



**DESIGN AND IMPLEMENTATION OF A UNIFIED
COMMAND AND CONTROL ARCHITECTURE FOR
MULTIPLE COOPERATIVE UNMANNED VEHICLES
UTILIZING COMMERCIAL OFF THE SHELF
COMPONENTS**

THESIS

Jeremy Gray, Civilian, Ctr
AFIT-ENV-MS-15-D-048

**DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY**

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

DISTRIBUTION STATEMENT A
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views expressed in this document are those of the author and do not reflect the official policy or position of the United States Air Force, the United States Department of Defense or the United States Government. This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

AFIT-ENV-MS-15-D-048

DESIGN AND IMPLEMENTATION OF A UNIFIED COMMAND AND
CONTROL ARCHITECTURE FOR MULTIPLE COOPERATIVE UNMANNED
VEHICLES UTILIZING COMMERCIAL OFF THE SHELF COMPONENTS

THESIS

Presented to the Faculty
Department of Systems Engineering and Management
Graduate School of Engineering and Management
Air Force Institute of Technology
Air University
Air Education and Training Command
in Partial Fulfillment of the Requirements for the
Degree of Master of Science in Systems Engineering

Jeremy Gray, Civilian, Ctr, B.S.

December 2015

DISTRIBUTION STATEMENT A
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

DESIGN AND IMPLEMENTATION OF A UNIFIED COMMAND AND
CONTROL ARCHITECTURE FOR MULTIPLE COOPERATIVE UNMANNED
VEHICLES UTILIZING COMMERCIAL OFF THE SHELF COMPONENTS

THESIS

Jeremy Gray, Civilian, Ctr, B.S.

Committee Membership:

Dr. David Jaques, PhD
Chair

Dr. John Colombi, PhD
Member

Maj Brian Woolley, PhD
Member

Abstract

Small unmanned systems provide great military application utility due to their portable and expendable design. These systems are, however, costly to develop, produce, and maintain, making it desirable to integrate available commercial off the shelf (COTS) components. This research investigates the development of a modular unified command and control (C2) architecture for heterogeneous and homogeneous vehicle teams to accomplish formation flocking and communication relay scenarios through the integration of COTS components. In this thesis, a vehicle agnostic architecture was developed to be applied across different vehicle platforms, different vehicle combination, and different cooperative missions. COTS components consisting primarily of open source hardware and software were integrated and tested based on the positional accuracy, precision, and other qualitative measures. The resulting system successfully demonstrated formation flocking in three of four vehicle combinations, with the fourth still demonstrating a leader follower relationship. The system achieved at best a mean relative positional error of 0.99m, a standard deviation of 0.44m, and a distance root mean square of 0.59m. The communication relay scenario was also demonstrated with two vehicle combinations for both distance and physical obstructions breaking the C2 link. This system demonstrated the desired capabilities and could easily be adapted to accomplish others through the use of the flexible architecture.

Acknowledgements

I would like to start off by thanking my research advisor Dr. David Jacques. He not only provided the technical guidance and mentorship I needed to complete this research, but also provided me this great opportunity to attend and work at AFIT.

I also want to thank Rick Patton for the technical expertise he provided me throughout the development of my hardware, for flying all of my research platforms, and most of all for being a tremendous mentor and friend throughout this process.

Finally and most importantly, I have to acknowledge and thank my fiancée. I asked her to be my wife shortly after starting AFIT and am reminded daily why I made that decision by her undying love, support, and patience. You have truly been my rock throughout this process and without you none of this would be possible. I love you and cannot wait to spend the rest of my life with you.

Jeremy Gray, Civilian, Ctr

Table of Contents

	Page
Abstract	iv
Acknowledgements	v
List of Figures	ix
List of Tables	xiv
List of Abbreviations	xv
I. Introduction	1
1.1 Introduction and Motivation	1
1.2 Problem Statement	3
1.3 Objective	4
1.4 Investigative Questions and Methodology	4
1.5 Scope	5
1.6 Assumptions and Limitations	5
1.7 Thesis Outline	5
II. Literature Review	7
2.1 Chapter Overview	7
2.2 Unmanned Systems	7
2.3 Cooperative Behavior Applications	9
Formation Flight	9
Communication Relay	10
Search and Surveillance	10
2.4 Command and Control Architectures	11
2.5 State of Practice for COTS, OSH, and OSS	14
Autopilot	14
Communication	17
Ground Control Station	22
2.6 Chapter Summary	23
III. Methodology	24
3.1 Chapter Overview	24
3.2 Command and Control Architecture Development	24
AV-1	25
Formation Flight Use Case and OV-1	26
Communication Relay Use Case and OV-1	27
Architecture Development Method	29

	Page
3.3 Software Development Procedure	29
3.4 Test and Verification Procedure	30
Formation Flocking Verification, Relative Accuracy and Precision Tests	30
Communication Relay Verification, Relative Accuracy and Precision Tests	32
System Latency	34
3.5 Chapter Summary	35
IV. Architecture	36
4.1 Chapter Overview	36
4.2 Operational Activities	36
4.3 System Elements and Functions	42
4.4 Chapter Summary	49
V. Results	50
5.1 Chapter Overview	50
5.2 Selected Hardware and Software	50
Communication System	50
Command and Control Software	50
Autopilot	51
Vehicles	52
5.3 Command and Control Software Development	55
5.4 Formation Flocking Test Results and Analysis	59
UGS Following Multi-Rotor UAS	60
Multi-Rotor UAS Following UGS	68
Multi-Rotor UAS Following Multi-Rotor UAS	84
Fixed Wing UAS Following Fixed Wing UAS	98
Formation Flight Analysis	108
5.5 Communication Relay Test Results and Analysis	111
Test Results and Analysis	112
5.6 Latency Test Results and Analysis	114
Results	114
Analysis	115
5.7 Chapter Summary	116
VI. Conclusion	117
6.1 Chapter Overview	117
6.2 Conclusion of Research	117
6.3 Recommended Future Work	121
Bibliography	124

	Page
Appendix	127
A Appendix A: Formation Flocking Leader Vehicle Script	127
B Appendix B: Formation Flocking Follower Vehicle Script	131
C Appendix C: Multi-Vehicle Function Module, as Tested	135
D Appendix D: Communication Relay Remote Vehicle Script	137
E Appendix E: Communication Relay Relay Vehicle Script	139
F Appendix F: Multi-Vehicle Function Module With Fixed Follower_Pos Calculation	142
G Appendix G: Traxxas EMAXX UGS Pixhawk Parameters	144
H Appendix H: X8 Multi-Rotor UAS Pixhawk Parameters	148
I Appendix I: Supper Sky Surfer UAS Pixhawk Parameters	153

List of Figures

Figure	Page
1	DOD Unmanned Systems Roadmap UAS Categories [1]. 8
2	DOD Unmanned Systems Roadmap UGS Categories [1]. 9
3	Pixhawk Autopilot Control Architecture [17]. 16
4	Centralized Communication Network Architecture [18]. 18
5	Decentralized Ad Hoc Communication Network Architecture [18]. 19
6	Decentralized Multi-Group Communication Network Architecture [18]. 20
7	Decentralized Multi-Layer Communication Network Architecture [18]. 21
8	OV-1: Formation Flight. 27
9	OV-1: Communication Relay. 28
10	Communication Relay Test Configuration. 33
11	Latency Stackup Test. 35
12	OV-5b: System Operational Activity Diagram. 38
13	OV-5b: Perform Cooperative Activity. 39
14	OV-5a: Operational Activity Decomposition Tree. 41
15	SV-1: System Interface Description. 44
16	SV-4: System Functionality Description. 45
17	SV-5a: System Function Traceability Matrix. 48
18	Traxxas E-Maxx UGS. 53
19	3DR X8 Multi-Rotor UAS. 54
20	Super Sky Surfer Fixed Wing UAS. 55
21	Follower Commanded Position Calculation Method. 57

Figure		Page
22	UGS Following Multi-Rotor UAS Test 1 Radial Position Error.	61
23	UGS Following Multi-Rotor UAS Test 1 Forward-Right Position Error.	61
24	UGS Following Multi-Rotor UAS Test 1 Vehicle Position.	62
25	UGS Following Multi-Rotor UAS Test 2 Radial Position Error.	63
26	UGS Following Multi-Rotor UAS Test 2 Forward-Right Position Error.	63
27	UGS Following Multi-Rotor UAS Test 2 Vehicle Position.	64
28	UGS Following Multi-Rotor UAS Test 3 Radial Position Error.	65
29	UGS Following Multi-Rotor UAS Test 3 Forward-Right Position Error.	65
30	UGS Following Multi-Rotor UAS Test 3 Vehicle Position.	66
31	Multi-Rotor UAS Following UGS Test 1 Radial Position Error.	70
32	Multi-Rotor UAS Following UGS Test 1 Forward-Right Position Error.	70
33	Multi-Rotor UAS Following UGS Test 1 Vehicle Position.	71
34	Multi-Rotor UAS Following UGS Test 2 Radial Position Error.	72
35	Multi-Rotor UAS Following UGS Test 2 Forward-Right Position Error.	72
36	Multi-Rotor UAS Following UGS Test 2 Vehicle Position.	73
37	Multi-Rotor UAS Following UGS Test 3 Radial Position Error.	74
38	Multi-Rotor UAS Following UGS Test 3 Forward-Right Position Error.	74

Figure		Page
39	Multi-Rotor UAS Following UGS Test 3 Vehicle Position.	75
40	Multi-Rotor UAS Following UGS Test 4 Radial Position Error.	76
41	Multi-Rotor UAS Following UGS Test 4 Forward-Right Position Error.	76
42	Multi-Rotor UAS Following UGS Test 4 Vehicle Position.	77
43	Multi-Rotor UAS Following UGS Test 5 Radial Position Error.	78
44	Multi-Rotor UAS Following UGS Test 5 Forward-Right Position Error.	78
45	Multi-Rotor UAS Following UGS Test 5 Vehicle Position.	79
46	Multi-Rotor UAS Following UGS Test 6 Radial Position Error.	80
47	Multi-Rotor UAS Following UGS Test 6 Forward-Right Position Error.	81
48	Multi-Rotor UAS Following UGS Test 6 Vehicle Position.	81
49	Multi-Rotor UAS Following Multi-Rotor UAS Test 1 Radial Position Error.	85
50	Multi-Rotor UAS Following Multi-Rotor UAS Test 1 Forward-Right Position Error.	86
51	Multi-Rotor UAS Following Multi-Rotor UAS Test 1 Vehicle Position.	86
52	Multi-Rotor UAS Following Multi-Rotor UAS Test 2 Radial Position Error.	87
53	Multi-Rotor UAS Following Multi-Rotor UAS Test 2 Forward-Right Position Error.	88
54	Multi-Rotor UAS Following Multi-Rotor UAS Test 2 Vehicle Position.	88
55	Multi-Rotor UAS Following Multi-Rotor UAS Test 3 Radial Position Error.	89

Figure		Page
56	Multi-Rotor UAS Following Multi-Rotor UAS Test 3 Forward-Right Position Error.	90
57	Multi-Rotor UAS Following Multi-Rotor UAS Test 3 Vehicle Position.	90
58	Multi-Rotor UAS Following Multi-Rotor UAS Test 4 Radial Position Error.	91
59	Multi-Rotor UAS Following Multi-Rotor UAS Test 4 Forward-Right Position Error.	92
60	Multi-Rotor UAS Following Multi-Rotor UAS Test 4 Vehicle Position.	92
61	Multi-Rotor UAS Following Multi-Rotor UAS Test 5 Radial Position Error.	93
62	Multi-Rotor UAS Following Multi-Rotor UAS Test 5 Forward-Right Position Error.	94
63	Multi-Rotor UAS Following Multi-Rotor UAS Test 5 Vehicle Position.	94
64	Multi-Rotor UAS Following Multi-Rotor UAS Test 6 Radial Position Error.	95
65	Multi-Rotor UAS Following Multi-Rotor UAS Test 6 Forward-Right Position Error.	96
66	Multi-Rotor UAS Following Multi-Rotor UAS Test 6 Vehicle Position.	96
67	Fixed Wing UAS Following Fixed Wing UAS Test 1 Radial Position Error.	100
68	Fixed Wing UAS Following Fixed Wing UAS Test 1 Forward-Right Position Error.	100
69	Fixed Wing UAS Following Fixed Wing UAS Test 1 Vehicle Position.	101
70	Fixed Wing UAS Following Fixed Wing UAS Test 2 Radial Position Error.	102

Figure		Page
71	Fixed Wing UAS Following Fixed Wing UAS Test 2 Forward-Right Position Error.	102
72	Fixed Wing UAS Following Fixed Wing UAS Test 2 Vehicle Position.	103
73	Fixed Wing UAS Following Fixed Wing UAS Test 3 Radial Position Error.	104
74	Fixed Wing UAS Following Fixed Wing UAS Test 3 Forward-Right Position Error.	104
75	Fixed Wing UAS Following Fixed Wing UAS Test 3 Vehicle Position.	105
76	Fixed Wing UAS Following Fixed Wing UAS Test 3 Vehicle Position NW Corner.	107
77	Formation Flocking Test Results Summary.	109
78	Multi-Rotor UAS Relaying to UGS Radial Position Error.	113
79	Multi-Rotor UAS Relaying to UGS Vehicle Position.	114

List of Tables

Table		Page
1	AV-1: System Overview and Summary Information.....	25
2	Formation Flocking Test Matrix	32
3	Communication Relay Test Matrix	34
4	Vehicle Agnostic Activity Descriptions for Each Type of Vehicle	42
5	System Function Descriptions	46
6	UGS Following Multi-Rotor UAS Test Parameter Matrix.....	60
7	UGS Following Multi-Rotor UAS Test Results	67
8	Multi-Rotor UAS Following UGS Test Parameter Matrix.....	69
9	Multi-Rotor UAS Following UGS Test Results	82
10	Multi-Rotor UAS Following Multi-Rotor UAS Test Parameter Matrix	84
11	Multi-Rotor UAS Following Multi-Rotor UAS Test Results	97
12	Fixed Wing UAS Following Fixed Wing UAS Test Parameter Matrix	99
13	Fixed Wing UAS Following Fixed Wing UAS Test Results	106
14	Latency Test Results	115

List of Abbreviations

Abbreviation	Page
C2	Command and Control 1
DOD	Department of Defense 2
OSH	Open Source Hardware 2
OSS	Open Source Software 2
COTS	Commercial Off the Shelf 2
UAV	Unmanned Aerial Vehicle 2
GPS	Global Positioning System 5
UAS	Unmanned Aerial Systems 7
UGS	Unmanned Ground Systems 7
ISR	Information, Surveillance, and Recognizance 7
SUAS	Small Unmanned Aerial Systems 7
GCS	Ground Control Station 10
PCB	printed circuit board 11
MAVLink	Micro Air Vehicle Link 15
MANET	Mobile Ad Hoc Networks 21
VANET	Vehicle Ad Hoc Networks 21
FANET	Flying Ad Hoc Networks 21
GUI	Graphical User Interface 22
DODAF	DOD Architecture Framework 24
OV-5a	Operational Activity Decomposition Tree 29
OV-5b	Operational Activity Model 29
SV-4	System Functionality Description 29

Abbreviation		Page
SV-1	Systems Interface Description	29
SV-5a	Operational Activity to System Function Traceability Matrix	29
DRMS	Distance Error Root Mean Square	30

DESIGN AND IMPLEMENTATION OF A UNIFIED COMMAND AND CONTROL ARCHITECTURE FOR MULTIPLE COOPERATIVE UNMANNED VEHICLES UTILIZING COMMERCIAL OFF THE SHELF COMPONENTS

I. Introduction

1.1 Introduction and Motivation

In recent years, ongoing conflicts in South West Asia have revealed the utility of unmanned vehicle systems to accomplish a myriad of different missions. Due to this realization, there is a growing desire and need to execute previously manned missions with unmanned vehicles. The missions reserved for these unmanned agents are generally too dull, dirty, dangerous, or difficult for onboard human pilots to complete. Additionally, the use of Small Unmanned Vehicles allows for a low cost, portable, and expendable solution for a number of different missions. These systems not only aid in the execution of missions previously fulfilled by humans, but can now accomplish more complicated tasks with a greater level of efficiency and effectiveness. Moving forward from their current capabilities, the Department of Defense states that currently fielded systems must be expanded to "achieve the levels of effectiveness, efficiency, affordability, commonality, interoperability, integration, and other key parameters needed to meet future operational requirements," while minimizing the overall cost of acquiring and maintaining the system [1].

One method of expanding the capabilities of these systems is to integrate multiple unmanned vehicles under a unified Command and Control (C2) architecture to cooperatively execute a common set of missions. Integrating multiple heterogeneous

or homogeneous vehicles allows for an increased number of sensors distributed across the team, allowing the team to survey a larger area and collect data faster than a lone agent. Also, with the sensors being distributed across multiple agents, the system can more easily handle the loss of sensors, allowing it to be more robust. Finally, with increasing levels of autonomy a single operator should be able to monitor and control more vehicles, allowing them to accomplish more complex tasks with much greater ease than multiple single vehicle pilots. Examples of such tasks include close formation flight with other aircraft, relaying communication around obstacles and across long distances, and surveying or searching a long perimeter or a large area for a target of interest. Due to the added data management and processing requirements, C2 architectures do have a much higher level of complexity than single vehicle architectures. Additionally, the weight, size, and power limitations of the vehicle can constrain the location of the data processing.

Coinciding with the increased use of unmanned vehicles by the Department of Defense (DOD), a rise in their use by the civilian population has also been seen. Civilians use these unmanned vehicles with varying levels of autonomy for different applications including those related to agriculture, videography, and hobbyist activities. These communities have driven a rapid evolution of different Open Source Hardware (OSH), Open Source Software (OSS), and Commercial Off the Shelf (COTS) products, which are utilized to achieve increased levels of autonomy and capabilities. These civilian technologies are even being utilized successfully by minimally funded war fighters of other nations as a means of collecting intelligence in the battle theater [2]. Though the cited case neutrally affects the United States, the availability of these technologies does introduce the possibility of someone using them for the wrong purpose at home or abroad. These scenarios range from invading someone's privacy using a camera mounted to a Unmanned Aerial Vehicle (UAV), to using an

aircraft to carry a dangerous payload, which could all be accomplished autonomously from a distance or while the perpetrator escapes. Due to these facts, it is critical that the capabilities of the technology be fully understood by the DOD to aid in the mitigation of these types of attacks. An additional need for understanding the capabilities of these technologies is to reduce the cost of acquiring and maintaining unmanned system through the integration of OSS, OSH, and COTS. The integration of these components can minimize the design work required to develop systems and reduce the amount of sensitive material contained in the system, allowing it to be more expendable if lost or captured.

This research focuses on the development and implementation of a C2 architecture for heterogeneous and homogeneous teams of multiple cooperative unmanned vehicles through the utilization of COTS products. Additionally, to aid in the mitigation of attacks at home or abroad, this research further investigates the capabilities COTS products can provide unmanned systems. The remainder of this chapter is an overview of the problem to be solved, the objectives of the research, the limitations and scope of the research, and the key assumptions made.

1.2 Problem Statement

Existing multi agent systems composed of purely COTS components do not effectively control the team to accomplish specific missions. Data latency, command authority, and other factors hinder the system's abilities. Additionally, it is desirable that COTS components be utilized due to the development, manufacturing, and maintenance of proprietary components and software for new systems which is a time consuming and costly processes. These systems must also allow for interoperable interfaces to allow for exchangeable component and software modules to increase the systems overall flexibility and utility.

1.3 Objective

The objective of this research is to develop a multi-agent C2 architecture for cooperative heterogeneous and homogeneous unmanned vehicles through the implementation of COTS components, OSH, and OSS. Furthermore, it is desired that the impact of this implementation be characterized to understand the capabilities of the system.

1.4 Investigative Questions and Methodology

The questions this research attempts to answer and an overview of the methods to answer these questions are outlined below.

- What are the desired missions to be accomplished by cooperative multi-agent systems? A literary review of desired missions for multi-agent cooperative unmanned systems will be accomplished.
- What is the structure and limitations of existing C2 architectures for cooperative unmanned vehicles? A literary review of previously developed C2 architectures for cooperative unmanned vehicles will be accomplished to determine their structure and limitations.
- What are the mission-specific qualitative and quantitative measures for the system? Performance measures are established to determine the system's ability to accomplish each mission.
- How well does this system perform using these performance measures? Tests are completed to determine the system's ability to meet each baseline requirement.
- What are the effects on system performance due to the utilization of COTS, OSH, and OSS? The system performance is reviewed to determine how the use of COTS may have affected the system performance.

1.5 Scope

This research will be limited to the modification of existing Small Unmanned Vehicle architectures and hardware which are currently used for academic research. The types of vehicles used will be limited to ground rovers, multi rotors, and fixed wing vehicles, with no use of any form of aquatic vehicles. Additionally, all components utilized for this research will be COTS components, OSS, or OSH. This research will only focus on the C2 of cooperative unmanned vehicles and not the interaction between manned and unmanned vehicles. However, there will be a safety pilot in the loop at all times to ensure the safety of test team and any observers. Finally, the missions to be accomplished by the architecture will be limited to the missions outlined in the next chapter.

1.6 Assumptions and Limitations

For this research, it is assumed that the Global Positioning System (GPS) will always be available to determine the location of the unmanned vehicles and other navigation techniques not utilizing GPS will not be utilized. All testing of hardware and software will be conducted with trained safety pilots. Test using planes or multi-rotors will be conducted at Atterbury Army Base and ground vehicle tests will be conducted on the Wright Patterson Air Force Base grounds. Testing will not be conducted during rainy or windy conditions.

1.7 Thesis Outline

In the next Chapter, a review of previous research and other works is accomplished to examine the existing foundational knowledge on the subject of cooperative C2 architectures for multiple vehicles. Chapter 3 outlines the methods of developing the

required C2 architecture through the integration of COTS components and the testing to be accomplished. Chapter 4 discusses the resulting architecture that is developed to perform the desired scenarios. Chapter 5 discusses the results of integrating these components through tests to measure the performance of the system. Finally, Chapter 6 reviews the entirety of this research, discusses conclusions drawn from the research accomplished, and recommends further work to be accomplished.

II. Literature Review

2.1 Chapter Overview

This section is an overview of previous work, theory, and technical information pertaining to the implementation of COTS, OSS, and OSH in C2 architectures for teams of unmanned vehicles.

2.2 Unmanned Systems

Unmanned Aerial Systems (UAS) and Unmanned Ground Systems (UGS) are systems that include the components and personnel required to control an unmanned vehicle to perform specific missions. These vehicles are used for a number of different missions, consisting mainly of Information, Surveillance, and Recognizance (ISR) and strike missions. The goal of using these systems is to reduce the probability of harm to humans in dull, dirty, or dangerous environments. Small Unmanned Aerial Systems (SUAS) sub categories of UAS, displayed in Figure 1, are limited mainly to ISR missions and are grouped according to weight, achievable altitude, and speed. UGS are grouped by capability, as displayed in Figure 2, including ISR and force protection tasks such as bomb defusing and detonation. The portability and low cost associated with small unmanned systems allows for deployment by ground troops with less worry of losing expensive equipment. These systems are operated remotely by a human usually with limited levels of autonomy, only relying on higher levels of autonomous behavior for extreme circumstances such as a loss of communication link. The DOD is, however, investigating the addition of levels of autonomy to decrease the cost of operating the system and decrease the workload of the pilots [1].

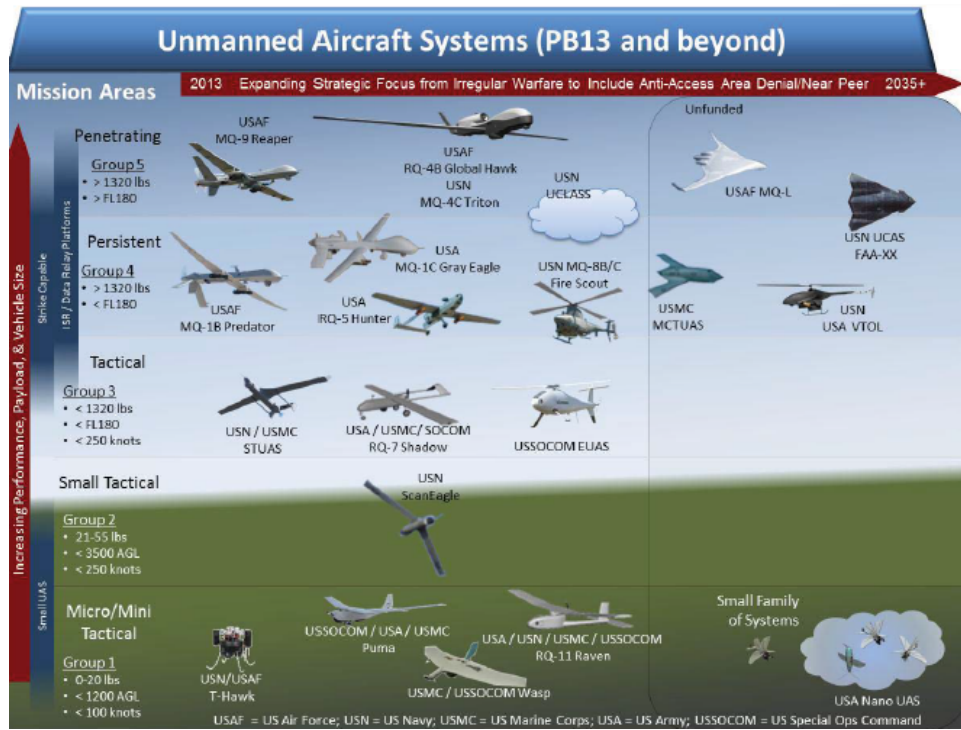


Figure 1. DOD Unmanned Systems Roadmap UAS Categories [1].

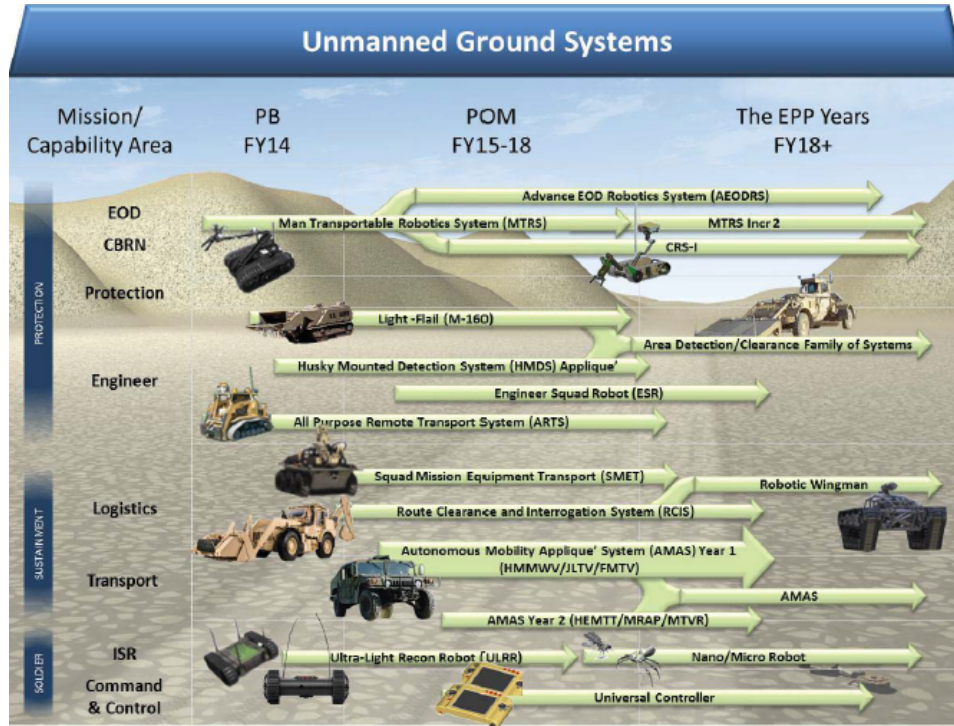


Figure 2. DOD Unmanned Systems Roadmap UGS Categories [1].

2.3 Cooperative Behavior Applications

Cooperative C2 architectures allow teams of unmanned vehicles to accomplish tasks more efficiently and effectively than individual vehicles, or even tasks which individual vehicles cannot accomplish alone. This section reviews common missions cooperative teams of vehicles are desired to accomplish.

Formation Flight.

The first task is formation flight, which can be utilized for autonomous air to air refueling or to utilize the leader's jet stream to reduce the amount of drag on the follower and thus reduce fuel usage. The goal of the required C2 architecture is to minimize the error between the desired position and current position of the follower relative to the leader. One of the difficulties accomplishing this task with a team

of fixed wing aircraft is reducing this error while flying in the turbulent flow of the wingtip vortexes. Also, this capability requires a higher controller command frequency as the desired span between vehicles decreases, limiting the ability to use processing at the Ground Control Station (GCS) and requiring more processing onboard [3, 4],

Communication Relay.

Another task requiring a cooperative C2 architecture is the relaying of communication links. The goal of this task is to provide a communication link from the GCS through a relaying agent to a remote agent which has a physical object obstructing their communication link or an obstruction due to distance from the desired point [5]. Also, this task allows a team of unmanned vehicles to collect and transmit data from remote sensors which are not networked with the data's desired destination [6]. Such tasks can require the team have the ability to obtain knowledge of the other members positions to avoid mid air collisions, but do not require as high of a refresh rate as formation flight.

Search and Surveillance.

The final task that is improved by or requires a cooperative C2 architecture is the cooperative persistent wide area surveillance problem. These applications include the persistent surveillance of an object with an obstructed viewing area (like a building's front door) and the surveillance of a long boarder, perimeter, or large area of interest which cannot be surveyed with a single sensor or agent. The benefit of utilizing multiple vehicles for these applications is it distributes the workload across the vehicles. Again, these tasks can require the team have the ability to obtain knowledge about other members locations to avoid mid air collisions and the ability to relay data through other vehicles in a network. Finally, it is an added benefit if the vehicles have

the ability to search intelligently, using information about where other vehicles have recently searched [7].

2.4 Command and Control Architectures

Cooperative C2 architectures have been successfully implemented on a number of different research projects. One implementation for the use by the NAVY is the Low-Cost UAV Swarming Technology (LOCUST) which was developed by the Office of Naval Research. This system utilizes a team of Coyote UAVs, which are launched from a tube array based launcher. Each UAV can communicate within the group and the system as a whole has demonstrated the ability to achieve close formation flight with up to nine vehicles [8].

Other implementations of formation flight through cooperative C2 architectures have been demonstrated using the Phastball UAV test bed, which include a team of custom SUAS. This test bed UAV is outfitted with a custom printed circuit board (PCB) autopilot with a GPS rated for 1.5m RMS, a 50Hz mechanical gyroscope, and 4 redundant IMUs which are over sampled to increase the resolution from 14 bit to 18 bit. This platform was used in a flight validation for a multi-UAV framework and wing wake encounter algorithms. This framework includes a linear quadratic inner loop controller algorithm which provides the desired trajectories. The flight tests were conducted using virtual and physical leaders. During testing, this configuration achieved a mean distance errors of 3.43m with a standard deviation less than 2m for straight portions of the flight path. The mean distance error increased to approximately 10m with a standard deviation of 3m during turning maneuvers [3].

Another implementation of a cooperative C2 architecture demonstrated by the Aerospace Controls Laboratory at the Massachusetts Institute of Technology utilized a flexible test bed architecture which was designed for research in the field of controls.

Each UAV contains a Piccolo autopilot, which controls the vehicle’s stabilization inner loop controls and waypoint navigation outer loop controls. Each autopilot contains a GPS with position errors of $\pm 2\text{m}$ and a transceiver to communicate with another mated transceiver at a dedicated GCS for each UAV. The GCS allows for outer loop commands to be sent to the autopilot and processing of formation flight algorithms to determine these commands. Due to the off board processing, the update rate of commands from the ground station to the vehicle is 1Hz. The demonstration of formation flight utilized fixed flight paths for both vehicles, varying their speeds to reduce the positional error between the UAVs to a 25m by 25m square around the desired position [9].

As displayed by Kingston et al. [10], COTS can also be implemented for other capabilities such as perimeter surveillance. This demonstration utilized a Kestrel Autopilot to validate a perimeter surveillance algorithm which allows for growth of the perimeter and the addition and subtraction of team members on a decentralized C2 network. The need for a decentralized C2 network is required because the size of the perimeter could lead the vehicle out of range of the communication link. It is proposed that the vehicle will collect data while patrolling, and dump the information once it is within range. In application, the system was able to survey the perimeter cooperatively, however it is not clearly stated nor does the autopilots utilized lead to conclusion that the vehicles were operated on a decentralized network.

This research directly follows an investigation into the effects implementing different configurations of low cost OSH and OSS for a cooperative C2 architecture. The latency and relative positional accuracy of each configuration were measured to determine the performance of that particular configuration. The first set of configurations tested utilized Mission Planner as the GCS, while varying the sub-application utilized to control multiple vehicles. The sub-applications used are the beta swarm applica-

tion and python scripting application. The python scripting application required the use of multiple instances of Mission Planner, either on the same or different GCS laptops with a hardwired ethernet connection to pass information between. The swarm application has limited capabilities due to its use of fixed formation movements relative to north as opposed to being relative to the heading of the lead vehicle. The python scripting on the other hand allows for greater flexibility in the formation of the vehicles and allows for added levels of autonomy. The results of these tests showed that the built in beta swarming application was the least latent with an update rate of approximately 0.4Hz. It was also found that operating both vehicles from the same ground station produced a higher latency than from individual ground stations, with the highest refresh rate being 0.3Hz and 0.2Hz respectively [11].

Additional work at AFIT includes the CUSS architecture, which was developed as a solution for cooperative surveillance of stationary and moving targets, along with a solution for the wide area search problem [7]. The architecture incorporated the Kestrel autopilot and the Virtual Cockpit GCS, which allows for control of multiple vehicles from a single application. Four BATCAM UAVs were successfully flown and provided simultaneous video feeds to survey the area. Another architecture developed at AFIT was the OWL architecture, which was a solution for relaying a communication link from the ground station to a remote vehicle that is out of range of the link [5]. To achieve a signal pass through capability on the relay vehicle, the link was passed through separate transceivers on the relay vehicle, which were each mated with the GCS and the remote vehicle transceivers. This does not allow for variation in the number of links due to the increased number of transceivers required and algorithms required to handle the data management.

As shown through previous work, the ability to C2 a team or teams of homogeneous vehicles exists. However, as the processing capability migrates away from the

airframe to a central processing location, the system's ability to handle close interactions with other agents degrades. Due to the latency of transmission between agents, a similar reaction occurs as the communication network transitions from an ad hoc network to a centralized network. However, the effects of this migration away from onboard processing and ad hoc networks are less noticeable during operations when agents are more highly distributed in space. From this review, it is found that the use of COTS, OSS, and OSH can and have been used to achieve homogeneous C2 architectures, with the consequence being a lower measure of capability compared to a system composed of proprietary components.

2.5 State of Practice for COTS, OSH, and OSS

COTS, OSS, and OSH allow for a quick, easy, and cheap solution for countless problems, with the majority of the time, effort, and cost being associated with the integration of components with one another or existing proprietary systems. For the application of unmanned vehicles, the integration of multiple sensors is required to achieve a more accurate and precise navigation solution versus a standalone sensor. With the recent expansion in availability of OSH and OSS for small unmanned vehicles, components are readily available and come pre-integrated with the required sensors or in a modular form allowing for the addition of sensors to achieve the desired capabilities. This section reviews the available COTS, OSS, and OSH required to attain a cooperative unmanned vehicle C2 architecture.

Autopilot.

The autopilot is an on board control mechanism which contains the inner and outer loop controllers required to perform stabilized flight and preprogrammed missions. The major difference between different brands of COTS autopilots is the availability

of information on the autopilot design. Autopilots such as the Pixhawk [12] and Ardupilot [13] are developed with open source chipsets where other autopilots, such as Pickelo [14] and Kestral [15], are developed with custom or non-specified chipsets, or for specific GCS software. The trend for open source and some proprietary autopilots is to exclusively use the Micro Air Vehicle Link (MAVLink) protocol, which is a message marshalling library of commands and responses specifically for small air vehicles [16]. The use of this common communication protocol allows for variations in the GCS software used, as long as that GCS software communicates using MAVLink. Also, open source autopilots utilize modular sensors such as GPS, compass, and airspeed sensors through common ports, allowing for easy integration of required sensors to achieve a more accurate navigation solution.

Due to their intended use for different RC vehicle platforms, open source autopilots utilize similar architectures. The Pixhawk autopilot controller architecture depicted in Figure 3 shows the general structure of these controllers. The autopilot receives and transmits MAVLink commands and telemetry via radio modems. These commands determine the outer loop command inputs, parameter settings, and vehicle mode. The inner loop controller commands are then achieved by monitors sensor readouts to determine attitude and a position estimates and commanding actuators to maintain the desired trajectory or flight characteristics. The control laws which govern the inner and outer loops are determined by the mode of the autopilot, which is set from the GCS or RC radio. Common modes among open source autopilots include a fully manual mode, which passes RC commands directly to the actuators, a stabilized mode, which controls only the inner loop controller to provide stabilized flight, and a fully autonomous mode, which controls both inner and outer loops to maintain stable flight and the desired trajectory. Other modes include a guided mode, which commands the aircraft to fly to a selected point and then loiter at that point.

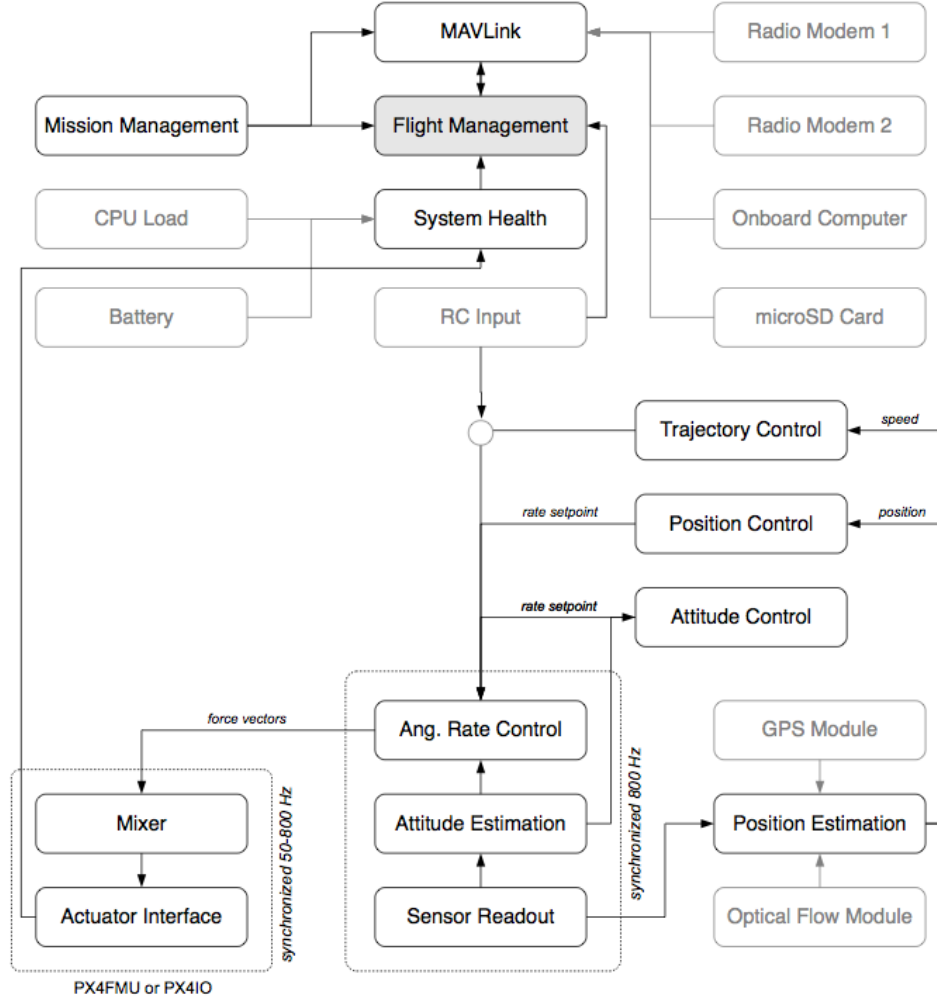


Figure 3. Pixhawk Autopilot Control Architecture [17].

The tradeoff between the different open source autopilots is their cost, robustness against failures, position estimation techniques used, and overall capabilities available. Based on these measures, the current market leader is the Pixhawk. Though a fraction more expensive than others, this autopilot provides a more robust architecture due to the optional redundant backup power supplies, GPS receivers, and IMUs. An Extended Kalman Filter is also utilized to provide a better position estimate in the presence of noise. Finally, this autopilot provides the user the fullest range of capabilities compared to other autopilots. These cheap and accessible autopilots coupled

with a GCS and a RC vehicle platform allow for near full autonomous behavior of the vehicle.

Communication.

The method of transmitting and receiving commands and information from sensors is a critical design point for all command and control architecture. This design requirement is dependent on the mission to be accomplished, and is even more demanding for cooperative unmanned vehicle C2 architectures due to their increased complexity and high computational demands. There are two general categories of communication network architectures which allow for certain forms of cooperative C2.

The first category of communication networks is the centralized network. As displayed in Figure 4, this architecture allows each agent to communicate to a central node, which then relays information to the other agents in the team. The three major drawbacks from this communication structure are the relatively high transmission latency for communication between vehicles, the constrained communication range, and the increased vulnerability due to the single point of failure at the central node. However, this network does allow for a central processing unit, allowing vehicles to require less board processing, thus reducing their total weight.



Figure 4. Centralized Communication Network Architecture [18].

The other communication network category is the decentralized network which has no centralized node through which all agents must obtain their information, allowing them to communicate directly or indirectly with other agents or ground stations. A subcategory of this is the ad hoc network architecture shown in Figure 5, which allows agents to relay information directly or indirectly to other agents through the network. In this configuration, a single UAV is used as the backbone link to the ground station, allowing for a single high power transceiver on the backbone UAV to extend the range of the team. This is unlike the centralized network which would require each UAV to have a high power receiver to achieve the same range. This single backbone and the relative proximity of the agents to each other allows for the majority of the team to use relatively lighter weight transceivers. Additionally, the optimal path of the communication link between vehicles can be determined through algorithms processed on the agent. One downfall of this architecture is the vulnerability of the system to the loss of the backbone node, which would disconnect all other agents from the central node. If the team were in close proximity of the central node, the backbone

node is unnecessary, reducing this vulnerability. Another downfall of this architecture is as the number of agents increase, the complexity of the network increases and the available bandwidth would decrease. This issue can be mitigated through the use of other decentralized networks including the multi-group networks and multi-layer ad hoc network. The multi-group network displayed in Figure 6 allows for centralized control of each team while maintaining decentralized control within each team, and the multi-layer network displayed in Figure 7 allows for decentralized control of each team and decentralized control within each team [19].

