



**EFFECTIVENESS OF INTER-VEHICLE COMMUNICATIONS AND
ON-BOARD PROCESSING FOR CLOSE UNMANNED AUTONOMOUS
VEHICLE FLIGHT FORMATIONS**

THESIS

Lidia Toscano, Civilian, USAF

AFIT-ENV-MS-17-M-230

**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 thesis are those of the author and do not reflect the official policy or position of the United States Air Force, 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.

**EFFECTIVENESS OF INTER-VEHICLE COMMUNICATIONS AND
ON-BOARD PROCESSING FOR CLOSE UNMANNED AUTONOMOUS
VEHICLE FLIGHT FORMATIONS**

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

Lidia Toscano, BS

Civilian, USAF

March 2017

DISTRIBUTION STATEMENT A.
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AFIT-ENV-MS-17-M-230

**EFFECTIVENESS OF INTER-VEHICLE COMMUNICATIONS AND
ON-BOARD PROCESSING FOR CLOSE UNMANNED AUTONOMOUS
VEHICLE FLIGHT FORMATIONS**

Lidia Toscano, BS

Civilian, USAF

Committee Membership:

Dr. John M. Colombi
Chair

Dr. David R. Jacques
Member

LtCol Amy Cox, PhD
Member

Abstract

The small size, full-featured autopilots, and on-board processing of UAVs make them highly versatile. They can be used to search in inaccessible areas, conduct Battle Damage Assessment (BDA), provide battle commanders with real-time intelligence, and, most importantly, protect our military force where loss of life is imminent.

The goal of this research was to integrate two or more small Unmanned Airborne Vehicles (UAVs) to successfully employ cooperative communication using three commercial-off-the-shelf (COTS) components: a BeagleBone Black on-board microprocessor, the Pixhawk autopilot, and Open Source Software (OSS) DroneKit-Python. These components are readily available and have a low-cost factor. To baseline functionality, this project prototyped and tested two Unmanned Ground Vehicles (UGVs). Successful integration of hardware and software components was demonstrated resulting in cooperative communication of two UGVs in formation. Experimentation found that the best distance root-mean-squared error was 4.0 m that occurred at an update rate 20Hz. Confidence level calculations indicated a follower UGV can safely navigate at a 2.5 m distance in close formation. A variety of qualitative observations were captured indicating the maturity of this cooperative architecture and readiness to implement across multiple UAVs in military applications.

Acknowledgments

We encounter many types of people along the highway of life and some create a deep impression leaving traces of their greatness imprinted in the lives of those whose lives they touch. Dr. John Colombi has been this kind of individual throughout my entire experience here at AFIT. His dedication and earnest endeavor to help me achieve excellence has been deeply appreciated, especially at times when I thought I had taken one step forward and two steps back; he always had a sustaining word of encouragement and a cheerful disposition. Thank you, Dr. Colombi, for your continued support, it will always be remembered.

I want thank my good friend, Dr. Rhonda Seamons, for all of her words of encouragement and support even before this entire journey began. She should graduate right along with me as she read and edited many of the papers written in this coursework. Thank you, Rhonda; I couldn't have done it without you. I could not ask for a finer friend.

Many thanks to my sister, Gabrielle, for always supporting me and being part of this experience. She drove many long hours to come and just be with me. Gaby, you're a ray of sunshine in my life and always the silver lining after a storm. God bless you always.

Finally, to my dear mom and dad, who sacrificed my presence that I may accomplish this objective. Mom and Dad, my love for you is immense, and I will forever be grateful to my Heavenly Father for allowing me to be your daughter.

Lidia Toscano, Civilian

Table of Contents

	Page
Abstract	iv
Table of Contents	vi
List of Figures	ix
List of Tables	xi
I. Introduction	1
General Issue	1
Research Objective	4
Research Focus	6
Investigative Questions	6
Approach & Methodology	6
Assumptions/Limitations	7
Implications	8
Preview	8
II. Literature Review	9
Overview	9
Cooperative Control	9
USAF and DoD Roadmaps for Small UAVs	10
Application of UAV On-board Processing Algorithms	13
Components of On-board Processing	17
Summary	21

III. Methodology	23
Overview	23
Hardware Architecture	23
Hardware	27
Software.....	31
Software Architecture.....	31
OSS Utilities.....	34
Software Architecture.....	35
Cooperative Communication.....	38
Test Plans.....	43
Summary.....	45
IV. Analysis and Results.....	47
Overview	47
Test Plan 1	47
Test Plan 2	48
Comparison with previous UGV research.....	70
Integration Anomalies	74
Summary.....	76
V. Conclusions and Recommendations	77
Overview	77
Conclusions of Research	77
Investigative Questions Answered	77
Study Limitations	80

Recommendations for Future Research.....	80
Significance of Research	82
Appendix A: Manufactured Cables	83
Appendix B: Use Case Specification.....	85
Appendix C: Leader-Follower.py	90
Appendix D: IP Configuration Table.....	96
Appendix E: Basic CodeVerify.py	97
References.....	101
References for Figures	107

List of Figures

	Page
Figure 1: Cooperative Communication Proposed Configuration	5
Figure 2: Worldwide DOD UAS Operations and Training	11
Figure 3: Drone Mosquito.....	12
Figure 4: DOD Unmanned Aircraft Systems.....	13
Figure 5: Autonomy Levels For Unmanned Systems (ALFUS)	15
Figure 6: USAF Fighter Aircraft Formations	16
Figure 7: NATO Heavy Aircraft Formations.....	17
Figure 8: BeagleBone Black	18
Figure 9: Pixhawk	20
Figure 10: Network Configuration Architecture.....	24
Figure 11: Leader Architecture Design.....	26
Figure 12: Follower Architecture Design	26
Figure 13: GCS Architecture Design.....	27
Figure 14: Minimal Data Processing Architecture Configuration.....	28
Figure 15: BBB Processor addition to Architecture Configuration.....	30
Figure 16: USR-TCPS232-24.....	30
Figure 17: Wave Relay	31
Figure 18: SV-1 Systems Interface Description	32
Figure 19: OV-6c Operational Event Trace Description	37
Figure 20: OV-1 Cooperative Communication from one UGV to another UGV.....	38
Figure 21: Mission Plan Box Pattern.....	39

Figure 22: Technical Performance Measures	42
Figure 23: Leader and Follower UGVs	46
Figure 24: Case 1 – Positive TI 5%	52
Figure 25: Case 2 - Negative TI 5%	53
Figure 26: Test Plan 2 Test 1	55
Figure 27: Test Plan 2 Test 1 DRMS Graphs	56
Figure 28: Test Plan 2 Test 2 part a	59
Figure 29: Test Plan 2 Test 2 b	60
Figure 30: Test Plan 2 Test 2 DRMS Graphs part a	61
Figure 31: Test Plan 2 Test 2 DRMS Graphs part b	62
Figure 32: Test Plan 2 Test 6 part a	64
Figure 33: Test Plan 2 Test 6 part b	65
Figure 34: Test Plan 2 Test 6 part c	66
Figure 35: Test Plan 2 Test 6 DRMS Graphs part a	67
Figure 36: Test Plan 2 Test 6 DRMS Graphs part b	68
Figure 37: Test Plan 2 Test 6 DRMS Graphs part c	69
Figure 38: Frequency DRMS Comparison	70
Figure 39: UGV Following Multi-Rotor UAS.....	71
Figure 40: Test Plan 2 Test 6 Comparison.....	73
Figure 41: Test Plan 2 Test 6 Lap 2 DRMS.....	79
Figure 42: Test Plan 2 Test 6 FULL Run DRMS	79
Figure 43: Pixhawk to USR-TCPS232-24 Manufactured Cable	83
Figure 44: BBB to PixHawk Manufactured Cable	84

List of Tables

	Page
Table 1: Processor Comparison	19
Table 2: Autopilot Comparison	21
Table 3: Software Tools on the BBB Micro SD Card	33
Table 4: Test Plan 1 - Code Verification	44
Table 5: Test Plan 2 Cooperative Communication	45
Table 6: Test Plan 2 Test 1 Results Summary	57
Table 7: Test Plan 2 Test 2 Results Summary	63
Table 8: Test Plan 2 Test 6 Results Summary	69
Table 9: Gray - Toscano Comparison	74
Table 10: IP Configuration	96

EFFECTIVENESS OF INTER-VEHICLE COMMUNICATIONS AND ON-BOARD PROCESSING FOR CLOSE UNMANNED AUTONOMOUS VEHICLE FLIGHT FORMATIONS

I. Introduction

General Issue

Since World War II the military has invested in research and development of vehicles that can be operated by remote control or Artificial Intelligence (AI), taking the pilot out of potentially dangerous situations. These vehicles are also known as Unmanned Aircraft Systems (UAS), Unmanned Airborne Vehicles (UAV), Unmanned Ground Vehicles (UGVs), and Drones, amongst other names. Industry has also recognized the importance these vehicles have for military and commercial implementation and has done a great deal to advance the technology. Educational institutions have also come to understand the significance of this major work and have committed time, money, and intellectual resources into UAS development. Recent technology improvement and component miniaturization has increased the technical capabilities of small UAS.

Cooperative communication in UAVs is the ability to send information between multiple UAVs in the air working together for a unified purpose or mission. The information shared may be a variety of data, such as navigation parameters; location information; or Intelligence, Surveillance, and Reconnaissance (ISR). The capability to cooperatively navigate presents an opportunity to increase autonomy in UAVs. Autonomy in UAVs enables them to process data *on-board* while they are airborne in order to communicate with one another without the need to relay information through a

ground station. It also affords the opportunity to increase the level of autonomy in small UAVs.

In a brief on autonomy initiatives in the DoD, the Autonomy Priority Steering Council (2012) defined *autonomy* as “having the capability and freedom to self-direct to achieve mission objectives. An autonomous system makes choices and has the human’s proxy for those decisions” (p. 5). The ability for several UAVs to work together in an autonomous manner using cooperative communication has a myriad of military applications, including “enemy or target detection, confusion effect, resource balancing, duty relief, and centralized information” (Kim & Seo, 2012, p. 870).

The demand for UAVs has exploded since they were first introduced; however, there is a lack of understanding and knowledge regarding:

- Which open source components can be integrated to achieve airborne cooperative control for military applications,
- How to integrate commercial off-the-shelf (COTS) on-board processing for cooperative control,
- Performance improvement for airborne on-board processing versus ground-based processing, and
- Which new applications are possible for low-cost airborne autonomy.

The Unmanned Systems Integrated Roadmap FY2011-2036 has a vision which encourages the use of COTS to avoid stove-piped and proprietary solutions which are expensive; it suggests that inexpensive open-source COTS components can foster creativity and innovation (Winnefeld & Kendal, 2011). This *Roadmap*’s executive summary identifies autonomy as the second highest challenge facing the Department of

Defense (DoD) with regards to unmanned systems and confirms that today's technology requires a high level of human interaction.

The United States Air Force Unmanned Aircraft Systems Flight Plan 2009-2047 mentions a plan to develop a UAS pilot career in order to relieve the UAS manpower burden. Currently, it takes more than a single pilot to conduct a UAS mission; at a minimum, one person flies the aircraft, a second person monitors the ground station, and a third person retrieves the aircraft (Donley & Schwartz, 2009). The goal is to have a single pilot control one, two, or even a swarm of autonomous UAS vehicles. One solution is developing the capability for cooperative communication to control multiple UAVs and reduce the number of operators required to execute UAS missions. Use of open-source COTS components to realize this capability presents a low-cost and flexible solution.

For approximately the past ten years, the Air Force Institute of Technology (AFIT) has been investigating cooperative communication from one UAV to another UAV in the air using COTS applications. Many researchers have dedicated themselves to exploring new methods and technology to develop UAVs that are lighter and more efficient, carry heavier payloads, and fly for longer periods of time. The next logical step is to orchestrate multiple vehicles that work together in unison by using cooperative communication as a group.

The Institute of Electrical and Electronics Engineers (IEEE) researchers in a 2011 GLOBECOM Workshop noted, "Teams of unmanned systems must be able to operate in dynamic environments where the state of tasks is always changing" (Kopeikin, Ponda, Johnson, Toupet, & How, 2011, p. 1303).

This project is a follow-on to work performed by Jeremy Gray, a recent AFIT graduate. In his thesis, entitled “Design and Implementation of a Unified Command and Control Architecture for Multiple Cooperative Unmanned Vehicles Utilizing Commercial Off The Shelf Components,” Gray, (2015) noted that “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” (p. 9).

Research Objective

The ultimate purpose of this research was to enable two UAVs to use cooperative communication directly from one UAV to the other UAV, using the ground station only as a monitoring link, but not as the main relay for communication. Another purpose was to streamline the equipment required and thus reduce communication latency and improve responsiveness; responsiveness is important in dynamic environments.

Figure 1 displays in black lines the current configuration which requires one Ground Control Station (GCS) per UAV deployed. The proposed configuration is shown in red lines and consists of one GCS to control at least two UAVs in flight formation.

As a first step, the research was able to prototype dynamic cooperative communication with Unmanned Ground Vehicles (UGVs) using a COTS BeagleBone Black (BBB) microprocessor. A follower UGV monitored the location and state of a leader UGV on the ground and manipulated its state in order to navigate towards the leader, thus functioning at a higher level of autonomy.

