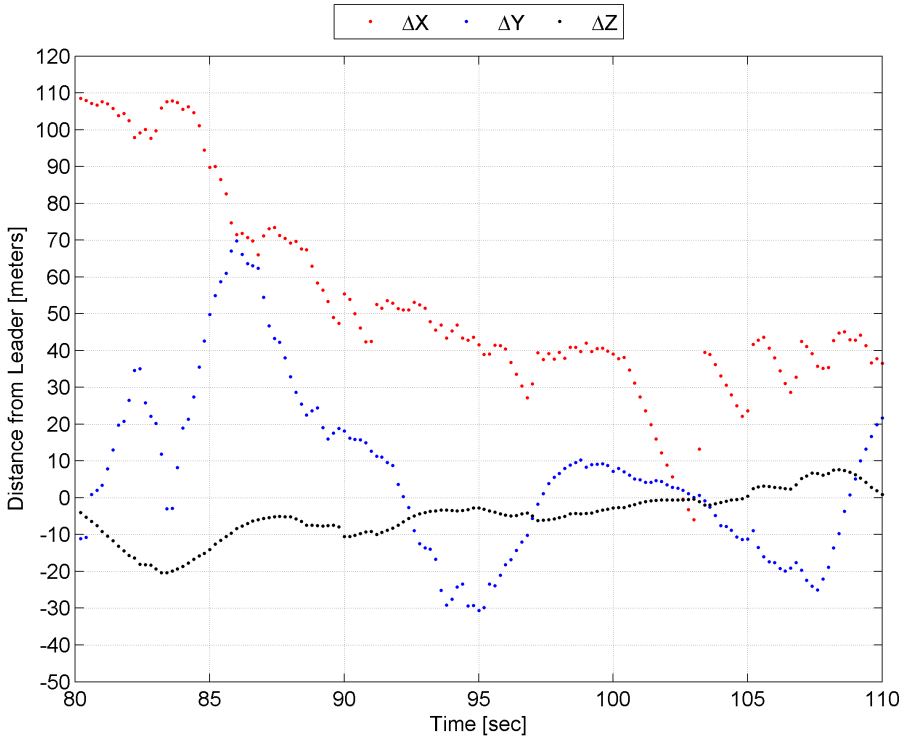


Figure 12.9: Best example of convergence for Huey and Dewey in a leader-follower formation, broken down by dimension in Dewey's LNAV reference frame

in Figure 12.10 the collision avoidance algorithm performs as designed, with the repulsive function kicking in to high gear as Huey quickly converges towards Dewey's position. Made more clear on Figure 12.9, this rapid convergence occurs mostly along the x-axis of Dewey's LNAV reference frame, but is coupled with the y-axis crossover as well, bringing the two UAVs much too close together.

Because there is rapid convergence along the x-axis, probable causes include air-speed changes made by Dewey or a brief tail-wind gust. A GPS position reading jump could have also occurred, however it would be a large coincidence to have the jump occur primarily along the x-axis of Dewey's LNAV reference frame. The

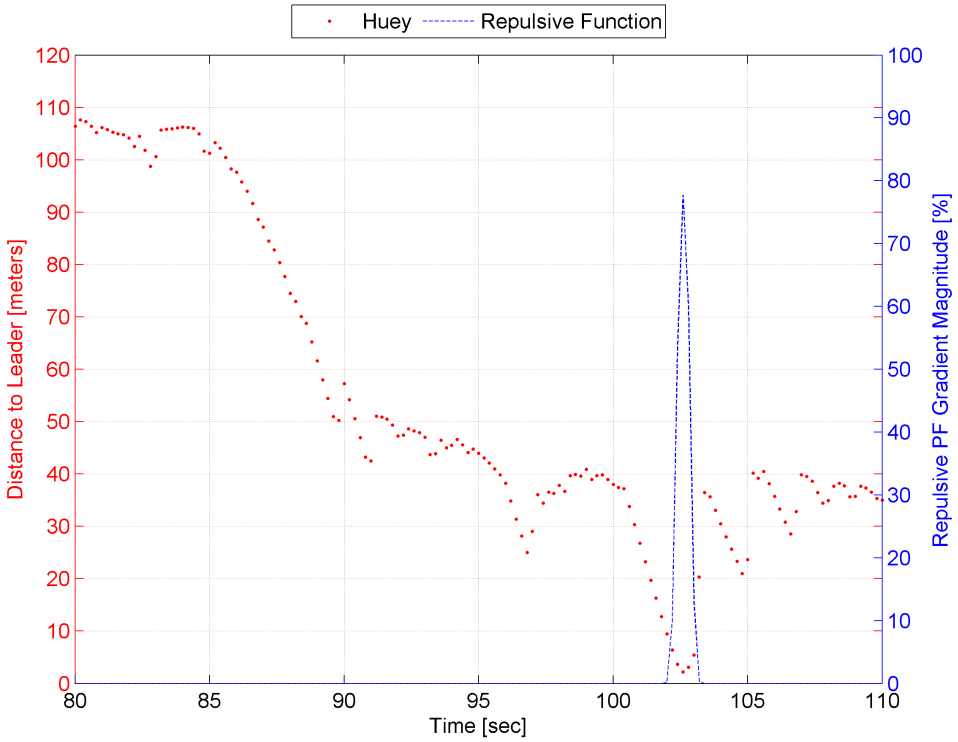


Figure 12.10: An example of the collision avoidance algorithm built in to PFG doing its job

communication problems mentioned earlier prevented meaningful demonstration of leader-follower convergence using the pre-defined loiter for the leader's flight path. By flying the leader manually in an exagerated racetrack pattern, longer straightaways made the follower less succeptable to communication losses, but also added another layer of complexity to the experiment. It is much more difficult for a human on the ground to fly the UAV along a steady flight path than it is for a moderately well-tuned autopilot to perform relatively simple altitude

and velocity hold control. Figure 12.11 shows that the leader does in fact have substantial velocity changes throughout the flight, which makes the followers job that much more difficult as he tries to track and converge on a leader that is not steady himself. While a robust controller could handle these fluctuations in speed,