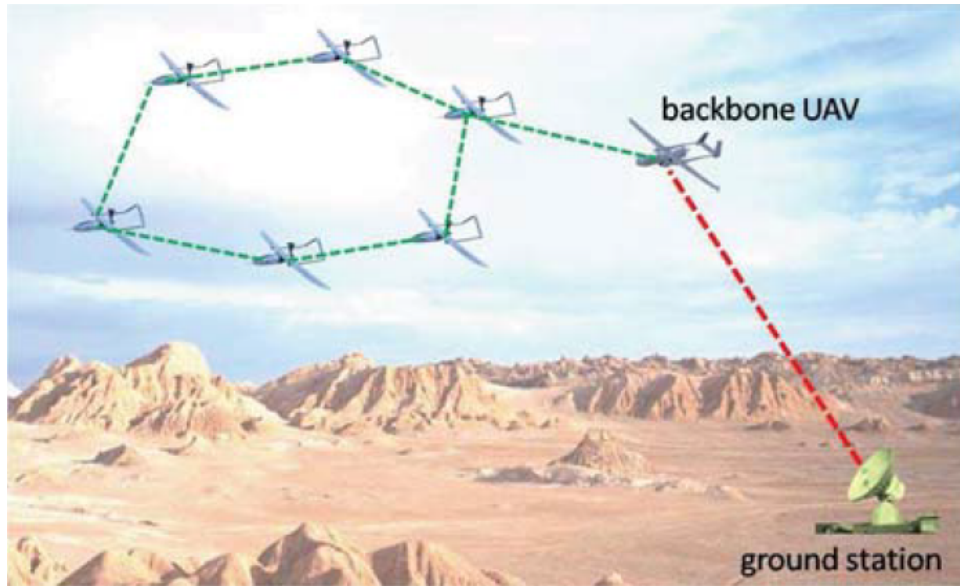


Figure 5. Decentralized Ad Hoc Communication Network Architecture [18].

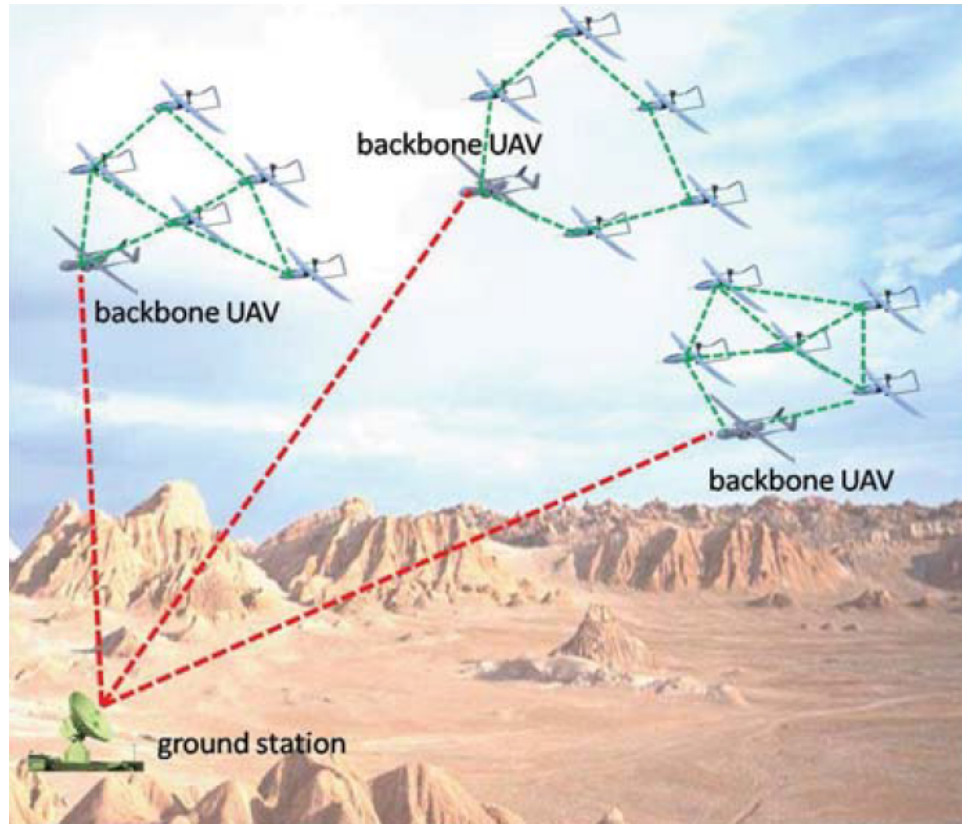


Figure 6. Decentralized Multi-Group Communication Network Architecture [18].

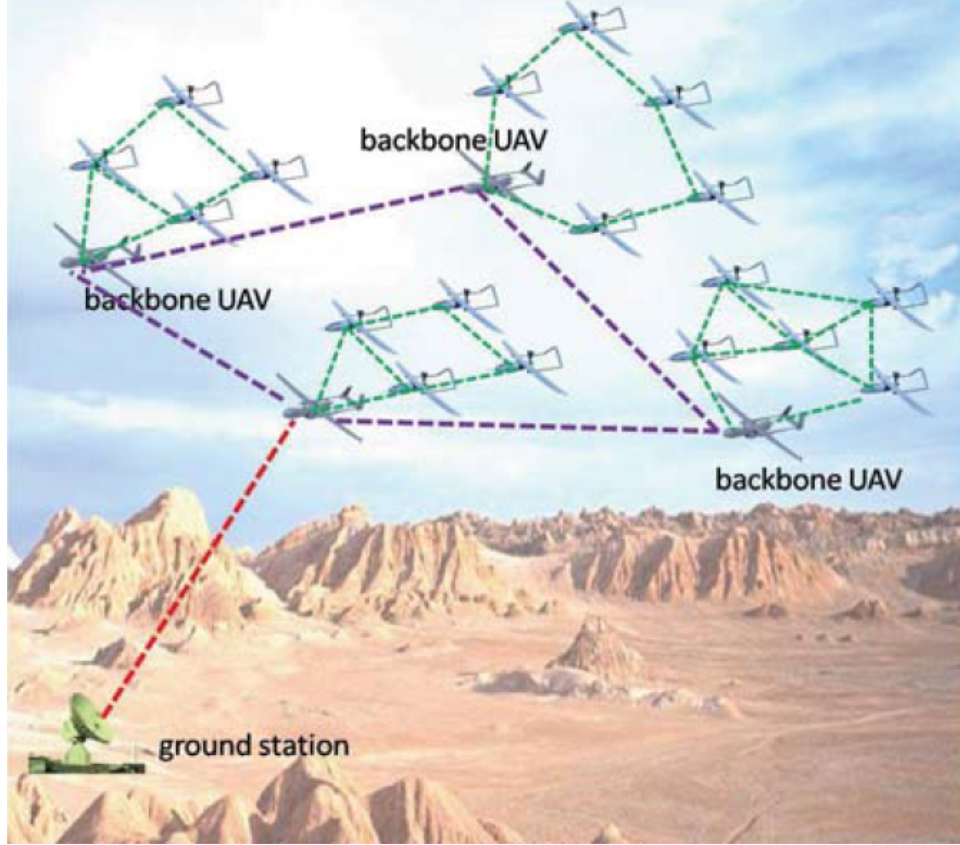


Figure 7. Decentralized Multi-Layer Communication Network Architecture [18].

These decentralized ad hoc networks can be accomplished through the use of COTS transceivers which run onboard networking protocol algorithms to determine the desired path to transfer data to its desired location. These components are used in Mobile Ad Hoc Networks (MANET) for person to person communication, Vehicle Ad Hoc Networks (VANET) for ground vehicle to ground vehicle communication, Flying Ad Hoc Networks (FANET) for flying vehicle to flying vehicle communication. The major difference between these networks is the node density and mobility. The node density affects the number of optional paths for the data to be transferred, and the node mobility affect the rate at which the topology of the network changes [18].

Ground Control Station.

One vital component of a C2 architecture is the central command station or GCS, which allows the user to command the system to accomplish different tasks and review information received from the vehicle. The GCS communicates to the autopilot via mated radio transceivers using the MAVLink protocol to send outer loop controller commands and change the mode of the autopilot, which changes the control laws governing the outer and inner loop controllers. The recent explosion of OSS and OSH has brought forth a number of easily obtainable ground control stations.

The first category of GCS is the PC based application which communicates with the vehicle via a radio telemetry link or WIFI. These GCS provide a user friendly Graphical User Interface (GUI) which allows for near full utilization of the full capability of the autopilot, which is limited by the capabilities the open source community wants in the GCS. Minimalized GCS allows users to remotely change flight modes, plan and send waypoint missions, and monitor telemetry. More advanced GCS such as Mission Planner [20] allow for the use of python scripts to add levels of autonomy to the system through logic and computations. Unlike APM Planner 2.0 [21] and Kestrel's proprietary GCS Virtual Pilot [15], Mission Planner does not have a robust method to control multiple vehicles. It does have a beta version of this capability. However, Hardy [11] successfully demonstrated the ability to use python scripts to pass information between multiple instances of Mission Planner to perform cooperative behavior. The GCS MavProxy stands out due to its use of the PC operating system's command line to send MAVLink protocol commands directly to the vehicle without the use of the GUI. This ground station is more flexible, modular, and allows for a higher utilization of the full capability of the autopilot while also containing the same capabilities of the other major GCS software. MavProxy does not require the use of a GUI, which can reduce the expected latency produced by this protocol layer.

Finally, like Mission Planner, this software allows for the use of scripting to achieve a higher level of autonomy [22].

The most recent expansion in GCS software are tablet and phone applications based GCS which utilize either a WIFI link or WIFI bridge to radio telemetry link to control the vehicle. These applications are more simplistic than their PC counterparts, providing a more limited number of the autopilot's built in capabilities. All of these ground control stations provide a GUI which displays streaming flight data and allows the ability to set waypoints for autonomous missions. Other advanced applications can utilize the GPS receiver on the control device to allow for a *follow me* mode in which the vehicle follows the ground station, usually while keeping its camera pointed at the target location. These applications are only for the most basic use of the autopilot and do not allow for any additional scripting or major modification to the base capabilities of the autopilot.

2.6 Chapter Summary

In this chapter, a definition, baseline information, and applications for cooperative unmanned systems were established. Also, previous work pertaining to the development of C2 architectures for multiple cooperative unmanned vehicles was reviewed. Finally, existing COTS, OSH, and OSS and their capabilities were discussed as it pertains to these architectures. In the next chapter a methodology is developed for designing, implementing, and testing a cooperative C2 architecture for heterogeneous unmanned vehicles.

III. Methodology

3.1 Chapter Overview

The purpose of this chapter is to articulate the methods followed to develop, test, and verify the functionality of a cooperative C2 architecture for teams of multiple vehicles. First, the process of developing the required architecture and software is described. Then, the test and verification procedures to measure key characteristics of the design are outlined.

3.2 Command and Control Architecture Development

In this section, the procedure and methods used for developing a C2 architecture for teams of multiple vehicles is developed. For this research, the DOD Architecture Framework (DODAF) 2.0 will be utilized as the collection of possible architectural views to create. From this collection, key views will be completed to design and define the system.

Before discussing the selected views to be created, an executive level summary of the system's primary goals, scope, and purpose is developed to scope the architecture development methods. To accomplish this, the Overview and Summary Information (AV-1), High-Level Operational Concept Graphic (OV-1), and brief use cases are created. The AV-1 is an executive level description of the vision, scope, and purpose of the desired system. Additionally, the use cases describe the activities the primary and supporting actors and system accomplish during operation. Finally, the vision is depicted using the OV-1, which visually portrays the operational concept. All of these views will be developed for the formation flight and communication relay operations.

AV-1.

The AV-1 in Table 1 provides a high level overview and summary of the vision, scope, and purpose of this system. The vision describes what the desired system is and how it will work, the scope describes the self imposed factors that will limit the development of the system, and the purpose describes why the system is needed to accomplish the vision.

Table 1. AV-1: System Overview and Summary Information

Architecture Vision	The vision for this system is a unified C2 architecture for teams of two heterogeneous or homogeneous vehicles that requires only a single operator. This architecture will allow the operator to control a team of autonomous vehicles to perform cooperative missions scenarios. These scenarios include performing formation flocking and performing communication relay. The formation flocking scenario requires a follower vehicle to autonomously maintain a fixed formation relative to a lead vehicle. The relaying scenario requires a relay vehicle to maintain a midpoint and pass information between the GCS and a remote vehicle that is out of communication range of the GCS.
Scope	For the initial prototype system, teams will be limited to two vehicles. All hardware and software will be limited to OSH, OSS, and COTS. The development and testing period is limited to six months.
Purpose	The purpose of this system is to provide a single operator the ability to control a team of multiple homogeneous and heterogeneous vehicles to accomplish missions requiring formation flocking or relaying communications through local vehicles.

Formation Flight Use Case and OV-1.

The following use case describes the actions taken during the scenario of formation flocking by all actors and system elements. The OV-1 in Figure 8 depicts the vision for this operation.

Formation Flight Use Case and OV-1.

An operator desires to have two vehicles cooperatively flock in formation to complete a mission. The primary actor is the GCS operator. The operator determines the mission trajectory for the leader vehicle to complete. The GCS operator saves the mission to the leader vehicle's autopilot. The GCS operator launches the lead vehicle. The GCS operator commands the lead vehicle to initialize a loiter maneuver. The GCS operator launches the follower vehicle. The GCS operator commands the follower vehicle to initialize a loiter maneuver. The GCS operator initializes the flocking C2 mode on the GCS. The GCS begins sending commands to the follower vehicle to continuously stay at the desired position relative to the leader. The GCS operator commands the leader vehicle to start the mission. This process ends when the mission is complete or when the GCS operator commands each vehicle to return home or to be recovered.

Formation Flight OV-1.

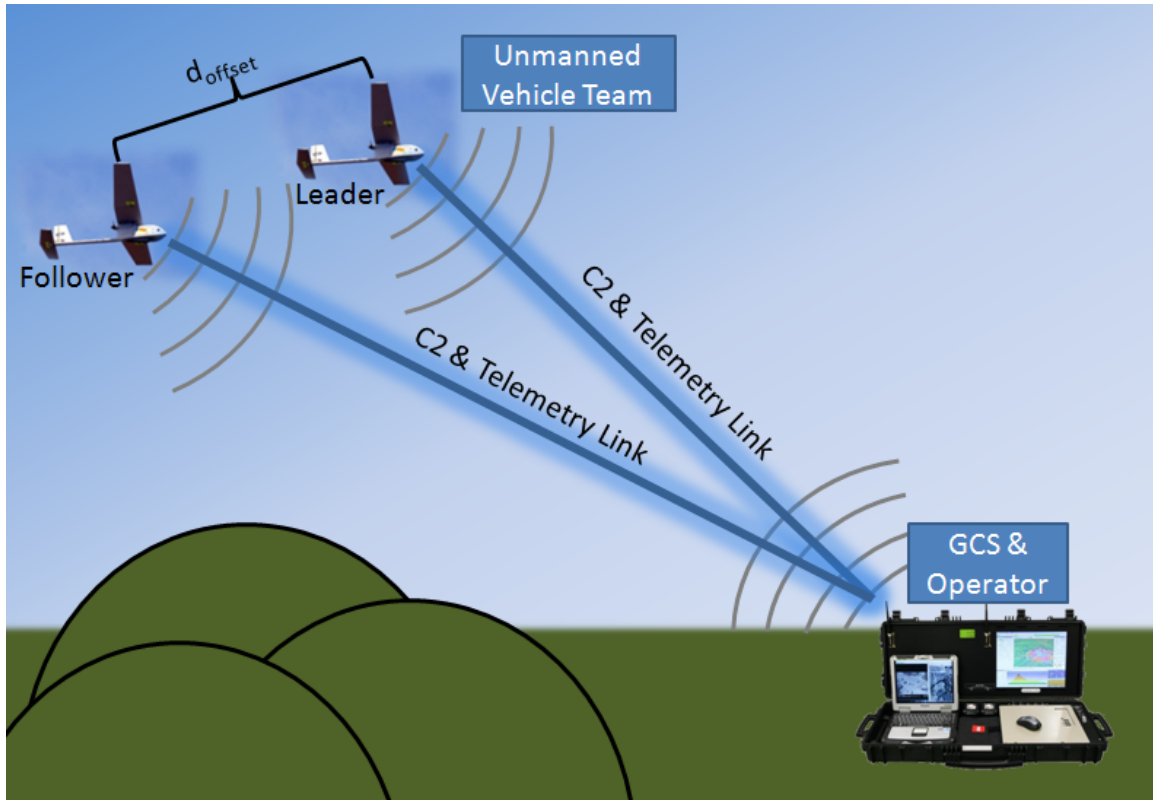


Figure 8. OV-1: Formation Flight.

Communication Relay Use Case and OV-1.

The following use case describes the actions taken during the scenario of communication relay by all actors and system elements. The OV-1 in Figure 9 depicts the vision for this operation.

Communication Relay Use Case.

An operator desires to have a relay vehicle pass information between a remote vehicle and the GCS. The primary actor is the GCS operator. The operator determines the mission trajectory for the remote vehicle to complete. The GCS operator saves the mission to the remote vehicle's

autopilot. The GCS operator launches the remote vehicle. The GCS operator commands the remote vehicle to initialize a loiter maneuver. The GCS operator launches the relay vehicle. The GCS operator commands the relay vehicle to initialize a loiter maneuver. The GCS operator initializes the relay C2 mode on the GCS. The GCS begins sending commands to the relay vehicle to continuously stay at the desired position relative to the remote vehicle. The GCS operator commands the remote vehicle to start the mission. This process ends when the mission is complete, when the GCS operator commands each vehicle to return home, or to be recovered.

Communication Relay OV-1.

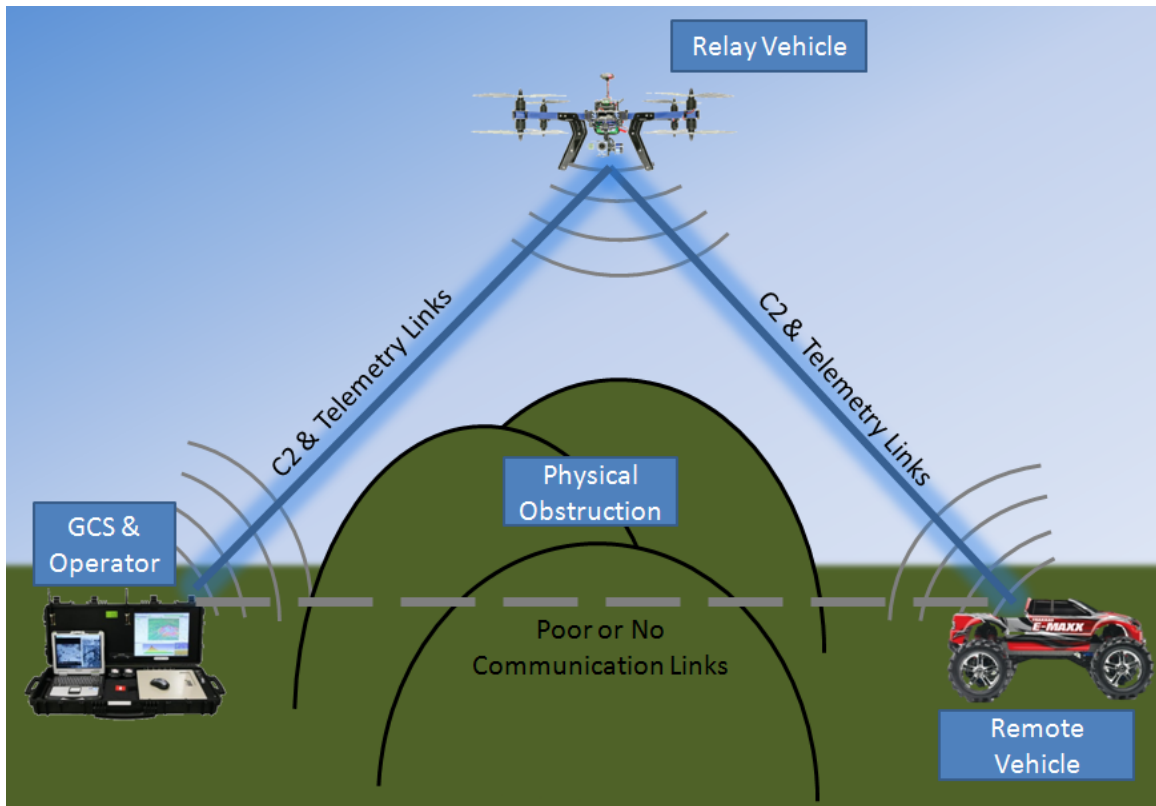


Figure 9. OV-1: Communication Relay.

Architecture Development Method.

The actions described in the use case summaries are decomposed into lower level operational activities in the Operational Activity Decomposition Tree (OV-5a). To aid in this decomposition, an Operational Activity Model (OV-5b) will be developed based on the logic described in each use case. The actors and high level components of the system depicted as swim lanes in the OV-5b are then decomposed into sub components and the functions they accomplish in a System Functionality Description (SV-4). These sub components are then grouped based on their physical location in each high level component of the system, which is depicted in the Systems Interface Description (SV-1). Finally, the leaf level functions depicted in the SV-4 will be traced to the leaf level operational activities of the OV-5a in an Operational Activity to System Function Traceability Matrix (SV-5a) to demonstrate concordance between architectural views [23].

3.3 Software Development Procedure

In this section, the procedure for developing the software for a C2 architecture for teams of multiple vehicles is discussed. Based on the use cases described in the previous section, a controller is required for processing information from vehicle 1 and sending commands to vehicle 2 from the GCS. These controllers will be able to command a follower vehicle to maintain a desired offset position relative to the leader and command a relay vehicle to maintain a midpoint position between the GCS and a remote vehicle. The commanded position calculation used in this controller for each scenario are developed in the results and analysis chapter.

3.4 Test and Verification Procedure

This section outlines the system test and verification methods performed. The tests methods described are developed to measure the latency and position accuracy of the system, while the verification methods are developed to validate the system's ability to accomplish the operations outlined in the previous section.

Formation Flocking Verification, Relative Accuracy and Precision Tests.

The first set of tests verifies the system's ability to perform formation flocking and measures the relative accuracy and precision achieved during formation flocking. The verification of the systems capability to perform formation flocking is based on qualitative pass fail measures. The system successfully demonstrates formation flocking if both vehicles demonstrates a leader follower relationship and the follower displays the tendency to stay near the desired position. The leader follower relationship is also demonstrated by the follower attempting to operate in the same operating region as the leader and the follower maneuvering in the same direction, clockwise or counters clockwise, as the leader.

For this research, accuracy is measured using the Distance Error Root Mean Square (DRMS), which is the measure of the average squared error between the follower vehicle's current and desired position at each time step. The equation for DRMS is shown as Equation 1, where N is the number of time steps, P_i is the i^{th} desired position for the follower, and \hat{P}_i is the i^{th} measured position of the follower.

$$DRMS(P, \hat{P}) = \sqrt{\frac{1}{N} \sum_i^N (P_i - \hat{P}_i)^2} \quad (1)$$

The precision is measured using the standard deviation of the measured errors in the follower's position relative to the desired position of the follower. The equation for

standard deviation is shown below as Equation 2, where N is the number of time steps, P_i is the i^{th} desired position for the follower, \hat{P}_i is the i^{th} measured position of the follower, and $E()$ is the expected value.

$$S(P, \hat{P}) = \sqrt{\frac{1}{N} \sum_i^N ((P_i - \hat{P}_i)_i - E(P_i - \hat{P}_i))^2} \quad (2)$$

These tests will be performed using four combinations of three vehicles. These combinations include a UGS following a multi-rotor UAS, a multi-rotor UAS following a UGS, a multi-rotor UAS following a multi-rotor UAS, and a fixed wing UAS following a fixed wing UAS. To aid in the identification of performance characteristics of the system while accomplishing straight and curved flight paths, the leader in each test will be commanded to autonomously perform both a box and circle path. The follower will simultaneously be commanded to perform formation flocking at a specific offset radius and offset angle. After the test, the flight data for both the leader and follower will be obtained from telemetry logs stored on the GCS. The error between the follower position and the desired follower position will then be evaluated from these telemetry logs.

The two tests with teams consisting of a UGS and a multi-rotor UAS test the system's ability to perform missions using teams of heterogeneous vehicles. The tests with teams consisting of two multi-rotor UAS and two fixed wing UAS test the system's ability to perform missions using teams of homogeneous vehicles. The combination of fixed wing aircraft and ground vehicle or fixed wing aircraft and multi-rotor are not tested due to the difference in operating speed between the vehicles, which would limit the team's ability to maintain formation. The test comprised of only multi-rotor or fixed wing aircraft will use an offset in altitude due to the increased speed of the vehicles and desire to avoid mid air collisions. Table 2 displays a test matrix of the different vehicle combinations being tested.

Table 2. Formation Flocking Test Matrix

		Follower		
		UGS	Multi-Rotor UAS	Fixed Wing UAS
Leader	UGS	No	Yes	No
	Multi-Rotor UAS	Yes	Yes	No
	Fixed Wing UAS	No	No	Yes

Communication Relay Verification, Relative Accuracy and Precision Tests.

Two tests will be performed to verify the system is capable of relaying the telemetry and C2 link between the GCS and a remote vehicle through a relay vehicle. To accomplish this, communication nodes at the GCS, relay vehicle, and remote vehicle will be configured on the same network, allowing each to communicate with all nodes on the network. Additionally, the transmission power will be turned down to the lowest value. This allows the remote vehicle to more easily move out of the range of the GCS node, forcing the path of communication through the relay vehicle at the midpoint. The qualitative measure for this scenario is simply verifying the system relays the C2 link. Finally, from the flight data logs stored on the GCS, the relative position accuracy and precision of the follower vehicle will be determined using the same methods outlined for formation flocking.

The procedure for this test is as follows. The remote vehicles will be moved away from the GCS in manual mode until the GCS loses the ability to communicate with

the remote node, as indicated by the percent of telemetry packets received in Mission Planner. The relay vehicle will then be moved in manual mode to the midpoint between the GCS and remote vehicle until the C2 link between the remote vehicle and GCS is reestablished. Doing this ensures the path of communication goes through the relay node to the remote vehicle due to the GCS and remote nodes physical distance from one another. Once the link is reestablished, the communication relay C2 script will be started to command the relay vehicle to maintain a midpoint position. The remote vehicle will then be driven manually in all directions to ensure the relay vehicle is maintaining a midpoint position. This verifies the system's ability to relay the communication and command links between vehicles and maintain a midpoint position to ensure the link is not lost again. Figure 10 depicts the test setup used to verify this capability.

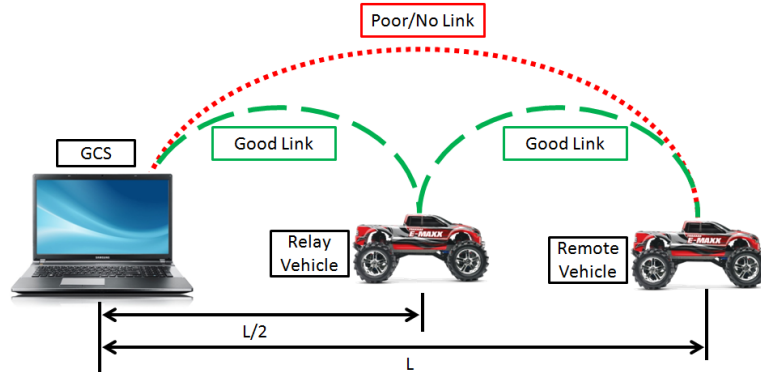


Figure 10. Communication Relay Test Configuration.

This test will be performed using two teams, one consisting of two UGS, and the other consisting of a multi-rotor UAS relaying the C2 link to a remote UGS. Table 3 details the combinations of vehicles and test performed for this capability.

Table 3. Communication Relay Test Matrix

		Relay		
		UGS	Multi-Rotor UAS	Fixed Wing UAS
Remote	UGS	Yes	Yes	No
	Multi-Rotor UAS	No	No	No
	Fixed Wing UAS	No	No	No

System Latency.

The final measure to be tested characterizes the latency within the system. For this research, latency is defined as the difference between the time information is sent, to the time the information is received and the desired action occurs. For the case of formation flight, the total latency is the difference between the time the leader vehicle sends its telemetry to the GCS, to the time when the follower vehicle receives the new commanded waypoint. For this system, the total system latency can be decomposed into sub-latencies (Figure 11). The first sub-latency, denoted t_1 , is the difference between the time one vehicle sends telemetry to the GCS and the time the GCS receives the telemetry. The second sub-latency, denoted t_2 , is the difference between the time the GCS receives the telemetry and the time the GCS sends a command up to the other vehicle. The third and final sub-latency, denoted t_3 , is the difference between the time the GCS sends a command up to the other vehicle and the time the other vehicle receives the command. These sub-latencies will be measured individually using C2 scripts on the GCS.

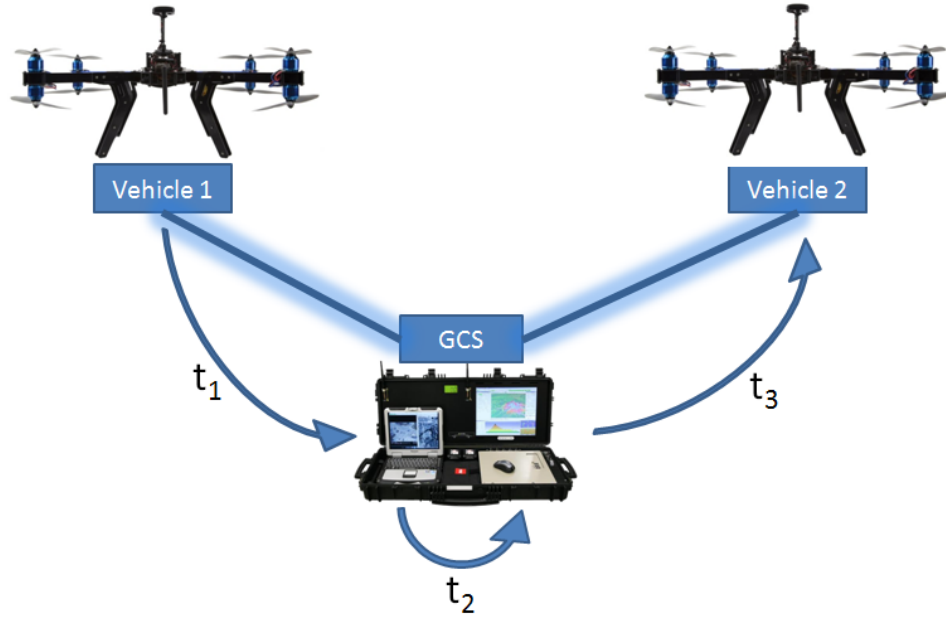


Figure 11. Latency Stackup Test.

3.5 Chapter Summary

This chapter reviewed the development, test, and verification methods used to ensure the system developed is capable of accomplishing all of the desired tasks. The following chapters will further develop the required architecture and analyze the results from the test described above.

IV. Architecture

4.1 Chapter Overview

The purpose of this chapter is to articulate the development of the system architecture to meet the architecture overview information developed in the AV-1, OV-1, and brief use cases from the previous chapter. For this architecture, all elements are developed to be vehicle agnostic, meaning the elements could apply to ground, quad rotor, or fixed wing vehicles. This is accomplished for the purpose of allowing the resulting hardware and software to be applied to any of these types of vehicles and in teams of any combination of these types of vehicles. Also, this architecture is developed to be modular, allowing for different missions to be accomplished through the variation of specific elements.

4.2 Operational Activities

In this section, the operational activities of the architecture are developed. To accomplish this, an OV-5b is created to outline the logical flow of activities to accomplish each use case. From these activities, a joint OV-5a is created to depict the decomposition from a high level abstract operation to the desired lower level operational activities for the system to accomplish.

The OV-5b shown in Figure 12 depicts the flow of activities performed by the GCS operator, GCS, and each vehicle for both the formation flocking and communication relay use cases. These use cases are combined into a single activity flow diagram due to the similarities in the activities performed and to drive the architecture to be agnostic towards the mission being accomplished. In this view, it is shown that the GCS operator initializes the use case by performing mission planning operations on the GCS, with the GCS providing visual information regarding the developed mission.

The GCS operator then commands the GCS to write the mission to the autopilot, causing the GCS to send a command to store the mission. Vehicle 1 receives this command and provides a verification receipt to the GCS, which visually displays the acceptance of the mission. After launching vehicle 1, the vehicle performs an idle maneuver to allow vehicle 2 to be launched and start performing the desired cooperative behavior before vehicle 1 proceeds to perform the mission. The follower will continue to perform the cooperative behavior until the mission is complete, at which time it will perform an idle maneuver until vehicle 1 is recovered. The flow of activities ends with the vehicle 2 being recovered.

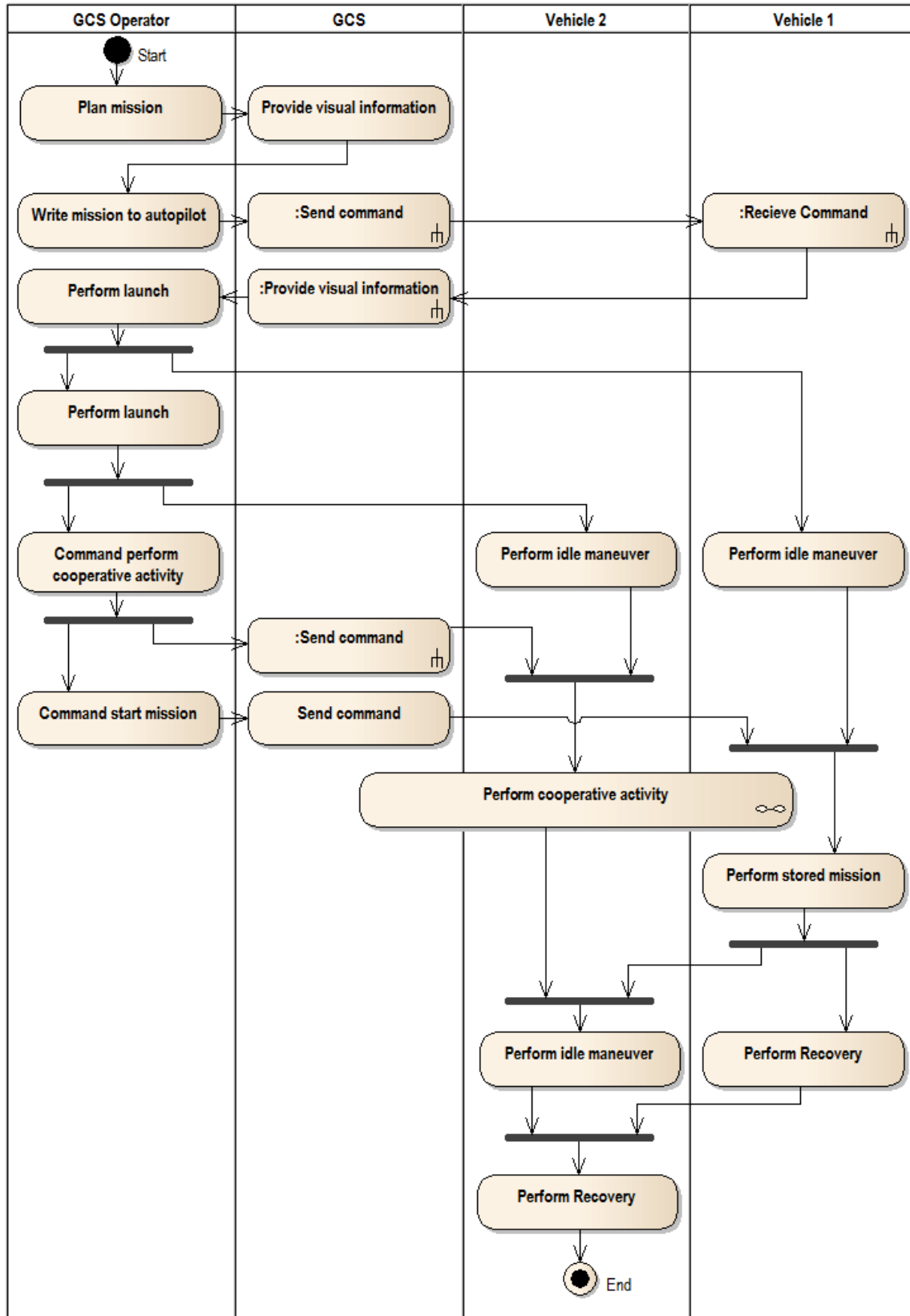


Figure 12. OV-5b: System Operational Activity Diagram.

The activity *Perform cooperative activity* is further refined in Figure 13 to further refine each vehicle's role in the activity. Additionally, this diagram aids in determining the communication structure required to pass vehicle telemetry to the GCS and commands to each vehicle. This activity involves the GCS software and both vehicles. The activity *Calculate desired vehicle 2 position* represents the algorithm performed by the GCS to determine the commanded go to location sent to vehicle 2 to accomplish each cooperative behavior. This element is the point where the architecture varies for each cooperative behavior, including formation flight and communication relay, to allow for different mission controller calculations.

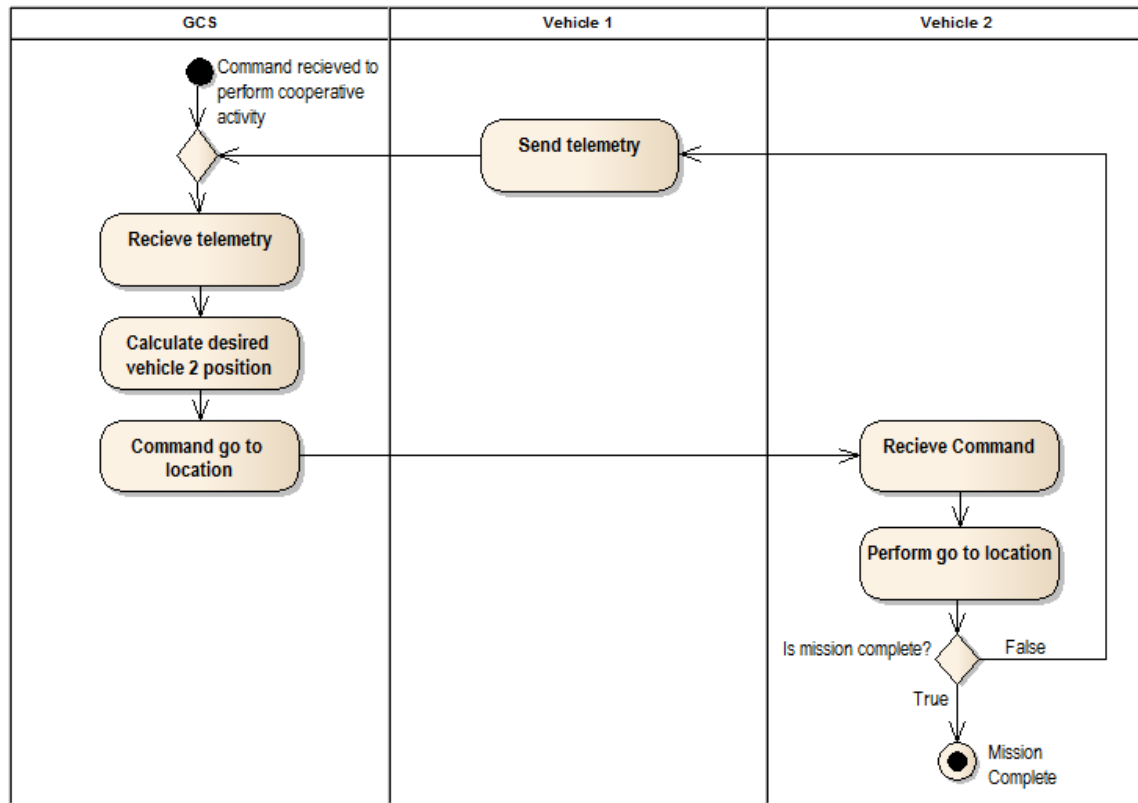


Figure 13. OV-5b: Perform Cooperative Activity.

The activities depicted in both OV-5b views are compiled into the OV-5a shown in Figure 14. Additional elements show in blue and outlined in black are derived

activities that are required to perform other activities contained in the OV-5b. The *Pass telemetry* and *Pass command* activities are derived from the need to pass information and commands to and from the remote vehicle through the relay vehicle for the communication relay use case. The *Maintain steady level movement* and *Provide navigation measurement* activities are derived from the need for the vehicle to stabilize itself and utilize sensors to perform waypoint based navigation. The *Maneuver vehicle* and *Move vehicle* activities are derived from the need for the vehicle to maneuver and propel itself during missions. Finally, the *Command idle maneuver* activity is derived from the need for the operator to sequence the launching of each vehicle by having them perform their idle maneuver to wait for further commands.

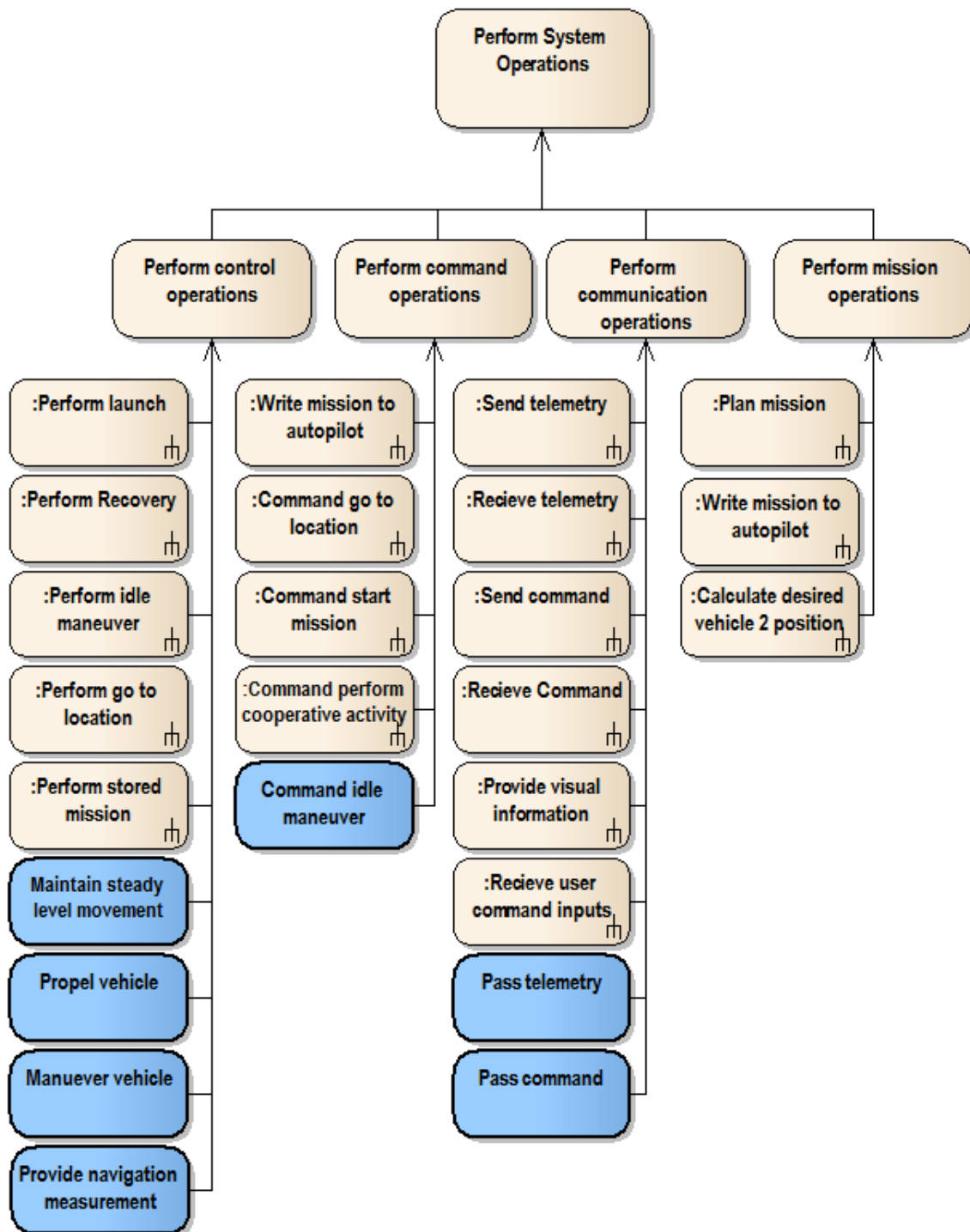


Figure 14. OV-5a: Operational Activity Decomposition Tree.