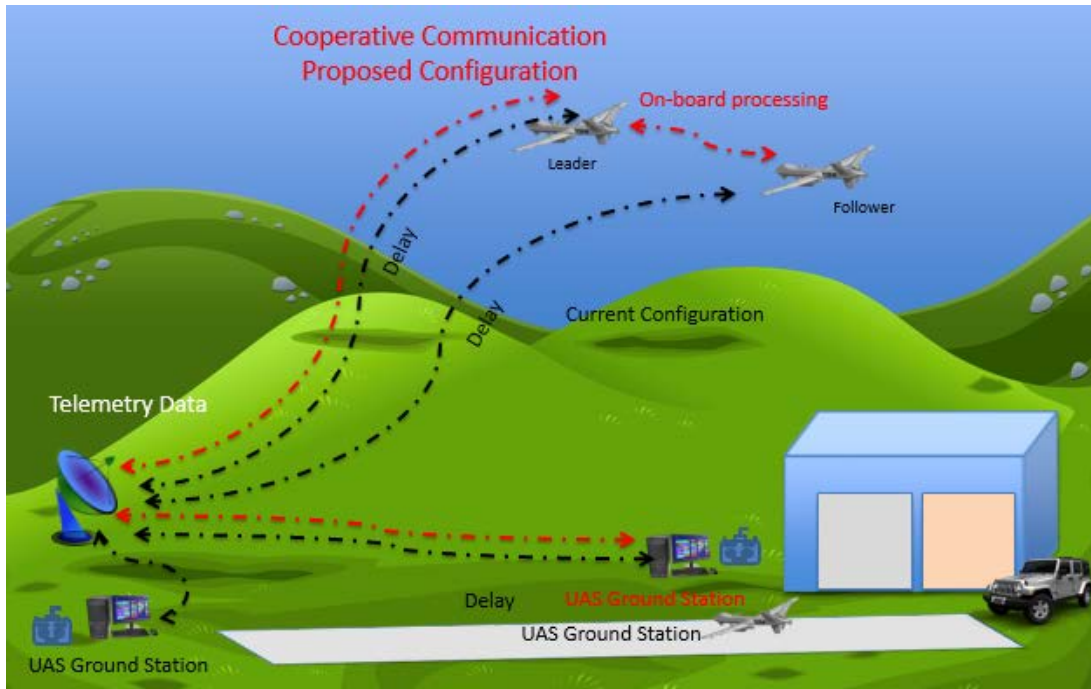


Figure 1: Cooperative Communication Proposed Configuration

In a report by the Civil UAV Capability Assessment prepared by NASA, Cox, Nagy, Skoog, Somers, and Warner (2004) indicate that “less direct human interaction in flying the UAV allows less on-station personnel, less on-station support infrastructure, and one operator to monitor several vehicles at a given time” (p. 5). Developing UAVs with a higher level of autonomy leverages the diminishing personnel resources and allows them agility in completing mission requirements. In the book *Cooperative Control and Optimization*, Passino et al. (2002) point out that while individual UAVs could perform missions independently, “the overall mission effectiveness can be improved via communications and coordination of their activities” (p. 235).

Research Focus

This research focuses on achieving Cooperative Communication with unmanned systems. As a first step UGVs were selected to facilitate rapid experimentation. Low cost, open source COTS solutions were intentionally chosen. The BBB w specifically chosen because its compact size, low cost, and open source architecture contribute to its effectiveness as a development tool. In addition, it can easily be accessed via several connectivity ports for development using Python, which is the Operational Support System (OSS) format for this project. Python has more than 96,000 packaged modules available, giving it an advantage over other languages (“PyPI - the Python Package Index,” n.d.).

Investigative Questions

The research of this thesis revolves around the integration of a hardware/software prototype that will provide cooperative communication between two UGVs. Several questions are explored.

- What is an architecture that supports on board processors and inter-vehicle communication?
- What is the Distance Root Mean Square (DRMS) between the follower’s navigation path and the commanded waypoint (WP)?
- What is the expected safe distance between UGVs using this architecture?

Approach & Methodology

This research consisted of two phases. The first phase focused on hardware and software architecture. Communication between the BBB and the Pixhawk was

established by connecting the BBB through a USB-TCP232-24 serial to an Ethernet module through a Wave Relay module to the Pixhawk autopilot. Once the connection was established and the BBB was communicating with the Pixhawk, the Python software algorithms were developed to execute the leader/follower formation. The second phase consisted of test and characterization using UGVs to ensure the hardware and software were executing Python algorithm commands correctly.

Assumptions/Limitations

This research project depended on the following assumptions:

1. BBB would be able to:
 - a. Communicate with at least two autopilots through a Wave Relay network.
 - b. Perform using the new DroneKit platform software.
 - c. Use an Ethernet network to communicate with a wireless Wave Relay radio.
2. DroneKit platform would have the ability to connect to multiple vehicles as communication algorithms were developed so that the BBB would direct the Leader and the Follower to adjust for distance behind the leader (Off_r), theta angle (Off_{θ}), and anticipated WPs (Off_{I1}).
3. Research would be limited to COTS Autopilots and BBB computer processors.
4. Mission Planner (MP) would be the main Ground Control Station (GCS) software Graphical User Interface (GUI).

5. Code previously developed to perform calculations would work as expected to calculate the follower's position.

Implications

The successful implementation of compact, open-source, inexpensive, COTS BBB computers for direct communication between two UAVs can enable other military applications. In the configuration of two UAVs, one acts as the leader, guiding and directing, while the other acts as the follower, receiving directions and anticipating the flight path of the leader. This opens the door to having a leader communicating with *multiple* followers.

Potentially, countless military missions can be executed using UAVs with the ability of cooperative communication. Cooperative communication facilitates UAV autonomy; UAV autonomy, in turn, leads to more efficient use of our limited manpower.

Preview

Chapter II provides a literature review of various journals and articles which support the purpose for this research. Topics addressed include the DOD's road map and vision for UAVs, their relevance to military applications, on-board processing, and components of on-board processing. Chapter III provides a description of hardware, software, cooperative communication architecture design, and test plans. Chapter IV discusses the equations and methods of in-depth analysis for the laboratory and field results. Chapter V provides conclusions and recommendations for future research.

II. Literature Review

Overview

Chapter II is a synopsis of a variety of engineering articles that explain the critical nature of UAVs both in military and commercial applications. These articles help provide the background and general technical knowledge for this thesis.

Cooperative Control

This research focuses on cooperative control of at least two UAVs flying in formation. The complexity of integrating cooperative control between multiple UAVs has been studied for several decades, and the flocking of birds has been used as a metaphor. In an IEEE journal article, Seong-Woo Kim and Seung-Woo Seo explain that, while cooperative communication of multiple autonomous vehicles has received a lot of attention, it has been slow in the making. They explain that the increase in complexity to the algorithms and the design of multiple cooperative UAVs is not trivial as compared to single UAVs (Kim & Seo, 2012, p. 871).

One of the goals of this project is to exchange telemetry data between UAVs effectively. Other researchers are working towards this same objective. In the handbook of Unmanned Aerial Vehicles, we find that implementation of COTS for cooperative tasks such as data routing and implementation of modularity is being explored in order to determine how to exchange data between UAVs effectively. The authors also indicate that “integration of intra- and inter-vehicle communication plays a significant, albeit subtle, role in the performance of unmanned aircraft system” (Elston, Stachura, Dixon, Argrow, & Frew, 2015, p. 793).

USAF and DoD Roadmaps for Small UAVs

The USAF and the DoD realize the important role UAVs play in military applications. The possible military applications are far-reaching: ISR in hazardous areas without exposing soldiers to lethal dangers (“RQ-11 Raven,” n.d.; “RQ-11B Raven,” n.d.), target identification in an unobstructed way (Linci & Mohanraj, 2015), and search and rescue in combat-zones or following natural disasters (Haddal & Gertler, 2010). The military’s need for UAVs has “experienced explosive growth” (Donley & Schwartz, 2009, p. 15). According to Col Spinneta, “at the start of the second Gulf War in 2003, the U.S. Air Force flew only one unmanned combat air patrol per day in support of the conflict” (2011, p. 30). Contrast that with a 2011 report prepared by the UAS Task Force Airspace Integration Integrated Product Team which shows there were over 500,000 flight hours by DoD UAS missions during 2010 (*see* Figure 2). Since 500,000 flight hours/year equates to 1,378 flight hours each day, the exponential increase in sorties performed each day is clear.

This explosive growth is not just in the military. The commercial market is also booming and is predicted to continue to increase at a fast pace. In 2014, the Teal Group, a professional research company, predicted that UAV spending would increase worldwide from \$6.4 billion annually to \$11.5 billion annually, totaling an unprecedented \$91 billion by 2024.

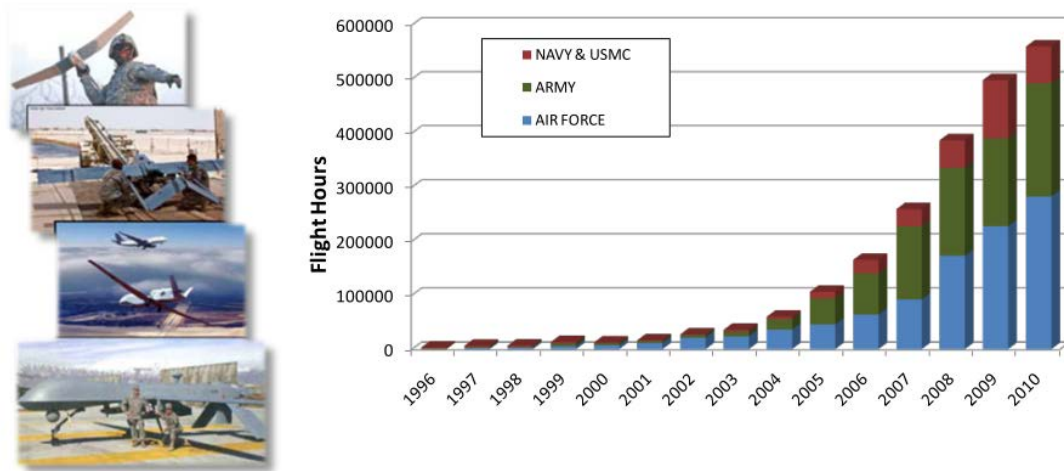


Figure 2: Worldwide DOD UAS Operations and Training

Finnegan, the director of corporate analysis for the Teal Group, noted that their “2014 UAV study calculates the UAV market at 89% military, 11% civil cumulative for the decade, with numbers shifting to 86% military and 14% civil by the end of the 10-year forecast” (as cited in “Teal Group,” 2014, n.p.).

The possibilities for *commercial* UAVs are immense: search and rescue (“Arizona,” n.d.), forest fire response (Gabbert, 2013), environmental monitoring, response to natural crises like earthquakes and hurricanes, and response to terrorist attacks (“Police,” 2012), to name a few. They may also prove very lucrative for innovators like Amazon who announced in 2013 a future “PrimeAir” to deliver packages to customers in 30 minutes or less using UAVs (“Amazon: Prime Air,” 2013.). Commercial interest has sparked creative development and improvement of UAV technology. There are a number of COTS applications available on the market that are compact and inexpensive, and that use OSS applications. OSS applications have the

advantage of a large user community that constantly contributes to their improvement and robustness.

However, since the 1950s, the US military continues to be the leader when it comes to UAV operation and development. The USAF has a powerful vision for Unmanned Aircraft Systems (UAS), and the DoD has created a clear “road map” for unmanned systems taking us to the year 2036. Because of this, it is anticipated that the military will continue to be the biggest market for UAS.

Over the past 60 years, UAVs have become lighter, they have increased in processing power, and they display greater functionality. Payload systems within the UAVs are more complex providing a variety of specialized capabilities ideal for military applications. Due to the usefulness and applicability of UAVs, the USAF and the DoD have written specific and detailed roadmaps for their future utilization. The DoD road map identifies five UAV groups based on size, weight, and capabilities. Group 1, for



Figure 3: Drone Mosquito

instance, would include lightweight nanotechnology through vehicles weighing 20 pounds. Such drones would be capable of surveillance and sensing (*see* Figure 3). Groups 4 and 5 include all unmanned vehicles over 1320 pounds. Such drones are capable of carrying heavier payloads

and could participate in recovery of personnel or objects (*see* Figure 4 for a complete list of requirements for all five groups).

The research for this project applies to Groups 1 and 2 as described in the USAF and DOD road maps. It builds on previous work done by AFIT graduates using small UAVs.