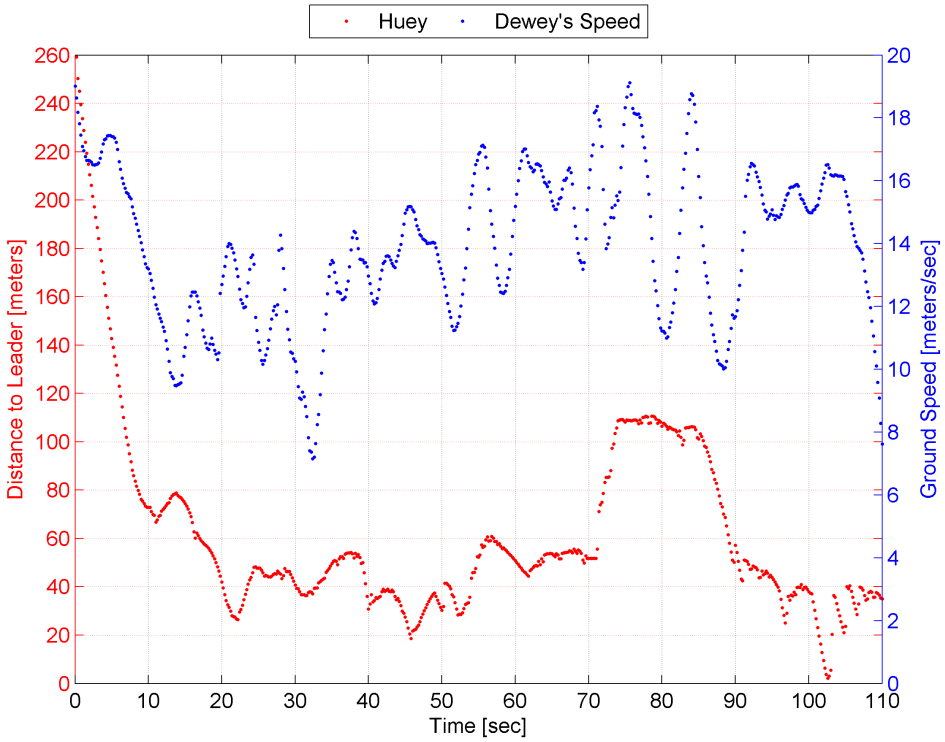


Figure 12.11: The relationship between leader velocity changes and follower convergence

these were not the conditions expected or tested in HIL simulation. Overlaying Dewey's velocity data on Huey's convergence data, it can be seen that larger fluctuations in velocity make it more difficult to converge. Notable exceptions are during turns, where dramatic changes in ground speed could be explained by wind directions, which are experienced by both the leader and follower, and therefore should not affect the convergence as substantially as velocity changes during the straightaway sections.

12.55 Data Collection and Reduction Techniques

The data collection process and reduction techniques used for the flight test data logs was similar to the process described in section 10.47, but without the many of the descrepencies regarding timing and sampling frequency that were found in the HIL simulation data logs. Having real GPS data meant that as long as there was a GPS lock, all the receivers had relatively identical UTC time stamps. These time stamps were used to synchronize different log types as well as synchronize the logs of different UAVs. As with the HIL simulation data, the FLOC APM Log Parser tool was utilized to parse the log files, convert the data to Matlab timeseries objects, and perform data reduction using linear interpolation functions to provide data with a uniform sampling rate for analysis and plotting. Error introduced through the data reduction process is the error associated with techniques such as linear interpolation and a zero-order hold. Data was preselected for reduction technique based on the type of value that was represented by the data. Continuous values, such as latitude, longitude, altitude, or velocity were all synchronized and reduced using linear interpolation. Discrete values and reference values, such satellite count, fix type, reference frame, or local leader ID were all synchronized and reduced with a zero order hold function, as a decimal value produced through linear interpolation would be meaningless. Data that was sampled at rates lower than the selected rate for data analysis and plotting was extrapolated using the same criteria for method selection.

13 Conclusion

Throughout this work, a formation flight control system architecture has been detailed with a design that builds off of previous work done at Cal Poly, using an implementation that utilizes a low cost, open source configuration, and demonstrating these systems through HIL simulation and flight testing.

The control system architecture adapts concepts presented in previous works to be integrated into the open source waypoint navigation-based autopilot software, ArduPlane, designed to run on the low cost APM 2 hardware. Swarm organization into leader-follower configurations is included with a PFG algorithm to provide commands to the Virtual Waypoint algorithm which interfaces with the supported waypoint navigation. Two new modes were introduced into the standard ArduPlane v2.66 software, supporting fully-autonomous formation flight and a manual flight mode designed to retain interaction with the rest of the formation. To make demonstration feasible in a flight test environment, the global goal was redefined to utilize a "spooled" goal concept, with the UAVs using distance traveled around the goal loiter to evaluate global leadership. Relative situational awareness is achieved through a GPS-based state-sharing method, shared over the inter-UAV FCOM network, which is enabled by XBee RF modules configured for peer-to-peer communication. The FCOM network makes use of the MAVLink protocol to send standard messages, as well as custom messages designed for compliance with MAVLink conventions. A HIL simulation setup using X-Plane 9 and the APM Mission Planner software was extrapolated to support multiple UAV HIL simulation, where the custom formation flight software could be verified. Several HIL simulation cases were presented in this work, verifying leader-follower convergence, three-UAV formation flight, dynamic leadership as-

signment and recovery, and collision avoidance. GPS testing was conducted to characterize the expected GPS error associated with relative distances calculated through the state-sharing method, providing guidance to establish a realistic error budget for the formation goal positions. Flight tests were conducted in multiple stages, starting with test-bed evaluation and ending with four separate attempts at formation flight demonstration.