The activities depicted in the OV-5b and OV-5a views are written to be agnostic

towards vehicle type to allow the architecture to be applied to different types of vehicles. Table 4 below contains descriptions of how these activities are performed by each type of vehicle.

Table 4. Vehicle Agnostic Activity Descriptions for Each Type of Vehicle

Activity	Action by UGS	Action by Multi-Rotor UAS	Action by Fixed Wing UAS
Perform Launch	Start Driving	Initialize propellers. Lift off of ground and hover at designated altitude and position.	Perform a hand launch or rolling launch and fly in a helix until the designated altitude is reached.
Perform Recovery	Start Driving	Transit to the landing site. Hover over landing site. Reduce altitude until vehicle touches down. Stop propellers.	Transit to the landing site. Reduce altitude and speed. Land the UAS in the landing area.
Perform idle maneuver	Stop driving.	Stop all movement and hover at the desired position or perform a circular loiter around the location.	Perform a circular loiter maneuver around the specified location.
Perform go to location	Drive to the location and perform an idle maneuver	Fly to location and perform an idle maneuver.	Fly to location and perform an idle maneuver.

4.3 System Elements and Functions

The SV-1 is a graphical representation of the physical architecture. This representation displays the allocation of components to sub-systems and the information exchanged between components. This diagram will also aid in ensuring each component can receive the specified data type.

Figure 15 depicts the SV-1 view for the desired system. Each element depicted is either directly responsible for one or more of the activities described in the OV-5a, or is required to aid other elements in the accomplishment of their activities. These elements which aid other elements in accomplishing their activities include the COM port, Monitor, and mouse and keyboard. These specific elements aid both the operator and communication transceiver with passing commands and information to and from the GCS computer and GCS software.

From this diagram, the operator provides inputs to the GCS via the keyboard and mouse, and receives information from the GCS via the monitor. The GCS sends commands and receives telemetry from each vehicle via the GCS communication transceiver. Each communication transceiver is on a network with the other communication transceivers. This configuration is required to allow one vehicle to act as a relay vehicle if the other is out of range of the GCS. This case can be shown by breaking the information passage between the GCS and vehicle 1. Due to the networked communication transceivers, information being passed across this link can still be passed through the vehicle 2 communication transceiver to the GCS. Each vehicle is identical except for the telemetry and commands sent to the autopilot. This is displayed this way to show that no processing is accomplished on the vehicle, and only commands addressed to the vehicle will reach it's autopilot. Finally, in each vehicle, the autopilot provides actuation commands to each component and receives GPS position information from the GPS receiver.

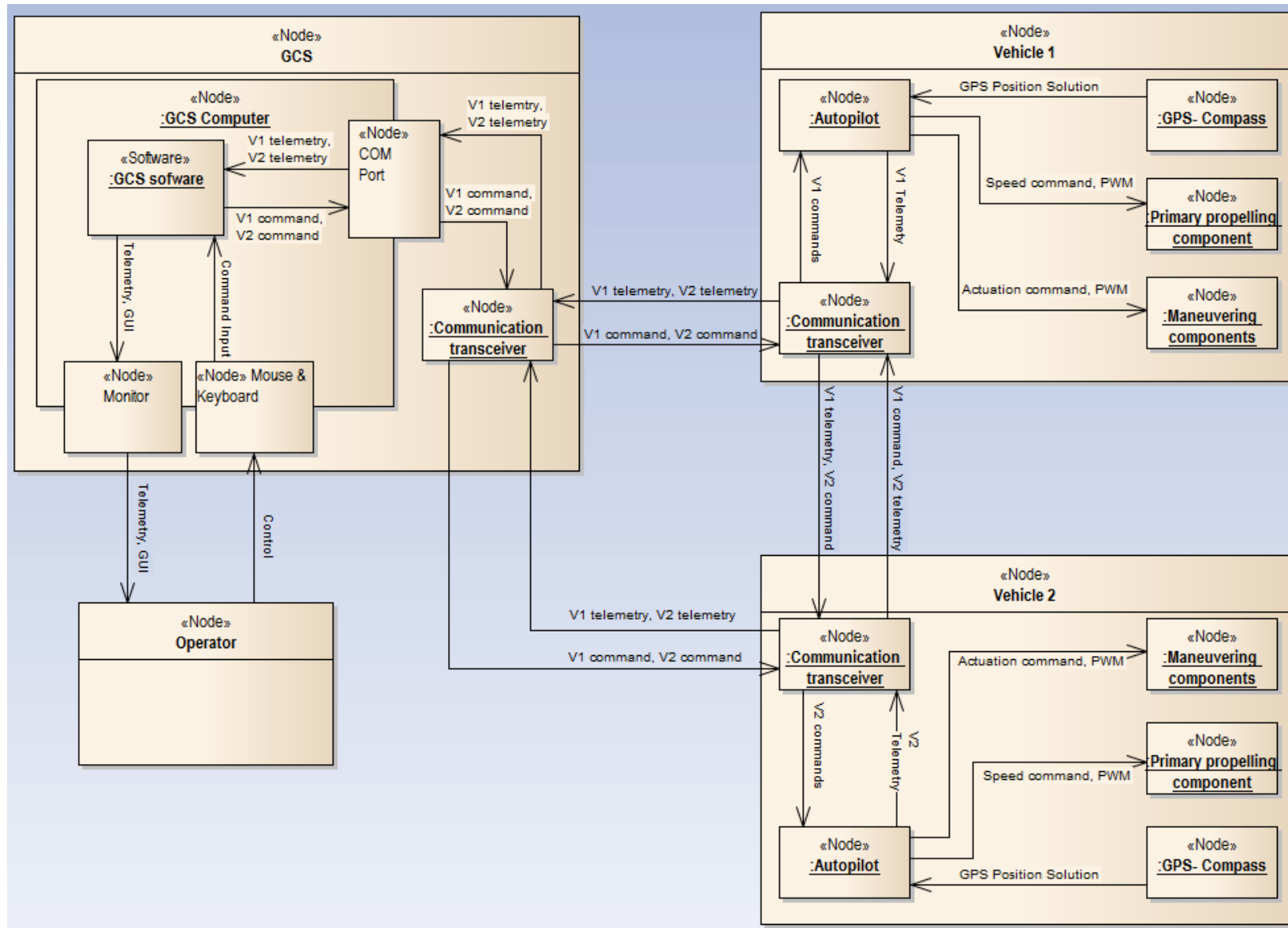


Figure 15. SV-1: System Interface Description.

The elements of the SV-1 are also defined by the system functions they perform. These elements are further refined into the system functions they perform in the SV-4 of Figure 16. Table 5 describes each of these functions based on the system element that performs the function, the required input, and the resulting output of each function.

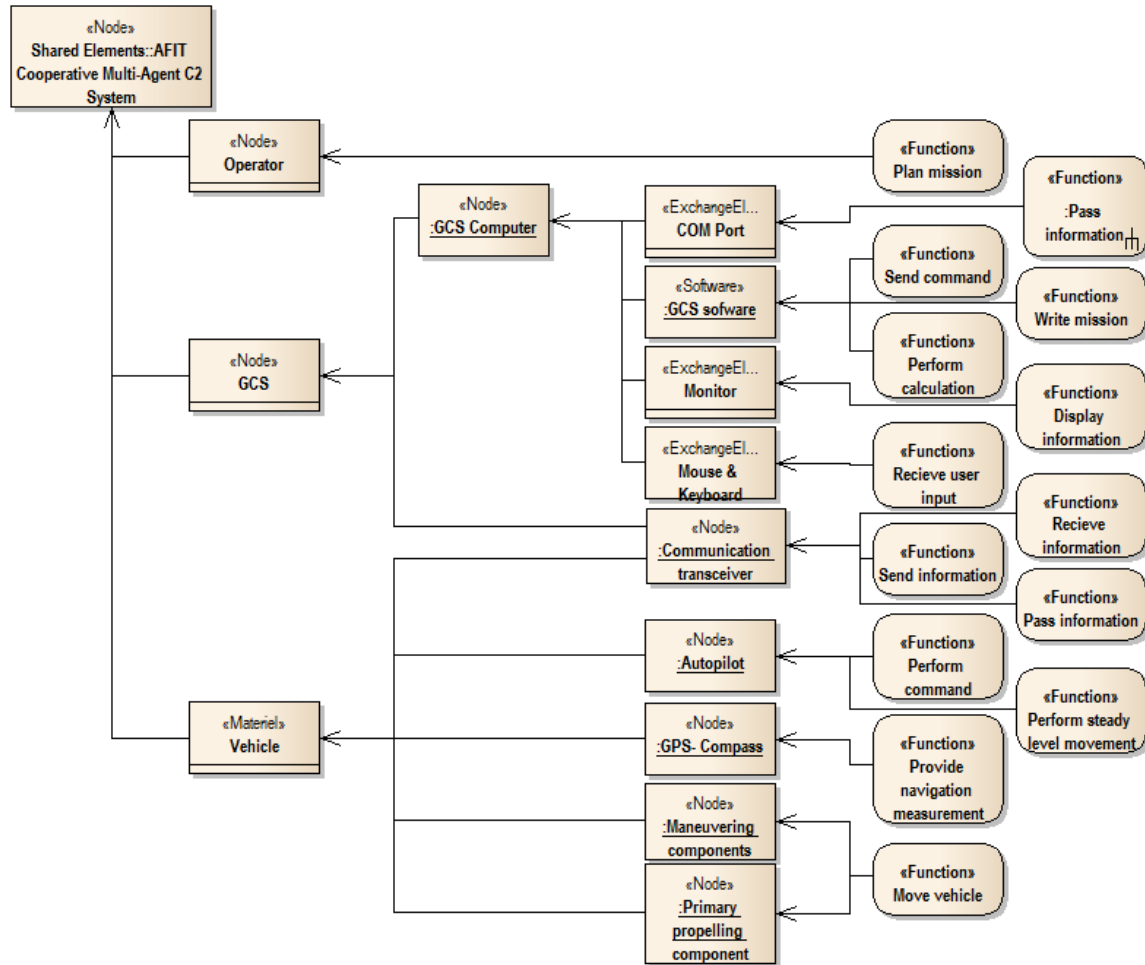


Figure 16. SV-4: System Functionality Description.

Table 5. System Function Descriptions

Function	System Element	Input	Output
Plan Mission	Operator	Mission specific requirements	Planned mission to be written to an autopilot
Send Command	GCS Software	Command to be sent and address of recipient	Command is sent to recipient
Write Mission	GCS Software	Mission to be written to autopilot	Confirmation that mission was written to autopilot
Perform Calculation	GCS Software	Required information for calculation to be performed	Commanded position to be sent to vehicle
Display Information	Monitor	Information from GCS to be displayed	Information is displayed
Receive User Input	Mouse and Keyboard	User mouse or keyboard input	Command in GCS software
Receive Information	Communication Transceiver	Information from sender and address of recipient	Information is received
Send Information	Communication Transceiver	Information and address of recipient	Information is sent
Pass Information	Communication Transceiver	COM Port	Information from sender to be passed and the address of the recipient Information is passed to recipient
Perform Command	Autopilot	Command from GCS	Vehicle actuator signals
Perform Steady Level Movement	Autopilot	Sensor measurements	Vehicle actuator signals
Provide Navigation Measurement	GPS- Compass	GPS signal and magnetic North signal	Sensor measurement
Move Vehicle	Maneuvering Components and Primary Propelling Component	Vehicle actuator signals	Vehicle movement

To ensure all of the operational activities are being completed by system elements, the SV-5a in Table 17 is used to map operational activities depicted in the OV-5a to system functions depicted in the SV-4. This view is used to both ensure each activity is able to be performed and ensure there are no extraneous components either completing the same activities or completing non required activities.

		System Function (SV-4)												
		Perform Command	Send Command	Receive Information	Send Information	Pass Information	Write Mission	Plan Mission	Perform Calculation	Display Information	Move Vehicle	Perform steady level movement	Provide Navigation Measurement	Receive User Input
Operational Activities (OV-5a)	Perform launch	X												
	Perform Recovery	X												
	Perform idle maneuver	X												
	Perform go to location	X												
	Perform stored mission	X												
	Command go to location		X											
	Command start mission		X											
	Command perform cooperative activity		X											
	Command idel maneuver		X											
	Send command		X											
	Receive telemetry			X										
	receive command			X										
	Send telemetry				X									
	Pass telemetry					X								
	Pass command					X								
	Write mission to autopilot						X							
	Plan Mission							X						
	Calculate desired vehicle 2 position								X					
	Provide visual information									X				
	Propel vehicle										X			
	Maneuver vehicle										X			
	Maintain steady level movement											X		
	Provide GPS solution												X	
	Receive user inputs													X

Figure 17. SV-5a: System Function Traceability Matrix.

4.4 Chapter Summary

In this chapter, architectural views were created to articulate the development of the system architecture to perform the conceptual operations described in the AV-1, OV-1, and brief use cases. First, multiple OV-5b views were developed to depict the logical sequence of activities required to perform the formation flocking and communication relay use cases. An OV-5a was also created to depict all of the activities depicted in the individual OV-5b views and how they related to the higher level operation of the system. An SV-1 and SV-4 were then developed to depict the system elements and functions required to accomplish the activities of the OV-5a and how these elements interact. Finally, an SV-5a was created to ensure each activity is being completed by a system function.

V. Results

5.1 Chapter Overview

In this section, the hardware and software selected to fulfill the roles and perform the functions developed in the previous chapter are discussed. Then, the C2 scripts to perform the formation flight and communication relay scenarios described in the previous chapter are developed. Finally, the results of the tests performed to validate and quantify the abilities of the system are outlined and analyzed.

5.2 Selected Hardware and Software

Communication System.

The Wave Relay MANET communication system was selected as the C2 link between the GCS and each vehicle. This COTS system utilizes a 2.3 GHz to 2.5 GHz radio frequency at up to 2.0 Watts to communicate with other nodes on the same IP network. The routing path for each link is optimized in real time based on the GPS location of each node and other factors proprietary to the system. Each node has the capability of routing data from other nodes to the desired IP address. This capability gives the system the ability to relay the C2 link from the GCS to a remote vehicle through an intermediate relay vehicle. An IP to TTL converter is used to convert the information being passed between the IP based Wave Relay node and the TTL based telemetry port of the autopilot. Additionally, this converter provides the autopilot with an IP address, allowing the GCS to connect via UDP or a virtual COM port.

Command and Control Software.

The command line based GCS software MAVProxy was selected as this system's primary GCS. This open source GCS software utilizes the MAVLink protocol to

communicate with the vehicle's autopilot through the C2 link. MAVProxy does not require a GUI like other GCS software, which reduces the processing power required and allows the GCS to run faster than other GUI based GCS. Also, this GCS software utilizes modules which increase the capabilities of the baseline GCS software. The one module used in this system is DroneAPI, which enables the GCS to achieve higher levels of autonomy through the use of Python 2.7 C2 scripts. These scripts allow the GCS to send *go to here* commands to its vehicle, receive telemetry data, and manipulate this information through the use of the full suite of tools available in Python 2.7. One downfall of this GCS and module is it does not allow one GCS and module pair to control more than one vehicle. Due to this, each vehicle requires its own instance of MAVProxy running DroneAPI. Finally, this GCS is also capable of sharing the vehicle's telemetry and command authority with other GCS software. On initialization, an IP socket is created to pass and receive this information. For the purposes of meeting requirements outlined in the AFIT Military Flight Release, Mission Planner is used as a heads up display for each vehicle during testing.

Autopilot.

The Pixhawk autopilot was selected to perform the autopilot element functions described in the previous chapter. This autopilot is composed of an open source chip set which utilizes inner loop stability control and outer loop waypoint navigation control algorithms to allow vehicles to perform autonomous missions. These inner and outer loop controllers fulfil the required *maintain steady level movement* function described in the previous chapter.

One method of waypoint control used by this autopilot is the guided waypoint, which is a single point that the vehicle will navigate to. Additionally, this waypoint can be sent from the C2 script to produce an updated navigation path for the vehicle.

The Pixhawk utilizes the MAVLink protocol, commonly used on many different open source ground control stations and Autopilots. This protocol allows the autopilot to both receive commands from the GCS and transmit telemetry down to the GCS. This functionality fulfills the *receive command* and *send telemetry* functions described in the previous chapter.

Another function of this autopilot is the ability to store missions from the GCS and perform stored missions. Waypoint based missions can be written to the autopilot's internal storage, allowing the autopilot to perform these missions when commanded to by the GCS.

Vehicles.

One of the goals of this system is to have the ability to control any small unmanned vehicle platform including UGS, multi-rotor UAS, and fixed wing UAS. Three COTS RC vehicles were chosen to fill these roles and each vehicle was retrofitted with the autopilot, sensors, and communication node described above. For the autopilot, each vehicle received a 3DR GPS/Compass. For the communication system, each vehicle received a GPS receiver, a 2.3 GHz to 2.5 GHz antenna, and an Ethernet to TTL converter.

For the UGS, as shown in Figure 18, a Traxxas E-Maxx RC truck was retrofitted with a component shelf which sits on the chase of the vehicle.

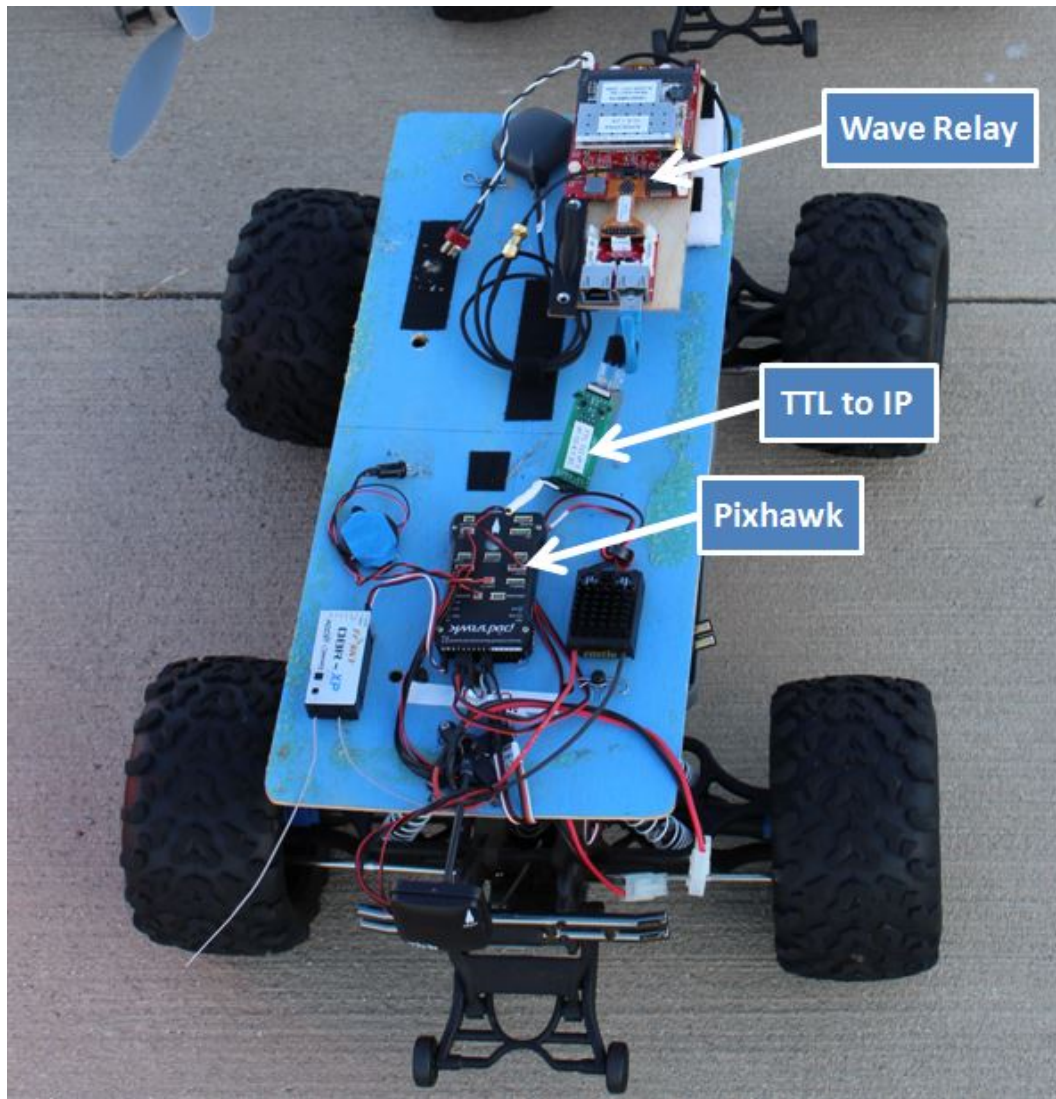


Figure 18. Traxxas E-Maxx UGS.

For the UAS multi-rotor aircraft, as shown in Figure 19, a 3DR X8 Octo-copter was used with no modifications to the base airframe.

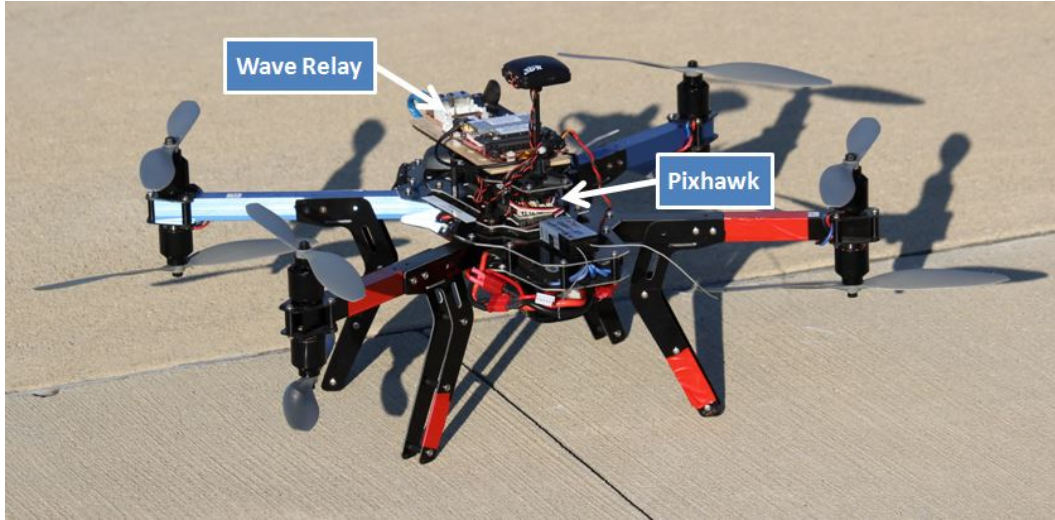


Figure 19. 3DR X8 Multi-Rotor UAS.

For the fixed wing UAS, as shown in Figure 20, a Banana Hobby Supper Sky Surfer was used. For this airframe, the rear control surface servos were moved from the center fuselage back to the tail, reducing the length and allowable bending of the wire connecting the servo arm to the surface control limb. Additionally, the Electronic Speed Control (ESC) was mounted to the bottom of the aircraft to increase the airflow across it and to reduce the space used in the fuselage.

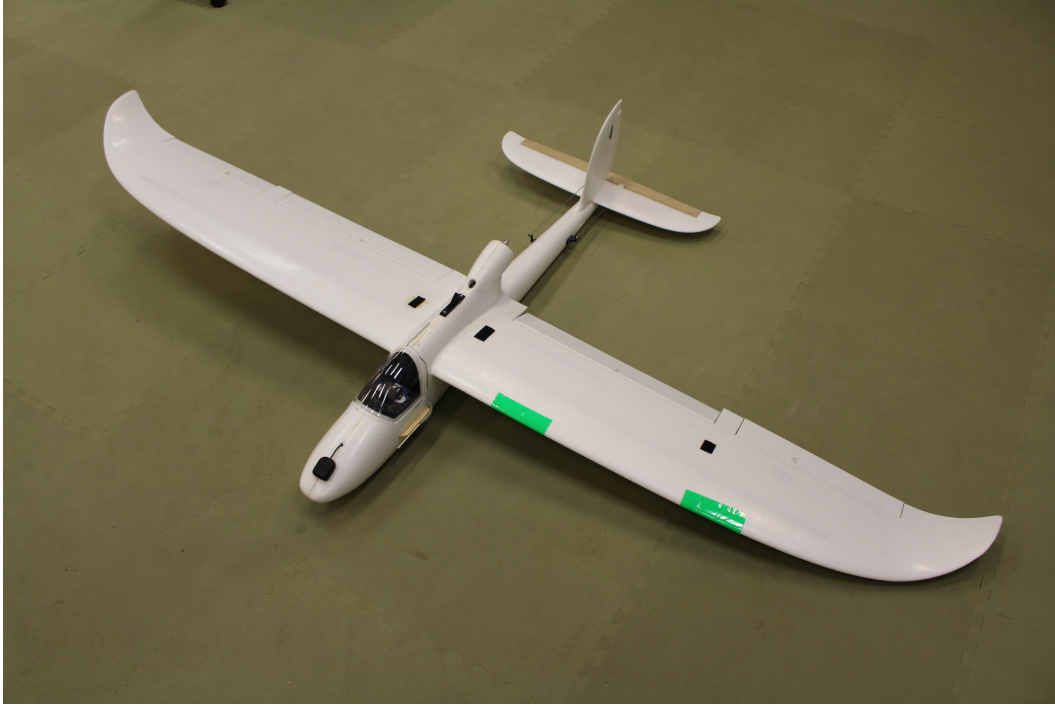


Figure 20. Super Sky Surfer Fixed Wing UAS.

5.3 Command and Control Software Development

After developing the system architecture and choosing the componentry required, it was discovered MAVProxy's DroneAPI module does not have the ability to control multiple vehicles from one C2 script. Due to this, each vehicle is controlled by its own instances of MAVProxy and DroneAPI C2 script. This section develops the methods of calculating the commanded position used to perform formation flocking and communication relay through the DroneAPI C2 scripts.

Commanding one vehicle to move to a specified point relative to the other vehicle is accomplished by sending a guided waypoint command from the GCS. This waypoint command progresses through two control states. The first control state forces the vehicle to change its ground course to navigate to the waypoint. The second state forces the vehicle to perform a loiter maneuver at the point. Depending on the type

of vehicle and rules set on the vehicle's autopilot, the vehicle will either stop inside a predesignated radius circle around the point or maneuver through the circle and performing a circle maneuver. In the case of the communication relay, it is desired the relay vehicle reaches this final state if no new positions are being sent.

In the case of maintaining formation during flight, it is desired the follower not perform a circle maneuver, as it would force the vehicle out of formation. In the case of the UGS and multi-rotor, the vehicle can stop in the circle and not inflict position error. However, the fixed wing aircraft is required to continue moving to maintain flight and will perform circular loiters, which can induce position error from the plane flying out of formation. This act of flying out of formation changes both the plane's heading and position, possibly also making it more difficult to recover the formation. Placing the waypoint forward of the desired position makes it less likely that the follower will reach the waypoint, and therefore reduces the probability of entering the final control state. An additional benefit of placing the waypoint forward of the desired position is it forces the follower to cut inside corners and catch up with a leader vehicle if it is out of range.

To avoid the end state described, the waypoint is placed forward of the desired position of the follower in the direction of the leaders ground course. The ground course is used to negate the effects of side slip with the multi-rotor and fixed wing UAS. Also, altitude will not be taken into account, as it will be fixed allowing each vehicle to operate freely in its own altitude plane without interference. Figure 21 depicts this method of control and is used to calculate the commanded waypoint. In this figure, the offset radius (r_{Offset}) is the radial distance from the desired follower position to the leader vehicle's position, the offset angle (θ_{Offset}) is the angular offset from the leader's ground course vector, the L_1 offset (L_1) is the forward offset along the leader's ground course vector, and the ground course angle (θ_{GC}) is the angle of

the ground course vector of the leader vehicle relative to north.

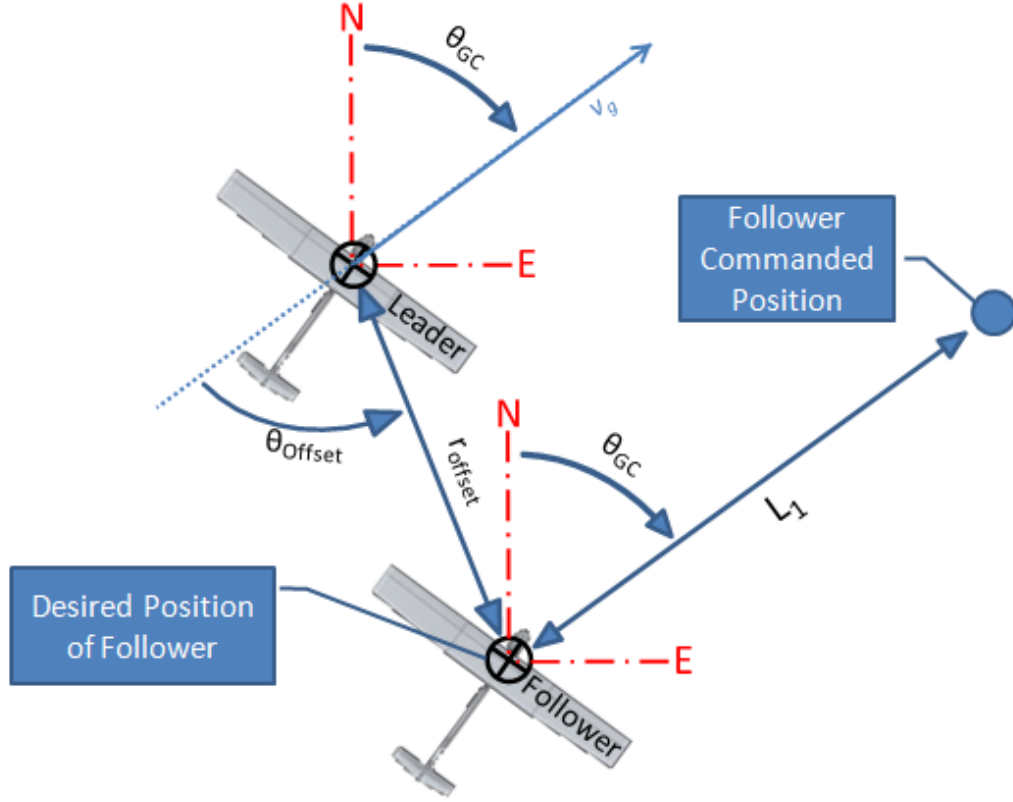


Figure 21. Follower Commanded Position Calculation Method.

One down side to a constant L_1 forward offset is that the follower is always commanded to fly ahead of its desired position. If the lead vehicle stops moving or slows, the follower will fly ahead of its desired position, forcing it out of formation. To mitigate this, the L_1 forward offset is defined as a function of the lead vehicles ground course velocity and a lead time constant, denoted as L_{1t} . The forward offset is now defined as the product of the ground course velocity and lead time constant. With this function, if the lead vehicle reaches a ground course velocity of zero, the follower will be commanded to maneuver directly to its desired position.

One python script for each vehicle is required to accomplish this method of control due to the required use of two instances of MAVProxy. For the case of formation flight,

these scripts consist of one leader sever script and one follower client script. For the case of the communication relay scenario, these scripts consist of one remote vehicle sever script and one relay vehicle client script.

The purpose of the leader server script is to acquire the position, ground course, and velocity of the leader vehicle, and provide that information to the follower client script through a UDP socket. The ground course is calculated using the vehicle's velocity in the X, Y, and Z directions relative to the body frame due to DroneAPI's inability to obtain the ground course directly from the vehicle's telemetry stream. The ground course of the vehicle is then rotated to be relative to North, the vehicle's position in latitude and longitude is provided relative to the geodetic frame (WGS-84), and the altitude is provided relative to the launch point of the vehicle. The python script created to perform this operation is contained in Appendix A

The purpose of the follower client script is to receive the information outlined from the leader server script through a UDP socket, calculate the next waypoint to send to the follower, and send the waypoint command to the follower autopilot. The python script created to receive the leader's information, call the offset position function, and send the resulting position to the follower autopilot is contained in Appendix B. The python function created to perform the commanded offset position calculation is contained in the function *follower_pos* in Appendix C.

The purpose of the remote vehicle sever script for communication relay is similar to the leader server script, with the difference being the information obtained and sent only includes the position of the remote vehicle. Also, for purposes of determining the midpoint for the relay vehicle to maintain, on starting the remote vehicle server script, the vehicle's first position is obtained and used as the position of the GCS. This script is contained in Appendix D.

The purpose of the relay client script is similar to the follower client script, with

the difference being the calculation performed to determine the desired position of the relay vehicle. The relay client script contained in Appendix E receives the leader’s information, calls the position function, and sends the guided waypoint command. The position is determined by finding the midpoint between the GCS and remote vehicle, and is contained in the function *relay_pos* in Appendix C.

The rate at which both of these scripts run is governed to control the amount of information stored in the UDP buffer. It was found during initial tests that if the server script is run at a higher rate than the client script, the buffer will get filled, causing the client script to send commands based on old information. For this reason, the client script is always run at twice the rate of the server to ensure the buffer does not fill.

5.4 Formation Flocking Test Results and Analysis

In this section, the results of the formation flocking tests outlined in the methodology section are discussed and analyzed. For these analyses, both the absolute position of each vehicle and the relative position error in meters will be used to display the collected data. The absolute position displayed in meters at a local level will be used to identify behavioral traits of the system while performing formation flocking, while the relative position error is used to quantify the system’s abilities. The relative position error is determined and displayed using the radial distance between the follower’s position and the follower’s desired position relative to the leader. Also, the relative position error of the follower forward and right of the desired position relative to the leader’s ground course is also displayed. These values are displayed at a higher level of precision than the GPS receiver can provide because the measurements are taken after the combination of the IMU and GPS solutions in the Kalman filter. For all tests except the fixed wing UAS tests, the position error will be calculated after the

follower has stabilized its position, typically about 50 seconds after the follower script is initiated. For all tests, the lead vehicle performs the specified path counter clockwise. Also, the GCS and GCS communication node was located on the East edge of each leader's path for all tests. All other test specific incidences or interferences will be outlined in each test's section.

UGS Following Multi-Rotor UAS.

The first set of tests performed included a team composed of an UGS in the role of the follower and a multi-rotor UAS in the role of the leader. This test was performed as described in the methodology section, with the exception of not performing circular paths. This change was caused by a time constraint while testing and the inability of the particular multi-rotor used to perform circular paths. The test parameters listed in Table 6 were varied with an offset radius of 2m and an offset angle of 0° .

Table 6. UGS Following Multi-Rotor UAS Test Parameter Matrix

Test Number	Flight Path	L_{1t}
1	Box	0s
2	Box	1s
3	Box	2s

Test 1: Box Path, $L_{1t} = 0s$.

For this run, a 15m box pattern was performed by the leader vehicle, with the follower vehicle using a 0s forward offset lead time. As shown in Figure 22, after allowing the follower to stabilize its position for 50s, the system achieved a mean radial position error of 3.43m with a standard deviation of 2.23m and a DRMS of 3.46m. Additionally, as shown in Figure 23, the system achieved mean forward and

right errors of -0.88m and 0.62m with standard deviations of 3.25m and 2.26m and DRMS of 2.84m and 1.98m. Figure 24 shows the path taken by both the leader and the follower, and the desired path of the follower based on the desired offset from the leader.

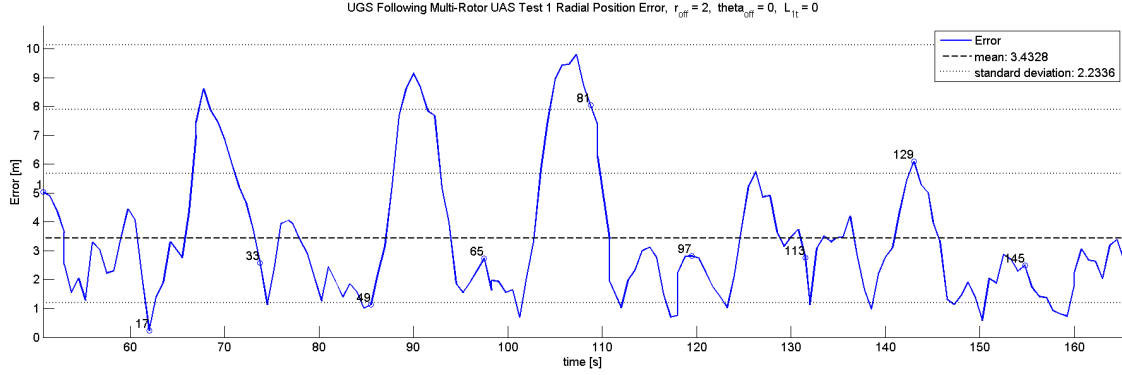


Figure 22. UGS Following Multi-Rotor UAS Test 1 Radial Position Error.

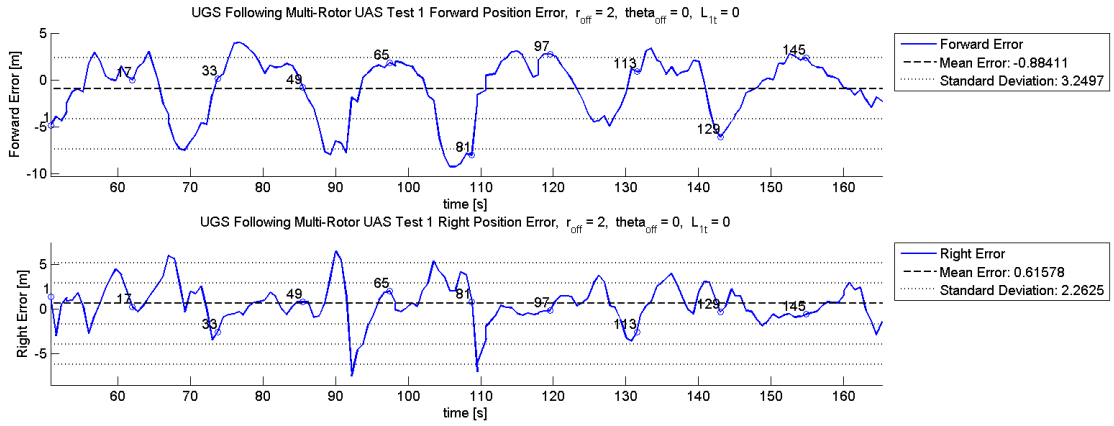


Figure 23. UGS Following Multi-Rotor UAS Test 1 Forward-Right Position Error.

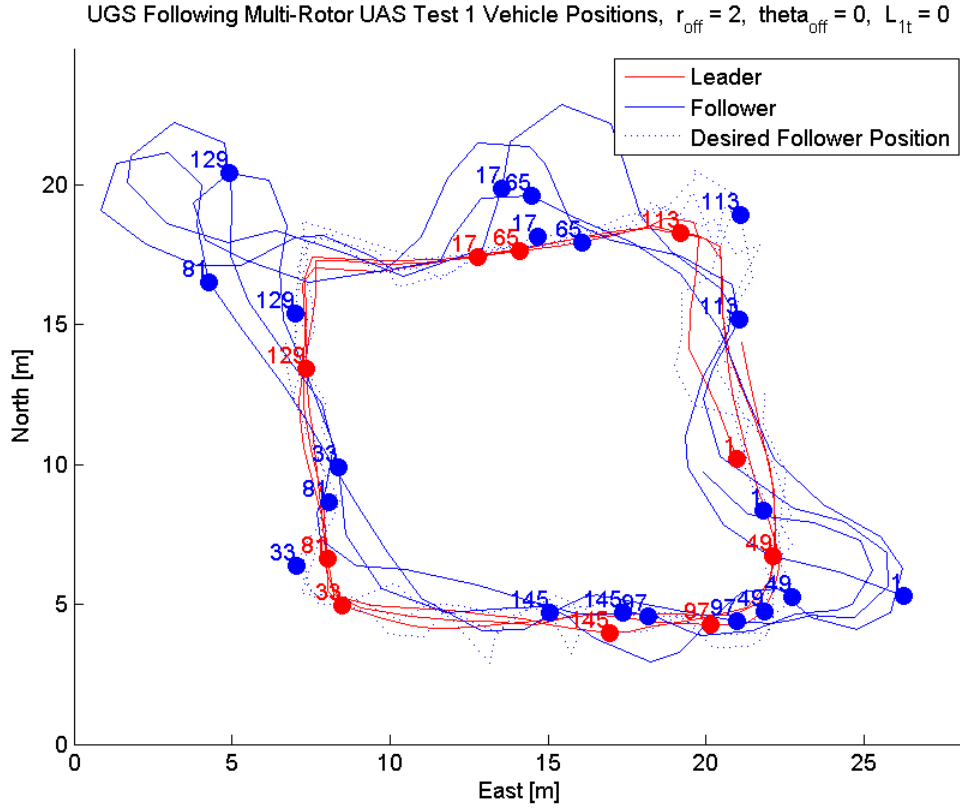


Figure 24. UGS Following Multi-Rotor UAS Test 1 Vehicle Position.

Test 2: Box Path, $L_{1t} = 1s$.

For this run, a 15m box pattern was performed by the leader vehicle, with the follower vehicle using a 1s forward offset lead time. As shown in Figure 25, after allowing the follower to stabilize its position for 50s, the system achieved a mean radial position error of 1.32m with a standard deviation of 0.83m and a DRMS of 1.40m. Additionally, as shown in Figure 26, the system achieved mean forward and right errors of 0.19m and 0.22m with standard deviations of 1.28m and 0.85m and DRMS of 1.16m and 0.79m. Figure 27 shows the path taken by both the leader and the follower, and the desired path of the follower based on the desired offset from the leader.