DoD Unmanned Aircraft Systems (As of 1 JULY 2011)					
General Groupings	Depiction	Name	(Vehicles/GCS)	Capability/Mission	Command Level
Group 5 • > 1320 lbs • > FL180		•USAF/USN RQ-4A Global Hawk/BAMS-D Block 10 •USAF RQ-4B Global Hawk Block 20/30 •USAF RQ-4B Global Hawk Block 40	•9/3 •20/6 •5/2	•ISR/MDA (USN) •ISR •ISR/BMC	•JFACC/AOC-Theater •JFACC/AOC-Theater •JFACC/AOC-Theater
		•USAF MQ-9 Reaper	•73/85* *MQ-1/MQ-9 same GCS	•ISR/RSTA/EW/ STRIKE/FP	•JFACC/AOC- Support Corps, Div, Brig, SOF
Group 4 • > 1320 lbs • < FL180		•USAF MQ-1B Predator	•165/85*	•ISR/RSTA/STRIKE/FP	•JFACC/AOC-Support Corps, Div, Brig
		•USA MQ-1C Warrior/MQ-1C Gray Eagle	•31/11	•(MQ-1C Only-C3/LG)	•NA
		•USN UCAS- CVN Demo	•2/0	•Demonstration Only	•NA
		•USN MQ-8B Fire Scout VTUAV	•14/8	•ISR/RSTA/ASW/ ASUW/MIW/OMCM/ EOD/FP	•Fleet/Ship
Group 3 • < 1320 lbs • < FL180 • < 250 knots		•USA MQ-5 Hunter	•45/21	•ISR/RSTA/BDA	•Corps, Div, Brig
		•USA/USMC/SOCOM RQ-7 Shadow	•368/265	•ISR/RSTA/BDA	•Brigade Combat Team
		•USN/USMC STUAS	•0/0	•Demonstration	•Small Unit
Group 2 • 21-55 lbs • < 3500 AGL • < 250 knots		•USN/SOCOM/USMC RQ-21A ScanEagle	•122/13	•ISR/RSTA/FORCE PROT	•Small Unit/Ship
Group 1 • 0-20 lbs • < 1200 AGL • < 100 knots		•USA / USN / USMC / SOCOM RQ-11 Raven	•5628/3752	•ISR/RSTA	•Small Unit
		•USMC/ SOCOM Wasp	•540/270	•ISR/RSTA	•Small Unit
		•SOCOM SUAS AECV Puma	•372/124	•ISR/RSTA	•Small Unit
		•USA gMAV / USN T-Hawk	•270/135	•ISR/RSTA/EOD	•Small Unit

Figure 4: DOD Unmanned Aircraft Systems

Application of UAV On-board Processing Algorithms

An autopilot like the Pixhawk or the ArduPilot Mega (APM) is used to control the UAV as it flies through a series of WPs on a mission plan. While autopilots do some basic data processing on-board, the data processing consists of flying the UAV through a pre-programmed flight algorithm defined on the ground station (“Choosing a ground station,” n.d.). To build more functionality into a UAV on the ground, designers need to

develop and implement a processor which provides the ability to run software algorithms *on-board* thus reducing delays to data processing. The on-board processor would provide the capability to enable the UAV to perform autonomously tasks such as sensing, analyzing, surveying, monitoring, and photographing in order to execute specialized tasks assigned by its operator.

The National Institute of Standards and Technology has identified five different levels of autonomy in Human Robot Interaction (HRI) which apply to UAVs. These five levels range from the most basic to the most complex in terms of human independence, mission complexity, and environmental difficulty (Huang, Messina, & Albus, 2007). For instance, the first level of autonomy consists of a basic UAV controlled completely by a human operator in the simplest of environments. The highest level of autonomy would have an UAV capable of performing complex missions in an extreme environment with almost zero human interaction (*see* Figure 5 for a listing of all levels). This project intends to raise the level of UAV autonomy from a level one to a level two, perhaps even a level three.

In some small UAV settings, for instance here at AFIT, it takes a minimum of three individuals to maneuver a single drone: a pilot, the person monitoring the GCS, and the observer. However, some UAVs require far more. Stracker (2007) explains the requirements for two different UAVs used by the Navy:

According to the Pioneer Navy Training System Plan (NTSP), there are a total of eight watch-station requirements for the Pioneer UAS. They consist of the External Pilot (EP), Internal Pilot (IP), Payload Operator (PO), Mission Commander (MC), an additional EP and IP for the PCS, and two requirements for

the Plane Captain and Crew Chief. The minimum crew required to operate the RQ-2 is three, to include the MC, IP and PO; an EP is also required but typically for takeoff and landing only. (p. 18)

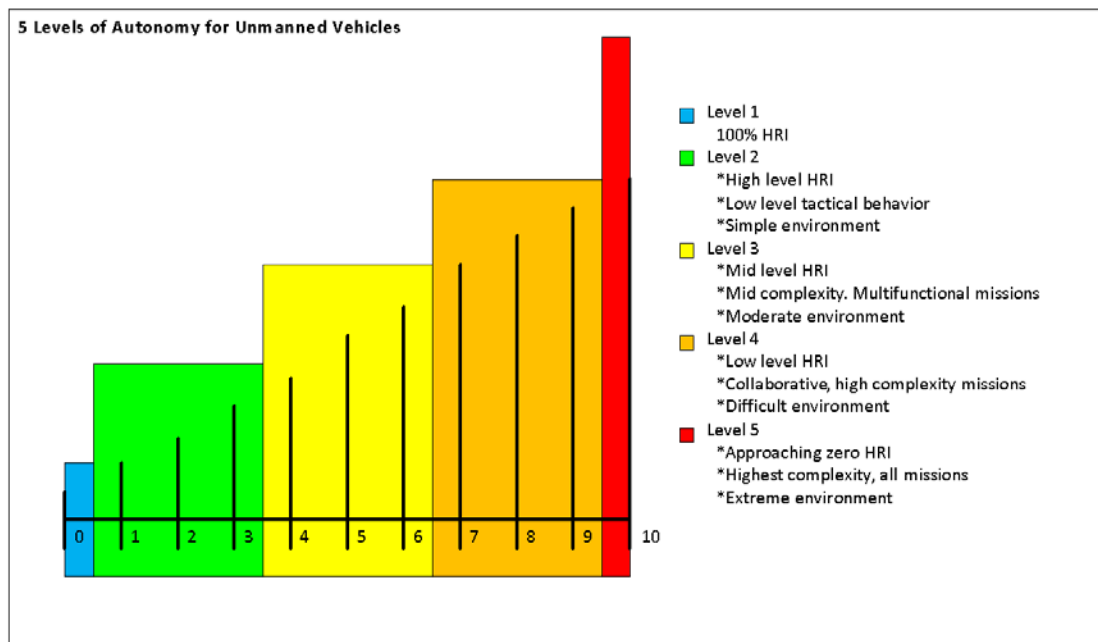


Figure 5: Autonomy Levels For Unmanned Systems (ALFUS)

The addition of on-board processing algorithms to increase autonomy is highly desired in military applications because it is through autonomy that the *capability* of limited manpower available may be increased. This reduces the workload necessary for vehicle control and can free up human resources for other system functions. Another important advantage is removing personnel from dangerous and lethal situations, helping them to “no longer be ‘in the loop’ but rather ‘on the loop’—monitoring the execution of certain decisions” (Donley & Schwartz, 2009, p. 41). This thesis focuses on processing

data on-board UAVs using COTS to achieve a high level of autonomy and the ability to perform the relatively complex task of formation flying.

Next, different flying formations are presented since this project will be running a leader-follower in formation WP pattern. Formation flying provides “mutual support for both tactical and non-tactical situations” (Lambach, 2014, p. 9). According to Reynolds (1987), it is loosely based on three principles: “collision avoidance, velocity matching, and flock centering” (p. 28). Formation flying includes a number of patterns with names that sometimes evoke images from nature—like *swarming* and *flocking*. The USAF has technical names it has assigned to various formations, such as “*two-ship fingertip*, *route formation*, *fighting wing cone*, *tactical line abreast*, and *wedge*” (“Air Force Manual,” 2013, pp. 71, 72, 80, 87, 99. , see Figure 6). The North Atlantic Treaty Organization

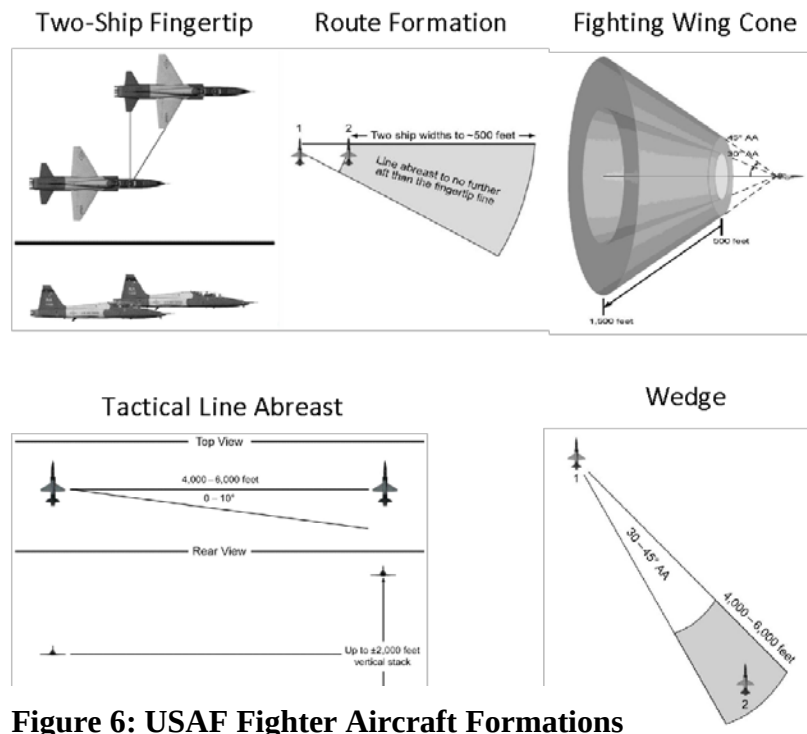


Figure 6: USAF Fighter Aircraft Formations

(NATO) also has names such as “*observation/contact/reform*, *en-route*, *SKE formation*,

and *echelon*.” (“ATP-56(B)”; pp. 2D1, 2D2; “Air Force Instruction 11-2C-17,” 2015, p. 206, see Figure 7).

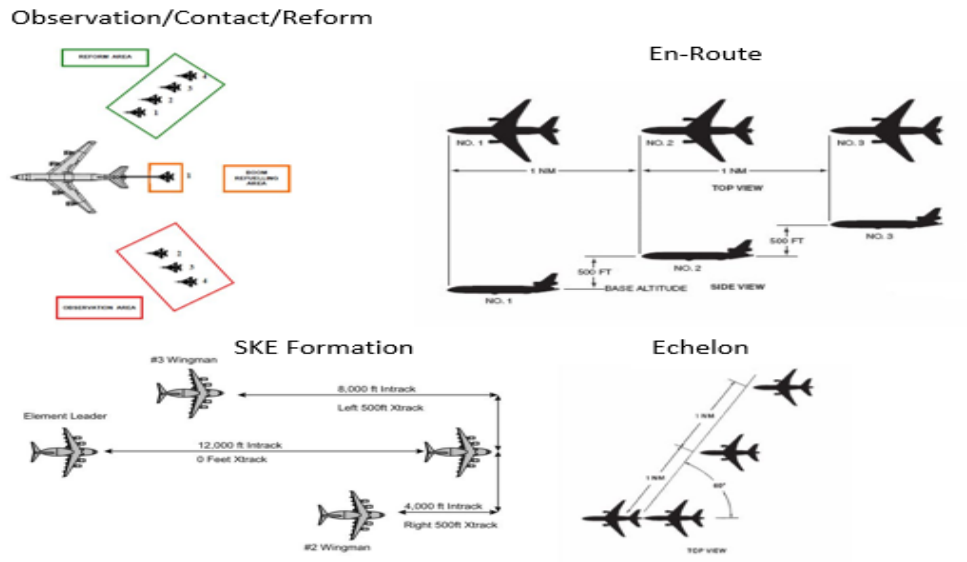


Figure 7: NATO Heavy Aircraft Formations

Components of On-board Processing

In today’s market, many commercial-off-the-shelf (COTS) processors are available, including BeagleBone Black and the Raspberry Pi. These processors are popular with radio control hobbyists who are using them to maneuver planes, cars, quadcopters, and First Person View (FPV) drones. Nayyar and Puri explain that “these boards are low cost, low power, easy deployable and has [sic] user-friendly configurable options” (2015, p. 55).

Manufacturers of the BeagleBone Black describe it as “a true open hardware, community-supported embedded computer for developers and hobbyists” (“Meet the Beagles,” n.d., n.p.). It has a Cortex-A8 processor and will boot Linux “in under 10

seconds” (“Meet the Beagles,” n.d., n.p.). Furthermore, it uses just a single USB cable, though it does have an Ethernet connection and a mini-USB port.

Raspberry Pi, which had had three upgrades by 2016, was originally a “small, barebones computer developed by The Raspberry Pi Foundation, a UK charity with the intention of providing low-cost computers and free software to students” (Verry, 2012, n.p.) in hopes of fostering science education. The Raspberry Pi 3 is “a 1.2GHz quad-core ARM processor . . . as well as integrated Wi-Fi and Bluetooth, a first for the series” (Hern, 2016, n.p.).

Intel’s NUC is “the first offered as a complete system by Intel” (“Mini PC Intel,” n.d., n.p.). Like the full-sized Intel towers, the NUC has a quad-core Intel Pentium processor. It is preinstalled with Windows, 8-channel audio, built-in WiFi, and Bluetooth (“Mini PC Intel,” n.d.).