While full three-UAV formation flight was not demonstrated through flight testing due to persistent hardware failures, several instances of leader-follower tracking and convergence were observed. In the fourth and final flight, steady convergence was observed with the follower UAV positioned at a relative distance between 30 and 40 meters $\pm$ 5.5 meters offset from the leader, an error between 15 and 25 meters with respect to the desired offset distance. This convergence was observed with the leader flown manually, instead of through using the pre-determined fully autonomous loiter flight path. Persistent communication failures plagued the final flight test, requiring the abandonment of the loiter-based global goal in favor of the more forgiving racetrack-like flight path flown by the leader UAV in manual control.

13.56 Lessons Learned

This work has provided intensive insight into the challenges of working in an interdisciplinary environment with hardware, software, simulation, and testing. While a significant amount of work was accomplished prior to this project, outlining a control scheme for formation flight, the jump from theoretical implementation to hardware implementation was not trivial. Even with some exposure to the nuances of hardware integration, understanding how to trouble-shoot a wide variety of problems that can arise due to poor power supply, individual compo-

nent failure, over-bearing memory requirements, or a number of other causes, proves to be a difficult endeavor. HIL simulation proved to be arguably the most important tool during this study, but developing an understanding of its limitations was equally as valuable. With the basic HIL implementation presented in this work, latency between the simulator and the autopilot control system accounts for significant performance losses, and renders the gains tuned for simulation relatively useless for flight testing purposes. A modular approach to flight testing is necessary to limit the scope of the risk matrix: insuring that untested hardware is not implemented with untested software on an untested airframe is crucial to managing the sources of errors that inevitably appear during testing. Standardized checklists help to mitigate many errors that can occur when attempting to prepare three separate UAVs for flight testing, especially with the help of others who are less familiar with the systems. More attention should be paid to performance constraints while designing the flight test experiments, such as reasonable throttle settings, loiter radii, maneuverability requirements, and available flight endurance. In general, a better understanding of the personnel required for flight test experiments should influence the size of group involved in a multiple UAV control study. Luckily, significant volunteer support was given to make this project possible, however future projects should include more team members with equal investment into the outcome.

A major lesson learned through the failures experienced through flight testing was that a robust control strategy, designed with redundancy, is priceless. A good example is the communication failures which in turn caused navigation failures, limiting the overall success of the flight testing. Having a persistent estimate of the aircraft state and its relative state with the other UAVs is necessary to mitigate the navigation failures caused by GPS and communication failures. Even

a poor estimate is better than the UAV's state effectively "freezing" during GPS or communication failure. While coupled GPS-INS solutions were investigated, they were evaluated from the perspective of increasing relative position accuracy, not from a standpoint of redundancy. When increasing the relative position accuracy was deemed outside of the scope for this work, the concept was soon neglected. However, even a loosely coupled GPS-INS solution would provide the type of state estimation that could have drastically improved the flight test results in the face of the failures experienced. If such an estimation process was extended to relative state estimation as well, through extended kalman filtering at best, or dead-reckoning at worst, the communication problems experienced would not have caused the navigation failures that derailed the loiter-based formation flight tests.

13.57 Future Work

This work can be characterized by saying that it covers a lot of ground, but does not focus intensely in any one particular direction. Much of the future work that could be inspired from this project would revolve around taking one aspect of this work and focusing on that concept with a greater level of depth.

Some obvious places to build off of this work include: enhanced modeling of the UAV through system identification, replication of the control system architecture in a higher-level simulation environment like Simulink, a higher quality HIL simulation environment that addresses the problematic latency associated with the current implementation, and the development of a robust relative state estimation technique to provide relative situational awareness.

Decentralized observation of the environment and other UAVs is a logical ex-

tension of this work, removing the communication dependency completely. Additionally, relative position accuracy increases could be investigated to enable the possibility of tighter formations. State estimation through inertial navigation and filtering could remove all dependency on GPS, further decentralizing the control scheme.

A lot of interesting studies could be derived from this work, moving away from the structured formation flight concept, focusing more on swarming techniques and loose cooperation for tasks such as optimized search and mapping. Additionally, more advanced control architectures could be investigated to move beyond the simple PID control architecture currently utilized.

Ultimately, this project introduced one example of bringing a multiple UAV control strategy from a theoretical implementation to a point where it can be demonstrated through flight testing. There are as many examples of routes to avoid as there are examples of promising directions taken. The hope is that this work will encourage a continuing legacy of multiple UAV control work at Cal Poly, focused more on realistic implementation than lofty theoretical solutions, which is what Cal Poly's "Learn By Doing" philosophy is all about.

BIBLIOGRAPHY

[1] Arduplane manual. Online: code.google.com/p/ardupilot-mega/wiki.

[2] Avl v3.32.

[3] Diy drones. Online: www.diydrones.com.

[4] Flightgear v2.10.

[5] Jsbsim v1.0.

[6] Mavlink micro air vehicle communication protocol. Online: http://qgroundcontrol.org/mavlink/start.

[7] Visual micro v1212.30.

[8] X-plane v9.

[9] Xflr5 v6.09.01.

[10] Xfoil.

[11] Mediatek - 3329 datasheet. Technical report, April 2010. Rev. A03.

[12] Atmega640/1280/1281/2560/2561. Technical report, ATMEL, 2012.

[13] Xbee/xbee-pro rf modules. Technical report, Digi, 2013.

[14] J. M. Abatti. Small power: the role of micro and small uavs in the future. Technical report, DTIC Document, 2005.

[15] F. Adachi. Wireless past and future–evolving mobile communications systems–. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 84(1):55–60, 2001.

[16] L. Allen. Evolution of flight simulation. In *AIAA Flight Simulation Technologies Conference, Monterey, CA*, pages 1–11, 1993.