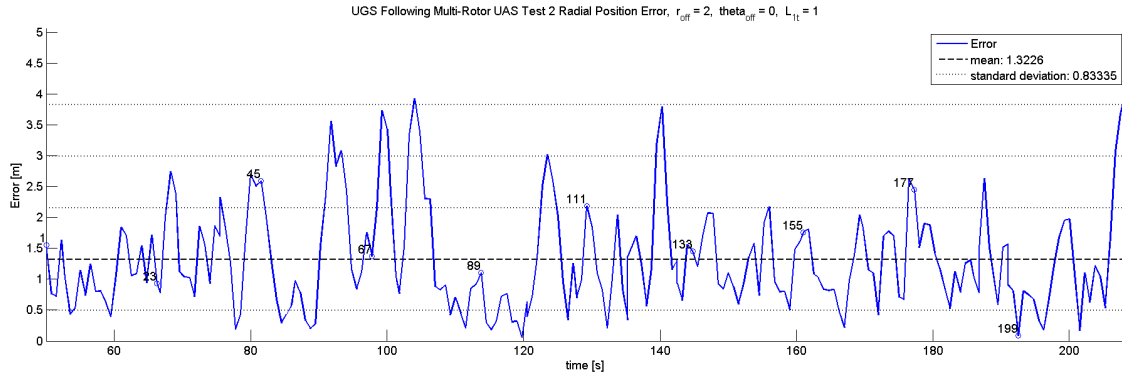


Figure 25. UGS Following Multi-Rotor UAS Test 2 Radial Position Error.

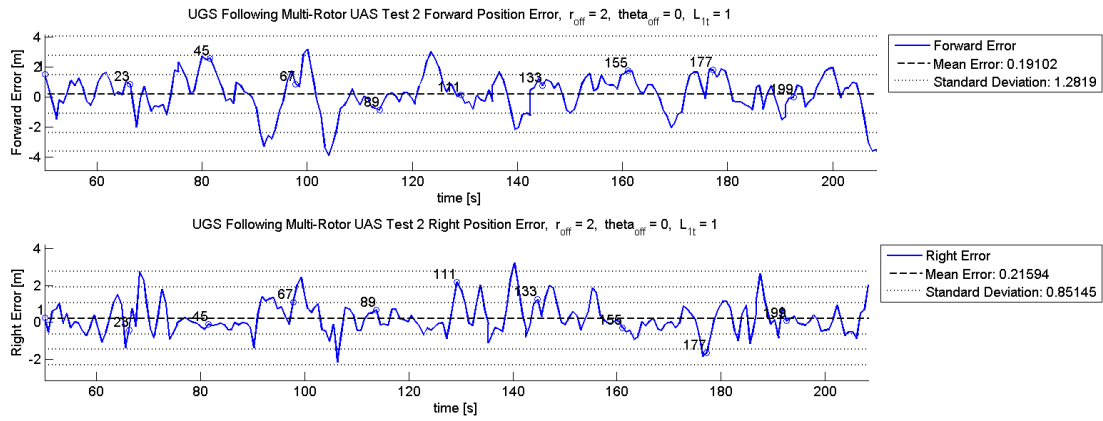


Figure 26. UGS Following Multi-Rotor UAS Test 2 Forward-Right Position Error.

Test 3: Box Path, $L_{1t} = 2s$.

64

multi-rotor and stop, indicating it reached the waypoint and was waiting for a new command.

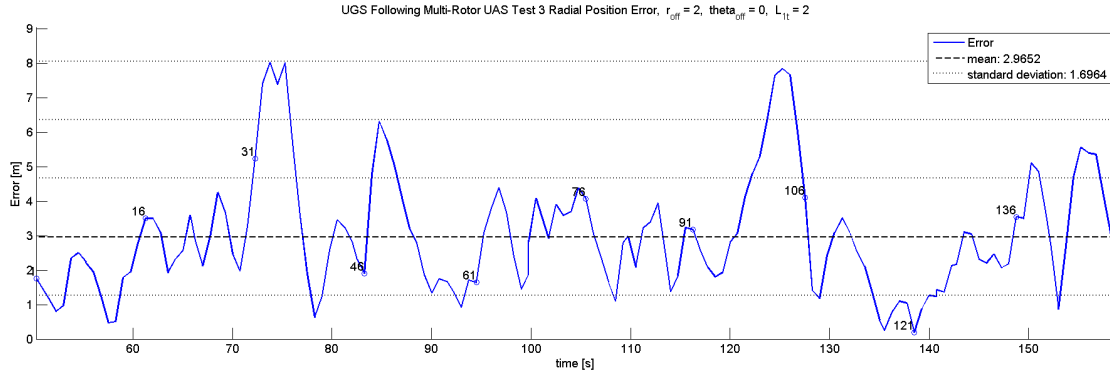


Figure 28. UGS Following Multi-Rotor UAS Test 3 Radial Position Error.

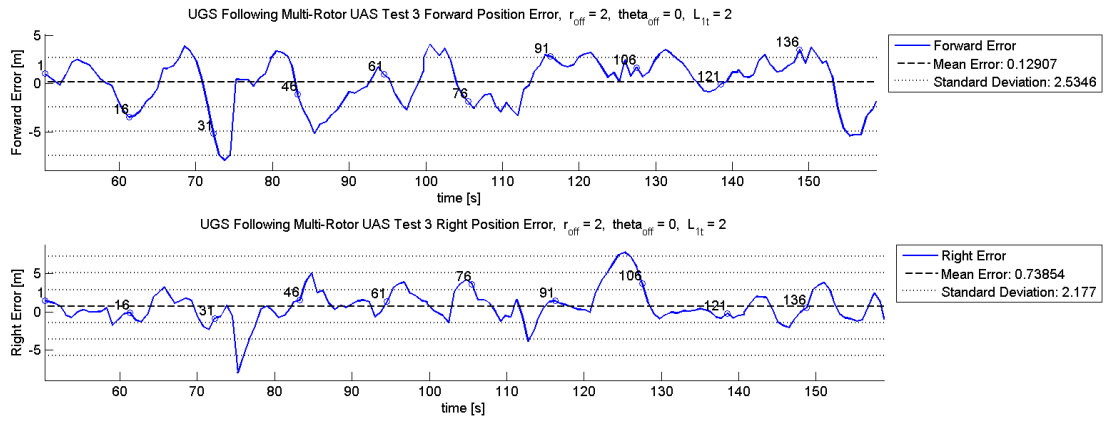


Figure 29. UGS Following Multi-Rotor UAS Test 3 Forward-Right Position Error.

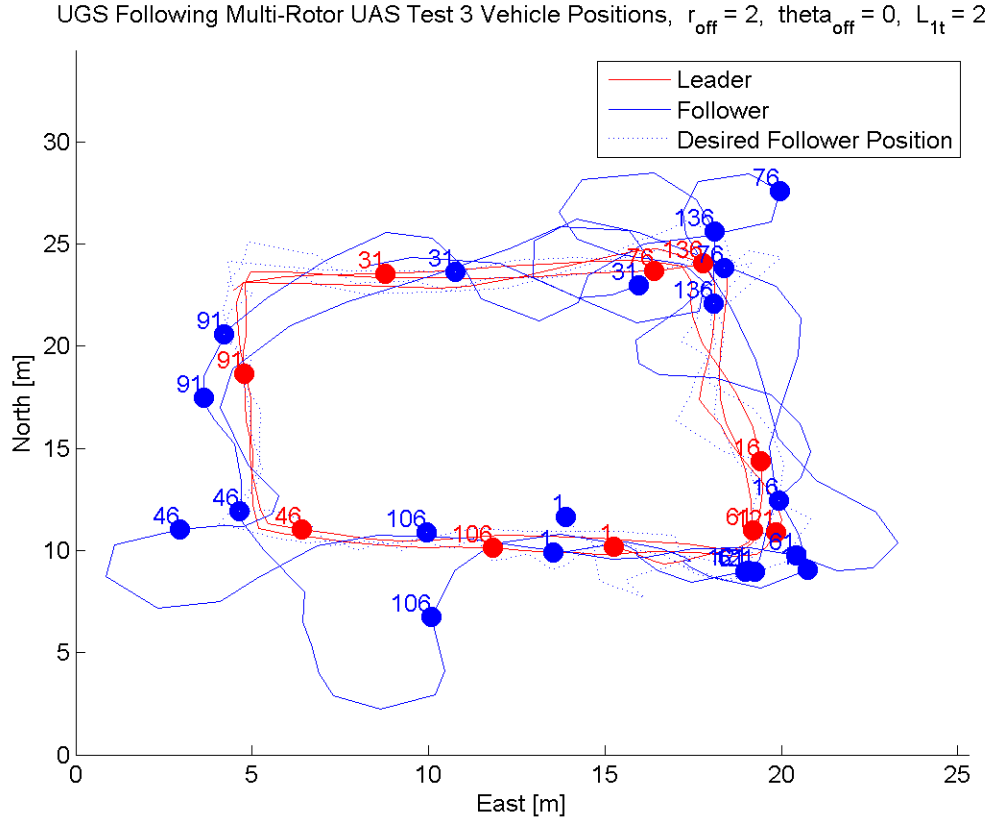


Figure 30. UGS Following Multi-Rotor UAS Test 3 Vehicle Position.

Team Analysis.

For this heterogeneous vehicle combination, the L_{1t} value of 1s resulted in the lowest position error for both the accuracy and precision measures. From this, it can be concluded that this team performs straight line paths with a higher level of positional accuracy and precision with this value of L_{1t} . The resulting measurements are summarized in Table 7 below.

Table 7. UGS Following Multi-Rotor UAS Test Results

	Forward Error (m)			Right Error (m)			Position Error (m)		
Test	μ	σ	DRMS	μ	σ	DRMS	μ	σ	DRMS
1	-0.88	3.25	2.84	0.64	2.26	1.98	3.43	2.23	3.46
2	-0.19	1.28	1.16	0.22	0.85	0.79	1.32	0.83	1.40
3	-0.13	2.53	2.07	0.74	2.18	1.88	2.97	1.70	2.79

One reoccurring issue seen during this test is the UGS's inability to maintain a low enough velocity to not drive past the desired position. This is caused by the motor and transmission's inability to provide the higher torque required to move at slower speeds, which for this specific platform is seen at velocities below 1.5 m/s. These factors also cause the vehicle to accelerate quickly from a stopped position. Additionally, the autopilot cannot command the vehicle to reverse or break, meaning if the vehicle comes to a point it needs to stop at, the autopilot reduces the throttle to zero and the vehicle rolls to a stop. The combination of these three factors cause the system to induce position error by driving or rolling past the desired positions.

Additional error is accrued due to the method the UGS uses to maneuver, meaning performing circular turns to change heading. After the previous issues occur and no new waypoint is received, the vehicle will perform a turning maneuver to navigate back to the passed point. This method of maneuvering can also induce errors at corners as seen at the North West corner at points 81 and 129 of Figure 24. At these corners, the vehicle drove through the commanded point, and after turning to the right to maneuver back to the point, received a new point further down the next leg of the path. This occurrence causes the vehicle to perform the equivalent of three right hand turns verse one left hand turn.

These errors are also shown to be induced more at corners, as indicated in the

spike in radial position error for tests 1 and 3. For test 1, three spikes in error occur at the North West corner and are due to the vehicle performing three right hand turns instead of performing a single left hand turn. For test 3, these spikes in error are also induced by the vehicles inability to perform a single left hand turn at the corners. Test 2 does not have noticeable spikes in error at corners due to the higher frequency of spikes across the entire time, which occurs at both turns and straight paths.

Finally, from the accuracy and precision measures for the forward and right errors, the system has less ability to maintain forward position opposed to right position. For each test, the standard deviation and DRMS of the right error are on average 0.59m and 0.48m lower than that of the forward error. However, as the L_{1t} increases, the forward mean error decreases, with no noticable trend in either dirrection for the standard deviation or DRMS. This means that the error ellipse formed by the standard deviation is centered closer to the desired location for higher values of L_{1t} .

Multi-Rotor UAS Following UGS.

The next set of tests performed included a team composed of a multi-rotor UAS in the role of the follower and a UGS in the role of the leader. This test was performed as described in the methodology section, with the test parameters listed in Table 10 varied with an offset radius of 2m and an offset angle of 0° . This test was performed in winds between 8 knots and 11 knots at between 170° and 210° .

Table 8. Multi-Rotor UAS Following UGS Test Parameter Matrix

Test Number	Flight Path	L_{1t}
1	Box	0 s
2	Box	1 s
3	Box	2 s
4	Circle	0 s
5	Circle	1 s
6	Circle	2 s

Test 1: Box Path, $L_{1t} = 0s$.

For this run, a 15m box pattern was performed by the leader vehicle, with the follower vehicle using a 0s forward offset lead time. The data collected had a gap of missing information making up the majority of the first 50s, so the follower was allotted an additional 50s to stabilize after the gap. As shown in Figure 31, after allowing the follower to stabilize its position for 50s after the data gap, the system achieved a mean radial position error of 4.24m with a standard deviation of 2.27m and a DRMS of 2.27m. Additionally, as shown in Figure 32, the system achieved mean forward and right errors of -3.19m and -1.16m with standard deviations of 2.93m and 1.89m and DRMS of 2.48m and 1.27m. Figure 33 shows the path taken by both the leader and the follower, and the desired path of the follower based on the desired offset from the leader.

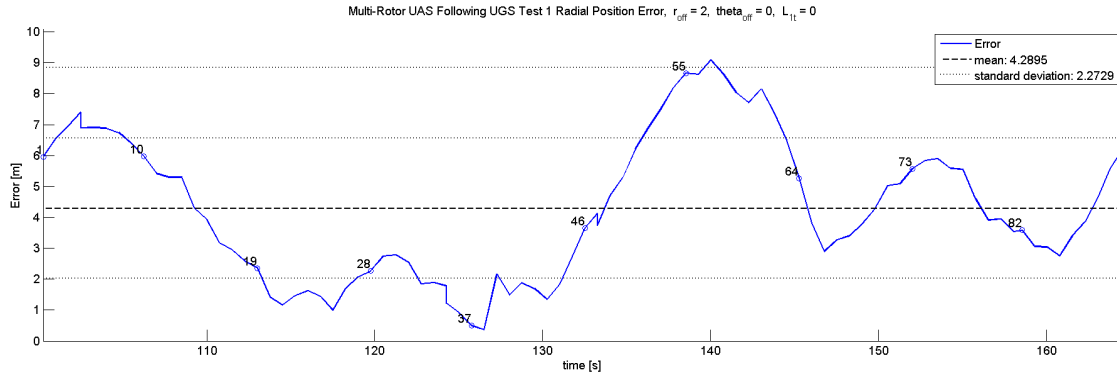


Figure 31. Multi-Rotor UAS Following UGS Test 1 Radial Position Error.

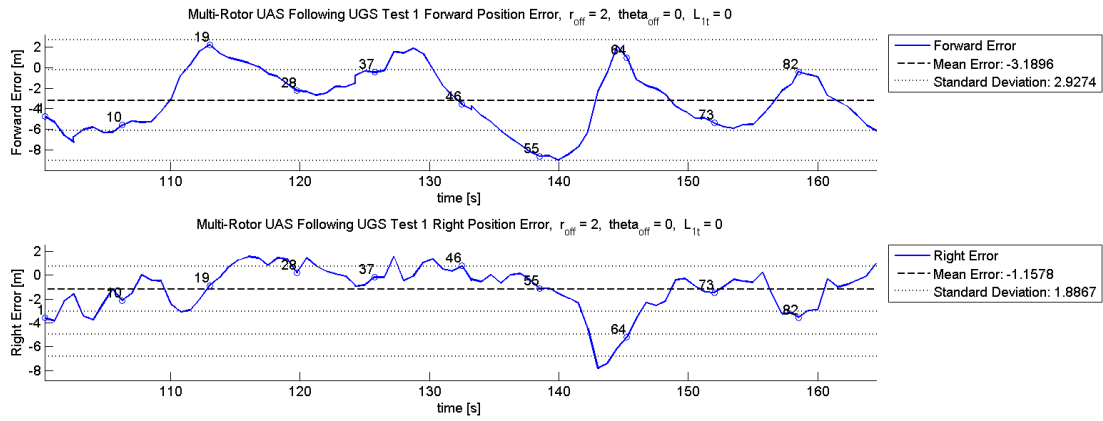


Figure 32. Multi-Rotor UAS Following UGS Test 1 Forward-Right Position Error.

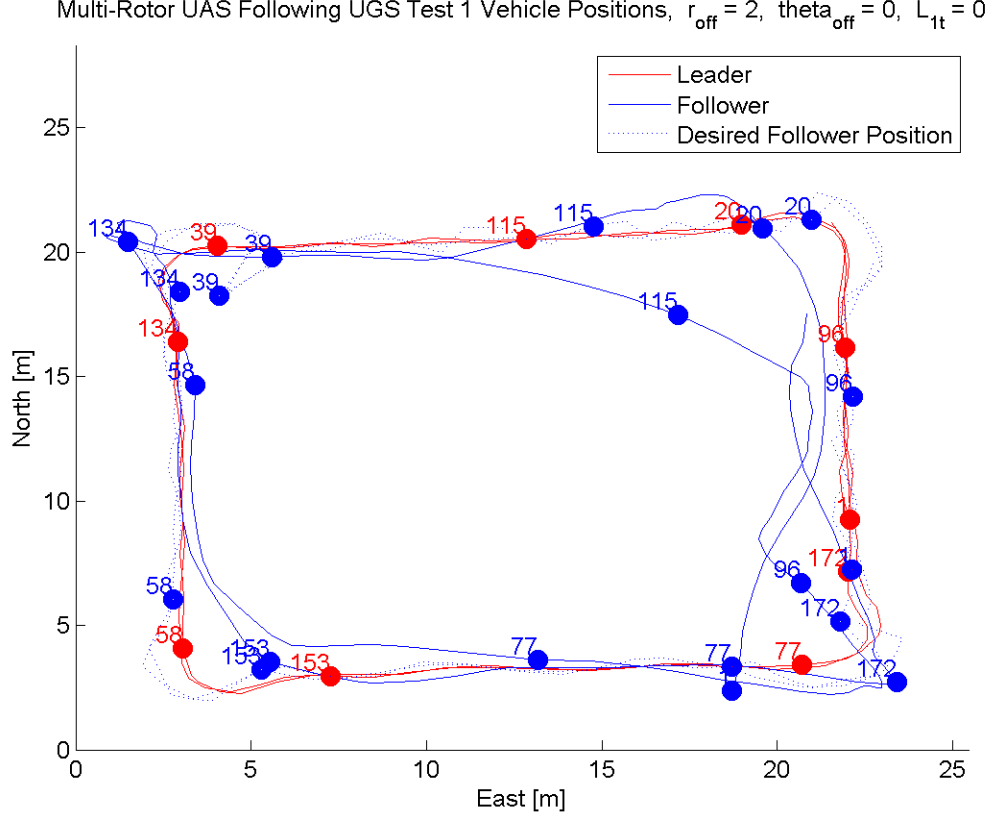


Figure 33. Multi-Rotor UAS Following UGS Test 1 Vehicle Position.

Test 2: Box Path, $L_{1t} = 1s$.

For this run, a 15m box pattern was performed by the leader vehicle, with the follower vehicle using a 1s forward offset lead time. At time 120s, a large gust of wind blew the lead vehicle off course causing a abnormally large spike in the error. As shown in Figure 34, after allowing the follower to stabilize its position for 50s and discounting errors after the gust of wind at time 120s, the system accomplished a mean radial position error of 1.59m with a standard deviation of 0.97m and a DRMS of 1.54m. Additionally, as shown in Figure 35, the system achieved mean forward and right errors of -0.85m and -0.17m with standard deviations of 1.29m and 0.92m and DRMS of 0.93m and 0.56m. Figure 36 shows the path taken by both the leader

and the follower, and the desired path of the follower based on the desired offset from the leader.

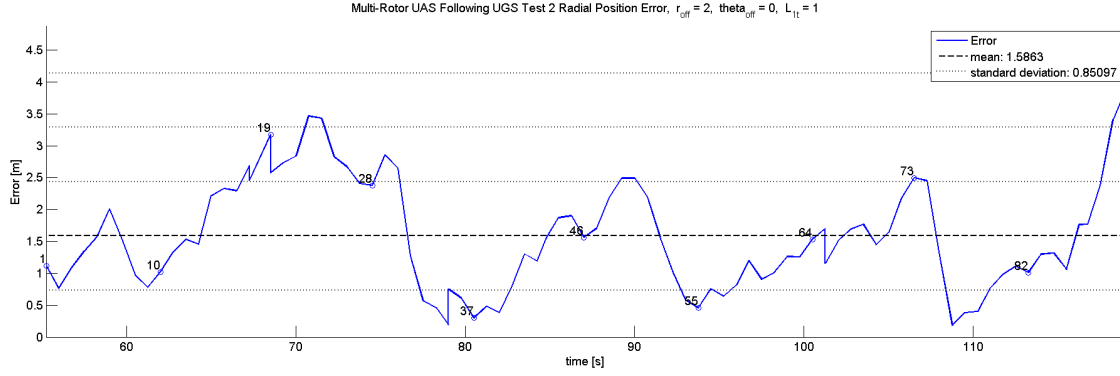


Figure 34. Multi-Rotor UAS Following UGS Test 2 Radial Position Error.

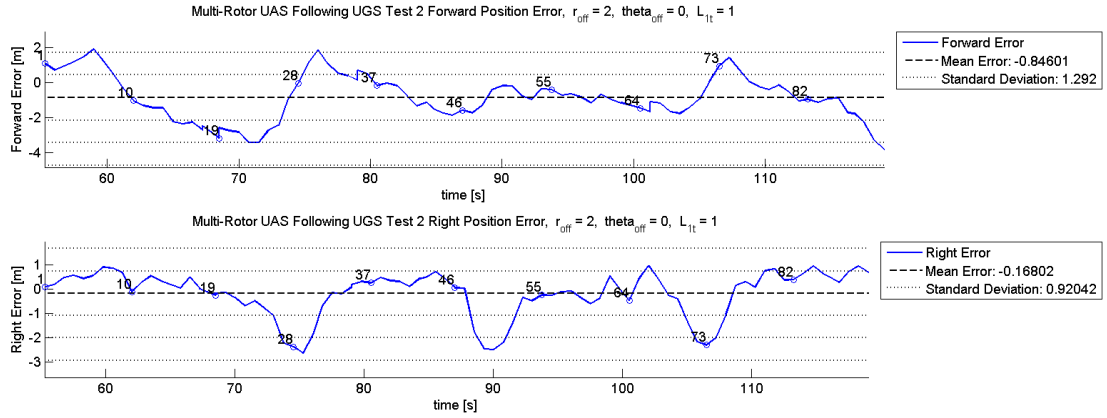


Figure 35. Multi-Rotor UAS Following UGS Test 2 Forward-Right Position Error.

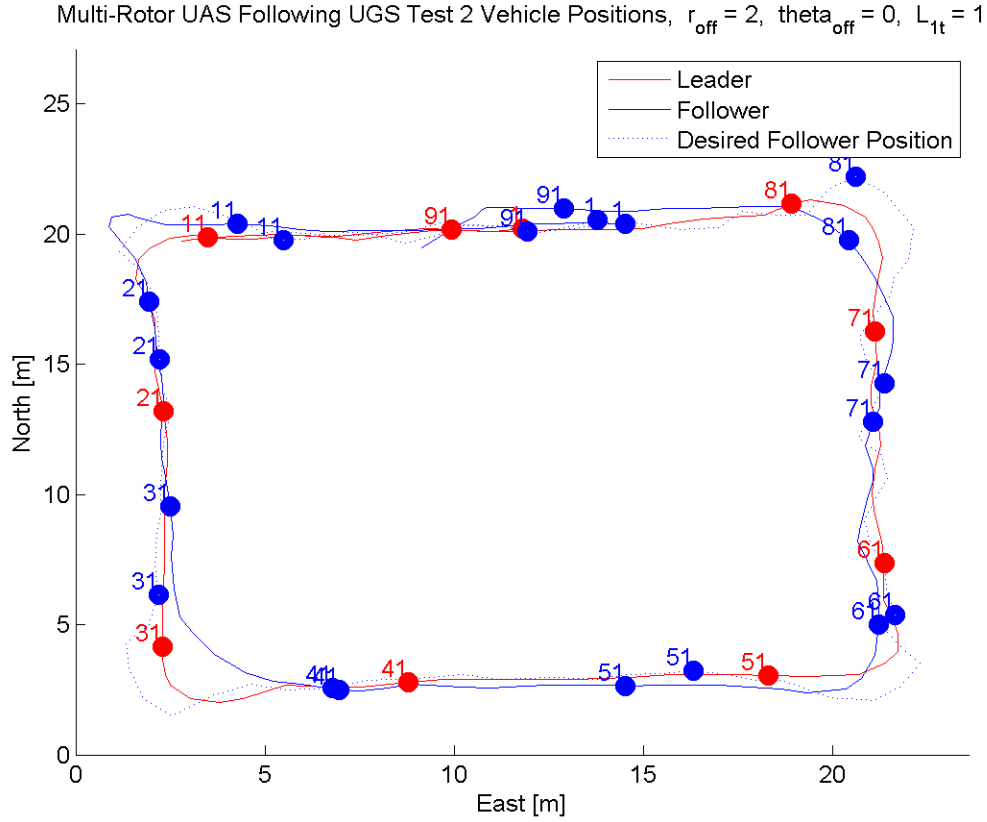


Figure 36. Multi-Rotor UAS Following UGS Test 2 Vehicle Position.

Test 3: Box Path, $L_{1t} = 2s$.

For this run, a 15m box pattern was performed by the leader vehicle, with the follower vehicle using a 2s forward offset lead time. As shown in Figure 37, after allowing the follower to stabilize its position for 50s, the system achieved a mean radial position error of 0.92m with a standard deviation of 0.43m and a DRMS of 0.90m. Additionally, as shown in Figure 38, the system achieved mean forward and right errors of -0.03m and -0.23m with standard deviations of 0.87m and 0.46m and DRMS of 0.77m and 0.46m. Figure 39 shows the path taken by both the leader and the follower, and the desired path of the follower based on the desired offset from the leader.

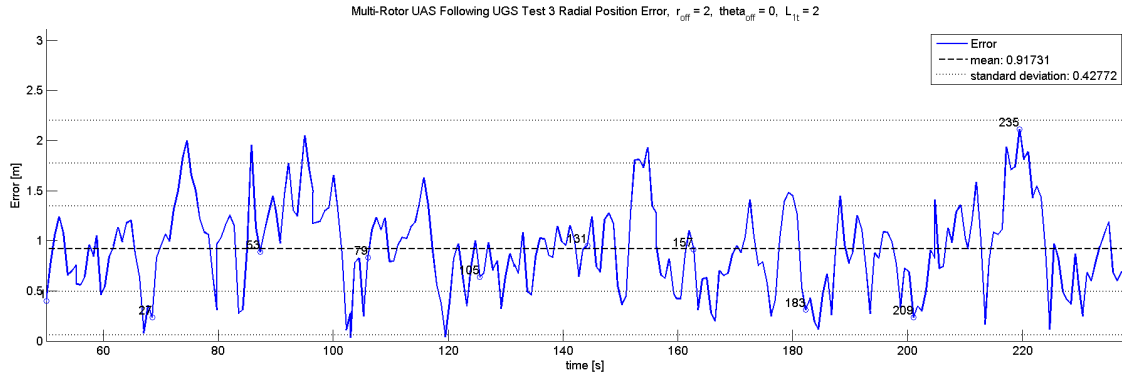


Figure 37. Multi-Rotor UAS Following UGS Test 3 Radial Position Error.

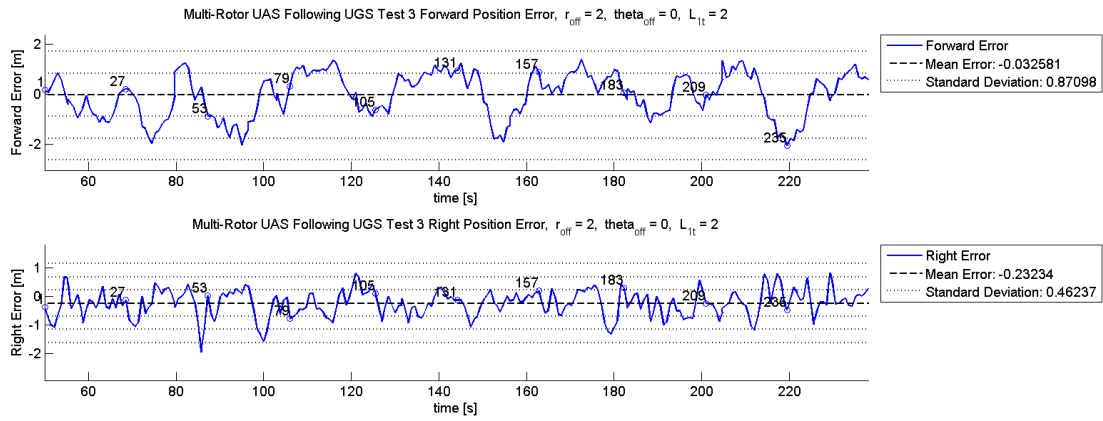


Figure 38. Multi-Rotor UAS Following UGS Test 3 Forward-Right Position Error.

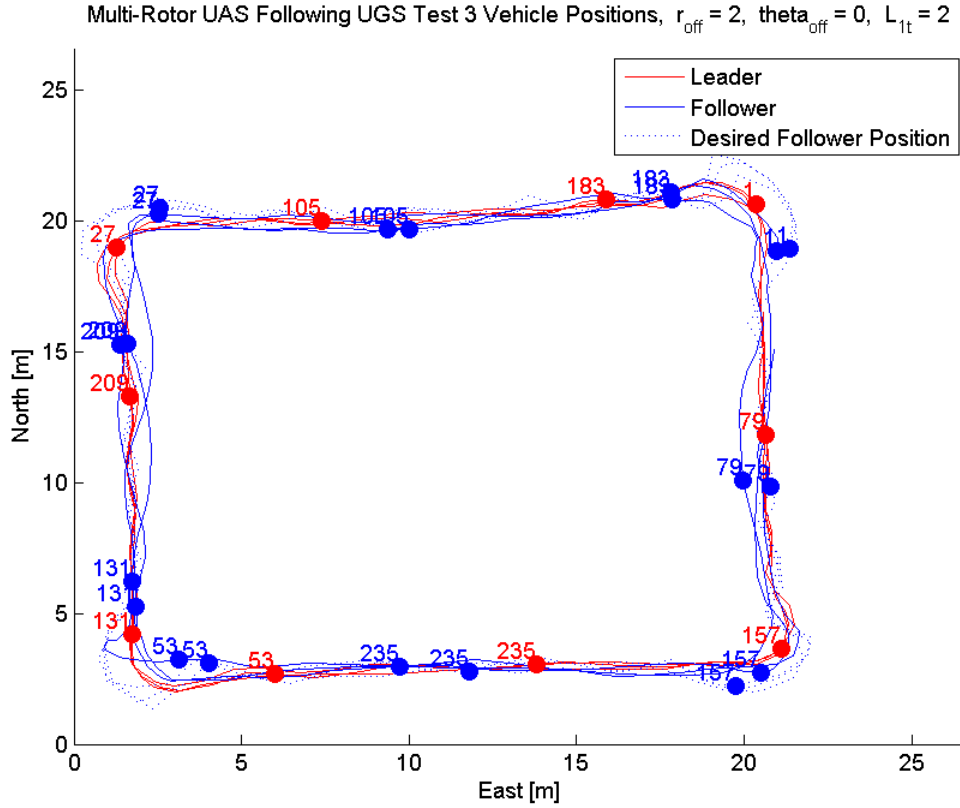


Figure 39. Multi-Rotor UAS Following UGS Test 3 Vehicle Position.

Test 4: Circle Path, $L_{1t} = 0\text{s}$.

For this run, a 15m radius circle pattern was performed by the leader vehicle, with the follower vehicle using a 0s forward offset lead time. During this test, the UGS leader stopped at approximately 165s due to a malfunction in its waypoint following operation and the operator had to manually override the truck. The remainder of the data after this point was removed due to it not allowing an additional 50s for the follower to stabilize. As shown in Figure 40, after allowing the follower to stabilize its position for 50s and removing the data after 165s, the system achieved a mean radial position error of 1.94m with a standard deviation of 0.97m and a DRMS of 1.54m. Additionally, as shown in Figure 41, the system achieved mean forward and

right errors of -0.50m and -0.42m with standard deviations of 1.45m and 1.47m and DRMS of 1.09m and 1.09m. Figure 42 shows the path taken by both the leader and the follower, and the desired path of the follower based on the desired offset from the leader.

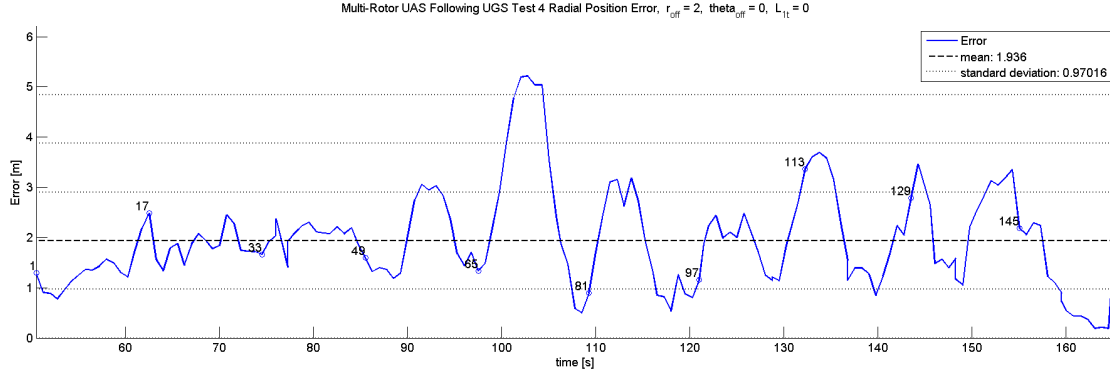


Figure 40. Multi-Rotor UAS Following UGS Test 4 Radial Position Error.

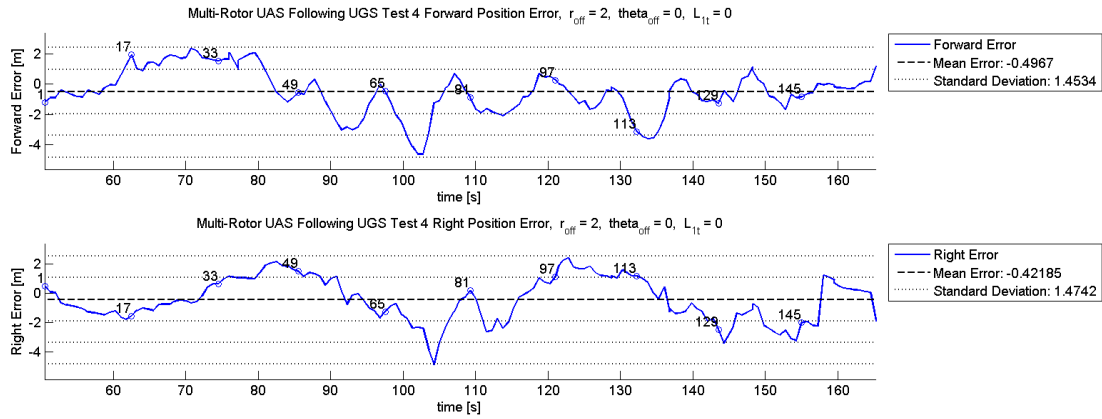


Figure 41. Multi-Rotor UAS Following UGS Test 4 Forward-Right Position Error.

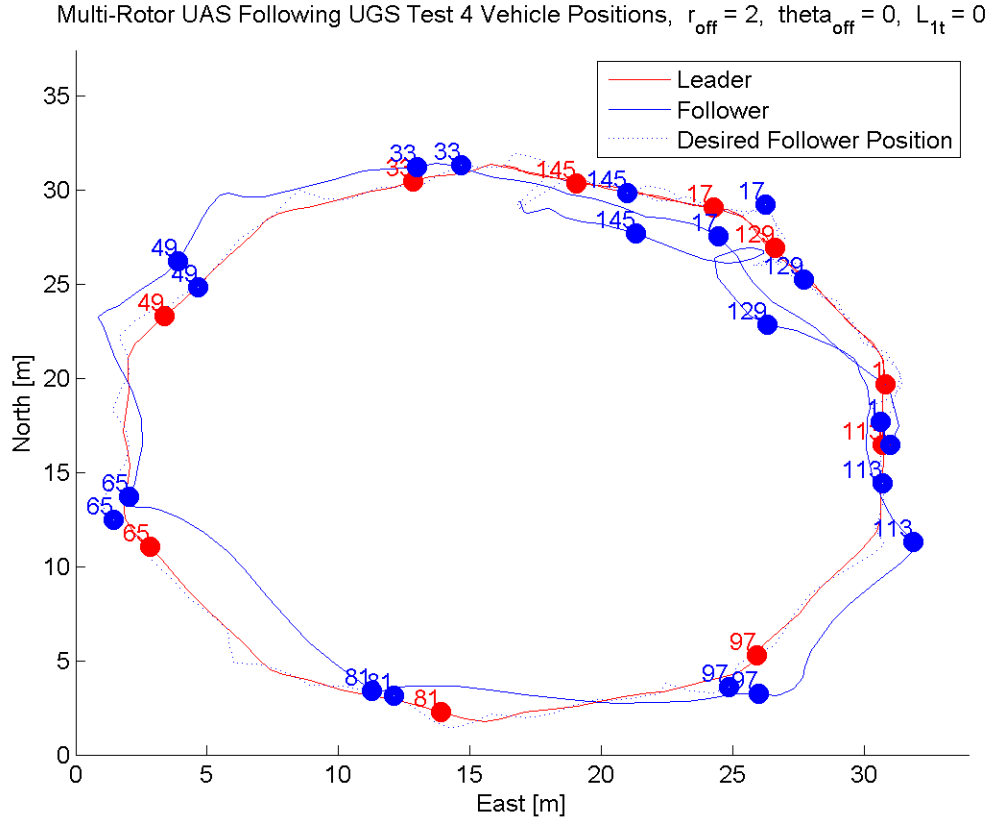


Figure 42. Multi-Rotor UAS Following UGS Test 4 Vehicle Position.

Test 5: Circle Path, $L_{1t} = 1s$.

For this run, a 15m radius circle pattern was performed by the leader vehicle, with the follower vehicle using a 1s forward offset lead time. As shown in Figure 43, after allowing the follower to stabilize its position for 50s, the system achieved a mean radial position error of 1.96m with a standard deviation of 1.72m and a DRMS of 2.13m. Additionally, as shown in Figure 44, the system achieved mean forward and right errors of -0.92m and -0.88m with standard deviations of 1.72m and 1.49m and DRMS of 1.60m and 1.41m. There is a noticeably longer stabilization time for this test, so after a total of 100s of stabilization the system achieved a mean position error of 1.01m with a standard deviation of 0.45m and a DRMS of 0.68m. Figure 45

shows the path taken by both the leader and the follower, and the desired path of the follower based on the desired offset from the leader.

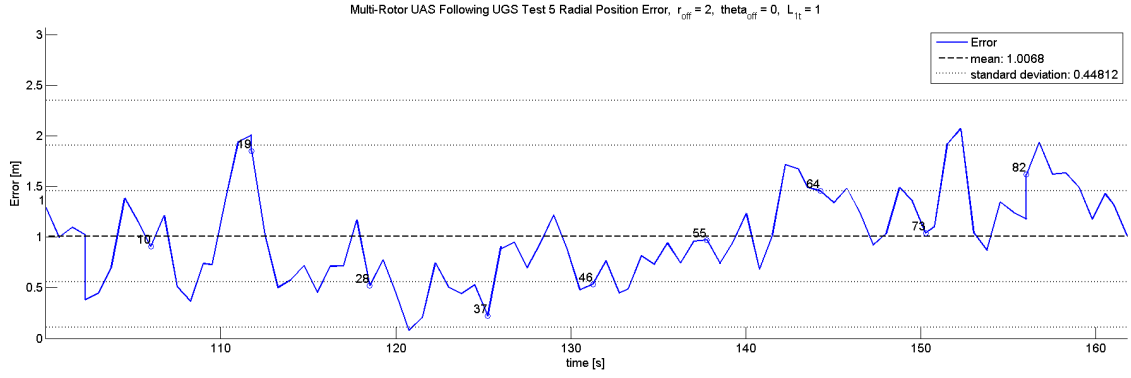


Figure 43. Multi-Rotor UAS Following UGS Test 5 Radial Position Error.

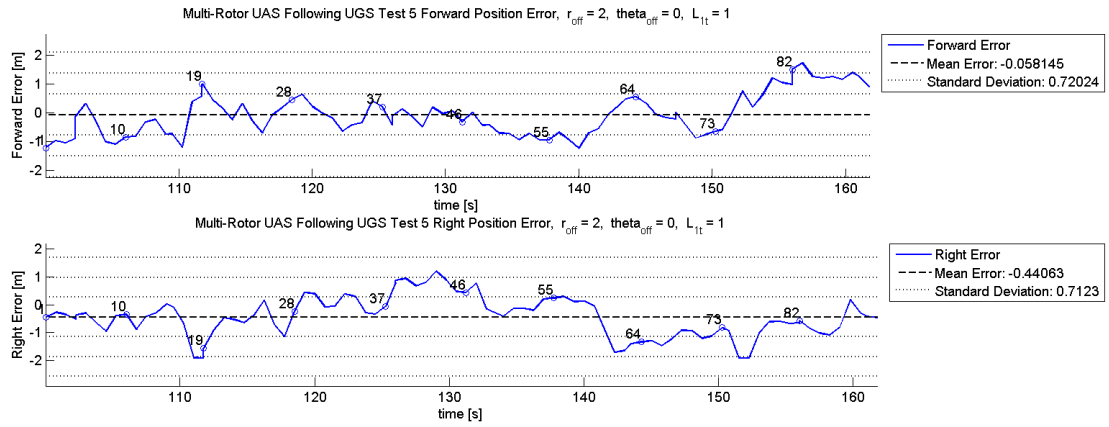


Figure 44. Multi-Rotor UAS Following UGS Test 5 Forward-Right Position Error.

point of the vehicle recovering a stable position, representing the best estimate for the mean and standard deviation for this configuration with no malfunction in the leader's operation. Before removing these errors the system achieved a mean position error of 6.22m with a standard deviation of 5.2m. After removing these error plateaus, the mean position error dropped to 1.47m with a standard deviation 1.01m and a DRMS of 1.34m. Additionally, as shown in Figure 47, the system achieved mean forward and right errors of -3.50m and 5.28m with standard deviations of 3.89m and 4.15m and DRMS of 3.03m and 4.18m. This measure does appear to be reasonable based on the close results of the two previous tests. However, this measure does have a lower level of confidence due to the manipulation of the data described and the reduced number of data points available. Figure 48 shows the path taken by both the leader and the follower, and the desired path of the follower based on the desired offset from the leader.