There are, literally, dozens of other processors on the market—and numerous websites where hobbyists and programmers alike attempt to compare and contrast the devices (Leonard, 2014; “An Overview and Comparison,” 2013). Most reviews focus on qualities such as ease of set-up, total cost, type and quantity of expansion connectors, processor speed, handling of graphics and audio, power consumption, and expandability. Although the reviewers are not always in agreement as to the “best” processor, BBB is

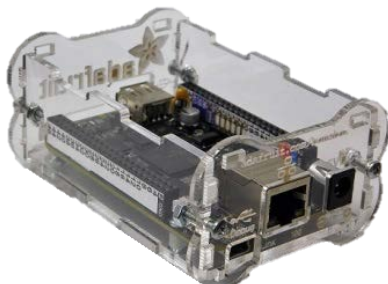


Figure 8: BeagleBone Black

consistently recognized for being widely supported by software developers (Leonard, 2014) and for having two 46-pin expansion connectors (“An Overview and Comparison,” 2013). The BBB (*see* Figure 8) has been selected

as the processor for this research project because it had high developer support, additional onboard eMMC memory, control buttons, and more GPIO pins (see Table 1).

Table 1: Processor Comparison

Processor Comparison				
Categories	Specifications	BBB	RaspberriPi	Intel NUC Kit
Support	High Developer Support	Yes	Yes	N/A
Price	Less than \$50	Yes	Yes	No
Weight	Less than 2 oz	Yes	Yes	No
Power	5V DC	Yes	Yes	Yes
CPU	1 Ghz	Yes	Yes	Yes
	3D Graphics engine	Yes	N/A	N/A
	2 X Progrmmable Real-Time Units	Yes	N/A	N/A
	32 bit			
RAM	512MB	Yes	Yes	Yes
Storage	eMMC 2/4 GB on-board flash	Yes	N/A	N/A
	Micro SD/SD Card slot	Yes	Yes	Yes
Control Buttons	Power, Reset	Yes	N/A	N/A
Connections	1 USB, 1 Mini USB, Ethernet, Micro-HDMI, 2x 46 pin headers	Yes	Yes	N/A
GPIO	65 Pins	Yes	N/A	N/A

A large number of COTS autopilots are also available, including the Pixhawk and the ArduPilot Mega (APM). These autopilots are designed to control Unmanned Ground Vehicles (UGVs), fixed-wing aircrafts, and both traditional and multi-rotor helicopters. These products are also designed for the academic, hobby, and industrial communities at relatively low costs.

According to its manufacturer, the Pixhawk (see Figure 9) is designed as “the industry standard” (“PX4 Autopilot,” n.d., n.p.). It includes “gyros, accelerometers, barometer, GPS, magnetometer/compass, and telemetry. It also has a MicroSD card slot for data logs” (Fabio, 2014, n.p.). It boasts abundant connectivity options and “an

integrated backup system for in-flight recovery and manual override” (“PX4 Autopilot,” n.d., n.p.).

The ArduPilot Mega, likewise, is a “professional quality IMU [inertial measurement unit] autopilot” (“ArduPilot Mega,” n.d., n.p.). It is capable of autonomous stabilization, waypoint-based navigation, and two-way telemetry (“ArduPilot Mega,” n.d.). The Ardupilot Mega has an on-board barometric sensor; further, “interfaces for off-board GPS and magnetometer/compass modules are provided” (Fabio, 2014, n.p.).

Again, many enthusiasts and scholars have debated the pros and cons of each product. They typically consider attributes

such as size and weight, cost, quality, support, flight stability, accuracy, on-board indicators, ease of setup, ability to fly autonomous missions, mission planner ground station compatibility, flight logging capability,



Figure 9: Pixhawk

upgradeability, and debugging difficulty (“Pixhawk versus APM,” 2015). Although reviewers often note that many flight controllers are “quite capable” (Fabio, 2014, n.p.), the Pixhawk typically outperforms its competitors in terms of accuracy, flight logging capability, and quality (“Pixhawk versus APM,” 2015). A comparison between the Pixhawk and the APM autopilots was conducted, and the Pixhawk was selected because it had more of the features required for this project.

Table 2: Autopilot Comparison

Autopilot Comparison			
Categories	Specifications	Pixhawk	ArduPilot Mega
Support	High Developer Support	Yes	Yes
Price	Less than \$50	No	Yes
Weight	Less than 2 oz	Yes	Yes
Power	5V DC	Yes	Yes
CPU	32 bit STM32F347	Yes	Yes
	168 MHz	Yes	N/A
	2 MB Flash	Yes	Yes
	32 bit STM32F103 failsafe co-processor	Yes	N/A
Sensors	3-axis Accelerometer/Gyroscope	Yes	Yes
Interfaces	5x UART	Yes	N/A
	2x can	Yes	N/A
	3.3V AND 6.6V ADC	Yes	Yes
	Internal microUSB	Yes	N/A
GPS	5 Hz	No	Yes
Altimeter	High Resolution	Yes	Yes

Summary

Chapter II reviewed literature that informed this study. The first section looked at the USAF and DoD road maps, particularly noting the explosive growth of UAVs over the past few decades and the possible applications in the military as well as the commercial market. The five groups of UAVs were explained, and it was noted that this project focuses on Groups 1 and 2. The second section described the data processing that happens—presently at a relay station—and how this research plans to implement it on-board UAVs in the future. The five ALFUS levels of autonomy were introduced, and it was indicated how this project intends to raise the level of UAV autonomy from a level one to a level two, perhaps even a level three. Formation patterns were also presented since formation navigation will be executed by a leader and a follower. Finally, a

comparison was given of three of the more popular processors and two of the autopilot controllers. Some of the unique characteristics of these products were described, along with reasons for their selection for this study. Chapter III provides a detailed description of the methodology used. Topics addressed include the hardware/software (HD/SW) components, architecture design, and test plans for the prototyping of UAV flight formations.

III. Methodology

Overview

The main goal of this chapter is to discuss the hardware, software, and test plans used to implement inter-vehicle communications and on-board processing for close UGV formations. It defines the architecture design and integration of the hardware and software used in this research project. It also briefly describes the test plans and the method of analysis that will be applied to obtain the results in Chapter IV.

Hardware Architecture

At the highest level, there are three main systems to the leader and follower network architecture configuration design: the leader, the follower, and the ground control station (*see* Figure 10). The *leader's* network configuration architecture incorporates a Pixhawk autopilot, a USB-TCP232-24, and a Wave Relay module. The Pixhawk was connected to the Wave Relay using a cable manufactured in-house for this purpose; pinout information is the first illustration in Appendix A.

The mission of the leader was simple: navigate through the box WP pattern until told to do otherwise. The methodology for creating the leader's architecture consisted of using Mission Planner (MP) to load the Pixhawk autopilot with a set of four WPs in a simple box pattern. A fifth WP was set up as a jump to the first WP to effectively command the leader to repeat the flight plan until it was placed in manual mode by the safety pilot or until the mission was aborted by the GCS operator.

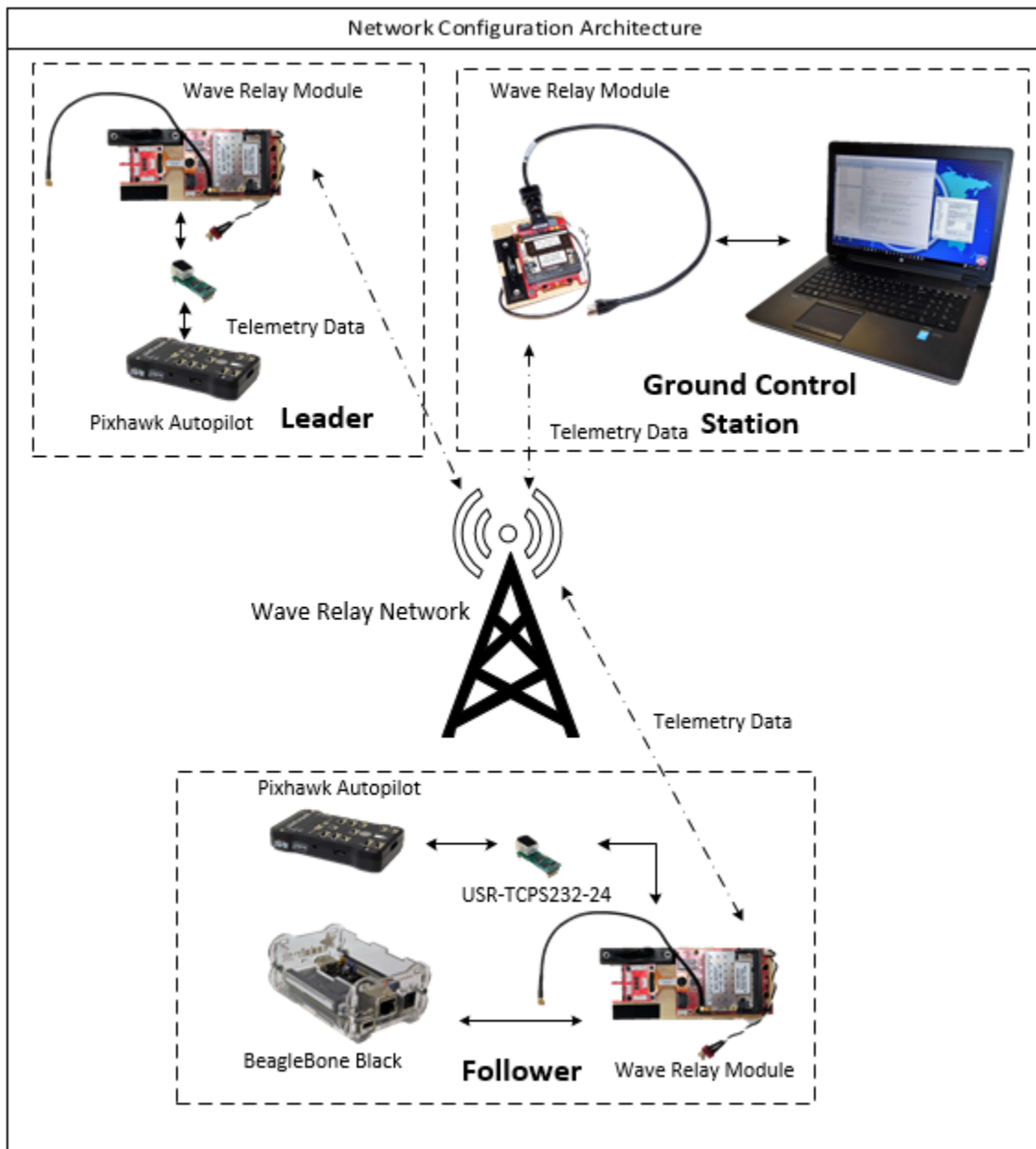


Figure 10: Network Configuration Architecture

The *follower's* architecture design integrated four components: the BBB, a Pixhawk autopilot, an USR-TCPS232-24, and a Wave Relay. A USR-TCPS232-24 was likewise used by the follower to convert the TTL telemetry data from the Pixhawk

autopilot into an Ethernet input signal to a Wave Relay module. The Wave Relay then broadcast the telemetry data out to the appropriate equipment linked via the GCS. The BBB was connected to the Wave Relay with an Unshielded Twisted Pair (UTP) Ethernet cable. The BBB was powered by connecting it to the voltage rail of the Pixhawk using a cable manufactured in-house for this purpose; its illustration and pinout information can be found in Appendix A. The methodology for the follower took advantage of the BBB's data processing power to give the follower the ability to appraise a situation, generate a decision, and act upon it. This gave it a small degree of self-government and autonomy and also built on the idea of command and control where a command was given through a software script to control a behavior or a response. In this case, the follower received the leader's telemetry data, processed that information to assess the leader's location, determined its location in relation to the leader, generated a decision based on a given command (for example, to stay back a certain distance, maintain a certain angle, or anticipate a desired location), calculated where it needed to be, and navigated to the desired location.

The final piece of architecture for the *ground control station* consisted of two hardware components: a laptop and a Wave Relay module. Other OSS components were used as part of the GCS to monitor and execute commands: the command line (part of most Windows operating systems and readily available), Putty, WinSCP, and a USR-VCOM virtual serial port server utility manager for the USR-TCP232-24. These components will be discussed in more detail in the software section of this chapter. Figures 11, 12, and 13 show schematic illustrations including Internet Protocol (IP) static addresses of the hardware architecture for the leader, follower, and GCS respectively.