[17] B. D. Anderson, B. Fidan, C. Yu, and D. Walle. Uav formation control: theory and application. In *Recent Advances in Learning and Control*, pages 15–33. Springer, 2008.

[18] K. Anderson. Rise of the drones: Unmanned systems and the future of war. 2010.

[19] R. Bamberger, D. Watson, D. Scheidt, and K. Moore. Flight demonstrations of unmanned aerial vehicle swarming concepts. *John Hopkins APL Technical Digest*, 27(1):41–55, 2006.

[20] L. Barnes, W. Alvis, M. Fields, K. Valavanis, and W. Moreno. Swarm formation control with potential fields formed by bivariate normal functions. In *Control and Automation, 2006. MED'06. 14th Mediterranean Conference on*, pages 1–7. IEEE, 2006.

[21] S. Bayraktar, G. E. Fainekos, and G. J. Pappas. Experimental cooperative control of fixed-wing unmanned aerial vehicles. In *Decision and Control, 2004. CDC. 43rd IEEE Conference on*, volume 4, pages 4292–4298. IEEE, 2004.

[22] R. W. Beard and T. W. McClain. *Small Unmanned Aircraft Theory and Practice*. Princeton University Press, 2012.

[23] R. W. Beard, T. W. McLain, D. B. Nelson, D. Kingston, and D. Johanson. Decentralized cooperative aerial surveillance using fixed-wing miniature uavs. *Proceedings of the IEEE*, 94(7):1306–1324, 2006.

[24] S. Berman, A. Halasz, V. Kumar, and S. Pratt. Bio-inspired group behaviors for the deployment of a swarm of robots to multiple destinations. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 2318 –2323, April 2007.

[25] G. A. Boyarko. System level simulation of artificial potential function guidance for a neutrally buoyant autonomous vehicle equipped with non-ideal actuators. In *47th AiAA Aerospace sciences meeting including the New Horizons Forum and Aerospace exposition, Orlando, Florida*, 2009.

[26] O. Cetin, I. Zagli, and G. Yilmaz. Establishing obstacle and collision free communication relay for uavs with artificial potential fields. *Journal of Intelligent & Robotic Systems*, 69(1-4):361–372, 2013.

[27] S.-M. Cheng, P. Lin, D.-W. Huang, and S.-R. Yang. A study on distributed/centralized scheduling for wireless mesh network. In *Proceedings of the 2006 international conference on Wireless communications and mobile computing*, pages 599–604. ACM, 2006.

[28] D. T. Cole, P. Thompson, A. H. Göktoğan, and S. Sukkarieh. System development and demonstration of a cooperative uav team for mapping and tracking. *The International Journal of Robotics Research*, 29(11):1371–1399, 2010.

[29] I. D. Couzin, J. Krause, R. James, G. D. Ruxton, and N. R. Franks. Collective memory and spatial sorting in animal groups. *Journal of theoretical biology*, 218(1):1–11, 2002.

[30] A. Deperrois. About stability analysis using xflr5, November 2010. Rev 2.1.

[31] A. R. Deri. " costless" war: American and pakistani reactions to the us drone war. *Intersect: The Stanford Journal of Science, Technology and Society*, 5, 2012.

[32] Y.-R. Ding, Y.-C. Liu, and F.-B. Hsiao. The application of extended kalman filtering to autonomous formation flight of small uav system. *International Journal of Intelligent Unmanned Systems*, 1(2):154–186, 2013.

[33] X. Dong, G. Cai, F. Lin, B. M. Chen, H. Lin, and T. H. Lee. Implementation of formation flight of multiple unmanned aerial vehicles. In *Control and Automation (ICCA), 2010 8th IEEE International Conference on*, pages 904–909. IEEE, 2010.

[34] J. Dorr, Less and A. Duquette. Fact sheet - unmanned aircraft systems. Memorandum, December 2010.

[35] J. L. Drury, L. Riek, and N. Rackliffe. A decomposition of uav-related situation awareness. In *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*, HRI '06, pages 88–94, New York, NY, USA, 2006. ACM.

[36] J. M. Fowler and R. D'Andrea. A formation flight experiment. *Control Systems, IEEE*, 23(5):35–43, 2003.

[37] B. Galibraith. Datcom+ pro v3.1. http://www.holycows.net/datcom/4901.html.

[38] F. Giulietti, L. Pollini, and M. Innocenti. Autonomous formation flight. *Control Systems, IEEE*, 20(6):34–44, 2000.

[39] Z. Gosiewski and L. Ambroziak. Formation flight control scheme for unmanned aerial vehicles. In *Robot Motion and Control 2011*, pages 331–340. Springer, 2012.

[40] W. Gracey. The experimental determination of the moments of inertia of airplanes bya simplified compound-pendulum method, 1948.

[41] Y. Gu, G. Campa, B. Seanor, S. Gururajan, and M. R. Napolitano. Autonomous formation flight–design and experiments. *Aerial Vehicles*, pages 235–258, 2009.

[42] C. E. Hanson, J. Ryan, M. J. Allen, and S. R. Jacobson. An overview of flight test results for a formation flight autopilot. 2002.

[43] A. Healey. Guidance laws, obstacle avoidance and artificial potential functions. *IEE CONTROL ENGINEERING SERIES*, 69:43, 2006.

[44] G. Healy. A bad week for drone lovers. 2013.

[45] P. Henry and H. Luo. Wifi: what's next? *Communications Magazine, IEEE*, 40(12):66 – 72, dec 2002.

[46] J. How, E. King, and Y. Kuwata. Flight demonstrations of cooperative control for uav teams. In *AIAA 3rd" Unmanned Unlimited" Technical Conference, Workshop and Exhibit*, volume 2012, page 9, 2004.

[47] J. P. How. Multi-vehicle flight experiments: Recent results and future directions. Technical report, DTIC Document, 2007.

[48] E. Johnson and S. Fontaine. Use of flight simulation to complement flight testing of low-cost uavs. In *AIAA Modeling and Simulation Technologies Conference and Exhibit*, pages 6–9, 2001.