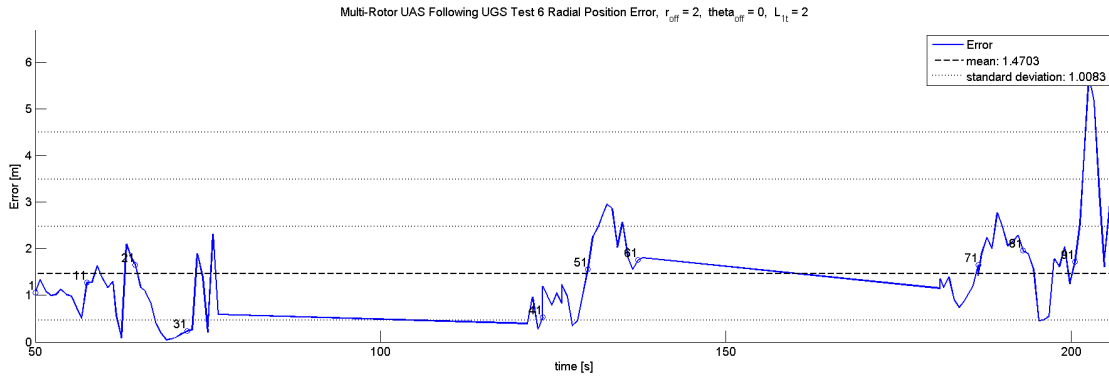


Figure 46. Multi-Rotor UAS Following UGS Test 6 Radial Position Error.

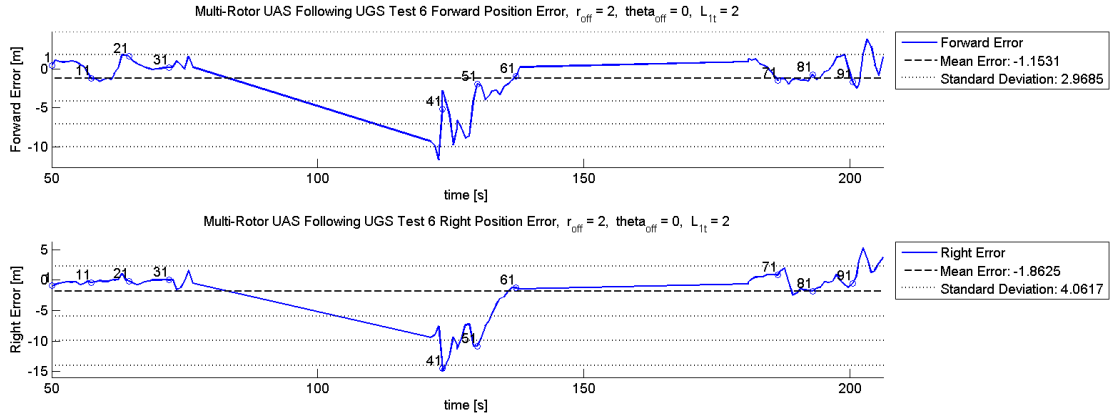


Figure 47. Multi-Rotor UAS Following UGS Test 6 Forward-Right Position Error.

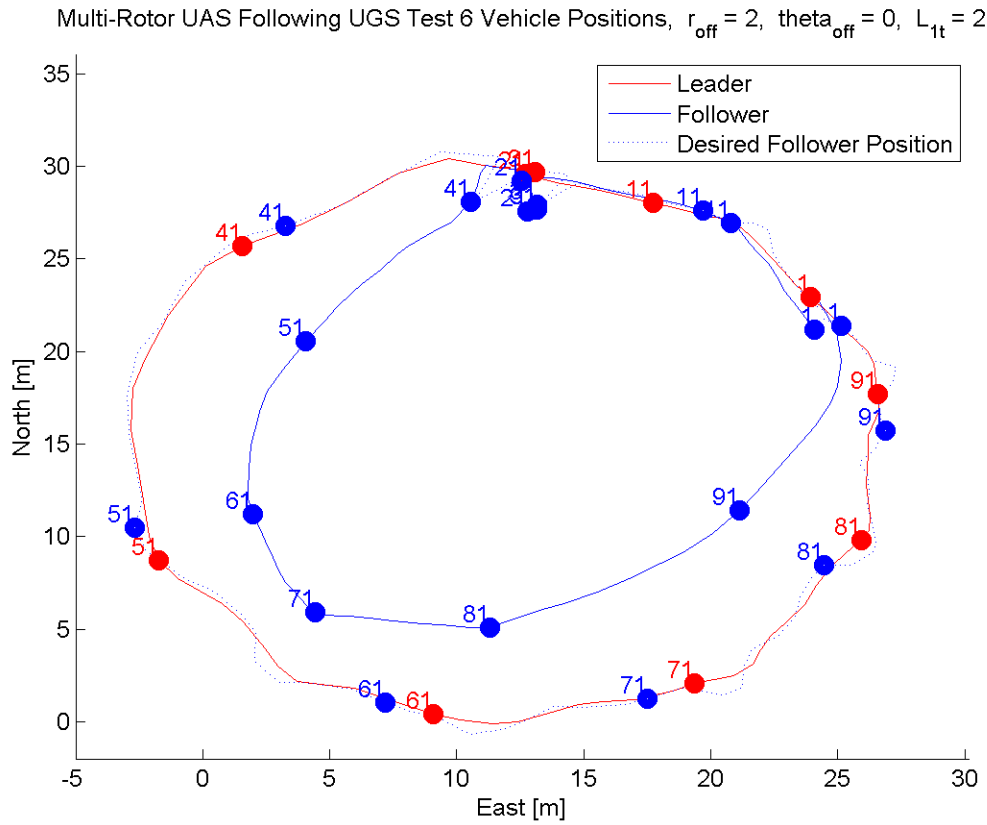


Figure 48. Multi-Rotor UAS Following UGS Test 6 Vehicle Position.

Team Analysis.

For this heterogeneous team, an L_{1t} value of 2s with the leader performing a box path resulted in the lowest radial mean position error and standard deviation. For this path, as L_{1t} increased, the mean position error, standard deviation, and DRMS decreases. The opposite was indicated for a circular path, which an L_{1t} value of 0s produced the best results in the forward and right directions, with the radial position results varying only. Also, it is shown that as L_{1t} increases, the mean position error, standard deviation, and DRMS increase. The resulting measurements are summarized in Table 9 below.

Table 9. Multi-Rotor UAS Following UGS Test Results

	Forward Error (m)			Right Error (m)			Position Error (m)		
Test	μ	σ	DRMS	μ	σ	DRMS	μ	σ	DRMS
1	-3.19	2.93	2.48	-1.16	1.89	1.27	4.24	2.27	2.27
2	-0.85	0.87	0.93	-0.17	0.92	0.56	1.59	0.85	1.08
3	-0.03	0.87	0.77	-0.23	0.46	0.46	0.92	0.43	0.90
4	-0.50	1.45	1.09	-0.42	1.47	1.09	1.94	0.97	1.54
5	-0.92	1.72	1.60	-0.88	1.49	1.41	1.96	1.72	2.13
6	-1.15	2.97	0.97	-1.86	4.06	0.92	1.47	1.01	1.34

Based on the results for the box pattern, this team performs straight line paths with a higher level of accuracy and precision for higher values of L_{1t} . One possible cause of this is as L_{1t} increases, the follower will cut inside corners more due to the commanded point being projected further forward from the follower's desired position. Also, with the point projected further, the follower has more time to maneuver to the point, as opposed to closer points which may require the follower to perform more

drastic maneuvers.

The team's ability to perform curved paths is, however, inversely affected by higher values of L_{1t} based on the results for the circle pattern. As the value of L_{1t} increases, the path created by the commanded points becomes larger than the path of the lead vehicle. This increased path radius causes the follower vehicle to fall behind due to each vehicle having a common cruise velocity.

Unlike a UGS or fixed wing aircraft, the method the multi-rotor airframe uses to maneuver allows the vehicle to move in any direction without performing turns. If the vehicle flies past a waypoint or is commanded to fly to a point off of the current ground course, it can simply pitch or roll in either direction to maneuver to the point. Due to this higher level of maneuverability, the vehicle is able to achieve a higher level of precision and accuracy when acting as the follower vehicle.

One possible cause of error can be seen at the corners of the box pattern for L_{1t} values greater than zero. As the UGS makes the turns, the desired follower position swings outside the box, creating an elbow on each corner. The higher this value gets, the more the elbow protrudes off the path, which can most easily be seen in Figure 39. The added length to the flight path can force the vehicle to fall behind. However, due to reasons stated above, the follower does cut some corners instead of following this path, allowing the follower to catch up instead of fall behind on the longer elbow path.

Finally, from the accuracy and precision measures for the forward and right errors, the system has less ability to maintain forward position opposed to right position while performing straight paths, as indicated by the box pattern. For the box tests, the standard deviation and DRMS of the right error are on average 0.61m and 0.63m lower than that of the forward error. Also, as the L_{1t} increases, the forward mean error, standard deviation, and DRMS decreased. This means that the error ellipse

formed by both the standard deviation is centered closer to the desired location for higher values of L_{1t} . The opposite was seen for the loiter tests, which indicate as this value increase, the center of the error ellipse migrates further away from the desired position.

Multi-Rotor UAS Following Multi-Rotor UAS.

The next set of test performed included a team composed of two multi-rotor UAS fulfilling the roles of both leader and follower. This test was performed as described in the methodology section, with the test parameters listed in Table 10 varied for an offset radius of 2m and an offset angle of 45° . The purpose of the 45° offset is to avoid wind interference on the part of the lower altitude follower from the leader's downward thrust. This test was performed in winds at approximately 8 knots at 180° .

Table 10. Multi-Rotor UAS Following Multi-Rotor UAS Test Parameter Matrix

Test Number	Flight Path	L_{1t}
1	Box	0 s
2	Box	1 s
3	Box	2 s
4	Circle	0 s
5	Circle	1 s
6	Circle	2 s

After testing was complete, an error was found in the method of calculating the offset position when using an angular offset and a L_{1t} greater than zero. This error caused the commanded waypoint to not be placed forward of the follower's desired position in the direction of the leader's ground course. This error does not effect the data presented which use an offset of 45° and a L_{1t} equal to zero. Due to this

issue, the mean error is expected to be higher than previous tests, but the standard deviation should still represent the team's ability to hold an accurate formation. The corrected follower script with this issue fixed is contained in Appendix F.

Test 1: Box Path, $L_{1t} = 0s$.

For this run, a 15m box pattern was performed by the leader vehicle, with the follower vehicle using a 0s forward offset lead time. As shown in Figure 49, after allowing the follower to stabilize its position for 50s, the system achieved a mean radial position error of 3.45m with a standard deviation of 1.11m and a DRMS of 2.97m. Additionally, as shown in Figure 50, the system achieved mean forward and right errors of -1.76m and 0.60m with standard deviations of 1.58m and 1.62m and DRMS of 1.94m and 1.41m. Figure 51 shows the path taken by both the leader and the follower, and the desired path of the follower based on the desired offset from the leader.

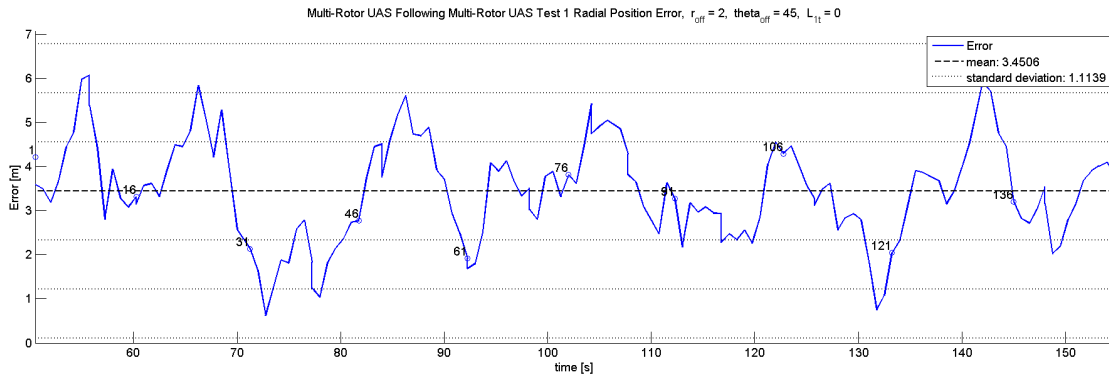


Figure 49. Multi-Rotor UAS Following Multi-Rotor UAS Test 1 Radial Position Error.

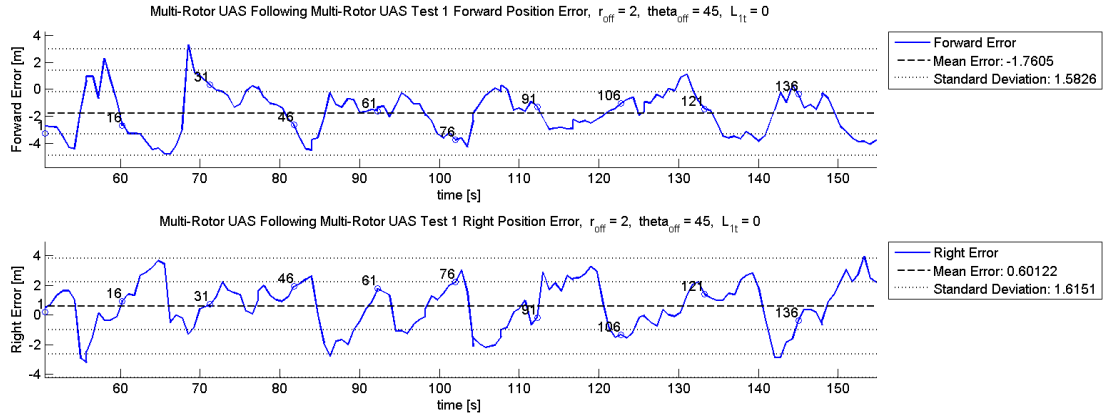


Figure 50. Multi-Rotor UAS Following Multi-Rotor UAS Test 1 Forward-Right Position Error.

Multi-Rotor UAS Following Multi-Rotor UAS Test 1 Vehicle Positions, $r_{off} = 2$, $\theta_{off} = 45$, $L_{lt} = 0$

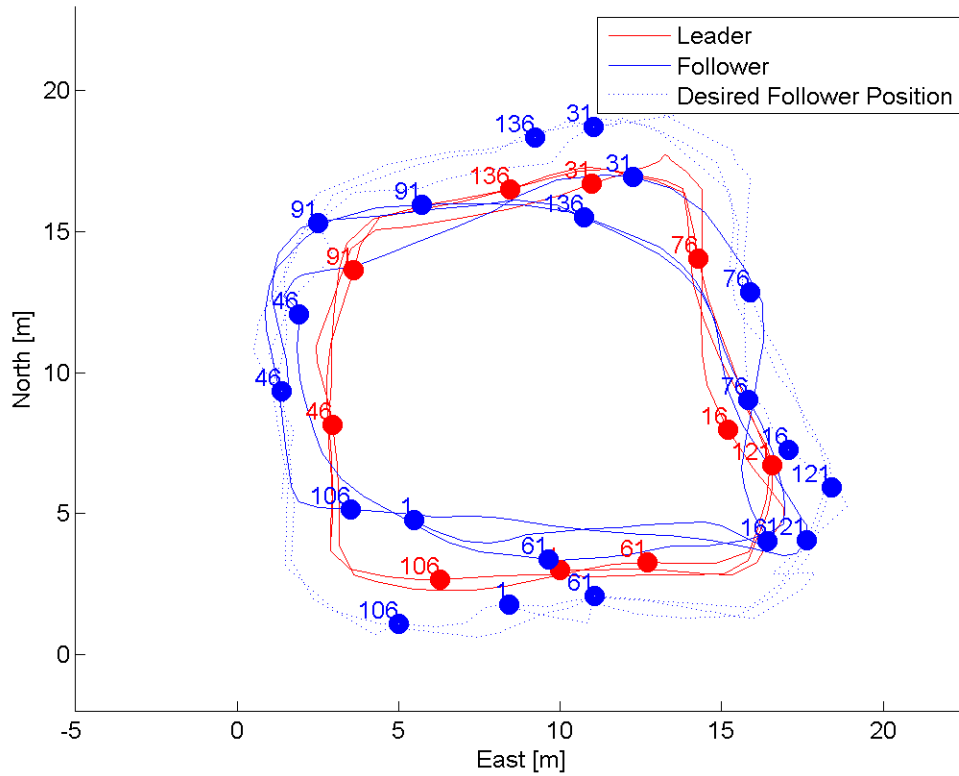


Figure 51. Multi-Rotor UAS Following Multi-Rotor UAS Test 1 Vehicle Position.

Test 2: Box Path, $L_{1t} = 1s$.

For this run, a 15m box pattern was performed by the leader vehicle, with the follower vehicle using a 1s forward offset lead time. During the test, the safety pilot of the follower vehicle performed a manual override and moved the vehicle off course. As shown in Figure 52, after allowing the follower to stabilize its position for 50s after the manual override, the system achieved a mean radial position error of 3.08m with a standard deviation of 0.63m and a DRMS of 1.77m. Additionally, as shown in Figure 53, the system achieved mean forward and right errors of -1.80m and 0.70m with standard deviations of 1.18m and 0.85m and DRMS of 1.21m and 0.62m. Figure 54 shows the path taken by both the leader and the follower, and the desired path of the follower based on the desired offset from the leader.

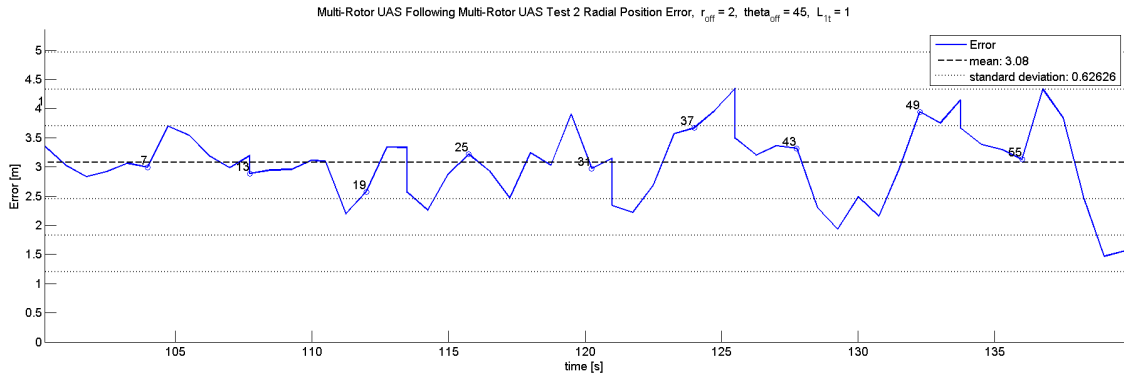


Figure 52. Multi-Rotor UAS Following Multi-Rotor UAS Test 2 Radial Position Error.

Test 3: Box Path, $L_{1t} = 2s$.

For this run, a 15m box pattern was performed by the leader vehicle, with the follower vehicle using a 2s forward offset lead time. As shown in Figure 55, after allowing the follower to stabilize its position for 50s, the system achieved a mean radial position error of 2.85m with a standard deviation of 1.32m and a DRMS of 2.53m. Additionally, as shown in Figure 56, the system achieved mean forward and right errors of -1.00m and 0.49m with standard deviations of 1.33m and 1.29m and DRMS of 1.34m and 1.11m. Figure 57 shows the path taken by both the leader and the follower, and the desired path of the follower based on the desired offset from the leader.

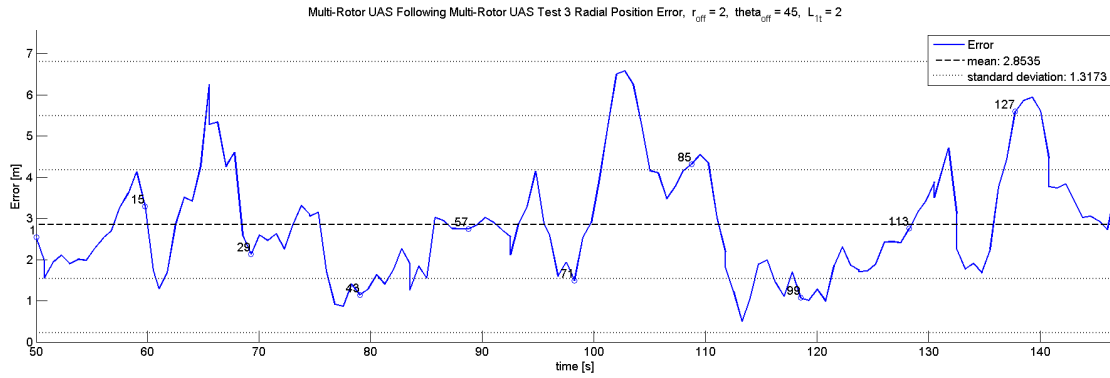


Figure 55. Multi-Rotor UAS Following Multi-Rotor UAS Test 3 Radial Position Error.

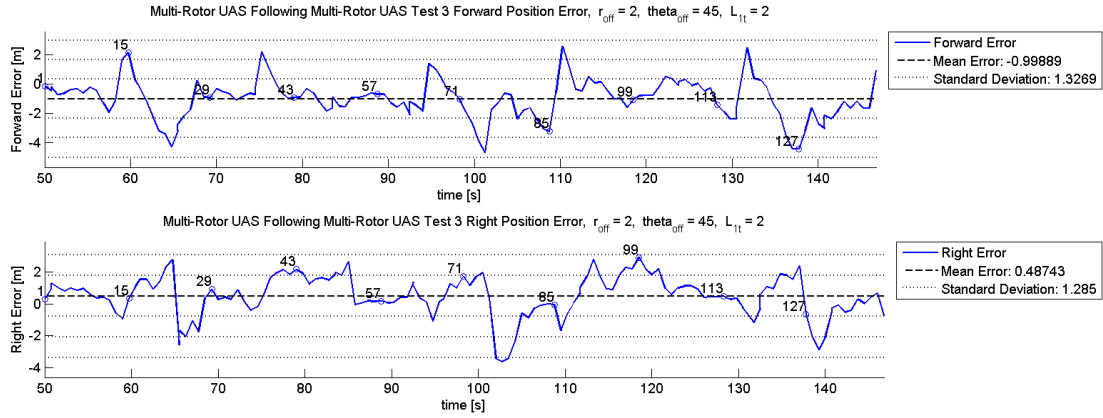


Figure 56. Multi-Rotor UAS Following Multi-Rotor UAS Test 3 Forward-Right Position Error.

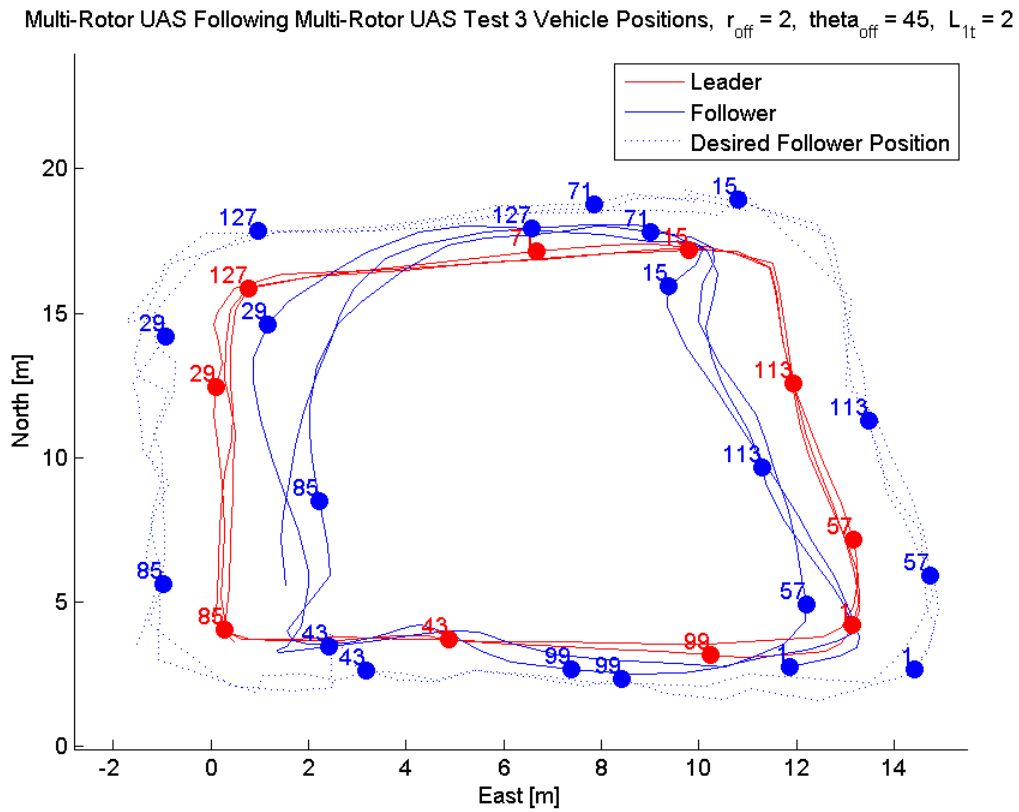


Figure 57. Multi-Rotor UAS Following Multi-Rotor UAS Test 3 Vehicle Position.

Test 4: Circle Path, $L_{1t} = 0s$.

For this run, a 10m radius circle pattern was performed by the leader vehicle, with the follower vehicle using a 0s forward offset lead time. As shown in Figure 58, after allowing the follower to stabilize its position for 50s, the system achieved a mean radial position error of 5.08m with a standard deviation of 2.16m and a DRMS of 4.17m. Additionally, as shown in Figure 59, the system achieved mean forward and right errors of -2.92m and 0.81m with standard deviations of 1.92m and 2.25m and DRMS of 2.64m and 1.80m. Figure 60 shows the path taken by both the leader and the follower, and the desired path of the follower based on the desired offset from the leader.

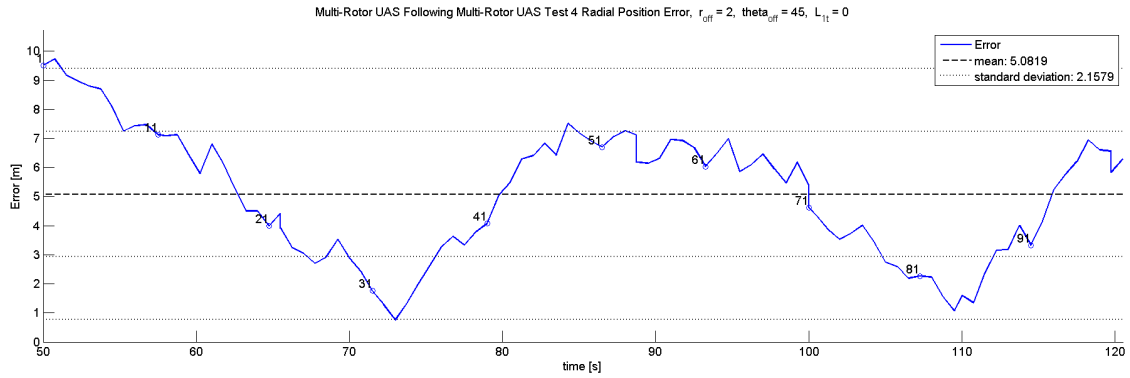


Figure 58. Multi-Rotor UAS Following Multi-Rotor UAS Test 4 Radial Position Error.

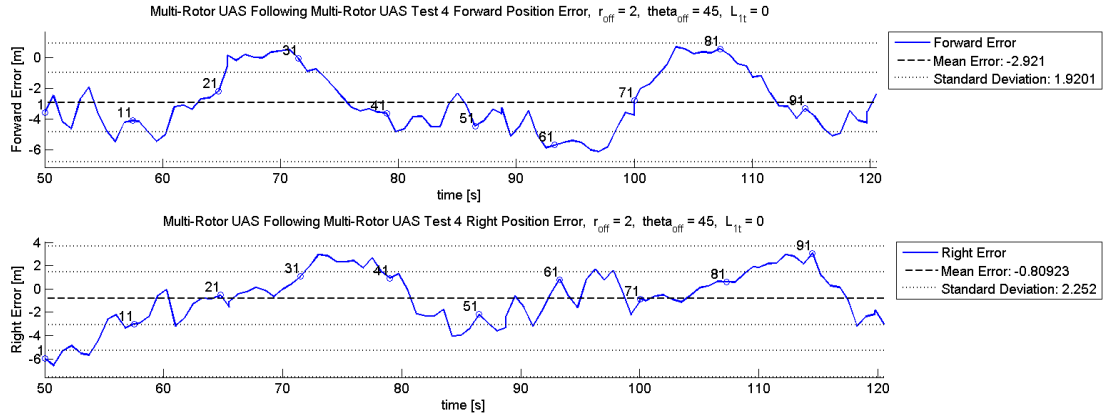


Figure 59. Multi-Rotor UAS Following Multi-Rotor UAS Test 4 Forward-Right Position Error.

Multi-Rotor UAS Following Multi-Rotor UAS Test 4 Vehicle Positions, $r_{off} = 2$, $\theta_{off} = 45$, $L_{lt} = 0$

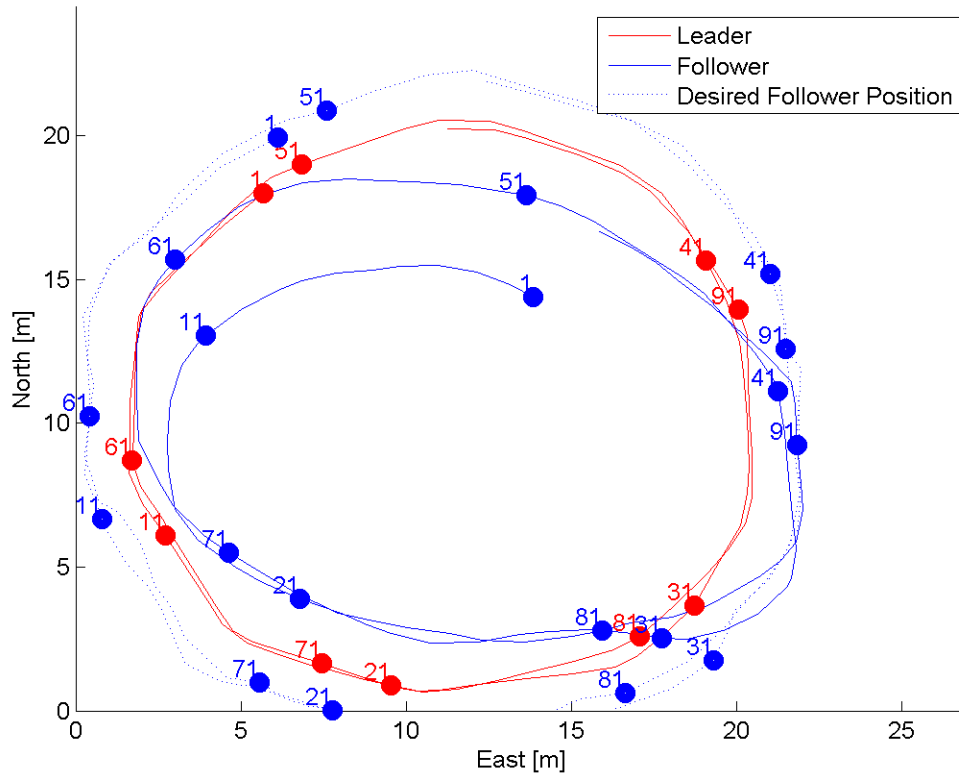


Figure 60. Multi-Rotor UAS Following Multi-Rotor UAS Test 4 Vehicle Position.

Test 5: Circle Path, $L_{1t} = 1s$.

For this run, a 10m radius circle pattern was performed by the leader vehicle, with the follower vehicle using a 1s forward offset lead time. As shown in Figure 61, after allowing the follower to stabilize its position for 50s, the system achieved a mean radial position error of 3.94m with a standard deviation of 1.68m and a DRMS of 3.51m. Additionally, as shown in Figure 62, the system achieved mean forward and right errors of -2.54m and 0.01m with standard deviations of 1.56m and 1.18m and DRMS of 2.44m and 0.96m. Figure 63 shows the path taken by both the leader and the follower, and the desired path of the follower based on the desired offset from the leader.

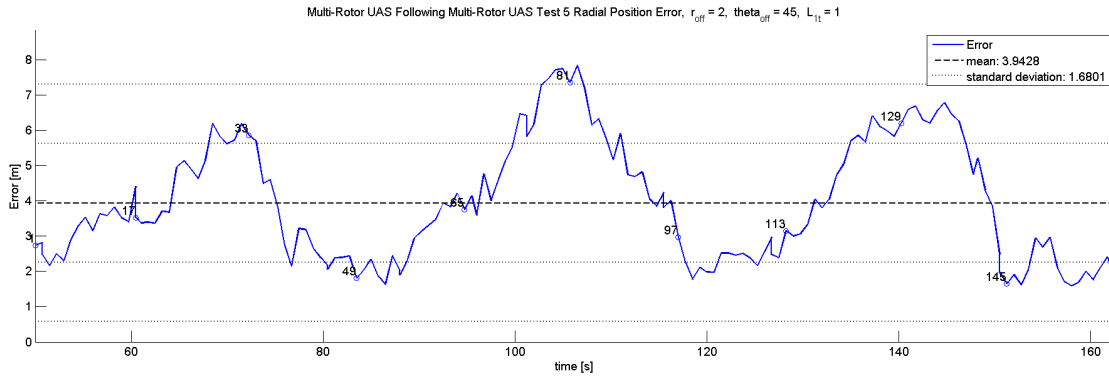


Figure 61. Multi-Rotor UAS Following Multi-Rotor UAS Test 5 Radial Position Error.

Test 6: Circle Path, $L_{1t} = 2s$.

For this run, a 10m radius circle pattern was performed by the leader vehicle, with the follower vehicle using a 2s forward offset lead time. As shown in Figure 64, after allowing the follower to stabilize its position for 50s, the system achieved a mean radial position error of 4.47m with a standard deviation of 2.29m and a DRMS of 3.67m. Additionally, as shown in Figure 65, the system achieved mean forward and right errors of -2.44m and 0.67m with standard deviations of 1.78m and 1.97m and DRMS of 2.21m and 1.52m. Figure 66 shows the path taken by both the leader and the follower, and the desired path of the follower based on the desired offset from the leader.

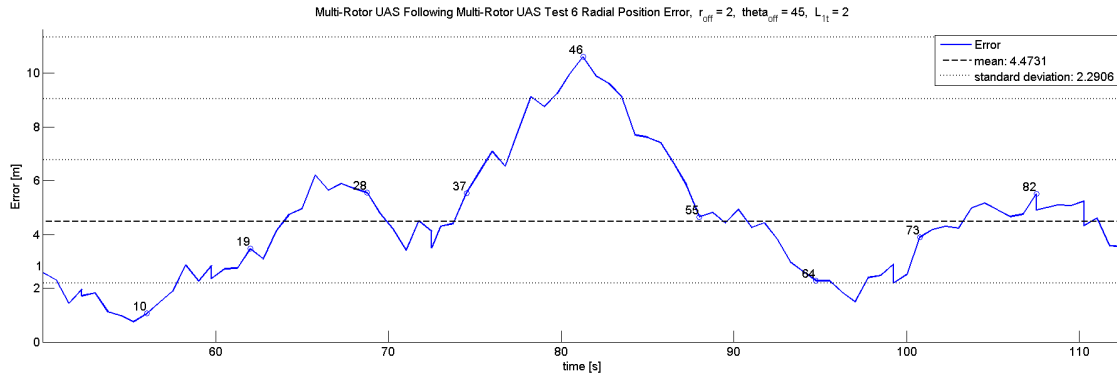


Figure 64. Multi-Rotor UAS Following Multi-Rotor UAS Test 6 Radial Position Error.

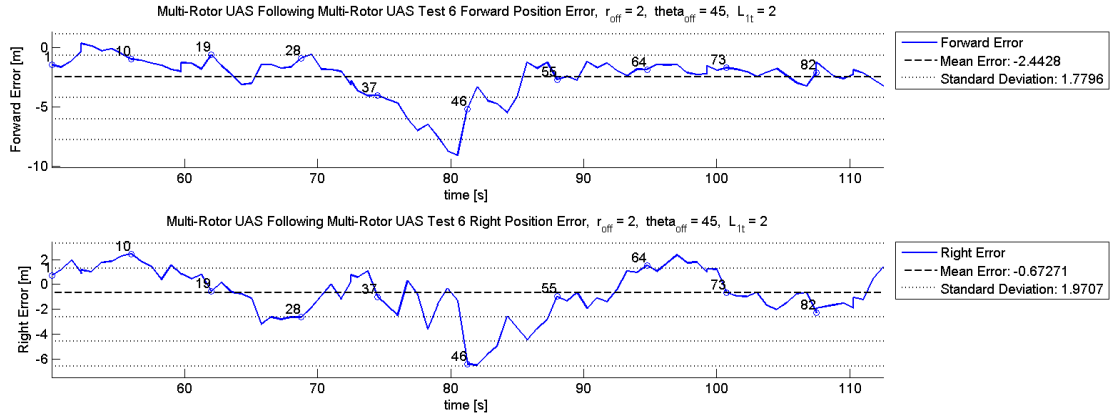


Figure 65. Multi-Rotor UAS Following Multi-Rotor UAS Test 6 Forward-Right Position Error.

Multi-Rotor UAS Following Multi-Rotor UAS Test 6 Vehicle Positions, $r_{off} = 2$, $\theta_{off} = 45$, $L_{lt} = 2$

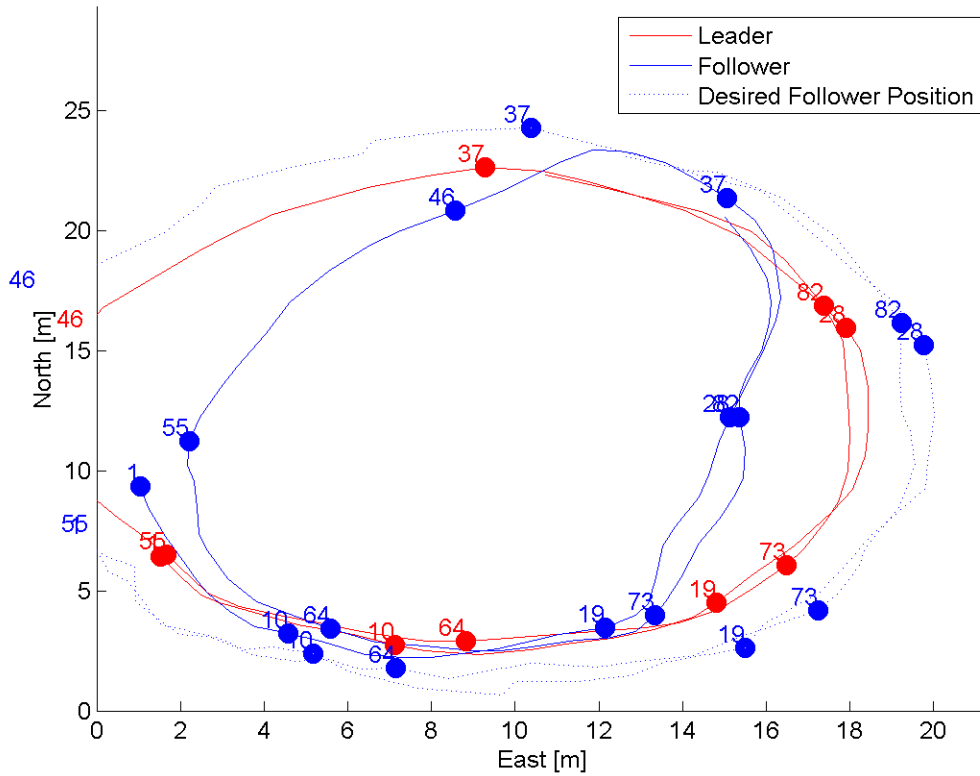


Figure 66. Multi-Rotor UAS Following Multi-Rotor UAS Test 6 Vehicle Position.

Team Analysis.

For this homogeneous team, an L_{1t} value of 1s with the leader performing a box path resulted in the lowest standard deviations and DRMS of the error, with the lowest mean error being seen for an L_{1t} value of 2s. Similarly for a circular path, an L_{1t} value of 1s produced the best results for the radial position error. These results indicate this team performs straight and curved paths with a higher level of precision and accuracy with this value of L_{1t} . The resulting measurements are summarized in Table 11.

Table 11. Multi-Rotor UAS Following Multi-Rotor UAS Test Results

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

As shown in the position plots for each test point, the follower's ground track noticeably varied from the desired follower position ground track. One known cause for this is the error in the method of calculating the commanded follower position described previously. However, another possible cause of this variation is the combination of identical vehicle cruise velocities and the difference in length of the desired follower ground track and leader ground track. Due to the follower's desired ground track on the outer circuit being longer than the leader's inner circuit, the follower

would have to fly at a higher ground speed than the leader around turns to maintain the desired separation. However, this is partially overcome by the follower cutting inside corners as the commanded point is placed further to the left after the leader turns. This allows the follower to gradually regain a similar ground track, as shown in Figure 51 at the North East corner at point 136. Due to the first cause of error stated not effecting tests with values of L_{1t} equal to zero, it is likely the second cause of error producing variations for both tests 1 and 4. For all other tests the product of variation is likely a combination of both causes.

Based on the previous team's results consisting of a multi-rotor UAS following a UGS leader and based on how well the vehicles visually maintained separation during the flight test, it is expected that a team consisting of two multi-rotor UAS would perform as well or better. It is likely this is true due to their shared flight characteristics which allow the airframe to maneuver in any direction.

Finally, from the accuracy and precision measures for the forward and right errors, the system has less ability to maintain forward position opposed to right position while performing straight paths, as indicated by the box pattern. For the box tests, the standard deviation and DRMS of the right error are on average 0.11m and 0.45m closer to zero than that of the forward error. However, as the L_{1t} increases, the forward mean error decreased. This means that the error ellipse formed by both the standard deviation is centered closer to the desired location for higher values of L_{1t} . No correlation of this kind is seen for the loiter tests.

Fixed Wing UAS Following Fixed Wing UAS.

The next set of tests performed included a team composed of two fixed wing UAS fulfilling both the leader and follower roles. This test was performed as described in the methodology section with the exception of the leader performing the loiter

due to time constraints. The test parameters listed in Table 12 were varied for an offset radius of 10m and an offset angle of 0° . This test was performed in winds at approximately 11 knots at 180° .

Table 12. Fixed Wing UAS Following Fixed Wing UAS Test Parameter Matrix

Test Number	Flight Path	L_{1t}
1	Box	0 s
2	Box	1 s
3	Box	2 s

Due to the highly oscillatory behavior of the position errors calculated, a 50 second stabilization time is not used in determining the mean, standard deviation, and DRMS of the data set.

Test 1: Box Path, $L_{1t} = 0s$.

For this run, a box pattern was performed by the leader vehicle, with the follower vehicle using a 0s forward offset lead time. As shown in Figure 67, for the full duration of the test, the vehicle accomplished a mean radial position error of 118.82m with a standard deviation of 64.96m and a DRMS of 135.38m. Additionally, as shown in Figure 68, the system achieved mean forward and right errors of -60.57m and 74.31m with standard deviations of 65.87m and 69.45m and DRMS of 89.42m and 101.65m. Figure 69 shows the path taken by both the leader and the follower, and the desired path of the follower based on the desired offset from the leader.