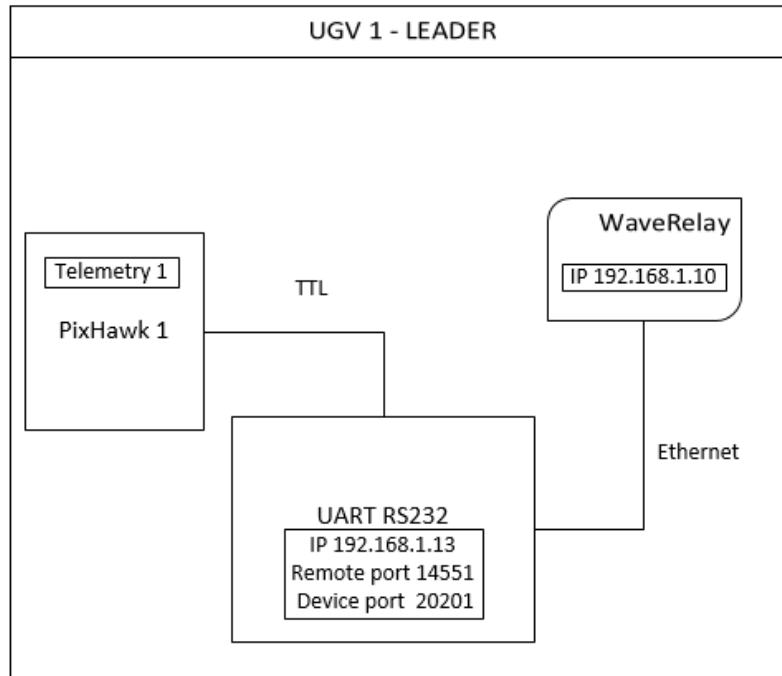


Figure 11: Leader Architecture Design

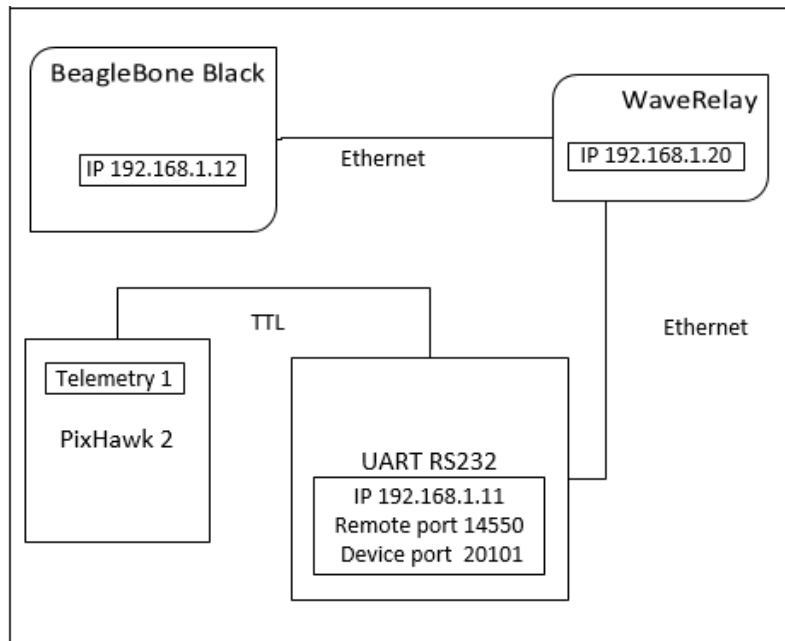


Figure 12: Follower Architecture Design

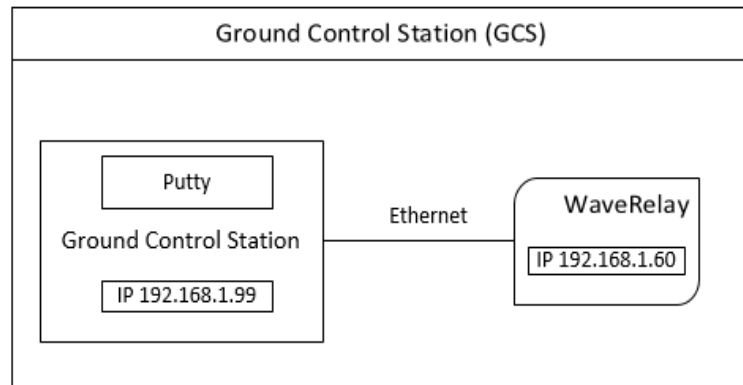


Figure 13: GCS Architecture Design

Hardware

The UGVs were obtained from the AFIT Autonomy and Navigation Technology Center (ANT) Lab. The first piece of hardware was the *autopilot*, which monitors real-time telemetry information such as altitude, longitude, and latitude. It is a very powerful tool as it maneuvers the UAV through a series of WPs in a flight path. The autopilot used for this research is the Pixhawk, which is one of the most popular autopilots currently on the market. It is small (about the size of a credit card), uses low power, and weighs only 1.31 ounces (“3DR Pixhawk,” n.d.). The Pixhawk performs a limited amount of on-board data processing to navigate the vehicle through a preset number of WPs programmed by the user or to loiter about a single WP. A very simple data processing architecture using the minimum number of components to navigate a UAV would be to use an autopilot in conjunction with a Ground Control Station (GCS) like MAVProxy or MP (see Figure 14). As the illustration indicates, this architecture requires one ground station to one autopilot. This configuration is equipment and operator intensive. It is

cumbersome for two UAVs and would be burdensome to implement in a swarm formation of many UAVs because the need to have one GCS per UAV requires it to be equipment and personnel intensive.

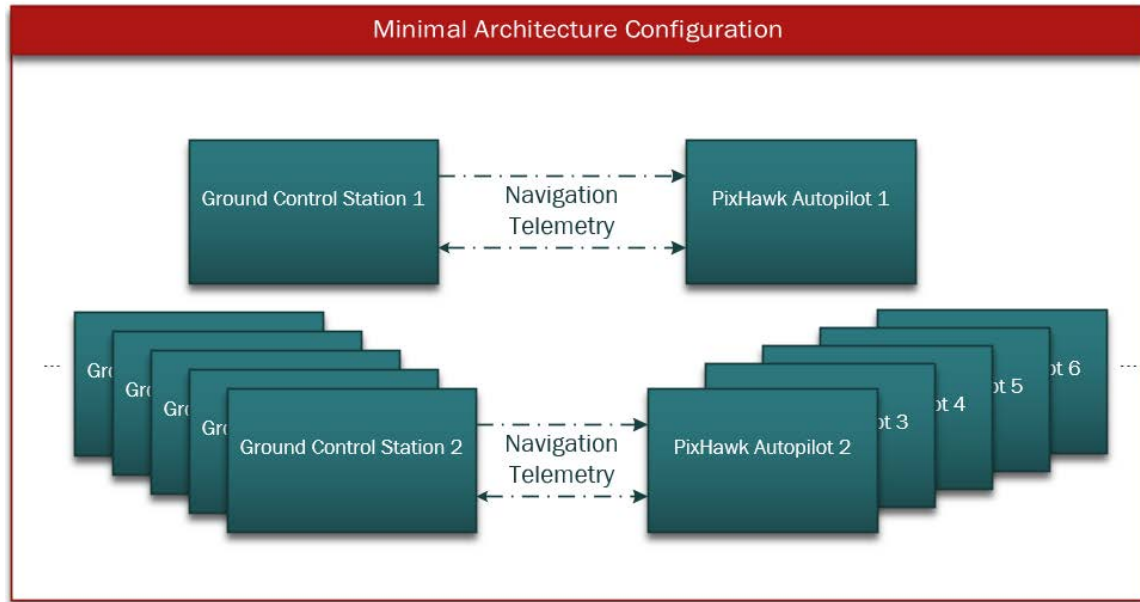


Figure 14: Minimal Data Processing Architecture Configuration

The GCS is the window through which the pilot can see the current state of the UAV and control it as if he/she were in the cockpit. It is briefly described and illustrated in this section because it does consist of a laptop for visual display; however, the GCS will be more fully described in the software section of this chapter.

The second piece of hardware, the *BeagleBone Black* (BBB), is a powerful microprocessor, and, like the Pixhawk, it is compact, uses low power, and weighs less than 1.5 ounces. This is important especially for Group 1 UAVs where small components that use less power and are light-weight provide the capacity of bigger payloads. The BBB has 512 MB of RAM and 4 GB of on-board flash storage. It is compatible with several software platforms (“BeagleBone Black,” n.d.). Adding a processor like the BeagleBone Black to the data processing architecture of a UAV (see Figure 15) increases

the on-board processing capacity and the number of functions which can be accomplished. For example, an on-board processor means a UAV can now process complex software algorithms to use an on-board camera for photographs or video, sensors to find hazardous chemicals in the air, or telemetry data to communicate with one or more UAVs in formation. It also raises the level of complexity in applications and the autonomy that can be developed with this new capability, one of which is Cooperative Control from one UAV to the UAV. *Cooperative Control* is defined in this prototyping study as the collection of intelligence (latitude, longitude, and velocity) of the leader UGV, assessing the information (to calculate where the follower UGV should be in relation to the leader) and taking action (navigating to that location.)

In Chapter II the different levels of autonomy were introduced. A UAV with a powerful on-board computer to process large and complex algorithms raises the level of autonomy from level one, which requires 100% human interaction, to at least level two, depending on the capabilities developed and implemented. This research adds the capability of cooperative communication with at least two UAVs, but that capability, once implemented, can be extended to even more UAVs as depicted in Figure 15. The advantage of this configuration can be conceptualized in the application of flying several UAVs in a swarm formation. One ground station and one BBB can deploy several UAVs streamlining the equipment and operators needed to perform this task.

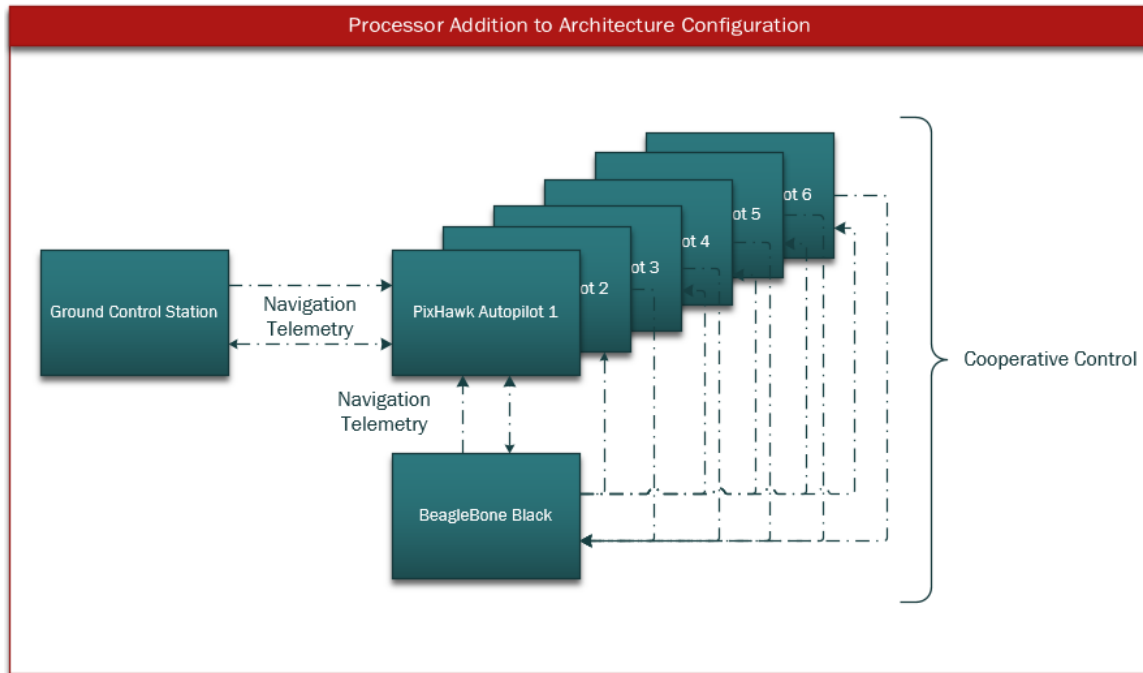


Figure 15: BBB Processor addition to Architecture Configuration

The *USR-TCPS232-24*, sometimes referred to as a UART, is a small module that converts TTL serial telemetry from the Pixhawk autopilot into an Ethernet format UDP/IP signal, enabling transmission to the Wave Relay signal (see Figure 16). The



Figure 16: USR-TCPS232-24

small module has the capability of using two different types of voltage levels: 3.3VDC or 5VDC. It has a current range of 150–200 ma. Transmittal of the telemetry data between the autopilot and the Wave Relay was achieved via two of these modules, one for each of the autopilots used. This converter was chosen for its functionality, compact size, and low cost.

The third piece of hardware was the *Wave Relay* module, Wave Relay modules were used as a switch to route the signals to the leader and the follower wirelessly and to provide the communication link for all components in this architecture design (see Figure 17). In the leader configuration, it connects the autopilot through the USB-TCP232-24 module to the GCS. In the follower configuration, it connects both the autopilot (with the use of a USB-TCP232-24) and the BBB to the GCS. In the GCS configuration, the Wave Relay links the leader and the follower.



Figure 17: Wave Relay

Software

This section discusses the software architecture and the OSS components used, as well as the methodology used to develop a single Python software script and how it was developed. It also briefly outlines how the architecture differed from Gray's architecture.

Software Architecture

Due to a Wave Relay being available at the, Ethernet protocols were selected. The Wave Relay served as the main communication link for all of the hardware and software components for this research. The SV-1 displays the system diagram (see Figure 18), and the relationship between each of the hardware and software components for the leader-follower design.