[49] E. N. Johnson, D. P. Schrage, J. Prasad, and G. J. Vachtsevanos. Uav flight test programs at georgia tech. In *Proceedings of the AIAA Unmanned Unlimited Technical Conference, Workshop, and Exhibit*, 2004.

[50] P. Jones, G. Vachtsevanos, and L. Tang. Multi-unmanned aerial vehicle coverage planner for area surveillance missions. 2007.

[51] D. Jung and P. Tsiotras. Modeling and hardware-in-the-loop simulation for a small unmanned aerial vehicle. *AIAA*, 1:1–13, May 2007.

[52] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The international journal of robotics research*, 5(1):90–98, 1986.

[53] D. H. Kim, H. Wang, and S. Shin. Decentralized control of autonomous swarm systems using artificial potential functions: Analytical design guidelines. *Journal of Intelligent and Robotic Systems*, 45(4):369–394, 2006.

[54] S. Kim and Y. Kim. Three dimensional optimum controller for multiple uav formation flight using behavior-based decentralized approach. In *Control, Automation and Systems, 2007. ICCAS'07. International Conference on*, pages 1387–1392. IEEE, 2007.

[55] D. H. Lyon. A military perspective on small unmanned aerial vehicles. *Instrumentation & Measurement Magazine, IEEE*, 7(3):27–31, 2004.

[56] Z. Mahboubi, Z. Kolter, T. Wang, G. Bower, and A. Ng. Camera based localization for autonomous uav formation flight. 2011.

[57] Z. Mahboubi and T. Wang. Formation flight cs 229 project: Final report. 2009.

[58] M. Manuel. Development and implementation of potential function guidance and virtual waypoint algorithm for uav navigation and collision avoidance. 2013.

[59] P. A. McCarthy. Characterization of uav performance and development of a formation flight controller for multiple small uavs. Technical report, DTIC Document, 2006.

[60] T. W. McLain, P. R. Chandler, S. Rasmussen, and M. Pachter. Cooperative control of uav rendezvous. In *American Control Conference, 2001. Proceedings of the 2001*, volume 3, pages 2309–2314. IEEE, 2001.

[61] E. A. Mehiel and M. J. Balas. A rule-based algorithm that produces exponentially stable formations of autonomous agents. In *Proc. AIAA Guidance, Navigation, and Control Conf*, 2002.

[62] N. Michael, D. Mellinger, Q. Lindsey, and V. Kumar. The grasp multiple micro-uav testbed. *Robotics & Automation Magazine, IEEE*, 17(3):56–65, 2010.

[63] S. Misra and P. Agarwal. Bio-inspired group mobility model for mobile ad hoc networks based on bird-flocking behavior. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 16:437–450, 2012. 10.1007/s00500-011-0728-x.

[64] J. P. Molin. Gps horizontal position accuracy. February 2007.

[65] J. Moon, R. Sattigeri, J. Prasad, and A. J. Calise. Adaptive guidance and control for autonomous formation flight. In *American Helicopter Society 63rd Annual Forum*. Citeseer, 2007.

[66] T. J. Mueller and J. D. DeLaurier. Aerodynamics of small vehicles. *Annual Review of Fluid Mechanics*, 35(1):89–111, 2003.

[67] R. C. Nelson. *Flight stability and automatic control*. McGraw-Hill (Boston, Mass.), 1998.

[68] S.-M. Oh and E. N. Johnson. Relative motion estimation for vision-based formation flight using unscented kalman filter. In *Collection of Technical Papers-AIAA Guidance, Navigation, and Control Conference*, volume 5, pages 5365–5381, 2007.

[69] R. Olfati-Saber. Flocking for multi-agent dynamic systems: Algorithms and theory. *Automatic Control, IEEE Transactions on*, 51(3):401–420, 2006.

[70] C.-S. Park, M.-J. Tahk, and H. Bang. Multiple aerial vehicle formation using swarm intelligence. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, pages 11–14, 2003.

[71] H. V. D. Parunak, L. M. Purcell, F. C. S. SIX, N. A. Station, and M. R. OConnell. Digital pheromones for autonomous coordination of swarming uav's. *Ann Arbor*, 1001:48105–1579, 2002.

[72] T. Paul, T. R. Krogstad, and J. T. Gravdahl. Uav formation flight using 3d potential field. In *Control and Automation, 2008 16th Mediterranean Conference on*, pages 1240–1245. IEEE, 2008.

[73] M. Pursifull. Hilstar 17f developer's guide. Technical report.

[74] P. Raj. Aircraft design in the 21st century - implications for design methods. In *Meeting Paper Archive*, pages –. American Institute of Aeronautics and Astronautics, 1998.

[75] C. W. Reynolds. Flocks, herds and schools: A distributed behavioral model. In *ACM SIGGRAPH Computer Graphics*, volume 21, pages 25–34. ACM, 1987.

[76] J. Rolfe and K. Staples. *Flight simulation*, volume 1. Cambridge University Press, 1988.

[77] A. Ryan, M. Zennaro, A. Howell, R. Sengupta, and J. Hedrick. An overview of emerging results in cooperative uav control. In *Decision and Control, 2004. CDC. 43rd IEEE Conference on*, volume 1, pages 602 –607 Vol.1, dec. 2004.

[78] M. Sadraey and R. Colgren. Uav flight simulation: credibility of linear decoupled vs. nonlinear coupled equations of motion. 2005.

[79] J. Sfeir, M. Saad, and H. Saliah-Hassane. An improved artificial potential field approach to real-time mobile robot path planning in an unknown environment. In *Robotic and Sensors Environments (ROSE), 2011 IEEE International Symposium on*, pages 208–213. IEEE, 2011.

[80] S. Shimoda, Y. Kuroda, and K. Iagnemma. High-speed navigation of unmanned ground vehicles on uneven terrain using potential fields. *Robotica*, 25(4):409–424, 2007.