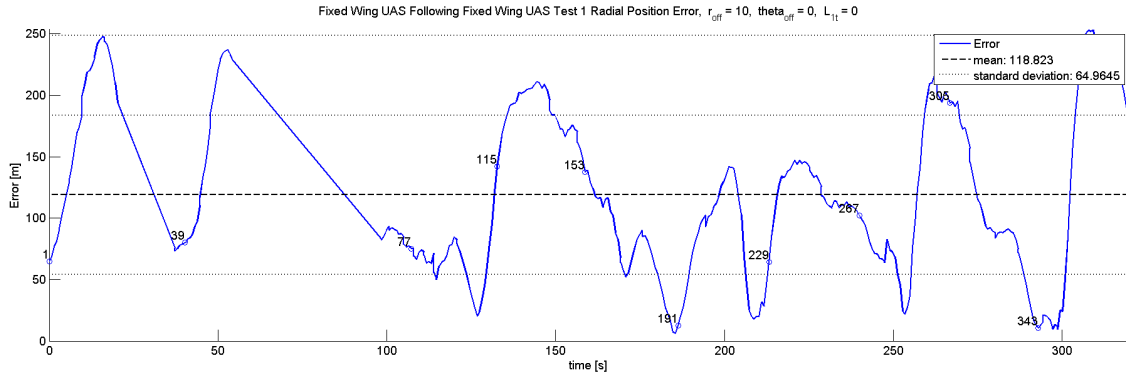


Figure 67. Fixed Wing UAS Following Fixed Wing UAS Test 1 Radial Position Error.

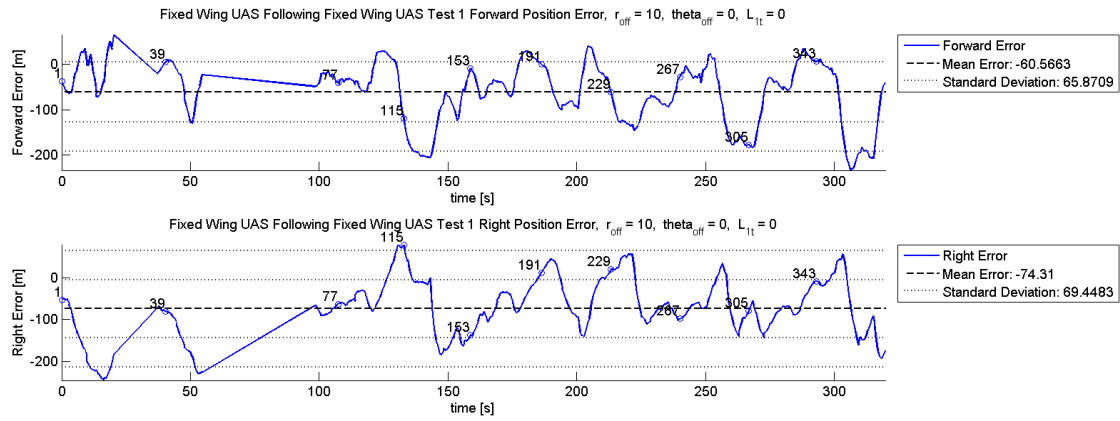


Figure 68. Fixed Wing UAS Following Fixed Wing UAS Test 1 Forward-Right Position Error.

Fixed Wing UAS Following Fixed Wing UAS Test 1 Vehicle Positions, $r_{\text{off}} = 10$, $\theta_{\text{off}} = 0$, $L_{1t} = 0$

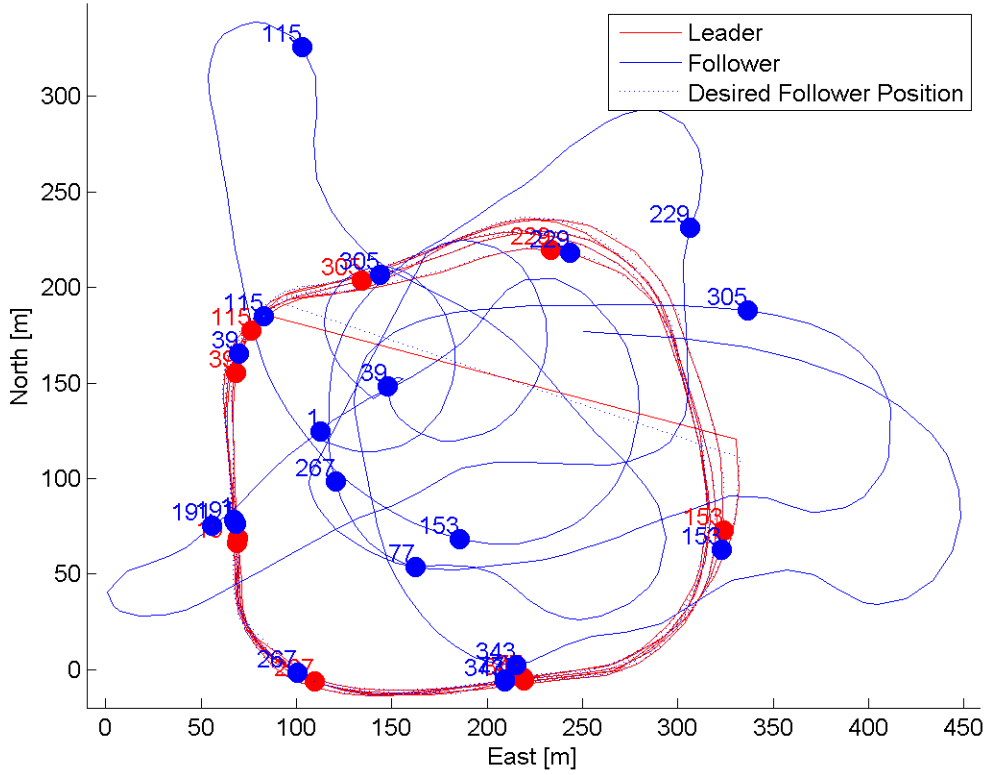


Figure 69. Fixed Wing UAS Following Fixed Wing UAS Test 1 Vehicle Position.

Test 2: Box Path, $L_{1t} = 1s$.

For this run, a box pattern was performed by the leader vehicle, with the follower vehicle using a 1s forward offset lead time. As shown in Figure 70, for the full duration of the test, the vehicle accomplished a mean radial position error of 104.01m with a standard deviation of 57.79m and a DRMS of 117.68m. Additionally, as shown in Figure 71, the system achieved mean forward and right errors of -48.41m and 56.77m with standard deviations of 72.39m and 58.06m and DRMS of 86.06m and 80.27m. Figure 72 shows the path taken by both the leader and the follower, and the desired path of the follower based on the desired offset from the leader.

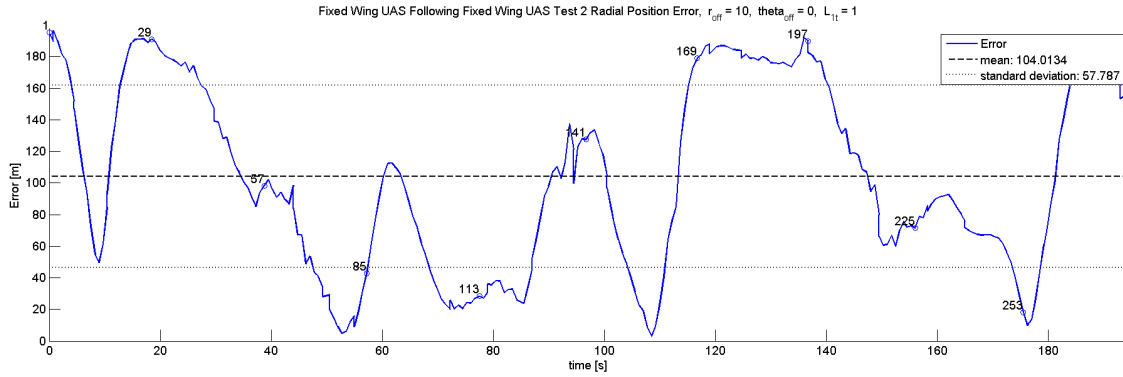


Figure 70. Fixed Wing UAS Following Fixed Wing UAS Test 2 Radial Position Error.

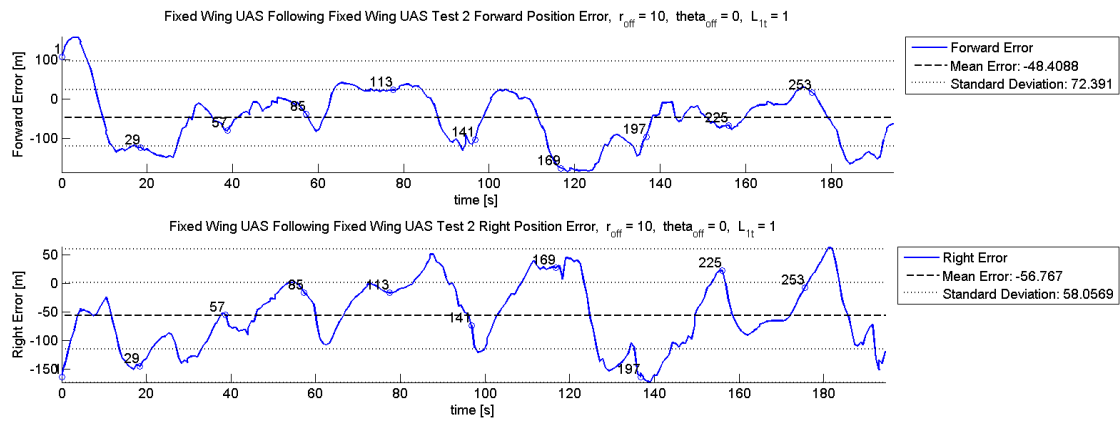


Figure 71. Fixed Wing UAS Following Fixed Wing UAS Test 2 Forward-Right Position Error.

Fixed Wing UAS Following Fixed Wing UAS Test 2 Vehicle Positions, $r_{\text{off}} = 10$, $\theta_{\text{off}} = 0$, $L_{1t} = 1$

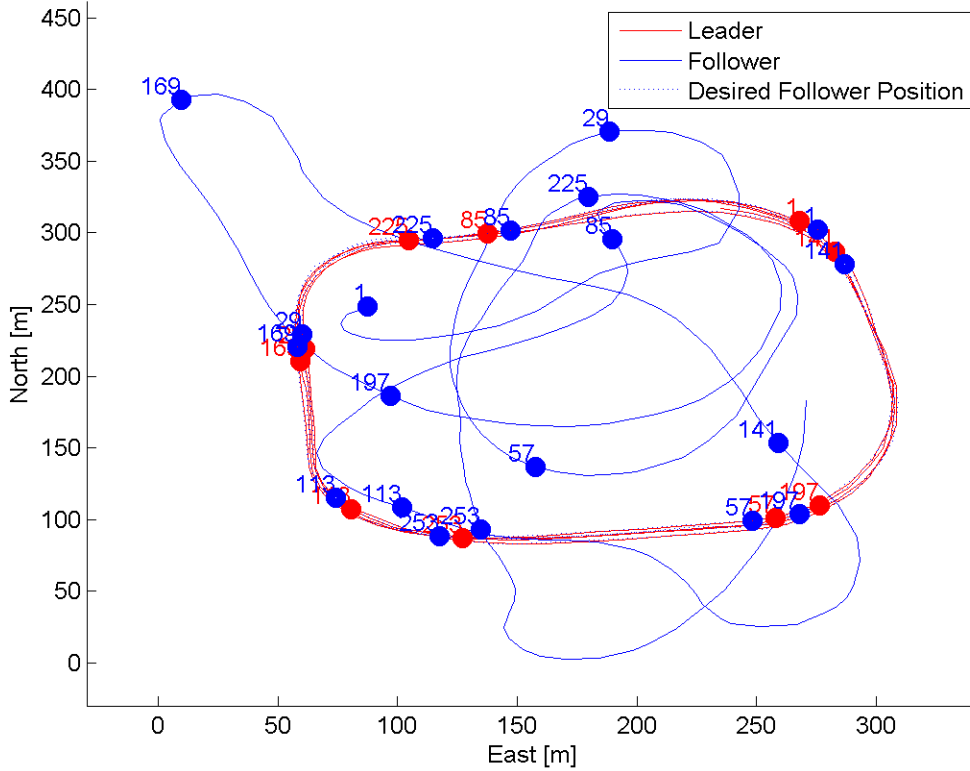


Figure 72. Fixed Wing UAS Following Fixed Wing UAS Test 2 Vehicle Position.

Test 3: Box Path, $L_{1t} = 2s$.

For this run, a box pattern was performed by the leader vehicle, with the follower vehicle using a 2s forward offset lead time. As shown in Figure 73, for the full duration of the test, the vehicle accomplished a mean radial position error of 121.78m with a standard deviation of 62.11m and a DRMS of 139.11m. Additionally, as shown in Figure 74, the system achieved mean forward and right errors of -60.26m and 74.90m with standard deviations of 59.07m and 76.17m and DRMS of 83.93m and 106.25m. Figure 75 shows the path taken by both the leader and the follower, and the desired path of the follower based on the desired offset from the leader.

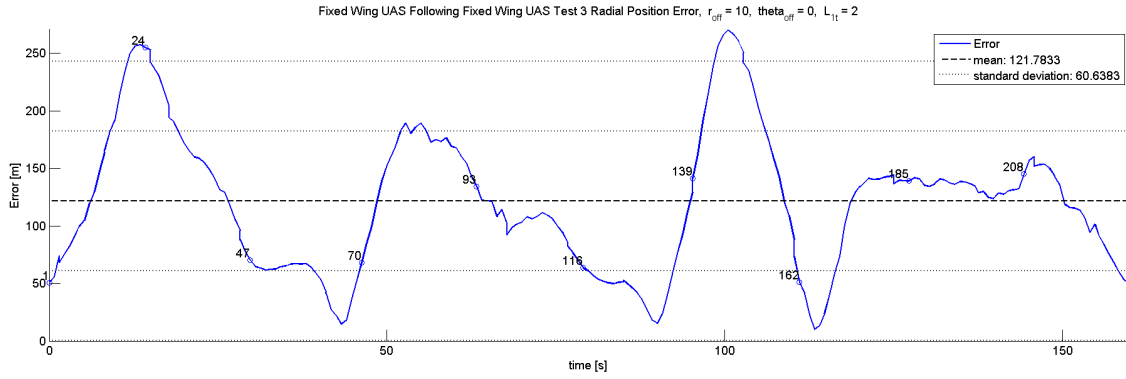


Figure 73. Fixed Wing UAS Following Fixed Wing UAS Test 3 Radial Position Error.

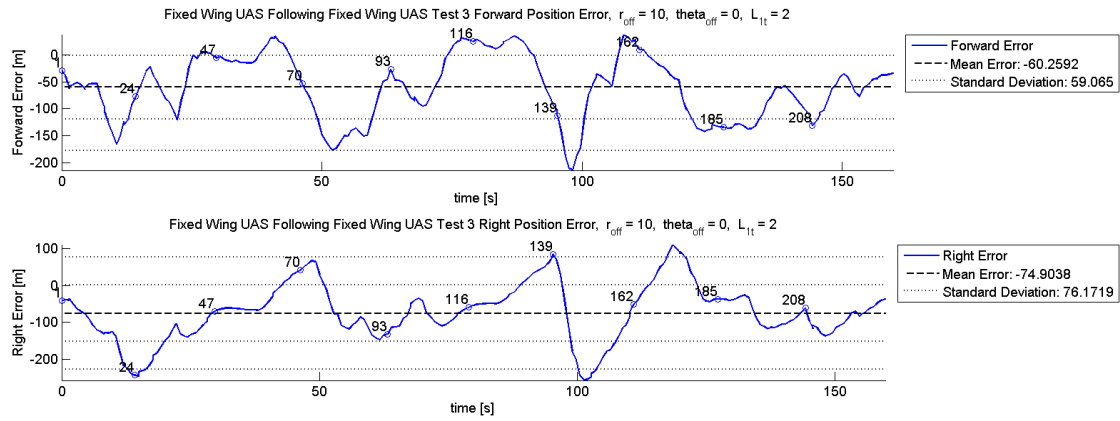


Figure 74. Fixed Wing UAS Following Fixed Wing UAS Test 3 Forward-Right Position Error.

Figure 10 is a 2D plot showing the trajectories of a leader (red solid line) and a follower (blue solid line) in a North vs. East coordinate system. The plot includes numbered points for each trajectory and a legend for 'Leader', 'Follower', and 'Desired Follower Position' (dotted blue line). The trajectories are complex, looping paths. The leader's path is generally smoother than the follower's path.

Team Analysis.

105

Table 13. Fixed Wing UAS Following Fixed Wing UAS Test Results

	Forward Error (m)			Right Error (m)			Position Error (m)		
Test	μ	σ	DRMS	μ	σ	DRMS	μ	σ	DRMS
1	-60.57	65.87	89.42	-74.31	69.45	101.65	118.82	64.96	135.38
2	-48.41	72.39	86.06	-56.77	58.06	80.27	104.01	57.79	117.68
3	-60.26	59.07	83.93	-74.90	76.17	106.25	121.78	62.11	139.11

As shown in the three position plots, the follower did not maintain a ground track that closely resembled the desired follower position's ground track. However, the vehicles did demonstrate a leader follower relationship. This relationship is indicated by the follower turning primarily in the same direction as the leader and operating in the same vicinity as the leader. Additionally, the ground course of the vehicle at most points are either pointing in the area of the desired position, as shown at point 208 in Figure 75, or are on a path which will point in the area of the desired position, as shown at point 115 in Figure 69.

Like the UGS, the fixed wing UAS is at a disadvantage compared to the multi-rotor because of how it is designed to maneuver. If it is commanded to fly to a point behind it or off the current ground course, it is required to perform a turn. However, compared to the UGS, this effect is amplified by the speed, larger turn radius, and the slower rate that commands are received at the extended distance from the GCS.

One indication of a slow rate of commands received and one common path error shared by all three tests is the lobe that occurs at the North West corner of the leader's path. A section of the flight path from Figure 75 containing this lobe is displayed in Figure 76 with the positions of both vehicles and the commanded position sent to the follower vehicle shown. Starting at time step 1, the follower vehicle looks to be

turning towards the desired position up to time step 10, at which time it maneuvers away from the desired flight path. The follower does not start to correct its heading towards the desired position until after time step 16. This 6s delay is likely caused by a weak C2 link connection, which did not allow the follower vehicle to receive an updated command between these points. A weak C2 link is also most likely to occur at this point, as it is one of the furthest points from the GCS communication node, inducing fewer packets received from the GCS. The likely cause of the vehicle performing a right turn, opposed to the desired left turn, is the autopilot was likely commanding the vehicle to either maneuver back to a previous waypoint or to perform a loiter around a waypoint.

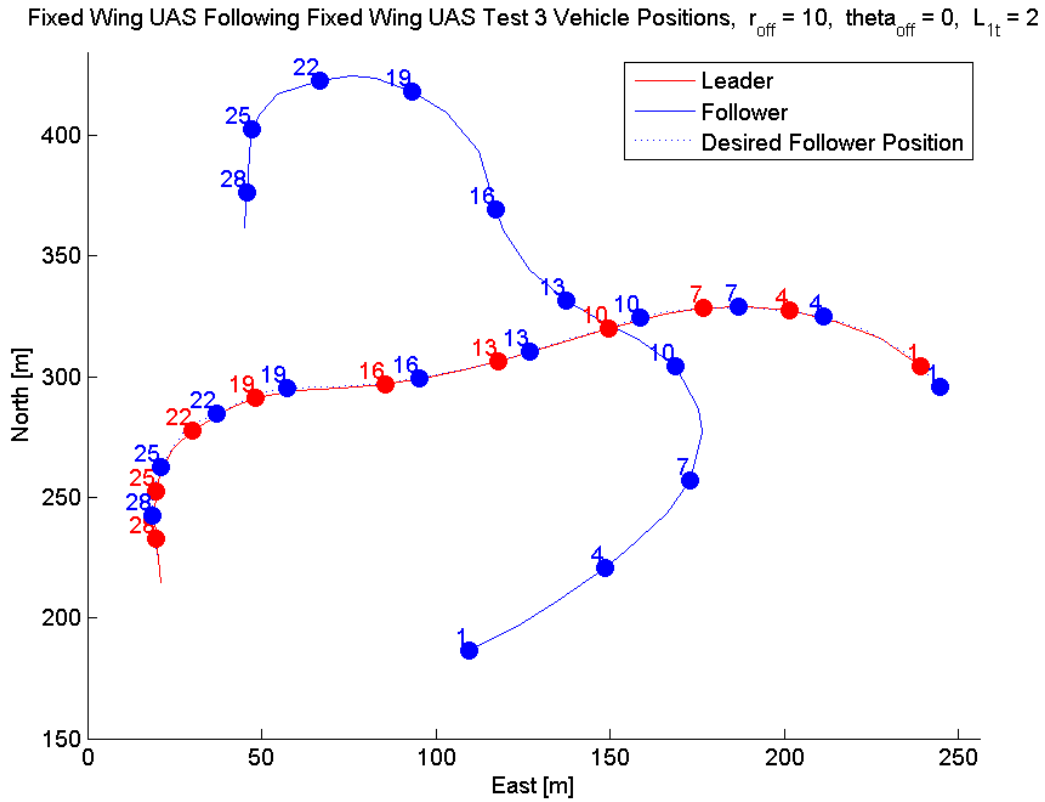


Figure 76. Fixed Wing UAS Following Fixed Wing UAS Test 3 Vehicle Position NW Corner.

Another possible cause of error is the method of introducing the vehicle into the flight path. For this test, the vehicle was not flown into a specific position or in a specific direction before starting the flocking script. This did not prep the vehicle to enter the circuit, and perhaps forced it to take more drastic maneuvers to fly to the correct position. An example of this can be seen in Figure 72, where the script was initialized at point 1 with the follower in the North West corner of the path heading West and the leader in the North East corner heading West. This caused the follower to perform a tight left hand turn and enter the circuit going the wrong direction. An example of the effects from a proper entrance can be seen in Figure 75 after point 185. The vehicle flew into the circuit tangent to the ground track of the leader, allowing the vehicle to maintain a higher level of precision for approximately 20 seconds as shown in the error plot.

Formation Flight Analysis.

The system's ability to control a team of two heterogeneous or homogeneous vehicles to perform formation flocking was verified. The only exception to this is the fixed wing aircraft team, which did not fully demonstrate formation flight, but did demonstrate a leader follower relationship. The resulting mean position errors with \pm one standard deviation and \pm one DRMS from the mean for each test at each value of L_{1t} is outlined in Figure 77 below.

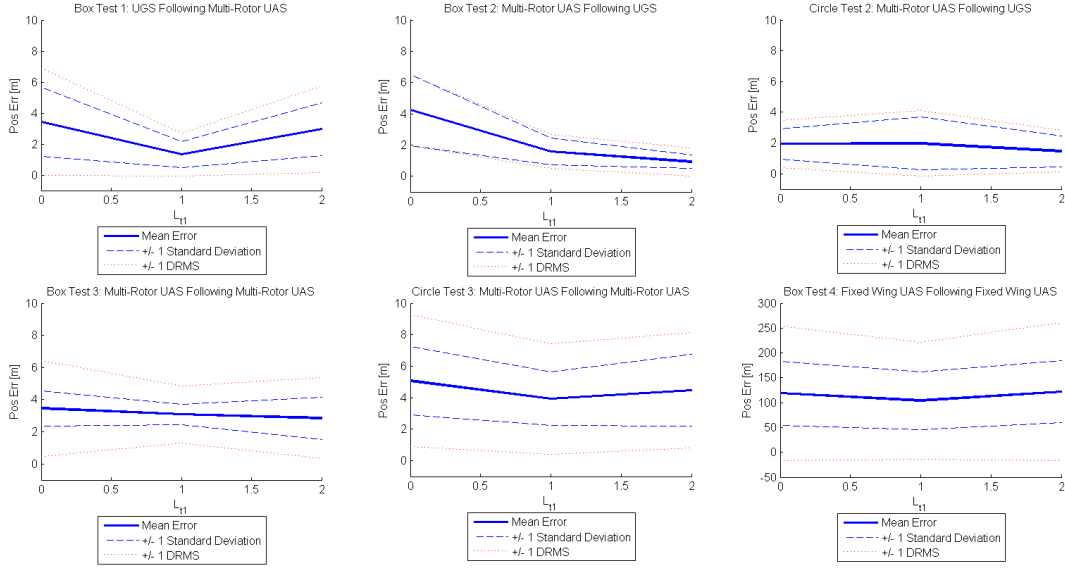


Figure 77. Formation Flocking Test Results Summary.

The effect of different L_{1t} on the position error and standard deviation varied for each vehicle combination and for each lead vehicle path performed. As expected, the values of DRMS and standard deviation have a close correlation; meaning both measures will increase or decrease together. Over all, the best performance was seen with the multi-rotor UAS following UGS performing a box pattern with an L_{1t} value of 2. Both results of Box Test 1 and Box Test 2 indicate a higher correlation between the value of L_{1t} and the position error due to their tendency to neck down at specific values. The other tests, however, indicate the value of L_{1t} has less of an effect on position error for those scenarios.

For Box Test 1, a minimum position error is indicated to exist around an L_{1t} value of 1s because the mean, standard deviations, and DRMS values are at their lowest at this point. For Box Test 2, as L_{1t} increases, the mean, standard deviation, and DRMS decreased by 3.32 m, 1.85 m, and 1.37m respectively. Unlike its box counterpart, Circle Test 2 indicates a maximum in position error and standard deviation for an L_{1t} value of one, with the standard deviation and DRMS decreasing as L_{1t} approaches

zero and mean, standard deviation, and DRMS decreasing as L_{1t} increases above one. For the last three tests, there is a less prominent correlation between standard deviation and L_{1t} . For Box Test 3 as the value of L_{1t} increased, the mean position error decreased by 0.60m. The standard deviation and DRMS values reached a minimum for this test at an L_{1t} value of 1s, with increased values at the other values of L_{1t} . Circle test three indicates a minimum position error, standard deviation, and DRMS for an L_{1t} value of 1s with increasing mean, standard deviation, and DRMS for other values of L_{1t} . Finally, the plane tests indicate almost no correlation between L_{1t} and the mean and standard deviation.

One benefit seen with the UGS and multi-rotor UAS in the role of follower is this control method's tendency to command the follower to cut inside corners during the box pattern. This action of cutting corners is caused by the commanded point being placed down the next leg of the box ahead of the leader after it performs a turn. This aids in the minimization of error, allowing the follower to take a shorter path through the corner and catch up. However, with this decrease in forward error comes an increase in right error due to the follower cutting the corner and going off track.

One common theme across all vehicle teams is the effect of the follower vehicle's method of maneuvering on the positional accuracy and precision of the system. As indicated by the multi-rotor following a UGS performing both box and circle patterns, the multi-rotor has the best ability to maintain a desired position relative to the leader. This is due to the airframe's ability to move in any direction, forwards or backwards, and its method of loitering, which is to maintain a stationary position. Both the UGS and fixed wing UAS are required to make turns, which can force them off the desired path and induce more error. Finally, the fixed wing UAS is at the biggest disadvantage due to statement above, the vehicles method of loitering which requires it to fly in a circle around a point, and its higher operating speed which

reduces the amount of time the aircraft can react to a new commanded position.

The biggest issue seen during these tests is the fixed wing UAS's inability to maintain an offset. As stated previously, one likely cause is a weak C2 link which decreases the percent of telemetry and command packets received, which in turn increases the latency of the system. For this vehicle combination to operate near the same level as the other teams the C2 link must be strengthened or the processing must be moved on board the aircraft to reduce the number of links made.

The final problem seen across all teams is the followers lesser ability to maintain a steady forward error. The forward and right errors for most tests indicated a greater ability to maintain a more precise and accurate right error than forward error. This inaccuracy and imprecision is likely due to the lack of command authority over the ground course velocity of the vehicle. This functionality was not included in the C2 architecture because the GCS and control module used does not allow for direct control of the ground course velocity. However, as indicated by the results previously discussed, the method of control used does allow for increased precision and accuracy of forward position for varying values of L_{1t} depending on the team and path taken.

5.5 Communication Relay Test Results and Analysis

In this section, the results of the communication relay tests outlined in the methodology section are discussed and analyzed. Both the absolute position of each vehicle at a local level in meters and the relative position error in meters will be used to display the collected data. The absolute position will be used to identify behavioral traits of the system while performing communication relay, while the relative position error is used for quantifying the system's abilities. The relative position error is determined and displayed using the radial distance between the relay vehicle's position and the relay vehicle's desired position relative to the remote vehicle and GCS.

Test Results and Analysis.

The communication relay test was performed as specified in the methodology section with one additional test to demonstrate the system's ability to relay a C2 link around obstructions. The first test, which verified the communication nodes would reestablish a lost link of a remote vehicle, was first tested with a team of UGS. All of the communication nodes, including each node on each vehicle and the GCS node, were set to their lowest power setting of 40mW. The remote UGS was driven to a distance of 114m until the C2 link was lost between it and the GCS. The relay vehicle was then manually driven to a halfway point. After 5 seconds, the remote vehicle's C2 link was reestablished with 80% packets received.

Another test was performed with this team to demonstrate the system's ability to relay the C2 link around an obstruction. For this test, the remote vehicle was driven next to a cement footer of a lamp post in line of sight with the relay vehicle communication node turned off. At this point the GCS was receiving 90% of telemetry packets. The remote vehicle was then driven behind the cement footer, out of line of sight, reducing the percent of packets received between 55% and 65%, with intermittent loss of the C2 link. The relay vehicle was then turned on and driven to a point where it would have visual line of sight of both the remote vehicle and GCS. The GCS then started receiving between 80% and 90% of the telemetry packets.

The final test was performed with a team consisting of a UGS in the role of the remote vehicle and a multi-rotor UAS in the role of a relay vehicle. Again, all communication nodes were turned down to their lowest setting, and the remote vehicle was driven away from the GCS until the link on the GCS was lost. The relay python script was then started on the GCS, and the relay vehicle was commanded to move to the midpoint between the GCS and the last known position of the remote vehicle. Once the relay vehicle reached its destination, the link was reestablished with 85%

of the telemetry packets being received. The remote vehicle operator then manually drove the vehicle around the area of the GCS. The positions of the remote and relay vehicles are displayed in Figure 79, and the associated error in position between the relay vehicle and its desired position is displayed in Figure 78. During this test, the relay vehicle achieved a mean error of 9.76m with a standard deviation of 6.17m and a DRMS of 11.55m after the vehicle came to a stabilized midpoint position. One point to note on Figure 78 are the spikes in error seen between data point 300 and 700. These spikes were caused by a lost link between the relay vehicle and GCS. These lost links were caused by the remote vehicle driving far enough away to pull the relay vehicle out of communication range of the GCS.

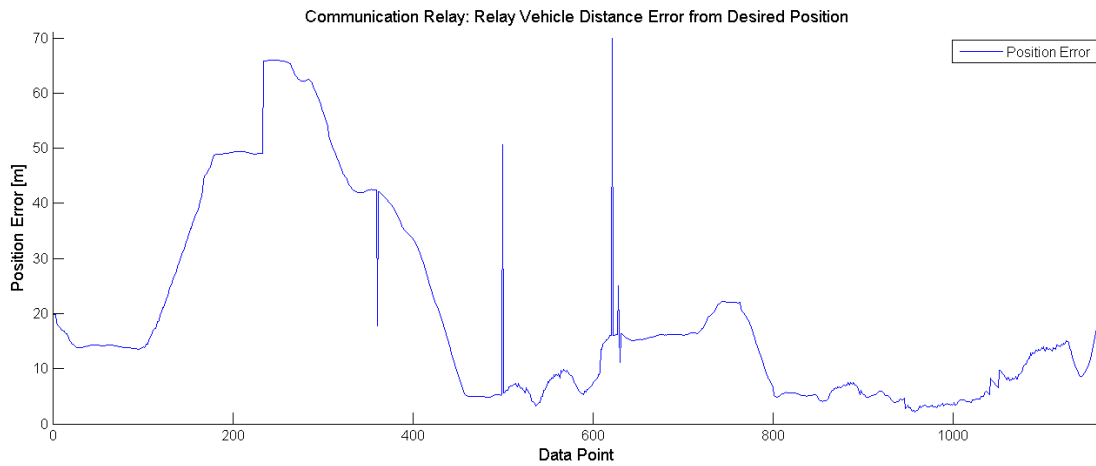


Figure 78. Multi-Rotor UAS Relaying to UGS Radial Position Error.

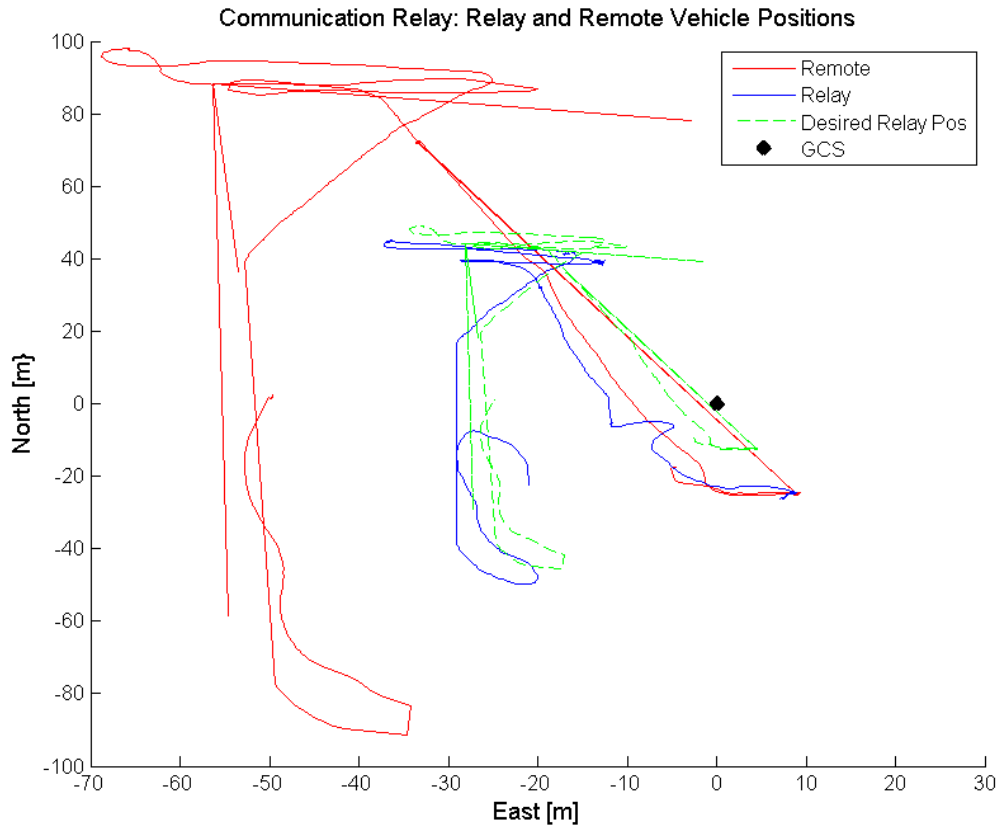


Figure 79. Multi-Rotor UAS Relaying to UGS Vehicle Position.

5.6 Latency Test Results and Analysis

Results.

Each component of the latency was measured as described in the methodology section using scripts on the GCS. To measure the telemetry down to GCS time, the GCS script collected the pitch and roll orientations from the telemetry at 20Hz to ensure the sampling rate was higher than the rate new telemetry is made available, which is indicated by multiple measurements of the same value for multiple adjacent time steps. This difference in time between the first instance of a measurement and an instance of a new measurement is determined as the telemetry down to GCS

time. The GCS processing time was measured by determining the start time of the leader flocking script, passing that time through the UDP socket, and calculating the total run time at the end of the follower flocking script. The command up and telemetry down time is measured by determining the time to send an RC channel PWM command from the GCS to the time the change is seen in the RC channel in the collected telemetry. These tests provided the resulting values outlined in Table 14 below. The GCS processing time is approximated due to the high rate of speed the GCS processing scripts run.

Table 14. Latency Test Results

Time Measurment	μ	σ
Telemetry Down to GCS	0.240s	0.005s
GCS Processing	< 0.003s	—
Command Up to Vehicle	0.220s	0.006s

Analysis.

The average total time for one vehicle to pass telemetry down to the GCS, to when the follower vehicle reacts to this telemetry is approximately 0.46s based on the times collected. Based on the standard deviations calculated, this total latency can vary between 0.493s and 0.427s for ± 3 standard deviations. With a 2.0 Hz update rate in the worst case, this system performs 400% faster than previous systems utilizing entirely all COTS components and OSS. This is compared to the work of Hardy [11], wherein Mission Planner’s built in swarming function was only able to achieve a maximum of 0.4Hz. This can be attributed to the use of MAVProxy as the primary GCS software, which as stated before does not require a more computationally taxing GUI. However, this measurement is for a best case scenario, with all vehicles in close

range of the GCS and GCS transceiver. This does not include the impact of lost telemetry packets over the communication network. As discussed in the formation flocking tests for a team of fixed wing UAS, as the fixed wing UAS flew further away from the GCS, the communication link degraded greatly. This caused slow updates of new waypoints and caused the vehicle to fly erratically compared to the lead vehicle.

5.7 Chapter Summary

In this chapter, the hardware and software selected to accomplish the functions of the developed architecture were discussed. The development of the C2 python scripts to control the vehicle was outlined. Finally, the results from the formation flight, communication relay, and latency tests were covered and the resulting data was analyzed.

VI. Conclusion

6.1 Chapter Overview

This section reviews the work accomplished in this research and the conclusions drawn from this research. Investigative questions outlined in the first chapter will be revisited to outline the conclusions drawn from each. Recommendations of actions that should be taken for future work are also outlined.

6.2 Conclusion of Research

In the first chapter, investigative questions were established to guide this research to obtain answers to each question. These questions will be restated and the conclusions for each are outlined.

What are the desired missions to be accomplished by cooperative multi-agent systems?

From the literature review, three groups of missions emerged. The first group is formation flocking, which is the act of controlling two or more vehicles to perform a mission in formation. The second group of missions complete the communication relay scenario, which is the act of passing information between a remote vehicle or sensor to a central GCS through an intermediate relay vehicle. The final group of missions is comprised of search and surveillance missions, which include wide area or perimeter searching or surveillance. For this research, the final group was not investigated as it was the only mission requiring a video system and it was desired the system be simplified for the time scope of the project. However, the architecture and system developed could possibly be applied to perform this mission with the addition of the required video system.

What is the structure and limitations of existing C2 architectures for cooperative unmanned vehicles?

Existing architectures range from being comprised of a mixture of proprietary and COTS components and software to being comprised of entirely of COTS components and OSS. All of these architectures have similar structures, with the primary variation being the location where the processing is being accomplished. Systems comprised of primarily proprietary components, as shown by Napolitano et al. [3], have the capability to perform decision making onboard the vehicle which increases the response time dramatically and allow for greater precision with position errors as low as 3.43m with a standard deviation less than 2m for fixed wing aircraft. As the processing migrates from onboard the vehicle to the GCS, as shown by How et al. [9], the response rate diminishes along with the precision with distance errors contained in a 25m box for fixed wing aircraft. Both of these systems utilize a proprietary GCS, which allows for greater flexibility in the processing of information and sending of commands. The final architecture examined is structured the same way, but utilizes entirely COTS components and OSS. With this architecture, shown by Hardy [11], the update rate further degraded to a maximum of 0.4 Hz for close range vehicles including UGS and multi-rotor UAS. These results show a degradation in system performance as more COTS components and OSS is integrated into the system and as the mission processing unit migrates from onboard to the GCS.

What are the mission-specific qualitative and quantitative measures for the system?

For this effort, it was decided the desired missions to be accomplished are the formation flocking and communication relay scenarios based on previously developed

architectures. For these scenarios, it was determined the desired quantitative measures include both the relative accuracy and precision of the vehicle's position error. The DRMS from the desired position was used to measure the relative accuracy of the system and the standard deviation was used to measure the precision. For formation flocking, the behavior of the system was used as a qualitative measure to determine if the system demonstrated a leader follower relationship. For communication relay, the percent telemetry packets received was used to quantitatively measure when the C2 link was lost, and the quality of the link once it was reestablished through the relay vehicle. Also, the ability to relay a communication link was used to qualitatively measure the system's ability to perform communication relay. Finally, the latency was measured using the time of each component of the system.

How well does this system perform using these performance measures?

The results of this question are contained in the results and analysis section. The best results for formation flocking were seen with a team consisting of a multi-rotor UAS following a UGS, which achieved a mean position error of 0.99m with a standard deviation of 0.44m and DRMS of 0.59m. Other results for combinations of these two vehicles ranged up to a mean position error of 5.08m, a standard deviation of 2.46m, and a DRMS of 4.17m for separate tests. The worst case not contained in the summary above was the team consisting of two fixed wing UAS aircraft, which resulted in errors two orders of magnitude higher than other tests. This test did not demonstrate the ability to perform formation flight, but did demonstrate a leader follower relationship.

For the communication relay scenario, two tests were accomplished to test the system's ability to perform this scenario. Both tests at similar ranges with different combinations of multi-rotor and UGS resulted in a similar percent of packets received,

ranging between 80% and 90% after the link was reestablished. This result was also seen when relaying the C2 link around a physical obstacle. Finally, the system achieved a mean error of 9.76m with a standard deviation of 6.17m and a DRMS of 11.55m while maintaining the follower's position at the midpoint.

Finally, the measurements from the latency test showed that the process of acquiring telemetry and sending commands makes up the majority of the latency, with the GCS processing only taking up a small portion. From these tests, it was found that the downlink time takes approximately 0.24s with a standard deviation of 0.005s, the GCS processing time takes less than 0.003s, and the command uplink time takes 0.22s with a standard deviation of 0.006s.

What are the effects on system performance due to the utilization of COTS, OSH, and OSS?

As indicated by the limitations of existing architectures, systems primarily or completely composed of COTS components have a tendency to have lower overall performance than completely proprietary systems. These COTS components are likely applied to roles that require the intended capabilities of the component be modified to meet the desired functionality. One example of this can be shown with the autopilot. On proprietary systems, the trend is to have the algorithms required to perform the mission be performed onboard the autopilot. This on board processing combined with the proper communication system that allow vehicle to vehicle communication could allow for a higher control loop frequency. Nonproprietary systems require this processing to be completed on the GCS, decreasing the control loop frequency. A decreased control loop frequency results in a longer period of time between commands being sent to and performed by the autopilot's outer loop, which increases the time vehicle can veer off course and induce position error.

Another trend for completely proprietary systems is to utilize custom GCS software that allows for full access to necessary measurements from the autopilot. The GCS software used and other COTS GCS software packages have a limited number of measurements available for manipulation in scripting modules. This again requires the extension of the intended capabilities of the software to accomplish the desired task.

6.3 Recommended Future Work

This research opened many doors for others to investigate new topics pertaining to this system and to apply new topics through the use of this system or similar systems. These recommended topics are outlined below.

This system performed relatively well for close range vehicles such as UGS and multi-rotors UAS. Due to this, it is suggested this system be utilized as a platform for future research on C2 algorithms of these close operating range vehicles. This system is not perfect and can still be improved on. Examples of possible improvements includes further developing the system to utilize a closed loop controller, investigating other methods of determining the commanded position sent to the follower, and investigating the addition of more vehicles.