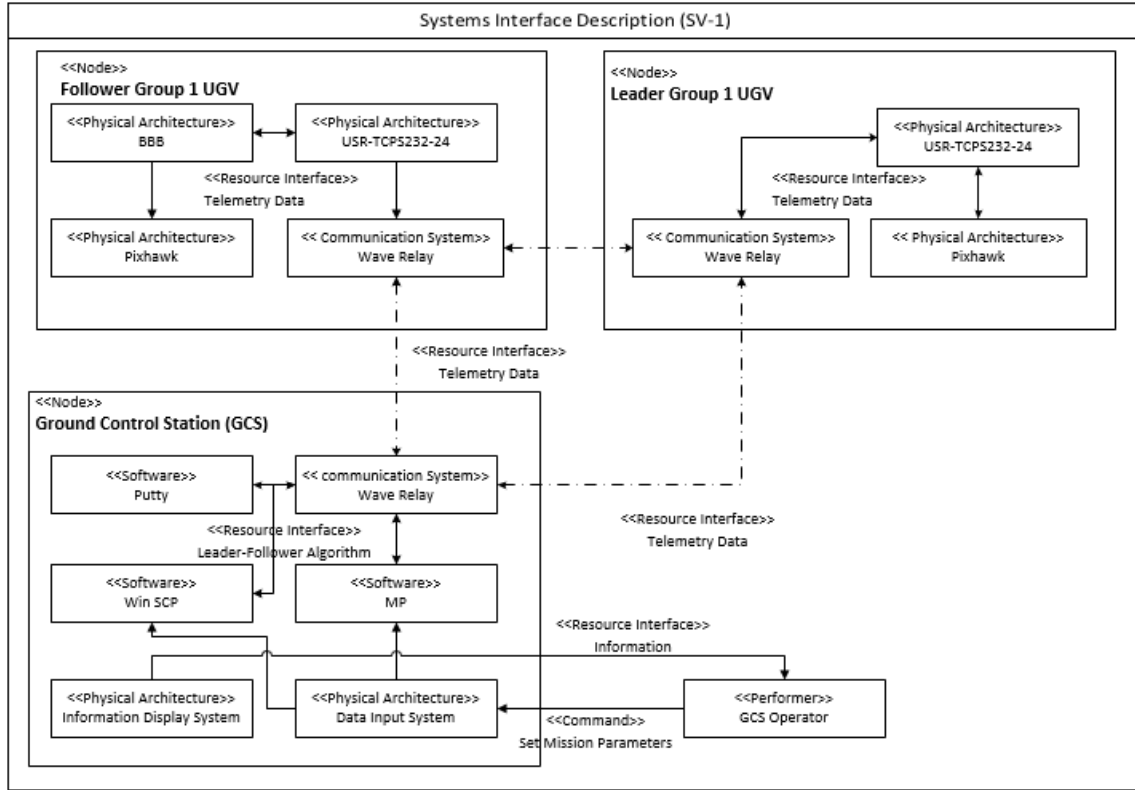


Figure 18: SV-1 Systems Interface Description

The software script was developed in two phases. The first phase was the development of the operating system software used by the BBB to communicate with the follower. A 32 GB removable flash memory micro-mini Secure Digital (SD) card was used to burn an image of the operating system (Linux kernel) and to load various OSS utilities needed for the operating system (see Table 1). The version of bone-debian image that was loaded was configured to boot up and operate explicitly with an Ethernet connection. The second phase was the development of the software script that directed how the leader and the follower performed cooperative communication to complete a mission.

Table 3: Software Tools on the BBB Micro SD Card

BBB Micro SD Card Tools
Autopoint
Bone-debian-8.5-lxqt-2gb-armhf-2016-06-19-2gb.img
DroneKit-Python
Libtool
MAVLink
MAVProxy
Numpy
Pymavlink
Python 2.7
Python-dev

Python was selected as the programming language for this project. Of the many object-oriented programming platforms on the market today, Python is a free OSS that has a large library of modules available for immediate application in development. It also has a large community of users that work continually to improve its applications to produce a programming platform that is stable and robust. Python was the programming language used in the research for which this project is a follow-on; therefore, it was advantageous to use the same platform.

DroneKit-Python is one of 3D Robotics' latest innovations. One of the most attractive features of their new platform is the ability to control multiple vehicles from one script, which is the reason it was chosen for this research. It is a library of OSS modules defined as Application Programming Interface (API) that interacts with a processor such as the Raspberry Pi or BeagleBone ("Companion Computers," n.d.) and communicates with autopilots like the ArduPilot or the Pixhawk ("About DroneKit," n.d.) to transmit real-time telemetry data.

Micro Air Vehicle Link (MAVLink) is a lightweight, connection protocol popular for UAV navigation. It provided the code, rules, or language necessary to recognize given directions in communication with UAVs. In essence, it is a message 17 bytes in length of the following structure: “message length = 17 (6 bytes header + 9 bytes payload + 2 bytes checksum)” (“Mavlink Tutorial for Absolute Dummies,” n.d., n.p.). It has been proven on several of the autopilot platforms on the market and is considered a backbone in communication (“MAVLink Micro Air Vehicle Communication Protocol - GroundControl GCS,” n.d.). This provided confidence that it is a robust platform for development of the leader-follower algorithm.

Mission Planner (MP) is a complete GCS; however, unlike MAVProxy, it is not a minimal application (“MAVProxy, n.d.). MP is an advanced graphical user interface which displays most of the telemetry components at a glance (“Mission Planner Overview” n.d.). It is designed to emulate a cockpit. The GCS operator can view on the screen live telemetry data, such as altitude, ground speed, distance, vertical speed, and attitude of the UAV. Another feature of MP is that it collects over 30 parameters of flight telemetry data into a t-log file, making it a dynamic tool for data analysis.

OSS Utilities

The other components or utilities used to create the GCS included, first, the *command line* available in most Windows operating systems. It was used to execute commands for administrative functions and other, similar tasks. It provided the ability to query all of the network systems and test connectivity using assigned static IP addresses.

The second component was *Putty*, a free Secure Shell (SSH) and telnet client for Windows. It has a simple GUI which allows users to establish remote, encrypted, login network services. Putty was the main connection portal between the GCS and the BBB through which all communication was made.

The third, and final, component was *WinSCP*, a free Secure File Transfer Protocol (SFTP), Secure Copy Protocol (SCP), and File Transfer Protocol (FTP) client for Windows. This utility enabled the easy transfer of files back and forth from the computer terminal through a GUI to the BBB.

Software Architecture

Gray's previous UAV software architecture utilized MAVProxy's DroneAPI, an earlier version of DroneKit-Python. The software platform he used did not allow control of multiple vehicles from within one script (Gray, 2015, p. 55). The software did not have the capability to do so. The new DroneKit-Python format has the added capability to accomplish this task. Communication between UAVs was previously carried out via two instances of MAVProxy, causing it to be necessary to have one laptop per GCS. One of the biggest differences in this research was the addition of the BBB, which allowed on-board cooperative control of multiple vehicles from within one script and one GCS. Cooperative communication in Gray's work was done by relaying telemetry data between two UGVs through two GCSs. This research designed the communication to take place on-board the BBB, enabling the vehicles to employ cooperative communication between the leader and the follower directly, bypassing the GCS and using it only for operator

control and to monitor a handful of telemetry parameters. The sequence of events for the cooperative communication script is described in the OV-6c Figure 19.

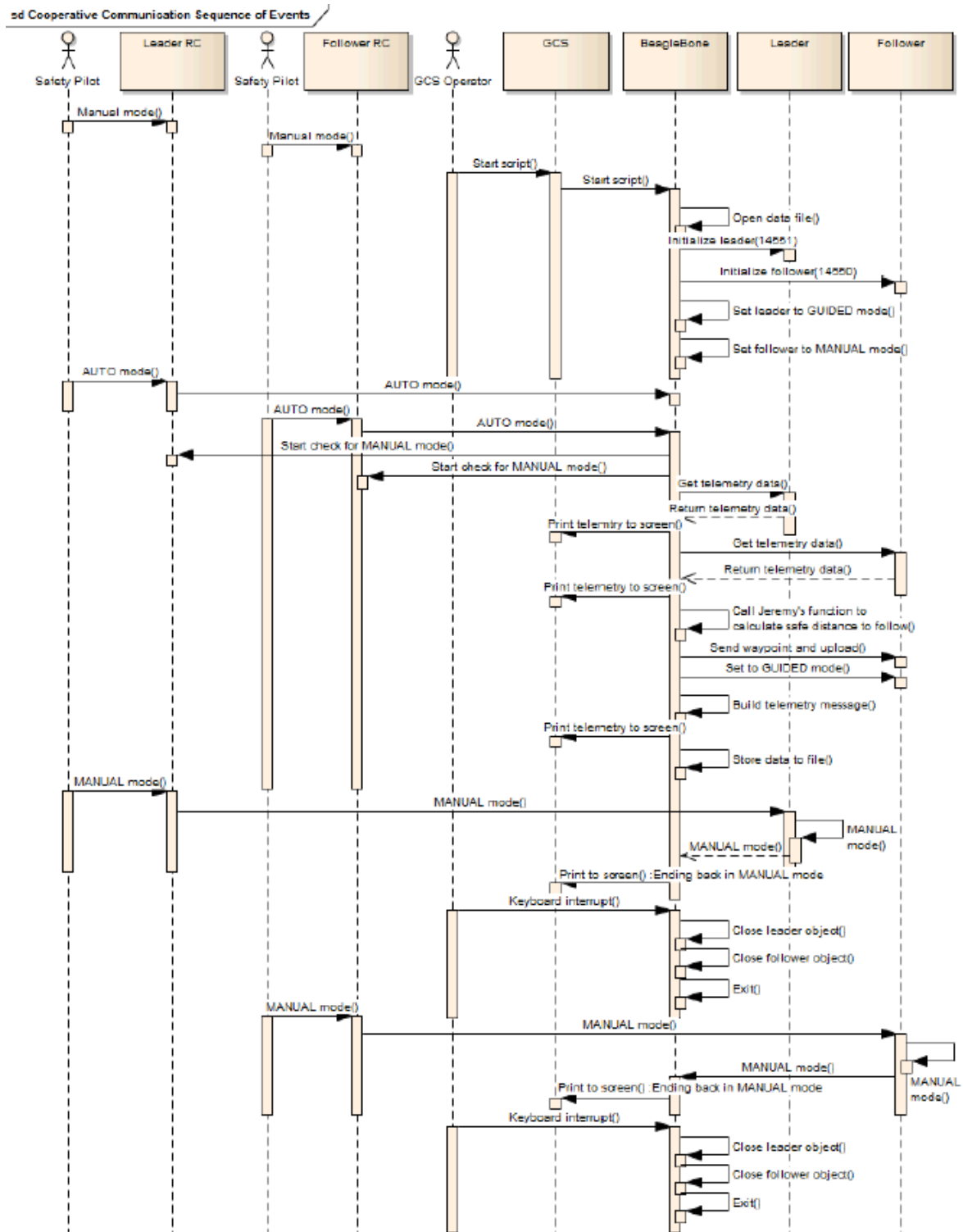


Figure 19: OV-6c Operational Event Trace Description

Cooperative Communication

The architecture to establish cooperative communication was developed with the goals to increase the effectiveness of inter-vehicle communications, to employ on-board processing, and to add a degree of autonomy for a team of two or more UGVs navigating in close formation. The main goal is to have the vehicles exchange latitude, longitude, altitude, velocity, heading, and ground course telemetry data to govern themselves for the duration of a pre-established mission as described by the OV-1 (see Figure 20).



Figure 20: OV-1 Cooperative Communication from one UGV to another UGV

In this research project, the leader UGV was pre-loaded with a basic box mission pattern (see Figure 21). The follower UGV was configured with the BBB microprocessor. The decision to equip the follower with the BBB was arbitrary; the BBB could have been installed on the leader without affecting how the UGVs behaved. The BBB was programmed with a cooperative communication algorithm to accomplish the mission. It provided additional memory and processing power to perform calculations, exchange telemetry data, and constantly monitor the location of both UGVs to communicate to the

follower information to maintain a distance, predetermined by the GCS operator, behind the leader, an angle (-x axis from the tail of the leader), and a commanded location to complete a certain pattern (Gray, 2015, p. 131). The follower responded to the telemetry data received from the leader and traveled towards the commanded location.