[81] B. L. Stevens and F. L. Lewis. *Aircraft control and simulation*. Wiley-Interscience, 2003.

[82] P. Sujit, D. Kingston, and R. Beard. Cooperative forest fire monitoring using multiple uavs. In *Decision and Control, 2007 46th IEEE Conference on*, pages 4875–4880. IEEE, 2007.

[83] M. Suzuki, K. Uchiyama, D. J. Bennet, and C. R. McInnes. Three-dimensional formation flying using bifurcating potential fields. In *AIAA Guidance, Navigation, and Control Conference*, 2009.

[84] S. Thakoor, J. M. Morookian, J. Chahl, B. Hine, and S. Zornetzer. Bees: Exploring mars with bioinspired technologies. *Computer*, 37(9):38–47, 2004.

[85] J. R. Thomas and S. Bhagat. zigbee-the latest buzz... *BSSS JOURNAL OF COMPUTER*, page 104, 2009.

[86] M. Tsuruta. An integrated formation flight algorithm via potential function guidance and biomimetics: A thesis. Master's thesis, California Polytechnic State University, 2008.

[87] J. Vaščák. Navigation of mobile robots using potential fields and computational intelligence means. *Acta Polytechnica Hungarica*, 4(1):63–74, 2007.

[88] S. A. Wallace. Development of a small and inexpensive terrain avoidance system for an umanned aerial vehicle via potential function guidance algorithm. Master's thesis, California Polytechnic State University, 2010.

[89] W. Williams, M. Harris, et al. The challenges of flight-testing unmanned air vehicles. In *Systems Engineering, Test & Evaluation Conference, Sydney, Australia*, 2002.

[90] D. XIANGXU. Development of sophisticated unmanned software systems and applications to uav formation. 2012.

[91] P. Yost. Rise of the drones. Television Broadcast, January 2013.

[92] P. Zipfel. *Modeling and simulation of aerospace vehicle dynamics*. Aiaa, 2000.

Appendix A: Checklists

Date: __/__/____

Hardware Integration Checklist

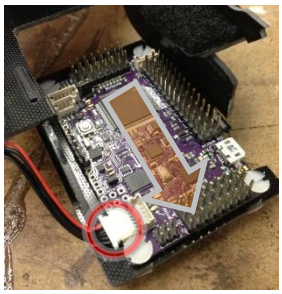
Flights: _____

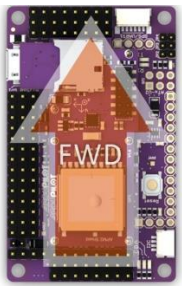
Huey / Dewey / Louie

Autopilot Installation

Autopilot Version: _____ **Additional Hardware:** _____

☐ External GPS installed

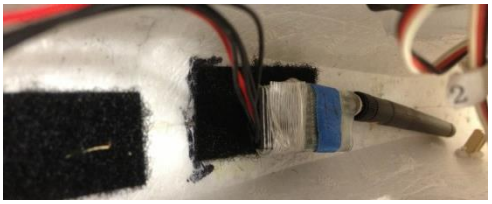




☐ APM 2.x secured to chassis w/ nylon screws

☐ APM is in the correct Orientation

☐ 3DR Radio mounted inside airframe



☐ LiPo Connected to BEC (3DR or Castle)

☐ LiPo Mounted inside Airframe

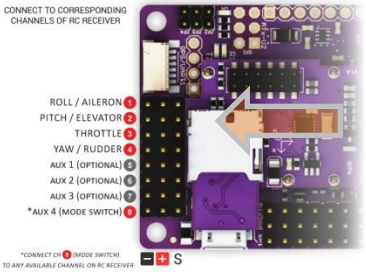
(Positioned with one edge lined up with the back of the cockpit opening on the Sky Surfer)

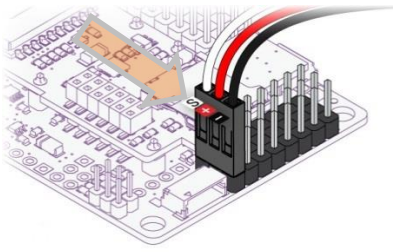
☐ Outputs connected correctly

☐ Inputs connected correctly

☐ **Verify** GND/Vcc/Serial connection for inputs and outputs

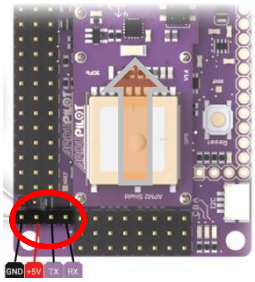
(Serial ALWAYS towards center of board)

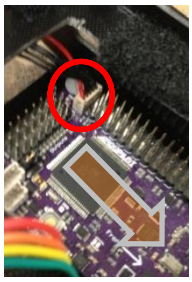




☐ 3DR Radio connected to UART0

(APM 2.0: Pins/ APM 2.5: Socket)



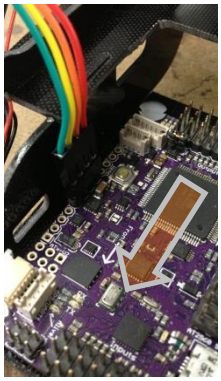


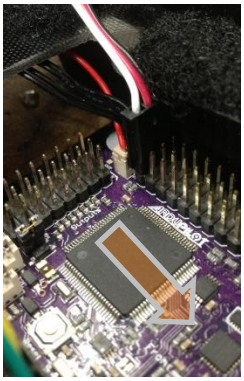
☐ XBee Radio connected to added UART2 pins