The system as it stands could be modified to accomplish additional cooperative vehicle tasks which require less positional accuracy. Examples of these tasks include the wide area search problem, persistent surveillance, and perimeter surveillance. These tasks could be accomplished by modifying the position calculation functions contained in the *multi_vehicle_toolbox* function contained in Appendix C. Also, with the addition of IP cameras, the communication system could additionally transmit video down to the GCS to perform visual based missions.

As shown in the formation flight test for a team of fixed wing UAS, this system

does not provide the capability for these airframes to perform formation flight. One major causes of this is the airframes relatively higher speed and the longer operating range from the GCS required. These factors increase the latency and increase the error induced between points. Due to this, it is suggested further investigation be conducted into moving the processing onboard the vehicle. Utilizing the vehicle to vehicle communication capability that is already present could reduce the communication link distance and total system latency. Also, by moving the processing on board, an outer loop controller could be developed to achieve a higher level of control authority. This could be achieved by injecting radio control commands from an on board micro controller into the autopilot input port while the vehicle is in a stabilized mode. This would utilize the inner loop stabilization while providing a higher rate of control over the motion of the aircraft.

Measuring the system latency was one of the more difficult tasks during this research. This is due to the inability to measure time differences between when physical and computational events occur. It is recommended the methods of measuring system latency be further investigated. System latency plays a large roll in positional accuracy of this system and, if better understood, could be used to better predict factors related to the system and therefore better control the system. Additionally, the effects of relatively long range communication on system latency has not been investigated for this or similar systems. This communication latency played a large role in the fixed wing tests performed, and if better understood, could be beneficial in future work related to fixed wing cooperative control.

Finally, it is recommended some of the tests performed be retested. The first set of tests to be accomplished again are the formation flocking tests for a team of two multi-rotor UAS due to the error in the commanded position described in the results. Additionally, the formation flocking test for a team of two fixed wing UAS should also be retested due to use of an inadequate antenna, which did not provide a sufficient C2 link with the aircraft.

Bibliography

1. “Unmanned Systems Integrated Roadmap FY2013-2038,” 2013.
2. Nicholas Lazaredes, “Ukraine’s DIY drone war: Self-taught soldiers facing up to Russian-backed war machine,” 2015.
3. Marcello R Napolitano, Yu Gu, Technical Officer, Curtis E Hanson, and Ms Theresa Stanley, “Cooperative Gust Sensing and Suppression for Aircraft Formation Flight Final Report Cooperative Gust Sensing and Suppression for Aircraft Formation Flight Motivation,” Tech. Rep., NASA, 2012.
4. Yu Gu, Giampiero Campa, Brad Seanor, Srikanth Gururajan, and Marcello R Napolitano, “Autonomous Formation Flight Design and Experiments,” in *Aerial Vehicles*, Thanh Mung Lam, Ed., chapter 12, pp. 236–258. InTech, 2009.
5. Matthew T. Seibert, Andrew J. Stryker, Jill T. Ward, and Chris T. Wellbaum, “SYSTEM ANALYSIS AND PROTOTYPING FOR SINGLE OPERATOR MANAGEMENT OF MULTIPLE UNMANNED AERIAL VEHICLES OPERATING BEYOND LINE OF SIGHT,” M.S. thesis, Air Force Institute of Technology, 2010.
6. Edison Pignaton De Freitas, Tales Heimfarth, Ivayr Farah Netto, Carlos Eduardo Lino, Carlos Eduardo Pereira, Armando Morado Ferreira, Flávio Rech Wagner, and Tony Larsson, “UAV relay network to support WSN connectivity,” in *2010 International Congress on Ultra Modern Telecommunications and Control Systems and Workshops, ICUMT 2010*, 2010.
7. Theodore T Diamond, Adam L Rutherford, and Jonathan B Taylor, “Cooperative Unmanned Aerial Surveillance Control System Architecture,” M.S. thesis, Air Force Institute of Technology, 2009.

8. David Smalley, "Locust: Autonomous, swarming uavs fly into the future," Online, April 2015.
9. Jonathan How, Ellis King, and Yoshiaki Kuwata, "Flight Demonstrations of Cooperative Control for UAV Teams," in *AIAA 3rd "Unmanned Unlimited" Technical Conference, Worksho and Exhibit*, 2004.
10. Derek Kingston, Randal W. Beard, and Ryan S. Holt, "Decentralized perimeter surveillance using a team of UAVs," 2008.
11. Stefan L Hardy, "IMPLEMENTING COOPERATIVE BEHAVIOR & CONTROL USING OPEN SOURCE TECHNOLOGY ACROSS HETEROGENEOUS VEHICLES," M.S. thesis, Air Force Institute of Technology, 2014.
12. "Pixhawk," "<https://pixhawk.org/choice>".
13. "Ardupilot 2.6," "<http://copter.ardupilot.com/wiki/common-apm25-and-26-overview/>".
14. "Pickelo Autopilot," "<http://www.cloudcaptech.com/products/auto-pilots>".
15. "Kestrel Autopilot," "<http://www.lockheedmartin.com/us/products/procerus/kestrel-autopilot.html>".
16. "MAVLink Protocol," "<http://qgroundcontrol.org/mavlink/start>".
17. "Pixhawk Control Architecture," "<https://pixhawk.org/dev/architecture>".
18. Ilker Bekmezci, Ozgur Koray Sahingoz, and amil Temel, "Flying Ad-Hoc Networks (FANETs): A survey," 2013.

19. Jun Li, Yifeng Zhou, and Louise Lamont, “Communication architectures and protocols for networking unmanned aerial vehicles,” in *2013 IEEE Globecom Workshops, GC Wkshps 2013*, 2013.
20. “Mission Planner,” ”<http://planner.ardupilot.com/>”.
21. “APM Planner 2.0,” ”<http://planner2.ardupilot.com/>”.
22. “MAVProxy,” ”<http://dronecode.github.io/MAVProxy/html/index.html>”.
23. Department of Defense, “Dod architecture framework version 2.0,” August 2010.

Appendix . Appendix

A Appendix A: Formation Flocking Leader Vehicle Script

```
1 #FlockingModeLeader (Jeremy Gray Aug 2015)
2 # Gets location request from follower and gives the leaders location
  and heading
3 #
4 # Prerequisites:
5 #     Two (2) instances of MAVProxy are operational
6 #     Vehicles are connected in both instances of MAVProxy
7 # Notes:
8 #     for best results , update system time
9
10 import socket
11 import sys
12 from droneapi.lib import VehicleMode
13 from droneapi.lib import Command
14 from droneapi.lib import mavutil
15 import numpy as np
16 import math
17 import time
18 from datetime import datetime
19 from LLA_ECEF_Convert import LLA_ECEF_Convert
20 from multi_vehicle_toolbox import follower_pos
21
22 '''INIT PARAMS'''
23 freq_control=4.0    #frequency of control loop, must be < follower , must
  be float (0.0)
24 freq_store=2.0      #frequency of data storage, must be float (0.0)
25 freq_print=1.0      #frequency of printed updates, must be float (0.0)
26 msg_size=128        #size of msg to be passed
27
28 '''DRONEAPI INIT'''
29 # Get a local APIConnection to the autopilot (from companion computer or
  GCS).
30 api = local_connect()
31
32 # Create vehicle objects for each vehicle from the APIConnection
33 v_leader = api.get_vehicles()[0]
34 print "Leader Vehicle Object Created"
35
36 '''DATA FILE INIT'''
37 timestr = time.strftime("%m-%d-%Y_%H-%M-%S")    #date-time for file name
38 file_name='leader-gcs-tel_' + timestr          #file name appended with
  date time
39 data_file = open(file_name, 'a')    #create txt doc to append to
40 print 'telemetry file open'
41
42 '''CONNECTION INIT'''
43 #Setup UDP link with leader_server
```

```

44 Port = 50005      # Port to TX/RX to/from follower_client
45 IP = '127.0.0.1'   #Local Host IP
46 s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)      # Create TCP
   socket object
47 print 'socket created'
48 address=(IP,Port)
49
50
51 '''Main LOOP'''
52 t_write=0  #forces first write to occure on start
53 t_print=0
54 print 'starting control loop'
55 #Current location is loc0, next location is loc1
56 while not api.exit:
57     try:
58         #get current time for sleep...
59         t1=time.time()
60
61         #get telemetry information
62         lat=str(v_leader.location.lat)           #latitude (deg)
63         lon=str(v_leader.location.lon)           #longitude (deg)
64         alt_asl = str(v_leader.location.alt)     #altitude above
   sea level (m)
65         p=float(np.deg2rad(v_leader.attitude.pitch)) #pitch (rad) of
   vehicle relative to NEU frame
66         r=float(np.deg2rad(v_leader.attitude.roll))  #roll (rad) of
   vehicle relative to NEU frame
67         y=float(np.deg2rad(v_leader.attitude.yaw))  #yaw (rad) of
   vehicle relative to NEU frame
68         v_b=v_leader.velocity                     #velocity
   vectory (m/s) relative to body
69         t_tel=time.time()                         #time telemetry
   was recieved
70
71         #flush data to leader
72         v_leader.flush()
73
74         #determine gc relative to vehicle frame (NED)
75         v_b=np.array([[v_b[0], v_b[1], v_b[2]]])
76         c_p=np.cos(p); s_p=np.sin(p)
77         c_r=np.cos(r); s_r=np.sin(r)
78         c_y=np.cos(y); s_y=np.sin(y)
79         R_v_b=np.array([ [c_p*c_y, c_p*s_y,
   -s_p ],
80                        [s_r*s_p*c_y-c_r*s_y, s_r*s_p*s_y+c_r*c_y,
   s_r*c_p ],
81                        [c_r*s_p*c_y+s_r*s_y, c_r*s_p*s_y-s_r*c_y,
   c_r*c_p ] ])
82         #rotation transform from vehicle to body
83         v_v=np.dot(R_v_b.T,v_b.T) #velocity vector relative to vehicle
   (NED) frame
84         gc=np.arctan2( v_v[1],v_v[0] ) #ground course (rad) relative

```

```

85     to North
86         v=np.linalg.norm(v-v)                                #velocity of leader,
used to calc Ll
87
88         #if V is too slow use yaw (rad) as ground course
89         if v < 0.25:
90             gc= y
91
92         #build telemetry msg to be a known length (msg_size)
93         tel_msg_raw = '%s %s %s %s %s' %(lat,lon,alt_asl, str(float(gc)),
str(v))    #build msg
94         tel_msg=msg_size* ' ',
95         if len(tel_msg_raw) < len(tel_msg):                    #set msg size to
known length
96             n_spaces=len(tel_msg)-len(tel_msg_raw)
97             tel_msg=tel_msg_raw + n_spaces * ' ',
98         else:
99             print 'err: udp message exceeds length. Increase msg_size'
100             break
101
102         #send leader telemetry to follower over UDP
103         s.sendto(str(tel_msg),address)
104
105         #append data w/ unix time on new line of data txt file , if 1/
freq_store has passed
106         if time.time() - t_write > 1/freq_store:
107             msg_data='%s %s' %(t_tel,tel_msg_raw)
108             data_file.write(msg_data + '\n')
109             t_write=time.time()
110
111         #print update message
112         if time.time() - t_print > 1/freq_print:
113             print 'telemetry sent & stored: ' + str(datetime.now().time
())
114             t_print=time.time()
115
116         #determine sleep time
117         t2=time.time()
118         t_remaining= ( 1/freq_control ) - ( t2 - t1 )
119         if t_remaining > 0:    #sleep for remainder of this control
cycle
120             time.sleep(t_remaining)
121         else:                #the operations in the while loop took
too long
122             print 'freq_control is too high'
123
124     except KeyboardInterrupt: #only way to stop the ride
125         data_file.close()
126         break
127
128     except:
129         print "Unexpected error:", sys.exc_info()[0]

```

```
129         data_file.close()
130         break
131
132     # exit
133     s.close()
134     print 'End of Script'
```

B Appendix B: Formation Flocking Follower Vehicle Script

```
1 #FlockingModeFollower (Jeremy Gray SEP 2015)
2 # Gets location of leader vehicle and sets waypoints to make follower
  vehicle follow at
3 # a fixed offset distance
4 #
5 # Prerequisites:
6 #     Two (2) instances of MAVProxy are operational
7 #     Vehicles are connected in both instances of MAVProxy
8 # Notes:
9 #     for best results , update system time
10
11 import socket
12 import sys
13 import math
14 import time
15 from datetime import datetime
16 import re
17 from numpy import matrix
18 import numpy as np
19 from droneapi.lib import VehicleMode, Location, Command, mavutil
20 from LLA_ECEF_Convert import LLA_ECEF_Convert
21 from multi_vehicle_toolbox import follower_pos
22
23
24 '''INIT PARAMS'''
25 #Follower offset parameters (relative to leader's body frame)
26 off_l1_s=2      #L1 lead time constant [s] for forward offset waypoint
27 off_r = 2      #radial distance [m] away from leader
28 off_theta = 45  #angle (deg) from -x axis (out of tail), CCW is (+)
  rotation
29 alt_agl_cmd=10 #alt agl [m] to be commanded, used in guided_pos
30
31 #timing parameters
32 t_freq=8.0      #control loop frequency, must be slower than follower
  and float (0.0)
33 freq_store=2.0  #frequency of storage of data to disk and must be
  float (0.0)
34 freq_print=1    #frequency of print statements (try to reduce this)
35
36 #other
37 msg_size=128    #size of msg to be passed
38
39 '''DRONEAPI INIT'''
40 # Get a local APIConnection to the autopilot (from companion computer or
  GCS).
41 api = local_connect()
42
43 # Create vehicle objects for follower vehicle from the APIConnection
44 v_follower = api.get_vehicles()[0]
45 print "Follower Vehicle Object Created"
```



```

46
47 '''DATA FILE INIT'''
48 timestr = time.strftime("%m-%d-%Y_%H-%M-%S") #date-time for file name
49 file_name='follower_gcs_tel_' + timestr #file name appended with
    date time
50 data_file = open(file_name, 'a') #create txt doc to append to
51 msg_data='%s %s %s' %(off_r, off_theta, off_ll_s)
52 data_file.write(msg_data + '\n')
53 print 'telemetry file open'
54
55 '''CONNECTION INIT'''
56 #Setup TCP link with leader_server
57 Port = 50005 # Port to TX/RX to/from leader_server
58 IP = '127.0.0.1' #Local Host IP
59 s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
60 print 'socket created'
61 s.bind((IP, Port)) # Connect socket
62 print 'Bound to port ' + str(Port)
63
64 '''Main Loop'''
65 rc_ch=v_follower.channel_readback
66 t_write=0 #forces first write to occure on start
67 t_print=0 #forces first print to occure on start
68 #Current location is pos0, next location is pos1
69 print 'starting control loop'
70 while not api.exit:
71     try:
72         #get current time for sleep...
73         t1=time.time()
74
75         if rc_ch['5'] > 1100: #MANUAL MODE FAIL SAFE, will not store data
76             v_follower.mode = VehicleMode("STABILIZE")
77
78             if time.time() - t_print > 1/freq_print:
79                 print "Follower Mode Set to Manual" + str(datetime.now().time
80 ())
81                 t_print=time.time()
82
83                 time.sleep(0.01)
84
85             else:
86                 #read leader tel from udp port
87                 tel_leader = s.recv(msg_size) #get "lat(deg) lon(deg) alt(m) gc(
88 rad) v(m/s)"
89
90                 #manipulate leader tel to parse out lat,lon,alt,heading/gc,
91                 velocity
92                 pattern = re.compile("[ ]") #Data patern (data seperated
93                 by [ ] i.e space)
94                 param = pattern.split(tel_leader) #split data based on data
95                 patern

```

```

92     pos_leader = np.array([float(param[0]), float(param[1]), float(
param[2])])
93                                     #leader pos [lat(deg) lon(deg) alt(m)]
94     heading_l = np.rad2deg(float(param[3])) #leader ground course
(rad)
95     v_l = float(param[4]) #leader velocity (m/s)
96
97     #calculate desired position
98     off_l1=off_l1_s*v_l #forward offset dist. ( [m] = [s] * [m/s])
99     pos1_f=follower_pos(off_r, off_theta, off_l1,
100                        pos_leader, heading_l) #pos1_f = [lat(deg)
lon(deg) alt(m)]
101
102     #Set new follower guided point
103     guided_pos=Location(pos1_f[0], pos1_f[1]-360, alt_agl_cmd,
is_relative=True)
104     if v_follower.mode != "GUIDED": #if not already in guided...go
guided
105         v_follower.mode = VehicleMode("GUIDED")
106         v_follower.commands.goto(guided_pos) #send guided point
107         v_follower.flush() #flush cmd to follower
108
109     #get telemetry information for storage
110     lat=str(v_follower.location.lat) #latitude (9 bytes CHECK)
111     lon=str(v_follower.location.lon) #longitude (9 bytes CHECK)
112     alt_asl = str(v_follower.location.alt) #altitude above sea level
(6 bytes CHECK)
113     p=float(np.deg2rad(v_follower.attitude.pitch)) #pitch (rad) of
vehicle relative to NEU frame
114     r=float(np.deg2rad(v_follower.attitude.roll)) #roll (rad) of
vehicle relative to NEU frame
115     y=float(np.deg2rad(v_follower.attitude.yaw)) #yaw (rad) of
vehicle relative to NEU frame
116     v_b= v_follower.velocity #velocity in x dir
relative to body (CHECK)
117
118     #determine gc relative to vehicle frame (NEU)
119     v_b=np.array([[v_b[0], v_b[1], v_b[2]]])
120     c_r=np.cos(r); s_r=np.sin(r)
121     c_p=np.cos(p); s_p=np.sin(p)
122     c_y=np.cos(y); s_y=np.sin(y)
123     R_v_b=np.array([ [c_p*c_y, c_p*s_y,
-s_p ],
124                      [s_r*s_p*c_y-c_r*s_y, s_r*s_p*s_y+c_r*c_y,
s_r*c_p ],
125                      [c_r*s_p*c_y+s_r*s_y, c_r*s_p*s_y-s_r*c_y,
c_r*c_p ] ])
126     #rotation transform from vehicle to body
127     v_v=np.dot(R_v_b.T, v_b.T) #velocity vector relative to vehicle (
NEU) frame
128     gc=np.arctan2(v_v[1], v_v[0]) #ground course relative to NEU (
CHECK)

```

```

129     v=np.linalg.norm(v-v)           #velocity of leader, used to calc L1
130     t_tel=time.time()
131
132     #if V is too slow use yaw as ground course
133     if v < 1:
134         gc= y
135
136     v_follower.flush()
137
138     #build telemetry data str
139     tel_msg_raw = '%s %s %s %s %s %s %s %s' %(lat,lon,alt_asl, str(float
140                                           str(pos1_f[0]), str(pos1_f
141                                           [1]), str(pos1_f[2]))
142
143     #append data with unix time on a new line of data txt file
144     if time.time() - t_write > 1/freq_store:
145         msg_data='%s %s' %(t_tel, tel_msg_raw)
146         data_file.write(msg_data + '\n')
147         t_write=time.time()
148
149     #print update message
150     if time.time() - t_print > 1/freq_print:
151         print 'cmd sent & telemetry stored: ' + str(datetime.now().
152         time())
153         t_print=time.time()
154
155     #determine sleep time
156     t2=time.time()
157     t_remaining= ( 1/t_freq ) - ( t2 - t1 )
158     if t_remaining > 0:           #sleep for remainder of this control cycle
159         time.sleep(t_remaining)
160     else:                         #the operations in the while loop took too
161         long
162         print 't_freq is too high'
163
164 except KeyboardInterrupt:    #only way to stop the ride
165     data_file.close()
166     break
167
168 except:
169     print "Unexpected error:", sys.exc_info()[0]
170     data_file.close()
171     break
172
173 # exit
174 s.close()
175 print 'End of Script'

```

C Appendix C: Multi-Vehicle Function Module, as Tested

```

1  '''
2  multi_vehicle_toolbox.py
3      Calculations required for multi-vehicle operations
4          1) flocking follower pos calculation
5          2) comm relay relay vehicle midpoint pos calc
6  '''
7
8  import numpy as np
9  from LLA_ECEF_Convert import LLA_ECEF_Convert
10
11 def follower_pos(off_r , off_theta , off_l1 , loc0_l , heading_l):
12     #function    description:    determines the next desired location of
13     #the follower
14     #            vehicle in lat lon alt (LLA)
15     #Inputs:    off_r:            radial distance away from leader [m]
16     #            off_theta:        angle (deg) from -x axis (out of tail),
17     #CCW is (+) rotation
18     #            off_l1:            distance the guided point is placed
19     #forward of the desired
20     #            follower location
21     #            loc0_l:            location of the leader at current
22     #increment of time
23     #            heading_l:        heading of the leader at current
24     #increment of time
25     #Outputs:    loc1_f:            current desired location of the follower
26
27     #Follower loc1 relative to leader body frame
28     off_theta+=270    #add 270 deg to make offset relative to east (+X
29     axis for math)
30     loc1_f=(off_r - off_l1) * np.array([    np.cos(np.deg2rad(off_theta)
31     ),
32                                           np.sin(np.deg2rad(off_theta)
33     ),
34                                           0
35     ])
36
37     #Follower loc1 relative to Local Level Frame (L, North-East-Up)
38     frame
39     cos_h=np.cos(np.deg2rad(heading_l))
40     sin_h=np.sin(np.deg2rad(heading_l))
41     R_BtoL=np.array([    [cos_h,    sin_h,    0],
42                           [-sin_h,    cos_h,    0],
43                           [0,    0,    1]    ])    #Rotation from body
44     to local
45     loc1_f=np.dot(loc1_f , R_BtoL)
46
47     #Follower loc1 from Local Level Frame (L, North-East-Up) to ECEF (E)
48     phi= np.deg2rad(loc0_l[0])    #latitude of leader
49     la= np.deg2rad(loc0_l[1])    #longitude of leader

```

```

40     sin_la= np.sin(la)
41     cos_la=np.cos(la)
42     sin_phi= np.sin(phi)
43     cos_phi=np.cos(phi)
44     R_LtoE=np.array([ [-sin_la ,  -sin_phi*cos_la ,    cos_phi*cos_la
45 ],
46                       [ cos_la ,  -sin_phi*sin_la ,    cos_phi*sin_la
47 ],
48                       [ 0 ,      cos_phi ,      sin_phi
49 ]   ]) #Rotation from local to ecef
50
51     T_LtoE=LLA_ECEF_Convert(np.rad2deg(phi),np.rad2deg(la),loc0_l[2], '
52 LLAtoECEF')
53     loc1_f= np.dot(R_LtoE,loc1_f) + T_LtoE.T
54
55     #Follower Location ECEF to lat lon alt (LLA)
56     loc1_f= LLA_ECEF_Convert(loc1_f[0], loc1_f[1], loc1_f[2], 'ECEFtoLLA
57 ')
58
59     return loc1_f
60
61 def relay_pos(pos_gcs_llh ,pos_rem_llh):
62     #function description:    Calculates the midpoint between the GCS
63     and remote vehicle
64     #
65     #                        to send the relay vehicle
66     #
67     #Inputs:    pos_gcs_llh:    pos of GCS in lat lon hae
68     #           pos_rem_llh:    pos of remote vehicle in lat lon hae
69     #
70     #Outputs:    pos_rel_llh:    calculated pos of relay vehicle
71     #
72     #Notation:
73     #           Remote vehicle: rem
74     #           Relay vehicle: rel
75     #           Ground Control: GCS
76
77     #convert pos of rem & gcs from llh to ecef
78     pos_rem_ecef=LLA_ECEF_Convert( pos_rem_llh[0], pos_rem_llh[1],
79                                     pos_rem_llh[2], 'LLAtoECEF')
80
81     pos_gcs_ecef=LLA_ECEF_Convert( pos_gcs_llh[0], pos_gcs_llh[1],
82                                     pos_gcs_llh[2], 'LLAtoECEF')
83
84     #calculate midpoint in ecef
85     pos_rel_ecef=pos_gcs_ecef + 0.5*(pos_rem_ecef-pos_gcs_ecef)
86
87     #convert pos of rel from ecef to llh
88     pos_rel_llh=LLA_ECEF_Convert( pos_rel_ecef[0], pos_rel_ecef[1],
89                                     pos_rel_ecef[2], 'ECEFtoLLA')
90
91     return pos_rel_llh

```

D Appendix D: Communication Relay Remote Vehicle Script

```
1 #FlockingModeLeader (Jeremy Gray Aug 2015)
2 # Gets location request from follower and gives the leaders location
  and heading
3 #
4 # Prerequisites:
5 #     Two (2) instances of MAVProxy are operational
6 #     Vehicles are connected in both instances of MAVProxy
7
8 import socket
9 import sys
10 from droneapi.lib import VehicleMode
11 from droneapi.lib import Command
12 from droneapi.lib import mavutil
13 from numpy import matrix
14 import numpy as np
15 import math
16 import time
17 from datetime import datetime
18 from LLA_ECEF_Convert import LLA_ECEF_Convert
19 from multi_vehicle_toolbox import follower_pos
20
21 '''INIT PARAMS'''
22 t_freq=5.0      #control loop frequency, must be slower than follower
23 freq_print=1.0  #rate of printing updates
24 msg_size=128    #size of msg to be passed
25
26 '''DRONEAPI INIT'''
27 # Get a local APIConnection to the autopilot (from companion computer or
  GCS).
28 api = local_connect()
29
30 # Create vehicle objects for each vehicle from the APIConnection
31 v_remote = api.get_vehicles()[0]
32 print "Leader Vehicle Object Created"
33
34 '''CONNECTION INIT'''
35 #Setup UDP link with leader_server
36 Port = 50005    # Port to TX/RX to/from follower_client
37 IP = '127.0.0.1'    #Local Host IP
38 s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)    # Create TCP
  socket object
39 print 'socket created'
40 address=(IP,Port)
41
42 '''Main LOOP'''
43 t_write=0    #forces first write to occur on start
44 t_print=0
45 #Current location is loc0, next location is loc1
46 while not api.exit:
47     try:
```

```

48     #get current time for sleep...
49     t1=time.time()
50
51     #get telemetry information
52     lat=str(v_remote.location.lat)           #latitude (deg)
53     lon=str(v_remote.location.lon)           #longitude (deg)
54     alt_asl = str(v_remote.location.alt)      #altitude above sea
level
55
56     #build telemetry msg to be a known length (msg_size)
57     tel_msg_raw = '%s %s %s' % (lat, lon, alt_asl)    #build msg
58     tel_msg=msg_size*','
59     if len(tel_msg_raw) < len(tel_msg):              #set msg size to
known length
60         n_spaces=len(tel_msg)-len(tel_msg_raw)
61         tel_msg=tel_msg_raw + n_spaces * ','
62
63     v_remote.flush()
64
65     s.sendto(str(tel_msg),address)
66     print 'telemetry sent: ' + str(datetime.now().time())
67
68     #print update message
69     if time.time() - t_print > 1/freq_print:
70         print 'telemetry sent & stored: ' + str(datetime.now().time
())
71         t_print=time.time()
72
73     #determine sleep time
74     t2=time.time()
75     t_remaining= ( 1/t_freq ) - ( t2 - t1 )
76     if t_remaining > 0:      #sleep for remainder of this control
cycle
77         time.sleep(t_remaining)
78     else:                    #the operations in the while loop took
too long
79         print 't_freq is too low'
80
81     except KeyboardInterrupt:
82         break
83
84     except:
85         print "Unexpected error:", sys.exc_info()[0]
86         break
87
88 # exit
89 s.close()
90 print 'End of Script'

```

E Appendix E: Communication Relay Relay Vehicle Script

```
1 #Comm Relay, Relay vehicle client script (Jeremy Gray SEP 2015)
2 # Gets location of remote vehicle and sets waypoints to make relay
  vehicle
3 # transit to a halfway point to relay comm
4 #
5 # Prerequisites:
6 #     Two (2) instances of MAVProxy are operational
7 #     Vehicles are connected in both instances of MAVProxy
8
9 import socket
10 import sys
11 import math
12 import time
13 from datetime import datetime
14 import re
15 import numpy as np
16 from droneapi.lib import VehicleMode, Location, Command, mavutil
17 from LLA_ECEF_Convert import LLA_ECEF_Convert
18 from multi_vehicle_toolbox import relay_pos
19
20 '''INIT PARAMS'''
21 alt_agl_cmd=10 #alt agl [m] to be commanded, used in guided_pos
22 t_freq=10.0 #control loop frequency, must be faster than leader
23 freq_print=1.0 #rate of printing updates
24 msg_size=128 #size of msg to be passed
25
26 '''DRONEAPI INIT'''
27 # Get a local APIConnection to the autopilot (from companion computer or
  GCS).
28 api = local_connect()
29
30 # Create vehicle objects for follower vehicle from the APIConnection
31 v_relay = api.get_vehicles()[0]
32 print "Follower Vehicle Object Created"
33
34 '''CONNECTION INIT'''
35 #Setup TCP link with leader_server
36 Port = 50005 # Port to TX/RX to/from leader_server
37 IP = '127.0.0.1' #Local Host IP
38 s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
39 print 'socket created'
40 s.bind((IP,Port)) # Connect socket
41 print 'Bound to port ' + str(Port)
42
43 '''GET HOME LOCATION'''
44 pos_home_lla=np.array([v_relay.location.lat, v_relay.location.lon,
  v_relay.location.alt])
45 print " Home WP: %s" % pos_home_lla
46
47 '''MAIN LOOP'''
```



```

48 msg_size=128
49 rc_ch=v_relay.channel_readback
50 t_write=0    #forces first write to occure on start
51 t_print=0
52 #remote vehicle is rem, relay is rel
53 while not api.exit:
54     try:
55         if rc_ch['5'] > 1100:
56             v_relay.mode = VehicleMode("STABILIZE")
57             print "Relay Mode Set to Manual"
58             time.sleep(0.25)
59         else:
60             t1=time.time()
61             tel_rem = s.recv(msg_size) #read port
62
63             #manipulate Rx data to parse out remote vehicle pos
64             pattern = re.compile("[ ]")    #Data patern (data seperated
by [ ] i.e space)
65             param = pattern.split(tel_rem)    #split data based on data
patern
66
67             pos_rem_lla = np.array([ float(param[0]) , float(param[1]) ,
float(param[2]) ])
68
69             #calculate desired relay position
70             pos_rel_lla=pos_home_lla , pos_rem_lla)
71             guided_pos=Location(    pos_rel_lla[0] , pos_rel_lla[1]-360,
72                                     alt_agl_cmd , is_relative=True    )
73
74             #Set new follower guided point
75             if v_relay.mode != "GUIDED":
76                 v_relay.mode = VehicleMode("GUIDED")
77                 v_relay.commands.goto(guided_pos)
78                 v_relay.flush()
79                 print 'cmd sent: ' + str(pos_rel_lla)
80
81             #print update message
82             if time.time() - t_print > 1/freq_print:
83                 print 'cmd sent & telemetry stored: ' + str(datetime.now
().time())
84                 t_print=time.time()
85
86             #determine sleep time
87             t2=time.time()
88             t_remaining= ( 1/t_freq ) - ( t2 - t1 )
89             if t_remaining > 0:    #sleep for remainder of this control
cycle
90                 time.sleep(t_remaining)
91             else:    #the operations in the while loop
took too long
92                 print 't_freq is too low'
93

```

```
94     except KeyboardInterrupt:    #only way to stop the ride
95         break
96
97     ##     except:
98     ##         print "Unexpected error:", sys.exc_info()[0]
99     ##         break
100
101
102 # exit
103 s.close()
104 print 'End of Script'
```

F Appendix F: Multi-Vehicle Function Module With Fixed Follower_Pos Calculation

```

1  '''
2  multi_vehicle_toolbox.py
3      Calculations required for multi-vehicle operations
4          1) flocking follower pos calculation
5          2) comm relay relay vehicle midpoint pos calc
6  '''
7
8  import numpy as np
9  from LLA_ECEF_Convert import LLA_ECEF_Convert
10
11 def follower_pos(off_r , off_theta , off_ll , loc0_l , heading_l):
12     #function description: determines the next desired location of
the follower
13     # vehicle in lat lon alt (LLA)
14     #Inputs: off_r: radial distance away from leader [m]
15     # off_theta: angle (deg) from -x axis (out of tail),
CCW is (+) rotation
16     # off_ll: distance the guided point is placed
forward of the desired
17     # follower location
18     # loc0_l: location of the leader at current
increment of time
19     # heading_l: heading of the leader at current
increment of time
20     #Outputs: loc1_f: current desired location of the follower
21
22
23     #Follower loc1 relative to leader body frame
24     off_theta+=270 #add 270 deg to make offset relative to east (+X
axis for math)
25     off_theta=np.deg2rad(off_theta)
26     loc1_f= off_r*np.array([np.cos(off_theta),np.sin(off_theta),0]) -
27         off_ll*np.array([1,0,0])
28
29     #Follower loc1 relative to Local Level Frame (L, North-East-Up)
frame
30     cos_h=np.cos(np.deg2rad(heading_l))
31     sin_h=np.sin(np.deg2rad(heading_l))
32     R_BtoL=np.array([ [cos_h, sin_h, 0],
33         [-sin_h, cos_h, 0],
34         [0, 0, 1] ]) #Rotation from body
to local
35     loc1_f=np.dot(loc1_f , R_BtoL)
36
37     #Follower loc1 from Local Level Frame (L, North-East-Up) to ECEF (E)
38     phi= np.deg2rad(loc0_l[0]) #latitude of leader
39     la= np.deg2rad(loc0_l[1]) #longitude of leader
40     sin_la= np.sin(la)
41     cos_la=np.cos(la)

```

```

42     sin_phi= np.sin(phi)
43     cos_phi=np.cos(phi)
44     R_LtoE=np.array([ [-sin_la ,  -sin_phi*cos_la ,    cos_phi*cos_la
45 ],
46                       [ cos_la ,  -sin_phi*sin_la ,    cos_phi*sin_la
47 ],
48                       [0 ,    cos_phi ,    sin_phi
49 ]    ]) #Rotation from local to ecef
50
51     T_LtoE=LLA_ECEF.Convert(np.rad2deg(phi),np.rad2deg(la),loc0_l[2], '
52     LLAtoECEF')
53     loc1_f= np.dot(R_LtoE,loc1_f) + T_LtoE.T
54
55     #Follower Location ECEF to lat lon alt (LLA)
56     loc1_f= LLA_ECEF.Convert(loc1_f[0], loc1_f[1], loc1_f[2], 'ECEFtoLLA
57     ')
58
59     return loc1_f
60
61 def relay_pos(pos_gcs_llh ,pos_rem_llh):
62     #function description:    Calculates the midpoint between the GCS
63     and remote vehicle
64     #
65     #                        to send the relay vehicle
66     #
67     #Inputs:    pos_gcs_llh:    pos of GCS in lat lon hae
68     #           pos_rem_llh:    pos of remote vehicle in lat lon hae
69     #
70     #Outputs:    pos_rel_llh:    calculated pos of relay vehicle
71     #
72     #Notation:
73     #           Remote vehicle: rem
74     #           Relay vehicle: rel
75     #           Ground Control: GCS
76
77     #convert pos of rem & gcs from llh to ecef
78     pos_rem_ecef=LLA_ECEF.Convert( pos_rem_llh[0], pos_rem_llh[1] ,
79     pos_rem_llh[2], 'LLAtoECEF')
80
81     pos_gcs_ecef=LLA_ECEF.Convert( pos_gcs_llh[0], pos_gcs_llh[1] ,
82     pos_gcs_llh[2], 'LLAtoECEF')
83
84     #calculate midpoint in ecef
85     pos_rel_ecef=pos_gcs_ecef + 0.5*(pos_rem_ecef-pos_gcs_ecef)
86
87     #convert pos of rel from ecef to llh
88     pos_rel_llh=LLA_ECEF.Convert( pos_rel_ecef[0], pos_rel_ecef[1] ,
89     pos_rel_ecef[2], 'ECEFtoLLA')
90
91     return pos_rel_llh

```

G Appendix G: Traxxas EMAXX UGS Pixhawk Parameters

#NOTE: 8/25/2015 12:26:27 PM	CAM_TRIGG_TYPE,0
AHRS_COMP_BETA,0.1	CH7_OPTION,1
AHRS_EKF_USE,0	CLI_ENABLED,0
AHRS_GPS_GAIN,1	COMPASS_AUTODEC,1
AHRS_GPS_MINSATS,6	COMPASS_DEC,-0.09905119
AHRS_GPS_USE,1	COMPASS_DEV_ID,73225
AHRS_ORIENTATION,0	COMPASS_DEV_ID2,131594
AHRS_RP_P,0.2	COMPASS_DEV_ID3,0
AHRS_TRIM_X,0	COMPASS_EXTERN2,0
AHRS_TRIM_Y,0	COMPASS_EXTERN3,0
AHRS_TRIM_Z,0	COMPASS_EXTERNAL,1
AHRS_WIND_MAX,0	COMPASS_LEARN,0
AHRS_YAW_P,0.2	COMPASS_MOT_X,0
AUTO_KICKSTART,0	COMPASS_MOT_Y,0
AUTO_TRIGGER_PIN,-1	COMPASS_MOT_Z,0
BATT_AMP_OFFSET,0	COMPASS_MOT2_X,0
BATT_AMP_PERVOLT,17	COMPASS_MOT2_Y,0
BATT_CAPACITY,3300	COMPASS_MOT2_Z,0
BATT_CURR_PIN,-1	COMPASS_MOT3_X,0
BATT_MONITOR,0	COMPASS_MOT3_Y,0
BATT_VOLT_MULT,10.1	COMPASS_MOT3_Z,0
BATT_VOLT_PIN,-1	COMPASS_MOTCT,0
BATT2_AMP_OFFSET,0	COMPASS_OFS_X,-101.8812
BATT2_AMP_PERVOL,17	COMPASS_OFS_Y,-12.68495
BATT2_CAPACITY,3300	COMPASS_OFS_Z,-101.4908
BATT2_CURR_PIN,3	COMPASS_OFS2_X,271.3355
BATT2_MONITOR,0	COMPASS_OFS2_Y,-438.0417
BATT2_VOLT_MULT,10.1	COMPASS_OFS2_Z,286.0888
BATT2_VOLT_PIN,2	COMPASS_OFS3_X,0
BRAKING_PERCENT,0	COMPASS_OFS3_Y,0
BRAKING_SPEEDERR,3	COMPASS_OFS3_Z,0
BRD_PWM_COUNT,4	COMPASS_ORIENT,0
BRD_SAFETYENABLE,1	COMPASS_ORIENT2,0
BRD_SBUS_OUT,0	COMPASS_ORIENT3,0
BRD_SER1_RTSCTS,2	COMPASS_PRIMARY,0
BRD_SER2_RTSCTS,2	COMPASS_USE,1
CAM_DURATION,10	COMPASS_USE2,1
CAM_SERVO_OFF,1100	COMPASS_USE3,1
CAM_SERVO_ON,1300	CRUISE_SPEED,1.5
CAM_TRIGG_DIST,0	CRUISE_THROTTLE,33

EKF_ABIAS_PNOISE,5E-05	GPS_AUTO_SWITCH,1
EKF_ACC_PNOISE,0.25	GPS_INJECT_TO,127
EKF_ALT_NOISE,1	GPS_MIN_DGPS,100
EKF_ALT_SOURCE,1	GPS_MIN_ELEV,-100
EKF_EAS_GATE,10	GPS_NAVFILTER,8
EKF_EAS_NOISE,1.4	GPS_RAW_DATA,0
EKF_FALLBACK,1	GPS_SBAS_MODE,2
EKF_FLOW_DELAY,25	GPS_SBP_LOGMASK,-256
EKF_FLOW_GATE,5	GPS_TYPE,1
EKF_FLOW_NOISE,0.15	GPS_TYPE2,0
EKF_GBIAS_PNOISE,8E-06	INITIAL_MODE,0
EKF_GLITCH_ACCEL,150	INS_ACC2OFFS_X,0
EKF_GLITCH_RAD,15	INS_ACC2OFFS_Y,0
EKF_GND_GRADIENT,2	INS_ACC2OFFS_Z,0
EKF_GPS_TYPE,0	INS_ACC2SCAL_X,1
EKF_GYRO_PNOISE,0.015	INS_ACC2SCAL_Y,1
EKF_HGT_GATE,10	INS_ACC2SCAL_Z,1
EKF_MAG_CAL,1	INS_ACC3OFFS_X,0
EKF_MAG_GATE,3	INS_ACC3OFFS_Y,0
EKF_MAG_NOISE,0.05	INS_ACC3OFFS_Z,0
EKF_MAGB_PNOISE,0.0003	INS_ACC3SCAL_X,0
EKF_MAGE_PNOISE,0.0003	INS_ACC3SCAL_Y,0
EKF_MAX_FLOW,2.5	INS_ACC3SCAL_Z,0
EKF_POS_DELAY,220	INS_ACCEL_FILTER,10
EKF_POS_GATE,10	INS_ACCOFFS_X,0
EKF_POSNE_NOISE,0.5	INS_ACCOFFS_Y,0
EKF_RNG_GATE,5	INS_ACCOFFS_Z,0
EKF_VEL_DELAY,220	INS_ACCSCAL_X,1
EKF_VEL_GATE,5	INS_ACCSCAL_Y,1
EKF_VELD_NOISE,0.7	INS_ACCSCAL_Z,1
EKF_VELNE_NOISE,0.5	INS_GYR2OFFS_X,0.007782747
EKF_WIND_PNOISE,0.1	INS_GYR2OFFS_Y,0.008182988
EKF_WIND_PSCALE,0.5	INS_GYR2OFFS_Z,-0.004160143
FORMAT_VERSION,16	INS_GYR3OFFS_X,0
FS_ACTION,2	INS_GYR3OFFS_Y,0
FS_GCS_ENABLE,0	INS_GYR3OFFS_Z,0
FS_THR_ENABLE,1	INS_GYRO_FILTER,10
FS_THR_VALUE,910	INS_GYROFFS_X,-0.002938097
FS_TIMEOUT,5	INS_GYROFFS_Y,0.03547082
GCS_PID_MASK,0	INS_GYROFFS_Z,0.006362518
GND_ABS_PRESS,98397.73	INS_PRODUCT_ID,5
GND_ALT_OFFSET,0	LEARN_CH,7
GND_TEMP,33.29029	LOG_BITMASK,65535

MAG_ENABLE,0
 MIS_RESTART,0
 MIS_TOTAL,6
 MNT_ANGMAX_PAN,4500
 MNT_ANGMAX_ROL,4500
 MNT_ANGMAX_TIL,4500
 MNT_ANGMIN_PAN,-4500
 MNT_ANGMIN_ROL,-4500
 MNT_ANGMIN_TIL,-4500
 MNT_DEFLT_MODE,3
 MNT_JSTICK_SPD,0
 MNT_K_RATE,5
 MNT_LEAD_PTCH,0
 MNT_LEAD_RLL,0
 MNT_NEUTRAL_X,0
 MNT_NEUTRAL_Y,0
 MNT_NEUTRAL_Z,0
 MNT_OFF_ACC_X,0
 MNT_OFF_ACC_Y,0
 MNT_OFF_ACC_Z,0
 MNT_OFF_GYRO_X,0
 MNT_OFF_GYRO_Y,0
 MNT_OFF_GYRO_Z,0
 MNT_OFF_JNT_X,0
 MNT_OFF_JNT_Y,0
 MNT_OFF_JNT_Z,0
 MNT_RC_IN_PAN,0
 MNT_RC_IN_ROLL,0
 MNT_RC_IN_TILT,0
 MNT_RETRACT_X,0
 MNT_RETRACT_Y,0
 MNT_RETRACT_Z,0
 MNT_STAB_PAN,0
 MNT_STAB_ROLL,0
 MNT_STAB_TILT,0
 MNT_TYPE,0
 MODE_CH,8
 MODE1,10
 MODE2,0
 MODE3,2
 MODE4,3
 MODE5,10
 MODE6,0