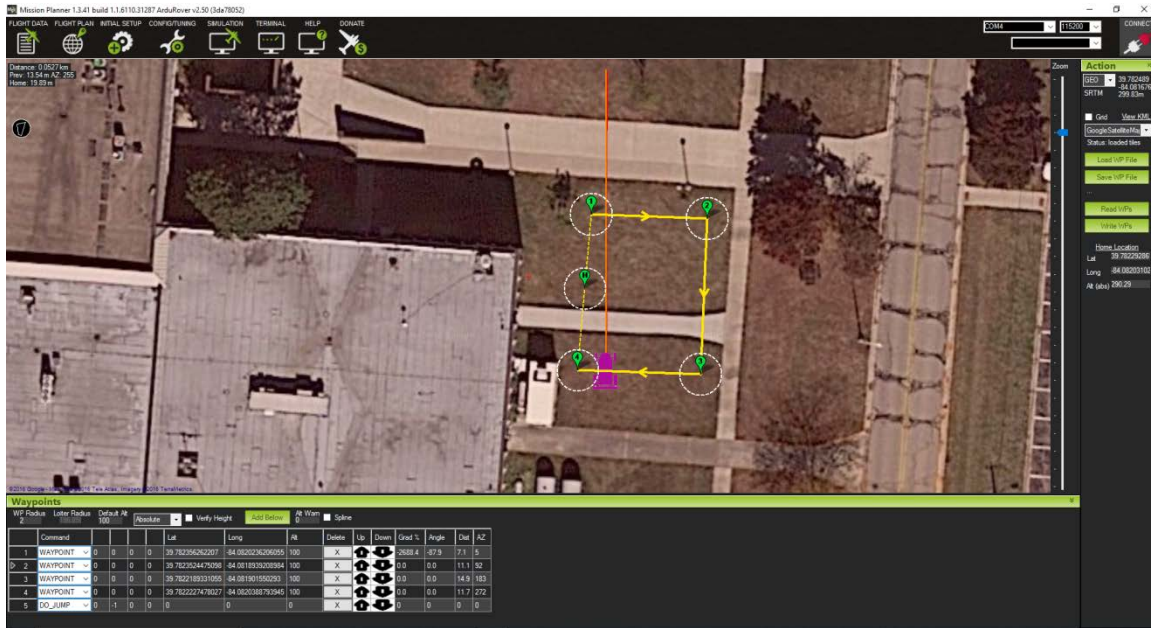


Figure 21: Mission Plan Box Pattern

A *use case* in software design provides a brief description of a goal and describes the basic flow of events. A *happy path* occurs when the flow of events happens without errors or exceptions. The use case for this project is briefly described below (the fully dressed use case can be found in Appendix B):

UGS operator wants to use two (or more) UGVs to navigate through a basic set of WPs using a formation flight pattern. A UGV is designated as the leader, and a second UGV is designated as the follower. The follower was

configured with a BeagleBone Black (BBB) microprocessor on-board; the BBB ran an algorithm to provide telemetry data from the leader to the follower.

Basic Flow – Employ Cooperative Communication

1. UGV pilot pre-loads mission plan on the designated leader UGV's autopilot using MP.
2. UGV pilot loads variable mission parameters into the software algorithm at the ground control station.
3. UGV pilot loads software algorithm onto BBB microprocessor residing on the follower UGV.
4. Safety pilots stand by with Radio Controls in manual mode.
5. UGV pilot starts software algorithm to initiate cooperative communication between the leader and the follower.
 - a. Software algorithm
 - i. opens a telemetry file
 - ii. initializes leader and follower
 - iii. monitors continually safety pilots' RC manual/auto switch
 - iv. sets leader and follower to "GUIDED" mode
 - v. monitors leader and follower locations
 - vi. sends follower
 1. radial distance away from the leader (Off_r)

2. angle away from the $-x$ axis from the tail of the leader
(Off_{θ})
 3. commanded WP (Off_{11}) for the follower
 4. frequency control (update rate in Hz)
- vii. appends telemetry data to file
6. Safety pilots flip MANUAL switch to AUTO as soon as they receive the command from the UGV pilot at the ground control station.
 7. Mission is terminated when the UGV pilot at the ground control station determines mission is complete or when one or both of the safety pilots determine(s) it is appropriate to do so.

The user pre-defines a set of four parameters or Technical Performance Measures (TPMs) to be used by the script: Off_r , Off_{θ} , Off_{11} , and the frequency control. Off_r is the distance away from the tail end of the leader, Off_{θ} is the angle away from the $-x$ -axis, Off_{11} is the follower's commanded WP, and frequency control is the update rate in Hz of telemetry data requested from the leader by the follower (see Figure 22) and the actual vehicles can be seen on Figure 23.

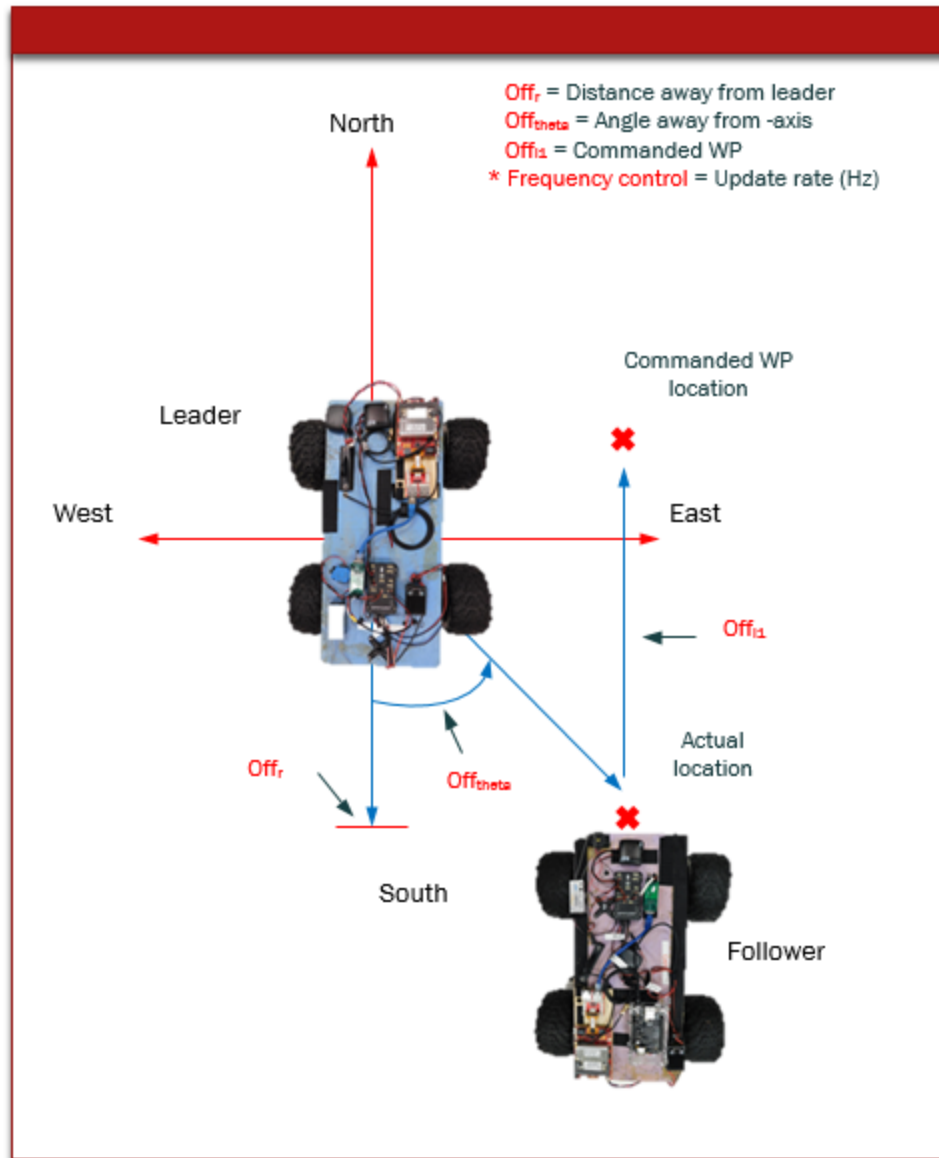


Figure 22: Technical Performance Measures

Since MP was used only to load the flight plan onto the leader UGV; a log file was not automatically recorded. Consequently, it became necessary to store specific key telemetry information for data analysis. The script opened a data file, initialized the leader and the follower, and set the leader to GUIDED mode and the follower to MANUAL mode. In the event that something went wrong, a failsafe feature was

designed into the script: the MANUAL mode of the safety pilot's radio controls was constantly monitored, so, if the safety pilot switched to MANUAL mode, the UGV was put in MANUAL mode until the safety pilot reverted to AUTO MODE or the GCS aborted the program. The next step was to call the calculation function modules. Two function calls were taken in their entirety from Gray's 2015 thesis work:

1. **LLA_ECEF_Convert**, which converted latitude, longitude, and altitude (LLA) coordinates to earth center/earth fixed (ECEF) coordinates and vice versa, and
2. **Multi_vehicle_toolbox_v3**, which included a function called `follower_pos` to determine the next desired position of the follower. This function takes five inputs: leader position, leader velocity, Off_r , Off_{θ} , and Off_{l1} .

The Leader-Follower software algorithm can be found in Appendix C.

It was key during the development of the software script to define all of the communication network assignments. A list of Internet Protocol (IP) was created in order to assign static IP addresses to all of the components (*see* Appendix D).

Test Plans

Two test plans were developed to assess the responsiveness of the software communication between vehicles on four different TPMs: Frequency Control, Off_r , Off_{θ} , and Off_{l1} .

The first test plan was designed to ensure all of the network components communicated through the Wave Relay and to verify the software algorithms were

performing as expected using the UGVs. It was conducted just outside the ANT Lab at AFIT. A basic box pattern of four WPs was used in conjunction with a jump to the first WP in order to place the leader in a loitering condition. The pattern was repeated for several laps to collect telemetry data. Once the leader's GC was verified, a small range of seven test combinations of distances, angles, and anticipated distances were performed to simulate possible mission plans (*see* Table 4).

Table 4: Test Plan 1 - Code Verification

Test Plan 1 Code Verification			
Test	1	2	3
Frequency Control (Hz)	1	10	20
Off-Radial (Meters)	2	2	2
Off-Theta (Degrees)	0	0	0
Off_l1 (Meters)	0	0	0

During the actual testing in the field it was decided an Off_r distance of 2.0 meters was sufficient due to the restricted space available for testing just outside of the ANT Lab. A printout of the verification code can be found in Appendix E

The second test plan added four additional tests to verify the ability of the BBB to manage the communication in a cooperative manner (*see* Table 5). The same box pattern of WPs was used to see how accurately the follower responded to commands from the leader. Gray (2015) did a similar analysis testing the “average squared error between follower vehicle’s current and desired position at each time step” (p. 30).

Table 5: Test Plan 2 Cooperative Communication

Test Plan 2 Cooperative Communication									
Test	1	2	3	4	5	6	7	8	9
Frequency Control (Hz)	1	10	10	10	10	20	20	20	20
Off-Radial (Meters)	2	2	2	2	2	2	2	2	2
Off-Theta (Degrees)	0	0	-45	-45	-45	0	-45	-45	-45
Off_l1 (Meters)	0	0	0	1	2	0	0	1	2

Summary

This chapter introduced the methodology for the research by listing various components used in the architecture design: Pixhawk, BBB, Python, DroneKit, MP, and MAVLink. Integration of hardware and software components was explained in detail, as was the ground/test procedure for data collection. The methodology for the test plans was also explained. Results of the tests and detailed data analysis will be the subject of Chapter IV.



Figure 23: Leader and Follower UGVs

IV. Analysis and Results

Overview

This chapter discusses the procedures, equations, and results of the two test plans introduced in Chapter III. It also considers the integration anomalies encountered, performs a comparison of this research with previous work, and provides answers to the investigative questions posed in Chapter I.

Test Plan 1

Test Plan 1 tested the network configuration. All of the IP addresses listed in Appendix D were tested using the Wave Relay. The process for Test Plan 1 was to configure the leader, the follower, and the GCS in the lab with their respective components as directed by the network configuration design presented in Chapter III. The Wave Relay network was put in the loop, and IP testing was performed.

The first attempt at putting the components together did not go as planned. All of the components had static IP addresses assigned; however, attempts at running the script determined the BBB was not recognized with the assigned “static” IP address. Lab tests were initially conducted using a router in the network configuration, but, even though all the instruments had been set to “static” IP addresses, there was no indication of connectivity failure until the Wave Relay was placed in the loop and tested in the field.

At first it was thought that the router was assigning an IP address dynamically to the BBB. The router’s software gave every indication the IP address assigned to it was “static” and was working as expected; however, this proved not to be the case. More testing was done to investigate why the BBB was not recognized. The router was

removed from the network configuration, and it was replaced with a network switch to eliminate the possibility that the router was creating the issue.

Testing the network configuration failed a second time in the field even though all of the IP addresses had worked well in the lab using the network switch. Upon further investigation in the field, it was discovered that the BBB was the root of the problem. Further research indicated that the BBB comes preinstalled with its own configuration manager called, ironically, “conman.” Conman was dynamically assigning an IP address to the BBB. This issue was resolved by removing conman. Once the configuration manager was removed and all the IP addresses were verified to make the appropriate connections using the Wave Relay, the next step was to start integrating the HW/SW and test the effectiveness of inter-vehicle communication between the UGVs.

During the code verification, as the frequency control was raised, there were errors encountered at about 18 Hz. These errors were caused due to 10 to 15 screen status updates that were sent to monitor the UGV status. These status updates were sent to the screen to monitor different states in the UGVs due to anomalies encountered with MP. MP was used only to load the box pattern onto the leader UGV. Once these status messages were streamlined down to a handful of status updates to the GCS, the frequency worked without issues at a rate of 20 Hz. The frequency might have updated at a higher rate; however, it was not attempted once the 20 Hz frequency worked without errors.

Test Plan 2

The results for Test Plan 2 are presented by way of a subset of three of the nine tests in the plan. The effectiveness of inter-vehicle communication between the leader

and the follower was analyzed using two equations in MATLAB and applying them to the navigational coordinates collected for the follower and the follower's WP. These two locations provided the information necessary for the calculation of how close the follower was to its desired location.