☐ **Verify** wire color order match picture

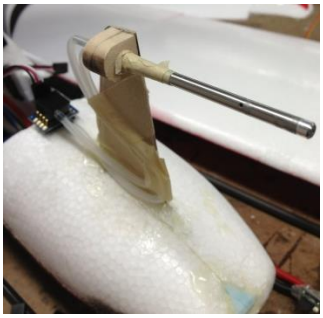
☐ Airspeed sensor connected to A0 pins

☐ **Verify** cable is not shifted over by one pin





☐ Airspeed sensor cable connected correctly to differential pressure sensor (see picture)



☐ Pitot Tube and Pressure Sensor secure

☐ Pitot hoses secure, no kinks

☐ All hardware is properly secured

Notes:

Date: __/__/____

Software Checklist

Flights: _____

Huey / Dewey / Louie

Software Configuration

Software Version: _____

☐ ”Formation Flight” **ENABLED**

☐ Telemetry out of UART2 **DISABLED**

☐ HIL mode **DISABLED**

☐ HIL debug configuration **DISABLED**

APM_Config.h

```
8  //Enable Formation Flight Firmware
9  # define FORMATION_FLIGHT    ENABLED
10
11 //Use the UART2 Pins on the APM Board for Telemetry
12 # define TELEMETRY_UART2     DISABLED
13
14 //Use HIL Firmware w/ Attitude Supplied
15 # define HIL_MODE            HIL_MODE_DISABLED
16 //Use the Serial1 Port for Debugging w/ Visual Micro
17 # define HIL_DEBUG           DISABLED
```

APM_Config_Formation.h

☐ Correct Airframe **UNCOMMENTED**

```
5  //Select which airframe you are using
6  //# define I_AM_HUEY
7  # define I_AM_DEWEY
8  //# define I_AM_LOUIE
```

☐ Inter-AC Coms using UART2 **ENABLED**

☐ Baud for FCOM set to **57600**

```
20 //INTER-AC COMMUNICATION NETWORK PARAMETERS
21 # define FCOM_UART2          ENABLED
22 //Baud rate for Inter-AC Coms
23 # define FCOM_BAUD           57600
```

Monitoring Inter-AC Coms with Ground Control Station Yes / No

☐ FCOM_GCS **ENABLED**

☐ **Verify** 64 bit Address for GCS XBee

```
24 //GCS link into the Inter-AC Coms
25 # define FCOM_GCS            ENABLED
26 //64-bit Address for QGCS XBee
27 # define QGCS_MSB            0x0013a200
28 # define QGCS_LSB            0x409a92ff
```

Logging Formation Flight Data Yes / No

☐ Formation Flight Data Log **ENABLED**

☐ Desired logs **ENABLED**

```
34 # define FF_LOGGING_ENABLED          ENABLED
35 # define LOG_FLOCK_STATUS            ENABLED
36 # define LOG_PF_FIELD                ENABLED
37 # define LOG_VWP                     ENABLED
38 # define LOG_REL_STATE               ENABLED
39 # define LOG_GPS_ERROR_ASSIST        ENABLED
```

☐ EFR “Home” Location **ENABLED**

☐ **Verify** GPS Coords/ Altitude for EFR

```
49  //HARDCODE HOME LOCATION AT EFR
50  # define HARDCODE_EFR_HOME ENABLED
51  # define HOME_ALT 6500 //cm
52  # define HOME_LAT 353284311 //1e7
53  # define HOME_LNG -1207524114 //1e7
```

☐ **Verify** GPS Coords/ Altitude for Goal

```
117 //Goal Waypoint to Loiter around
118 //CalPoly EFR - Arround Lake
119 # define    GOAL_LAT        353283260
120 # define    GOAL_LON        -1207533502
121 # define    GOAL_ALT        15000
```

☐ **Verify** Standard Flight Modes

```
66  //Pre-defined Flight Modes
67  # define FLIGHT_MODE_1      START_LOCATIONS
68  # define FLIGHT_MODE_2      FORMATION
69  # define FLIGHT_MODE_3      STABILIZE
70  # define FLIGHT_MODE_4      MANUAL_IN_FORMATION
71  # define FLIGHT_MODE_5      RTL
72  # define FLIGHT_MODE_6      MANUAL
```

☐ Verify PFG and VWP Parameters

```
76  ////////////////////////////////////
77  //PFG/VWP PARAMETERS
78  ////////    Parameter       |   Value
79  # define    PFG_CHI             30
80  # define    PFG_TAU             5000
81  # define    PFG_ZETA            800
82  # define    PFG_SIGMA           1
83  # define    PFG_X_LAMBDA        20
84  # define    PFG_Y_LAMBDA        20
85  # define    PFG_Z_LAMBDA        20
86  # define    PFG_x_OFFSET        1000
87  # define    PFG_y_OFFSET        1000
88  # define    OFFSET_ANGLE        13500
89  # define    PFG_z_OFFSET        0
90  # define    PFG_VWP_XY_OFFSET   60 //M
91  # define    PFG_VWP_Z_OFFSET    2 //me
92
93  # define    PFG_DEFAULT_SIDE    -1
94
95  # define    PFG_MIN_ALT         10000
96  # define    PFG_MAX_ALT         30000
97  # define    PFG_MIN_AIRSPEED_CM 800
98  # define    PFG_MAX_AIRSPEED_CM 2000
99
100 //Airspeed Controller Gains
101 # define    PFG_K_P_dV          100
102 # define    PFG_K_I_dV          100
103
104 # define    FORMATION_THROTTLE  60
105
```

☐ Software **Uploaded** to board

Date: __/__/____

Flight Test Log

Flight #: ___

Test: 3 Flock Member Leadership Changes **Crew:** _____

Configuration

Airframe: Huey I

Auto Pilot *Hardware:* _____ *Software:* _____

Additional Sensors/Electronics: _____

Tx: _____ **Mode1:** _____ **Mode4:** _____

Rx: _____ **Mode2:** _____ **Mode5:** _____

Mode3: N/A **Mode6:** MANUAL

Parameter File: _____

Start Location: Start Location A

Pre-Day

☐Batteries Charged ☐Software Checklist ☐Logs Cleared ☐Hardware Checklist

Pre-Flight

☐Battery #:	☐Verify CG Location:	☐Verify TOGW:
☐Visual Inspection of airframe for cracks in foam, mount separation, and tail alignment		
☐Verify control horns are securely mounted to control surfaces		
☐Verify wings are secured with rubber band		
☐Verify all components are secured to airframe		