NAVL1_DAMPING,0.9
 NAVL1_PERIOD,8
 PIVOT_TURN_ANGLE,30
 RC1_DZ,30
 RC1_MAX,1760
 RC1_MIN,1248
 RC1_REV,1
 RC1_TRIM,1496
 RC10_DZ,0
 RC10_FUNCTION,0
 RC10_MAX,1900
 RC10_MIN,1100
 RC10_REV,1
 RC10_TRIM,1500
 RC11_DZ,0
 RC11_FUNCTION,0
 RC11_MAX,1900
 RC11_MIN,1100
 RC11_REV,1
 RC11_TRIM,1500
 RC12_DZ,0
 RC12_FUNCTION,0
 RC12_MAX,1900
 RC12_MIN,1100
 RC12_REV,1
 RC12_TRIM,1500
 RC13_DZ,0
 RC13_FUNCTION,0
 RC13_MAX,1900
 RC13_MIN,1100
 RC13_REV,1
 RC13_TRIM,1500
 RC14_DZ,0
 RC14_FUNCTION,0
 RC14_MAX,1900
 RC14_MIN,1100
 RC14_REV,1
 RC14_TRIM,1500
 RC2_DZ,30
 RC2_FUNCTION,0
 RC2_MAX,2009
 RC2_MIN,1298
 RC2_REV,1

RC2_TRIM,1500
RC3_DZ,0
RC3_MAX,2017
RC3_MIN,1120
RC3_REV,1
RC3_TRIM,1529
RC4_DZ,0
RC4_FUNCTION,0
RC4_MAX,2016
RC4_MIN,992
RC4_REV,1
RC4_TRIM,1502
RC5_DZ,0
RC5_FUNCTION,0
RC5_MAX,2017
RC5_MIN,2015
RC5_REV,1
RC5_TRIM,2017
RC6_DZ,0
RC6_FUNCTION,0
RC6_MAX,2017
RC6_MIN,2015
RC6_REV,1
RC6_TRIM,2016
RC7_DZ,0
RC7_FUNCTION,0
RC7_MAX,1146
RC7_MIN,1145
RC7_REV,1
RC7_TRIM,1146
RC8_DZ,0
RC8_FUNCTION,0
RC8_MAX,2017
RC8_MIN,991
RC8_REV,1
RC8_TRIM,2016
RC9_DZ,0
RC9_FUNCTION,0
RC9_MAX,1900
RC9_MIN,1100
RC9_REV,1
RC9_TRIM,1500
RCMAP_PITCH,3

RCMAP_ROLL,1
RCMAP_THROTTLE,2
RCMAP_YAW,4
RELAY_DEFAULT,0
RELAY_PIN,54
RELAY_PIN2,55
RELAY_PIN3,-1
RELAY_PIN4,-1
RNGFND_DEBOUNCE,2
RNGFND_FUNCTION,0
RNGFND_GNDCLEAR,10
RNGFND_MAX_CM,700
RNGFND_MIN_CM,20
RNGFND_OFFSET,0
RNGFND_PIN,-1
RNGFND_PWRRNG,0
RNGFND_RMETRIC,1
RNGFND_SCALING,3
RNGFND_SETTLE,0
RNGFND_STOP_PIN,-1
RNGFND_TRIGGR_CM,100
RNGFND_TURN_ANGL,45
RNGFND_TURN_TIME,1
RNGFND_TYPE,0
RNGFND2_FUNCTION,0
RNGFND2_GNDCLEAR,10
RNGFND2_MAX_CM,700
RNGFND2_MIN_CM,20
RNGFND2_OFFSET,0
RNGFND2_PIN,-1
RNGFND2_RMETRIC,1
RNGFND2_SCALING,3
RNGFND2_SETTLE,0
RNGFND2_STOP_PIN,-1
RNGFND2_TYPE,0
RSSI_PIN,-1
RST_SWITCH_CH,0
SCHED_DEBUG,0
SERIAL0_BAUD,115
SERIAL1_BAUD,57
SERIAL1_PROTOCOL,1
SERIAL2_BAUD,57
SERIAL2_PROTOCOL,1

SERIAL3_BAUD,38	SR2_EXTRA2,1
SERIAL3_PROTOCOL,5	SR2_EXTRA3,1
SERIAL4_BAUD,38	SR2_PARAMS,10
SERIAL4_PROTOCOL,5	SR2_POSITION,1
SKID_STEER_IN,0	SR2_RAW_CTRL,1
SKID_STEER_OUT,0	SR2_RAW_SENS,1
SKIP_GYRO_CAL,0	SR2_RC_CHAN,1
SPEED_TURN_DIST,2	SR3_EXT_STAT,1
SPEED_TURN_GAIN,1	SR3_EXTRA1,1
SPEED2THR_D,0.5	SR3_EXTRA2,1
SPEED2THR_I,0.5	SR3_EXTRA3,1
SPEED2THR_IMAX,5000	SR3_PARAMS,10
SPEED2THR_P,0.5	SR3_POSITION,1
SR0_EXT_STAT,2	SR3_RAW_CTRL,1
SR0_EXTRA1,6	SR3_RAW_SENS,1
SR0_EXTRA2,6	SR3_RC_CHAN,1
SR0_EXTRA3,1	STEER2SRV_D,0.2
SR0_PARAMS,10	STEER2SRV_FF,0
SR0_POSITION,2	STEER2SRV_I,0.1
SR0_RAW_CTRL,4	STEER2SRV_IMAX,5000
SR0_RAW_SENS,1	STEER2SRV_MINSPD,1
SR0_RC_CHAN,2	STEER2SRV_P,1.5
SR1_EXT_STAT,4	STEER2SRV_TCONST,0.75
SR1_EXTRA1,4	SYS_NUM_RESETS,213
SR1_EXTRA2,4	SYSID_MYGCS,255
SR1_EXTRA3,4	SYSID_SW_TYPE,20
SR1_PARAMS,10	SYSID_THISMAV,1
SR1_POSITION,4	TELEM_DELAY,0
SR1_RAW_CTRL,4	THR_MAX,100
SR1_RAW_SENS,4	THR_MIN,0
SR1_RC_CHAN,4	THR_SLEWRATE,100
SR2_EXT_STAT,1	TURN_MAX_G,1.1
SR2_EXTRA1,1	WP_RADIUS,2

H Appendix H: X8 Multi-Rotor UAS Pixhawk Parameters

#NOTE: 10/15/2015 3:47:42 PM	ACRO_YAW_P,3
ACRO_BAL_PITCH,1	AHRS_COMP_BETA,0.1
ACRO_BAL_ROLL,1	AHRS_EKF_USE,0
ACRO_EXPO,0.3	AHRS_GPS_GAIN,1
ACRO_RP_P,4.5	AHRS_GPS_MINSATS,6
ACRO_TRAINER,2	AHRS_GPS_USE,1

AHRS_ORIENTATION,0
 AHRS_RP_P,0.1
 AHRS_TRIM_X,-0.003521048
 AHRS_TRIM_Y,0.01564041
 AHRS_TRIM_Z,0
 AHRS_WIND_MAX,0
 AHRS_YAW_P,0.1
 ANGLE_MAX,4500
 ARMING_CHECK,1
 ATC_ACCEL_RP_MAX,72000
 ATC_ACCEL_Y_MAX,18000
 ATC_RATE_FF_ENAB,1
 ATC_RATE_RP_MAX,9000
 ATC_RATE_Y_MAX,9000
 ATC_SLEW_YAW,1000
 BAROGLTCH_ACCEL,1500
 BAROGLTCH_DIST,500
 BAROGLTCH_ENABLE,1
 BATT_AMP_OFFSET,0
 BATT_AMP_PERVOLT,17
 BATT_CAPACITY,6000
 BATT_CURR_PIN,3
 BATT_MONITOR,4
 BATT_VOLT_MULT,10.1
 BATT_VOLT_PIN,2
 BATT_VOLT2_MULT,1
 BATT_VOLT2_PIN,-1
 BRD_PWM_COUNT,4
 BRD_SAFETYENABLE,1
 BRD_SER1_RTSCCTS,2
 BRD_SER2_RTSCCTS,2
 CAM_DURATION,10
 CAM_SERVO_OFF,1100
 CAM_SERVO_ON,1300
 CAM_TRIGG_DIST,0
 CAM_TRIGG_TYPE,0
 CH7_OPT,18
 CH8_OPT,0
 CHUTE_ALT_MIN,10
 CHUTE_ENABLED,0
 CHUTE_SERVO_OFF,1100
 CHUTE_SERVO_ON,1300
 CHUTE_TYPE,0

CIRCLE_RADIUS,1000
 CIRCLE_RATE,20
 COMPASS_AUTODEC,1
 COMPASS_DEC,0
 COMPASS_DEV_ID,73225
 COMPASS_DEV_ID2,131594
 COMPASS_DEV_ID3,0
 COMPASS_EXTERNAL,1
 COMPASS_LEARN,0
 COMPASS_MOT_X,0
 COMPASS_MOT_Y,0
 COMPASS_MOT_Z,0
 COMPASS_MOT2_X,0
 COMPASS_MOT2_Y,0
 COMPASS_MOT2_Z,0
 COMPASS_MOT3_X,0
 COMPASS_MOT3_Y,0
 COMPASS_MOT3_Z,0
 COMPASS_MOTCT,0
 COMPASS_OFS_X,-91
 COMPASS_OFS_Y,-28
 COMPASS_OFS_Z,-148
 COMPASS_OFS2_X,-59
 COMPASS_OFS2_Y,111
 COMPASS_OFS2_Z,219
 COMPASS_OFS3_X,0
 COMPASS_OFS3_Y,0
 COMPASS_OFS3_Z,0
 COMPASS_ORIENT,0
 COMPASS_PRIMARY,0
 COMPASS_USE,1
 DCM_CHECK_THRESH,0.8
 EKF_ABIAIS_PNOISE,0.0001
 EKF_ACC_PNOISE,0.25
 EKF_ALT_NOISE,1
 EKF_CHECK_THRESH,0.8
 EKF_EAS_GATE,10
 EKF_EAS_NOISE,1.4
 EKF_GBIAS_PNOISE,1E-06
 EKF_GLITCH_ACCEL,150
 EKF_GLITCH_RAD,15
 EKF_GPS_TYPE,0
 EKF_GYRO_PNOISE,0.015

EKF_HGT_GATE,10
 EKF_MAG_CAL,1
 EKF_MAG_GATE,3
 EKF_MAG_NOISE,0.05
 EKF_MAGB_PNOISE,0.0003
 EKF_MAGE_PNOISE,0.0003
 EKF_POS_DELAY,220
 EKF_POS_GATE,10
 EKF_POSNE_NOISE,0.5
 EKF_VEL_DELAY,220
 EKF_VEL_GATE,6
 EKF_VELD_NOISE,0.7
 EKF_VELNE_NOISE,0.5
 EKF_WIND_PNOISE,0.1
 EKF_WIND_PSCALE,0.5
 ESC,0
 FENCE_ACTION,1
 FENCE_ALT_MAX,100
 FENCE_ENABLE,0
 FENCE_MARGIN,2
 FENCE_RADIUS,300
 FENCE_TYPE,3
 FLOW_ENABLE,0
 FLTMODE1,3
 FLTMODE2,16
 FLTMODE3,3
 FLTMODE4,0
 FLTMODE5,6
 FLTMODE6,2
 FRAME,1
 FS_BATT_ENABLE,1
 FS_BATT_MAH,20
 FS_BATT_VOLTAGE,14
 FS_GCS_ENABLE,1
 FS_GPS_ENABLE,2
 FS_THR_ENABLE,1
 FS_THR_VALUE,975
 GND_ABS_PRESS,98177.3
 GND_ALT_OFFSET,0
 GND_TEMP,37.70488
 GPS_AUTO_SWITCH,1
 GPS_HDOP_GOOD,200
 GPS_MIN_DGPS,100

GPS_NAVFILTER,8
 GPS_TYPE,1
 GPS_TYPE2,0
 GPSGLITCH_ACCEL,1000
 GPSGLITCH_ENABLE,1
 GPSGLITCH_RADIUS,200
 HLD_LAT_P,1
 INAV_TC_XY,2.5
 INAV_TC_Z,5
 INS_ACC2OFFS_X,1.147846
 INS_ACC2OFFS_Y,1.140518
 INS_ACC2OFFS_Z,1.171686
 INS_ACC2SCAL_X,1.040946
 INS_ACC2SCAL_Y,0.9906676
 INS_ACC2SCAL_Z,0.9855135
 INS_ACC3OFFS_X,0
 INS_ACC3OFFS_Y,0
 INS_ACC3OFFS_Z,0
 INS_ACC3SCAL_X,0
 INS_ACC3SCAL_Y,0
 INS_ACC3SCAL_Z,0
 INS_ACCOFFS_X,-0.004180954
 INS_ACCOFFS_Y,-0.1249053
 INS_ACCOFFS_Z,-0.1089551
 INS_ACCSCAL_X,1.004815
 INS_ACCSCAL_Y,0.9986967
 INS_ACCSCAL_Z,0.9888687
 INS_GYR2OFFS_X,-0.003148957
 INS_GYR2OFFS_Y,0.0173818
 INS_GYR2OFFS_Z,-0.009323909
 INS_GYR3OFFS_X,0
 INS_GYR3OFFS_Y,0
 INS_GYR3OFFS_Z,0
 INS_GYROFFS_X,-0.0135583
 INS_GYROFFS_Y,0.03729856
 INS_GYROFFS_Z,0.008076278
 INS_MPU6K_FILTER,0
 INS_PRODUCT_ID,0
 LAND_REPOSITION,1
 LAND_SPEED,50
 LOG_BITMASK,26622
 LOITER_LAT_D,0
 LOITER_LAT_I,0.5

LOITER_LAT_IMAX,1000
 LOITER_LAT_P,1
 LOITER_LON_D,0
 LOITER_LON_I,0.5
 LOITER_LON_IMAX,1000
 LOITER_LON_P,1
 MAG_ENABLE,1
 MIS_RESTART,0
 MIS_TOTAL,2
 MNT_ANGMAX_PAN,4500
 MNT_ANGMAX_ROL,4500
 MNT_ANGMAX_TIL,0
 MNT_ANGMIN_PAN,-4500
 MNT_ANGMIN_ROL,-4500
 MNT_ANGMIN_TIL,-9000
 MNT_CONTROL_X,0
 MNT_CONTROL_Y,0
 MNT_CONTROL_Z,0
 MNT_JSTICK_SPD,0
 MNT_MODE,3
 MNT_NEUTRAL_X,0
 MNT_NEUTRAL_Y,0
 MNT_NEUTRAL_Z,0
 MNT_RC_IN_PAN,0
 MNT_RC_IN_ROLL,0
 MNT_RC_IN_TILT,6
 MNT_RETRACT_X,0
 MNT_RETRACT_Y,0
 MNT_RETRACT_Z,0
 MNT_STAB_PAN,0
 MNT_STAB_ROLL,0
 MNT_STAB_TILT,0
 MOT_SPIN_ARMED,70
 MOT_TCRV_ENABLE,1
 MOT_TCRV_MAXPCT,93
 MOT_TCRV_MIDPCT,52
 OF_PIT_D,0.12
 OF_PIT_I,0.5
 OF_PIT_IMAX,100
 OF_PIT_P,2.5
 OF_RLL_D,0.12
 OF_RLL_I,0.5
 OF_RLL_IMAX,100

OF_RLL_P,2.5
 PHLD_BRAKE_ANGLE,3000
 PHLD_BRAKE_RATE,8
 PILOT_ACCEL_Z,250
 PILOT_VELZ_MAX,250
 POSCON_THR_HOVER,414
 RALLY_LIMIT_KM,0.3
 RALLY_TOTAL,0
 RATE_PIT_D,0.005
 RATE_PIT_I,0.1999
 RATE_PIT_IMAX,5000
 RATE_PIT_P,0.1999
 RATE_RLL_D,0.005
 RATE_RLL_I,0.1999
 RATE_RLL_IMAX,5000
 RATE_RLL_P,0.1999
 RATE_YAW_D,0.005
 RATE_YAW_I,0.02
 RATE_YAW_IMAX,1000
 RATE_YAW_P,0.16
 RC_FEEL_RP,15
 RC_SPEED,490
 RC1_DZ,30
 RC1_MAX,1931
 RC1_MIN,1080
 RC1_REV,1
 RC1_TRIM,1506
 RC10_DZ,0
 RC10_FUNCTION,0
 RC10_MAX,1900
 RC10_MIN,1100
 RC10_REV,1
 RC10_TRIM,1500
 RC11_DZ,0
 RC11_FUNCTION,0
 RC11_MAX,1900
 RC11_MIN,1100
 RC11_REV,1
 RC11_TRIM,1500
 RC12_DZ,0
 RC12_FUNCTION,0
 RC12_MAX,1900
 RC12_MIN,1100

RC12_REV,1	RC7_MAX,1510
RC12_TRIM,1500	RC7_MIN,1084
RC13_DZ,0	RC7_REV,1
RC13_FUNCTION,0	RC7_TRIM,1510
RC13_MAX,1900	RC8_DZ,0
RC13_MIN,1100	RC8_FUNCTION,0
RC13_REV,1	RC8_MAX,1900
RC13_TRIM,1500	RC8_MIN,1100
RC14_DZ,0	RC8_REV,1
RC14_FUNCTION,0	RC8_TRIM,1509
RC14_MAX,1900	RC9_DZ,0
RC14_MIN,1100	RC9_FUNCTION,7
RC14_REV,1	RC9_MAX,1520
RC14_TRIM,1500	RC9_MIN,1000
RC2_DZ,30	RC9_REV,1
RC2_MAX,1933	RC9_TRIM,1500
RC2_MIN,1082	RCMAP_PITCH,2
RC2_REV,1	RCMAP_ROLL,1
RC2_TRIM,1508	RCMAP_THROTTLE,3
RC3_DZ,30	RCMAP_YAW,4
RC3_MAX,1851	RELAY_PIN,54
RC3_MIN,1169	RELAY_PIN2,-1
RC3_REV,1	RNGFND_FUNCTION,0
RC3_TRIM,1171	RNGFND_GAIN,0.8
RC4_DZ,40	RNGFND_MAX_CM,700
RC4_MAX,1935	RNGFND_MIN_CM,20
RC4_MIN,1084	RNGFND_OFFSET,0
RC4_REV,1	RNGFND_PIN,-1
RC4_TRIM,1509	RNGFND_RMTRIC,1
RC5_DZ,0	RNGFND_SCALING,3
RC5_FUNCTION,0	RNGFND_SETTLE_MS,0
RC5_MAX,1892	RNGFND_STOP_PIN,-1
RC5_MIN,1196	RNGFND_TYPE,0
RC5_REV,1	RSSI_PIN,-1
RC5_TRIM,1500	RSSI_RANGE,5
RC6_DZ,0	RTL_ALT,2000
RC6_FUNCTION,0	RTL_ALT_FINAL,0
RC6_MAX,1851	RTL_LOIT_TIME,5000
RC6_MIN,1169	SCHED_DEBUG,0
RC6_REV,1	SERIAL0_BAUD,115
RC6_TRIM,1568	SERIAL1_BAUD,57
RC7_DZ,0	SERIAL2_BAUD,57
RC7_FUNCTION,0	SERIAL2_PROTOCOL,2

SIMPLE,0	SUPER_SIMPLE,0
SR0_EXT_STAT,0	SYSID_MYGCS,255
SR0_EXTRA1,0	SYSID_SW_MREV,120
SR0_EXTRA2,0	SYSID_SW_TYPE,10
SR0_EXTRA3,0	SYSID_THISMAV,1
SR0_PARAMS,10	TELEM_DELAY,0
SR0_POSITION,0	TERRAIN_ENABLE,1
SR0_RAW_CTRL,0	TERRAIN_SPACING,100
SR0_RAW_SENS,0	THR_ACCEL_D,0
SR0_RC_CHAN,0	THR_ACCEL_I,1
SR1_EXT_STAT,0	THR_ACCEL_IMAX,800
SR1_EXTRA1,0	THR_ACCEL_P,0.5
SR1_EXTRA2,0	THR_ALT_P,1
SR1_EXTRA3,0	THR_DZ,100
SR1_PARAMS,0	THR_MAX,1000
SR1_POSITION,0	THR_MID,450
SR1_RAW_CTRL,0	THR_MIN,130
SR1_RAW_SENS,0	THR_RATE_P,5
SR1_RC_CHAN,0	TRIM_THROTTLE,414
SR2_EXT_STAT,0	TUNE,0
SR2_EXTRA1,0	TUNE_HIGH,1000
SR2_EXTRA2,0	TUNE_LOW,0
SR2_EXTRA3,0	WP_YAW_BEHAVIOR,2
SR2_PARAMS,0	WPNAV_ACCEL,250
SR2_POSITION,0	WPNAV_ACCEL_Z,100
SR2_RAW_CTRL,0	WPNAV_LOIT_JERK,1000
SR2_RAW_SENS,0	WPNAV_LOIT_SPEED,200
SR2_RC_CHAN,0	WPNAV_RADIUS,100
STB_PIT_P,4	WPNAV_SPEED,200
STB_RLL_P,4	WPNAV_SPEED_DN,200
STB_YAW_P,2.5	WPNAV_SPEED_UP,200

I Appendix I: Supper Sky Surfer UAS Pixhawk Parameters

#NOTE:	AFS_AMSL_LIMIT,0
10/15/2015 12:11:10 PM	AFS_ENABLE,0
Plane: Skywalker	AFS_HB_PIN,-1
	AFS_MAN_PIN,-1
ACRO_LOCKING,0	AFS_MAX_COM_LOSS,0
ACRO_PITCH_RATE,180	AFS_MAX_GPS_LOSS,0
ACRO_ROLL_RATE,180	AFS_QNH_PRESSURE,0
AFS_AMSL_ERR_GPS,-1	AFS_RC_FAIL_MS,0

AFS_TERM_ACTION,0
 AFS_TERM_PIN,-1
 AFS_TERMINATE,0
 AFS_WP_COMMS,0
 AFS_WP_GPS_LOSS,0
 AHRS_COMP_BETA,0.1
 AHRS_EKF_USE,0
 AHRS_GPS_GAIN,1
 AHRS_GPS_MINSATS,6
 AHRS_GPS_USE,1
 AHRS_ORIENTATION,0
 AHRS_RP_P,0.2
 AHRS_TRIM_X,0
 AHRS_TRIM_Y,0
 AHRS_TRIM_Z,0
 AHRS_WIND_MAX,0
 AHRS_YAW_P,0.2
 ALT_CTRL_ALG,0
 ALT_HOLD_FBWCM,0
 ALT_HOLD_RTL,10000
 ALT_MIX,1
 ALT_OFFSET,0
 ARMING_CHECK,1
 ARMING_DIS_RUD,0
 ARMING_REQUIRE,0
 ARSPD_AUTOCL,0
 ARSPD_ENABLE,1
 ARSPD_FBW_MAX,22
 ARSPD_FBW_MIN,9
 ARSPD_OFFSET,2.130668
 ARSPD_PIN,15
 ARSPD_RATIO,1.9936
 ARSPD_SKIP_CAL,0
 ARSPD_TUBE_ORDER,2
 ARSPD_USE,0
 AUTO_FBW_STEER,0
 AUTOTUNE_LEVEL,6
 BATT_AMP_OFFSET,0
 BATT_AMP_PERVOLT,17
 BATT_CAPACITY,3300
 BATT_CURR_PIN,3
 BATT_MONITOR,0
 BATT_VOLT_MULT,10.1

BATT_VOLT_PIN,2
 BATT2_AMP_OFFSET,0
 BATT2_AMP_PERVOL,17
 BATT2_CAPACITY,3300
 BATT2_CURR_PIN,3
 BATT2_MONITOR,0
 BATT2_VOLT_MULT,10.1
 BATT2_VOLT_PIN,2
 BRD_PWM_COUNT,4
 BRD_SAFETYENABLE,1
 BRD_SER1_RTSCTS,2
 BRD_SER2_RTSCTS,2
 CAM_DURATION,10
 CAM_SERVO_OFF,1100
 CAM_SERVO_ON,1300
 CAM_TRIGG_DIST,0
 CAM_TRIGG_TYPE,0
 COMPASS_AUTODEC,1
 COMPASS_DEC,0
 COMPASS_DEV_ID,73225
 COMPASS_DEV_ID2,131594
 COMPASS_DEV_ID3,0
 COMPASS_EXTERN2,0
 COMPASS_EXTERN3,0
 COMPASS_EXTERNAL,1
 COMPASS_LEARN,1
 COMPASS_MOT_X,0
 COMPASS_MOT_Y,0
 COMPASS_MOT_Z,0
 COMPASS_MOT2_X,0
 COMPASS_MOT2_Y,0
 COMPASS_MOT2_Z,0
 COMPASS_MOT3_X,0
 COMPASS_MOT3_Y,0
 COMPASS_MOT3_Z,0
 COMPASS_MOTCT,0
 COMPASS_OFS_X,-0.3281624
 COMPASS_OFS_Y,5.074333
 COMPASS_OFS_Z,-1.829881
 COMPASS_OFS2_X,0.7644874
 COMPASS_OFS2_Y,10.79797
 COMPASS_OFS2_Z,-2.574864
 COMPASS_OFS3_X,0

COMPASS_OFS3_Y,0
 COMPASS_OFS3_Z,0
 COMPASS_ORIENT,0
 COMPASS_ORIENT2,0
 COMPASS_ORIENT3,0
 COMPASS_PRIMARY,0
 COMPASS_USE,1
 COMPASS_USE2,1
 COMPASS_USE3,1
 EKF_ABIAS_PNOISE,0.0002
 EKF_ACC_PNOISE,0.5
 EKF_ALT_NOISE,0.5
 EKF_EAS_GATE,10
 EKF_EAS_NOISE,1.4
 EKF_FALLBACK,1
 EKF_FLOW_DELAY,25
 EKF_FLOW_GATE,3
 EKF_FLOW_NOISE,0.3
 EKF_GBIAS_PNOISE,1E-06
 EKF_GLITCH_ACCEL,150
 EKF_GLITCH_RAD,20
 EKF_GND_GRADIENT,2
 EKF_GPS_TYPE,0
 EKF_GYRO_PNOISE,0.015
 EKF_HGT_GATE,20
 EKF_MAG_CAL,0
 EKF_MAG_GATE,3
 EKF_MAG_NOISE,0.05
 EKF_MAGB_PNOISE,0.0003
 EKF_MAGE_PNOISE,0.0003
 EKF_MAX_FLOW,2.5
 EKF_POS_DELAY,220
 EKF_POS_GATE,30
 EKF_POSNE_NOISE,0.5
 EKF_RNG_GATE,5
 EKF_VEL_DELAY,220
 EKF_VEL_GATE,6
 EKF_VELD_NOISE,0.5
 EKF_VELNE_NOISE,0.3
 EKF_WIND_PNOISE,0.1
 EKF_WIND_PSCALE,0.5
 ELEVON_CH1_REV,0
 ELEVON_CH2_REV,0

ELEVON_MIXING,0
 ELEVON_OUTPUT,0
 ELEVON_REVERSE,0
 FBWA_TDRAG_CHAN,0
 FBWB_CLIMB_RATE,2
 FBWB_ELEV_REV,0
 FENCE_ACTION,0
 FENCE_AUTOENABLE,0
 FENCE_CHANNEL,0
 FENCE_MAXALT,0
 FENCE_MINALT,0
 FENCE_RET_RALLY,0
 FENCE_RETALT,0
 FENCE_TOTAL,0
 FLAP_1_PERCNT,0
 FLAP_1_SPEED,0
 FLAP_2_PERCNT,0
 FLAP_2_SPEED,0
 FLAP_IN_CHANNEL,0
 FLAP_SLEWRATE,75
 FLAPERON_OUTPUT,0
 FLOW_ENABLE,0
 FLOW_FXSCALER,0
 FLOW_FYSCALER,0
 FLTMODE_CH,8
 FLTMODE1,10
 FLTMODE2,2
 FLTMODE3,2
 FLTMODE4,2
 FLTMODE5,2
 FLTMODE6,0
 FORMAT_VERSION,13
 FS_BATT_MAH,0
 FS_BATT_VOLTAGE,0
 FS_GCS_ENABL,0
 FS_LONG_ACTN,0
 FS_LONG_TIMEOUT,20
 FS_SHORT_ACTN,0
 FS_SHORT_TIMEOUT,1.5
 GLIDE_SLOPE_MIN,15
 GND_ABS_PRESS,98518.57
 GND_ALT_OFFSET,0
 GND_TEMP,25

GPS_AUTO_SWITCH,1
 GPS_MIN_DGPS,100
 GPS_MIN_ELEV,-100
 GPS_NAVFILTER,8
 GPS_SBAS_MODE,2
 GPS_TYPE,1
 GPS_TYPE2,0
 GROUND_STEER_ALT,0
 GROUND_STEER_DPS,90
 INS_ACC2OFFS_X,1.483593
 INS_ACC2OFFS_Y,2.251354
 INS_ACC2OFFS_Z,2.305809
 INS_ACC2SCAL_X,1
 INS_ACC2SCAL_Y,1
 INS_ACC2SCAL_Z,1
 INS_ACC3OFFS_X,0
 INS_ACC3OFFS_Y,0
 INS_ACC3OFFS_Z,0
 INS_ACC3SCAL_X,0
 INS_ACC3SCAL_Y,0
 INS_ACC3SCAL_Z,0
 INS_ACCOFFS_X,0.4568798
 INS_ACCOFFS_Y,1.079645
 INS_ACCOFFS_Z,-0.103158
 INS_ACCSCAL_X,1
 INS_ACCSCAL_Y,1
 INS_ACCSCAL_Z,1
 INS_GYR2OFFS_X,0.03132736
 INS_GYR2OFFS_Y,0.04376069
 INS_GYR2OFFS_Z,0.004333549
 INS_GYR3OFFS_X,0
 INS_GYR3OFFS_Y,0
 INS_GYR3OFFS_Z,0
 INS_GYROFFS_X,0.008482047
 INS_GYROFFS_Y,0.02937811
 INS_GYROFFS_Z,0.0008407365
 INS_MPU6K_FILTER,0
 INS_PRODUCT_ID,5
 INVERTEDFLT_CH,0
 KFF_RDDRMIX,0.5
 KFF_THR2PTCH,0
 LAND_FLAP_PERCNT,0
 LAND_FLARE_ALT,3

LAND_FLARE_SEC,2
 LAND_PITCH_CD,0
 LEVEL_ROLL_LIMIT,5
 LIM_PITCH_MAX,2000
 LIM_PITCH_MIN,-2500
 LIM_ROLL_CD,4500
 LOG_BITMASK,65535
 MAG_ENABLE,1
 MIN_GNDSPD_CM,0
 MIS_RESTART,0
 MIS_TOTAL,0
 MIXING_GAIN,0.5
 MNT_ANGMAX_PAN,4500
 MNT_ANGMAX_ROL,4500
 MNT_ANGMAX_TIL,4500
 MNT_ANGMIN_PAN,-4500
 MNT_ANGMIN_ROL,-4500
 MNT_ANGMIN_TIL,-4500
 MNT_CONTROL_X,0
 MNT_CONTROL_Y,0
 MNT_CONTROL_Z,0
 MNT_JSTICK_SPD,0
 MNT_LEAD_PTCH,0
 MNT_LEAD_RLL,0
 MNT_MODE,0
 MNT_NEUTRAL_X,0
 MNT_NEUTRAL_Y,0
 MNT_NEUTRAL_Z,0
 MNT_RC_IN_PAN,0
 MNT_RC_IN_ROLL,0
 MNT_RC_IN_TILT,0
 MNT_RETRACT_X,0
 MNT_RETRACT_Y,0
 MNT_RETRACT_Z,0
 MNT_STAB_PAN,0
 MNT_STAB_ROLL,0
 MNT_STAB_TILT,0
 NAV_CONTROLLER,1
 NAVL1_DAMPING,0.75
 NAVL1_PERIOD,18
 OVERRIDE_CHAN,0
 PTCH2SRV_D,0.2
 PTCH2SRV_I,0.1

PTCH2SRV_IMAX,1500
 PTCH2SRV_P,2.25
 PTCH2SRV_RLL,1
 PTCH2SRV_RMAX_DN,0
 PTCH2SRV_RMAX_UP,0
 PTCH2SRV_TCONST,0.5
 RALLY_LIMIT_KM,5
 RALLY_TOTAL,0
 RC1_DZ,30
 RC1_MAX,1900
 RC1_MIN,1100
 RC1_REV,1
 RC1_TRIM,1500
 RC10_DZ,0
 RC10_FUNCTION,0
 RC10_MAX,1900
 RC10_MIN,1100
 RC10_REV,1
 RC10_TRIM,1500
 RC11_DZ,0
 RC11_FUNCTION,0
 RC11_MAX,1900
 RC11_MIN,1100
 RC11_REV,1
 RC11_TRIM,1500
 RC12_DZ,0
 RC12_FUNCTION,0
 RC12_MAX,1900
 RC12_MIN,1100
 RC12_REV,1
 RC12_TRIM,1500
 RC13_DZ,0
 RC13_FUNCTION,0
 RC13_MAX,1900
 RC13_MIN,1100
 RC13_REV,1
 RC13_TRIM,1500
 RC14_DZ,0
 RC14_FUNCTION,0
 RC14_MAX,1900
 RC14_MIN,1100
 RC14_REV,1
 RC14_TRIM,1500

RC2_DZ,30
 RC2_MAX,1900
 RC2_MIN,1100
 RC2_REV,1
 RC2_TRIM,1500
 RC3_DZ,30
 RC3_MAX,1900
 RC3_MIN,1100
 RC3_REV,1
 RC3_TRIM,1500
 RC4_DZ,30
 RC4_MAX,1900
 RC4_MIN,1100
 RC4_REV,1
 RC4_TRIM,1500
 RC5_DZ,0
 RC5_FUNCTION,0
 RC5_MAX,1900
 RC5_MIN,1100
 RC5_REV,1
 RC5_TRIM,1500
 RC6_DZ,0
 RC6_FUNCTION,0
 RC6_MAX,1900
 RC6_MIN,1100
 RC6_REV,1
 RC6_TRIM,1500
 RC7_DZ,0
 RC7_FUNCTION,1
 RC7_MAX,1900
 RC7_MIN,1100
 RC7_REV,1
 RC7_TRIM,1500
 RC8_DZ,0
 RC8_FUNCTION,0
 RC8_MAX,1900
 RC8_MIN,1100
 RC8_REV,1
 RC8_TRIM,1500
 RC9_DZ,0
 RC9_FUNCTION,0
 RC9_MAX,1900
 RC9_MIN,1100

RC9_REV,1	SCALING_SPEED,15
RC9_TRIM,1500	SCHED_DEBUG,0
RCMAP_PITCH,2	SERIAL0_BAUD,115
RCMAP_ROLL,1	SERIAL1_BAUD,57
RCMAP_THROTTLE,3	SERIAL2_BAUD,57
RCMAP_YAW,4	SERIAL2_PROTOCOL,1
RELAY_DEFAULT,0	SKIP_GYRO_CAL,0
RELAY_PIN,54	SR0_EXT_STAT,2
RELAY_PIN2,55	SR0_EXTRA1,10
RELAY_PIN3,-1	SR0_EXTRA2,10
RELAY_PIN4,-1	SR0_EXTRA3,2
RLL2SRV_D,0.07	SR0_PARAMS,10
RLL2SRV_I,0.2	SR0_POSITION,3
RLL2SRV_IMAX,1500	SR0_RAW_CTRL,1
RLL2SRV_P,2.5	SR0_RAW_SENS,2
RLL2SRV_RMAX,0	SR0_RC_CHAN,2
RLL2SRV_TCONST,0.5	SR1_EXT_STAT,1
RNGFND_FUNCTION,0	SR1_EXTRA1,1
RNGFND_LANDING,0	SR1_EXTRA2,1
RNGFND_MAX_CM,700	SR1_EXTRA3,1
RNGFND_MIN_CM,20	SR1_PARAMS,10
RNGFND_OFFSET,0	SR1_POSITION,1
RNGFND_PIN,-1	SR1_RAW_CTRL,1
RNGFND_RMETRIC,1	SR1_RAW_SENS,1
RNGFND_SCALING,3	SR1_RC_CHAN,1
RNGFND_SETTLE,0	SR2_EXT_STAT,1
RNGFND_STOP_PIN,-1	SR2_EXTRA1,1
RNGFND_TYPE,0	SR2_EXTRA2,1
RNGFND2_FUNCTION,0	SR2_EXTRA3,1
RNGFND2_MAX_CM,700	SR2_PARAMS,10
RNGFND2_MIN_CM,20	SR2_POSITION,1
RNGFND2_OFFSET,0	SR2_RAW_CTRL,1
RNGFND2_PIN,-1	SR2_RAW_SENS,1
RNGFND2_RMETRIC,1	SR2_RC_CHAN,1
RNGFND2_SCALING,3	STAB_PITCH_DOWN,2
RNGFND2_SETTLE,0	STALL_PREVENTION,1
RNGFND2_STOP_PIN,-1	STEER2SRV_D,0.005
RNGFND2_TYPE,0	STEER2SRV_I,0.2
RSSI_PIN,-1	STEER2SRV_IMAX,1500
RSSI_RANGE,5	STEER2SRV_MINSPD,1
RST_MISSION_CH,0	STEER2SRV_P,1.8
RST_SWITCH_CH,0	STEER2SRV_TCONST,0.75
RTL_AUTOLAND,0	STICK_MIXING,1

SYS_NUM_RESETS,3
 SYSID_MYGCS,255
 SYSID_SW_TYPE,0
 SYSID_THISMAV,1
 TECS_CLMB_MAX,5
 TECS_HGT_OMEGA,3
 TECS_INTEG_GAIN,0.1
 TECS_LAND_ARSPD,-1
 TECS_LAND_DAMP,0.5
 TECS_LAND_SINK,0.25
 TECS_LAND_SPDWGT,1
 TECS_LAND_TCONST,2
 TECS_LAND_THR,-1
 TECS_PITCH_MAX,0
 TECS_PITCH_MIN,0
 TECS_PTCH_DAMP,0
 TECS_RLL2THR,10
 TECS_SINK_MAX,5
 TECS_SINK_MIN,2
 TECS_SPD_OMEGA,2
 TECS_SPDWEIGHT,1
 TECS_THR_DAMP,0.5
 TECS_TIME_CONST,5
 TECS_VERT_ACC,7
 TELEM_DELAY,0
 TERRAIN_ENABLE,1
 TERRAIN_FOLLOW,0
 TERRAIN_LOOKAHD,2000
 TERRAIN_SPACING,100
 THR_FAILSAFE,1

THR_FS_VALUE,950
 THR_MAX,75
 THR_MIN,0
 THR_PASS_STAB,0
 THR_SLEWRATE,100
 THR_SUPP_MAN,0
 THROTTLE_NUDGE,1
 TKOFF_FLAP_PCNT,0
 TKOFF_ROTATE_SPD,0
 TKOFF_TDRAG_ELEV,0
 TKOFF_TDRAG_SPD1,0
 TKOFF_THR_DELAY,2
 TKOFF_THR_MAX,0
 TKOFF_THR_MINACC,0
 TKOFF_THR_MINSPD,0
 TKOFF_THR_SLEW,0
 TRIM_ARSPD_CM,1200
 TRIM_AUTO,0
 TRIM_PITCH_CD,0
 TRIM_THROTTLE,45
 VTAIL_OUTPUT,0
 WP_LOITER_RAD,60
 WP_MAX_RADIUS,0
 WP_RADIUS,90
 YAW2SRV_DAMP,0.1
 YAW2SRV_IMAX,1500
 YAW2SRV_INT,0
 YAW2SRV_RLL,0.25
 YAW2SRV_SLIP,0

REPORT DOCUMENTATION PAGE					Form Approved OMB No. 0704-0188	
The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.						
1. REPORT DATE (DD-MM-YYYY) 24-12-2015		2. REPORT TYPE Master's Thesis			3. DATES COVERED (From — To) Sept 2014 — Dec 2015	
4. TITLE AND SUBTITLE Design and Implementation of a Unified Command and Control Architecture for Multiple Cooperative Unmanned Vehicles Utilizing Commercial Off the Shelf Components					5a. CONTRACT NUMBER 5b. GRANT NUMBER 5c. PROGRAM ELEMENT NUMBER 5d. PROJECT NUMBER 5e. TASK NUMBER 5f. WORK UNIT NUMBER	
6. AUTHOR(S) Jeremy Gray, Civilian, Ctr					8. PERFORMING ORGANIZATION REPORT NUMBER AFIT-ENV-MS-15-D-048	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 Hobson Way Wright-Patterson AFB OH 45433-7765					10. SPONSOR/MONITOR'S ACRONYM(S) AFIT/ENV	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Department of Systems Engineering and Management (AFIT) 2950 Hobson Way Wright-Patterson AFB OH 45433-7765 Dr. David Jacques COMM 937-255-3636 x3329 Email: david.jacques@afit.edu					11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION / AVAILABILITY STATEMENT DISTRIBUTION STATEMENT A: APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.						
13. SUPPLEMENTARY NOTES This work is declared a work of the U.S. Government and is not subject to copyright protection in the United States						
14. ABSTRACT Small unmanned systems provide great utility to military applications due to their portable and expendable design. These systems are, however, costly to develop, produce, and maintain, making it desirable to integrate available commercial off the shelf (COTS) components. This research investigates the integration of COTS components through the development of a modular unified command and control (C2) architecture for heterogeneous and homogeneous vehicle teams to accomplish formation flocking and communication relay scenarios. A vehicle agnostic architecture was developed to be applied across different vehicle platforms, different vehicle combination, and different cooperative missions. COTS components consisting primarily of open source hardware and software were integrated and tested based on the positional accuracy, precision, and other qualitative measures. The resulting system successfully demonstrated formation flocking in three of four vehicle combinations, with the forth still demonstrating leader follower behaviors. The system achieved at best a mean positional error of 0.99m, a standard deviation of 0.44m, and a DRMS of 0.59m....						
15. SUBJECT TERMS Cooperative, Command and Control, Multi-vehicle, Multi-agent, Architecture, Unmanned, UAS, UGS, COTS, OSH, OSS, Formation Flight, Flocking, Communication Relay						
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT		18. NUMBER OF PAGES	
a. REPORT	b. ABSTRACT	c. THIS PAGE	U		177	
U	U	U	19a. NAME OF RESPONSIBLE PERSON Dr. David Jacques, AFIT/ENV			
						19b. TELEPHONE NUMBER (include area code) (937) 255-3636, x3329; david.jacques@afit.edu