Pre-Launch Procedures

1. Turn on Transmitter
2. Connect BEC power cable to AUX input on receiver (**Do not** plug flight battery into ESC yet)
3. Hold airframe level until APM is finished initializing (Changes from Blue/Red flashing to solid orange with slow flashing blue light)
4. In APM Mission Planner, connect Telemetry link

☐ All parameters are retrieved	☐ Attitude changes of airframe are shown in APM MP

5. In the APM MP configuration tab, load parameter file
6. Modify Mode 1 for “Stabilize” and then write parameter file
7. Ensuring that throttle is off, and Tx in set to **Manual Mode** (Gear switch is ALWAYS 1), connect flight battery to ESC, and listen for the motor arming.
8. Switch Tx into Stabilize Mode, which should be set to the Mode 1 slot (Gear = 0, Elev D/R = 0, Flap = 0)
9. Rotate airframe and verify that Stabilize mode is providing deflections for corrective behavior, reverse deflections in the APM MP configuration tab as necessary
10. Switch Tx back into Manual Mode, and verify manual deflections, reversing inputs on Tx as necessary
11. In APM MP’s configuration tab, change Mode 1 back to Auto (10) and write parameter
12. In APM MP’s Flight Planner tab, load start location WP file and write WPs
13. If linking ground station to the Inter-AC communication network, connect XBee to QGCS
14. Verify GPS lock and Communication Links

☐ GPS LED is solid blue	☐ XBee LEDs (Din and Dout) blinking at 1/2Hz (at least)
☐ APM Mission Planner shows GPS 3D Lock	***If QGCS is linked to Inter-AC communication:*** ☐ QGroundControl shows GPS Location of aircraft

15. Test throttle and verify prop is spinning in the correct direction
16. Note flight conditions:

17. Other notes:

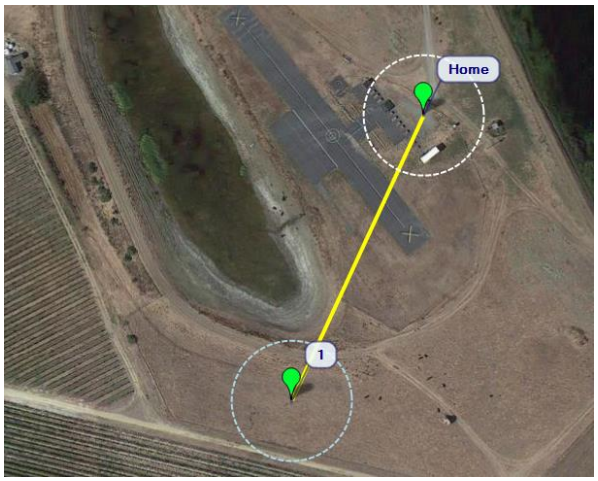
Date: __/__/____

Flight Test Log

Flight #: ___

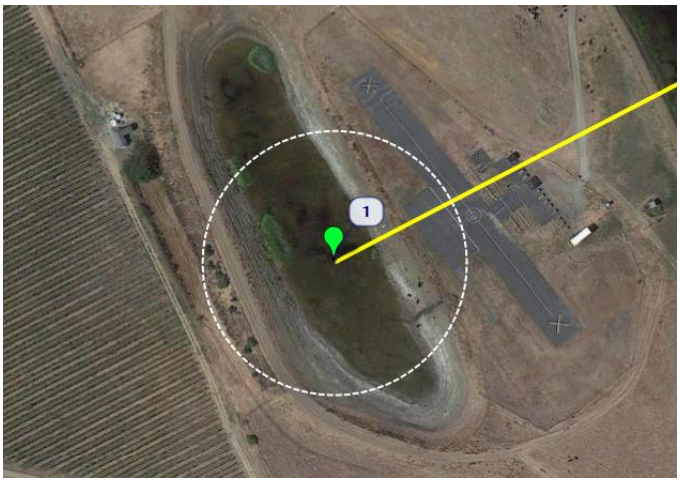
Flight Procedures:

1. Take off and climb to **~ 120** meters AGL
2. **Trim** Huey for **50-60%** Throttle Setting
3. Switch mode to **"Start Location"**
4. Huey should proceed to this waypoint:



   Does he get there? YES NO

5. Allow Huey to loiter in start location until Dewey and Louie are ready as well
   a. Keep an eye on **altitude**. If Huey is struggling to maintain altitude, you can help him out by bumping it up with **elevator stick input.**
6. Once Dewey and Louie are also at their start position, wait for the signal, and **then engage Formation Flight mode** (all three at the same time).
   a. They should attempt to their formation around this loiter position:



If Huey is not the leader, skip step 7

If Huey is the leader

7. When signaled, switch to **manual**, and steer **away** from formation

If Huey does not become the leader, skip step 8-10

If Huey becomes the leader after the original leader "disappears":

8. When signaled, switch to **Manual-In-Formation** mode
9. Fly in a smooth pattern and try not to maneuver too aggressively.
10. Try not to out-run the trailing aircraft

Observations:

Is Huey the original leader? YES NO

Does the formation re-converge? YES NO

Does Huey become the leader? YES NO

If not:

- Does he follow the new leader? YES NO
- Does he pass the new leader? YES NO
- Is he zig-zagging behind? YES NO
- Is he matching altitude YES NO

Other Observations:

11. When the signal is given, switch to **manual**, and come back to land
12. Unplug ESC from battery
13. Unplug BEC cable from Rx
14. Bring back to work area for log download

☐ Logs are downloaded	☐ Logs are cleared