The Law of Cosines equation used to measure the GPS distance was drawn from a Blog Spot sponsored by BlueMM ("Excel Formula," n.d.). In an article entitled "Computing Distances," Bob Chamberlain contends that while the equation may be mathematically correct, the results are not exact, especially when measuring very short distances on the surface of the earth: "It is a perfectly fine formula when it is used for the right purposes. Solving for short distances on the surface of the Earth is not one of those purposes" ("Calculate Distance," n.d., n.p.). More detailed information on the haversine formula can be found in Sinnott's "Virtues of the Haversine" (1984). In another article, entitled "Calculate Distance, Bearing and More Between Latitude/Longitude Points," other scholars indicate that the law of cosines is just fine: "The simple spherical law of cosines formula . . . gives well-conditioned results down to distances as small as a few meters on the earth's surface" (n.d. n.p.). It appears to be a factor of the precision of the computer and software being used. MATLAB provides a precision of 16 digits by default. According to MathWorks, MATLAB can also be set to provide 32 digits of accuracy and higher ("Digits," n.d.). For the purposes of this study it was decided that 16 digits was sufficient.

This first equation was used to provide the Global Positioning System (GPS) distance between two points:

$$(L_i - \hat{L}) = ACOS[SIN(LAT1) * SIN(LAT2) + COS(LAT1) * COS(LAT2) \\ * COS(LON2 - LON1)] * 6371000$$

where:

$$\begin{aligned} L_i &= \text{Commanded position} \\ \hat{L} &= \text{Desired WP} \\ LAT1 &= \text{Commanded Latitude} \\ LON1 &= \text{Commanded Longitude} \\ LAT2 &= \text{Actual Latitude} \\ LON2 &= \text{Actual Longitude} \end{aligned}$$

This second equation was used to calculate the Distance Root Mean Square (DRMS):

$$DRMS(L_i, \hat{L}) = \sqrt{\frac{1}{N} \sum_i^N (L_i - \hat{L})^2}$$

where:

$$\begin{aligned} L_i &= \text{Follower Actual Location} \\ \hat{L} &= \text{Follower Commanded Location} \\ N &= \text{Number of samples taken} \end{aligned}$$

These Tolerance Interval (TI) equations were used to calculate the upper and lower TIs:

$$Y_L = \bar{Y} - k_2 s; \quad Y_U = \bar{Y} + k_2 s$$

where:

$$\begin{aligned} \bar{Y} &= \text{Mean} \\ k_2 &= \sqrt{\frac{v \left(1 + \frac{1}{N}\right) z_{(1-p)/2}^2}{\chi_{1-\gamma, v}^2}} \\ s &= \text{Standard Deviation} \end{aligned}$$

The TIs were used to set a lower bound on how close the follower was getting to the leader and to find a safe radial distance for the follower. The first equation describes a positive 5% TI in relation to the Commanded Waypoint (CW):

$$TI\ 5\% - CW > 0$$

where:

$$TI\ 5\% > CW$$

The second equation describes a negative 5% TI in relation to the CW:

$$-TI\ 5\% + CW > 0$$

where:

$$TI\ 5\% < CW$$

While cooperative communication was prototyped on the ground using UGVs, this architecture can be implemented in the air using UAVs. The TI represents a safety consideration that is important when implementing close formation of fixed wing vehicles for drag reduction or fuel conservation.

The distribution of the follower position about the commanded waypoint, seen in blue in Figures 24 and 25, depends on several factors, including velocity, wind, atmosphere, time delays, GPS signal, and vehicle; however, for this analysis these factors were not taken into account. These parameters were held relatively constant for this research.

To illustrate the advantages of observing safe distance two cases are presented. This project obtained positive DRMS lag results while other researchers in the ANT lab conducting a similar project during the same time period realized negative DRMS lag results. For this research, the positive lag obtained was defined as distance behind the

CW (away from the leader). A positive lag provided a safety distance from the CW where the $TI\ 5\% - CW > 0$; therefore, in a situation where the vehicle has a positive lag, it can be commanded ahead of the leader and have less than TI 5% probability of impact (see Figure 24).

For the other researchers, a negative lag was defined as distance ahead of the CW (closer to the leader). A negative lag did not provide a safety distance from the CW and the $-TI\ 5\% + CW < 0$ caused a collision. In such situations, the vehicle must be commanded behind the leader so that the $-TI\ 5\% + CW > 0$ in order to have a probability of impact less than 5% TI (see Figure 25).

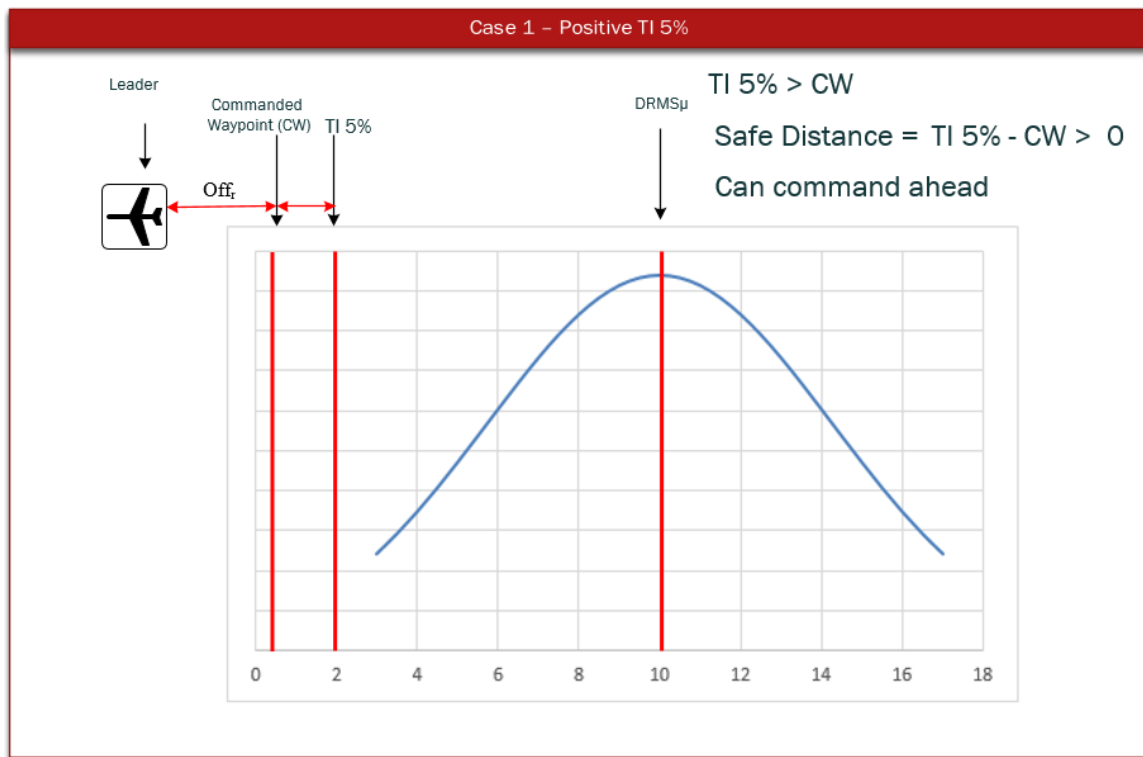


Figure 24: Case 1 – Positive TI 5%

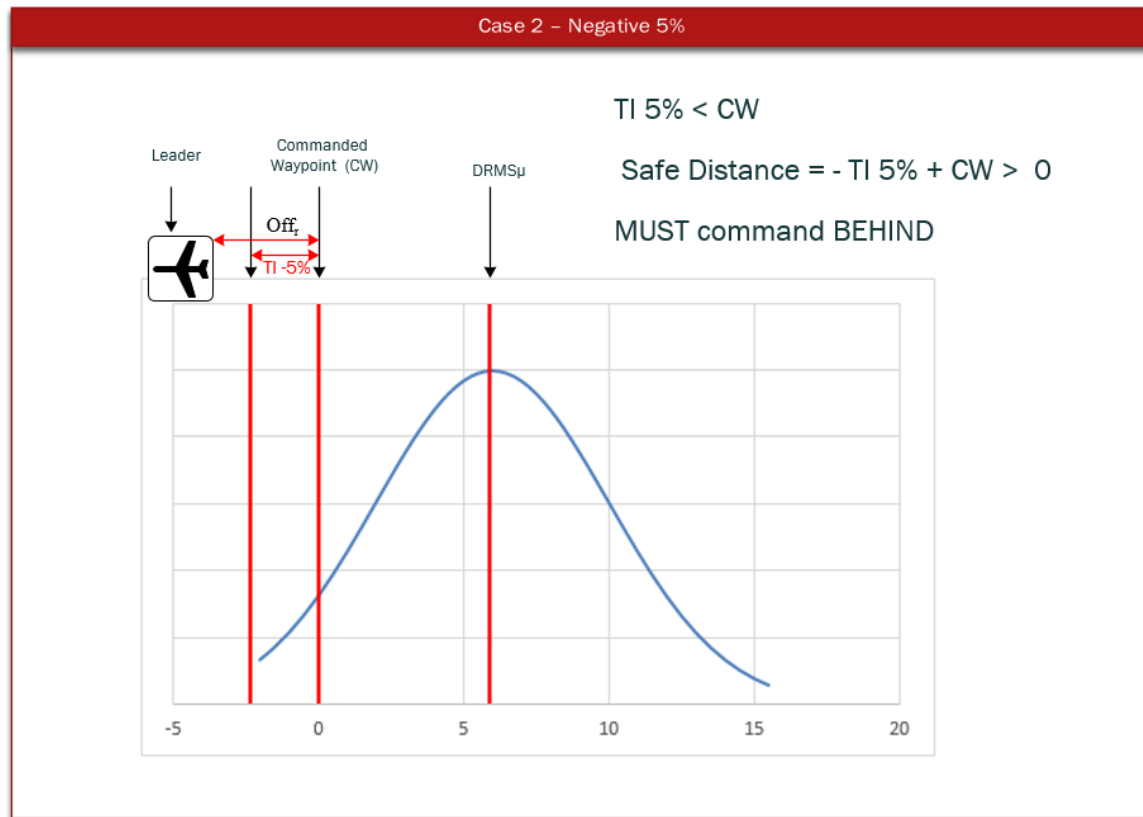


Figure 25: Case 2 - Negative TI 5%

Test Plan 2 provided data of the two UGVs used to execute cooperative communication. The data illustrates how the leader's location, the follower's actual location, and the commanded WP relate to each other. Test 1 yielded two laps, Test 2 yielded five laps, and Test 6 yielded six laps. The Off_r , Off_{theta} , and Off_{l1} were set to zero. The three different tests show results obtained at three different frequencies: 1 Hz, 10 Hz, and 20 Hz respectively. For closer examination, the runs in each of the tests have been separated into individual laps.

Test Plan 2 - Test 1 consisted of two complete laps of the box pattern outside the ANT Lab. All of the graphs have been oriented in accordance with the aerial photograph

of the testing area (*see* Figure 21). Looking at the first lap (*see* Figure 26), a box pattern was completed by the leader with very smooth turns; however, the commanded WP displayed jagged transitions and the actual path was deformed. In lap two, the leader slightly overdrove at the top left corner of the graph, but it was still within form. The commanded WP had more extreme jagged distortions and it did a cross-over turn at the bottom left of the graph. The follower's actual path resulted in an oval shape as it tried to adhere to the lopsided commanded WP. The distortion in the commanded WP might be attributed to a combination of the low frequency control of 1 Hz and a poor GPS signal, which may have been caused by the proximity to the building. Also, a large metal enclosure located near the ANT lab may have contributed as the commanded signal pull appears to be in the direction of the structure (*see* Figure 21).

Figure 27 displays the DRMS for the full run and each of the three individual laps. The full run DRMS resulted in 6.56 m, a standard deviation of 2.87 m, and a 5% Tolerance Interval (TI) of 0.01 m as shown on Table 6.

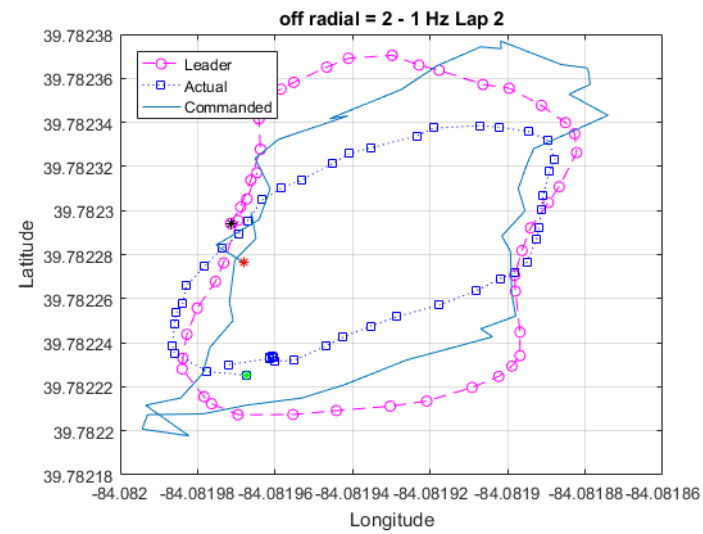
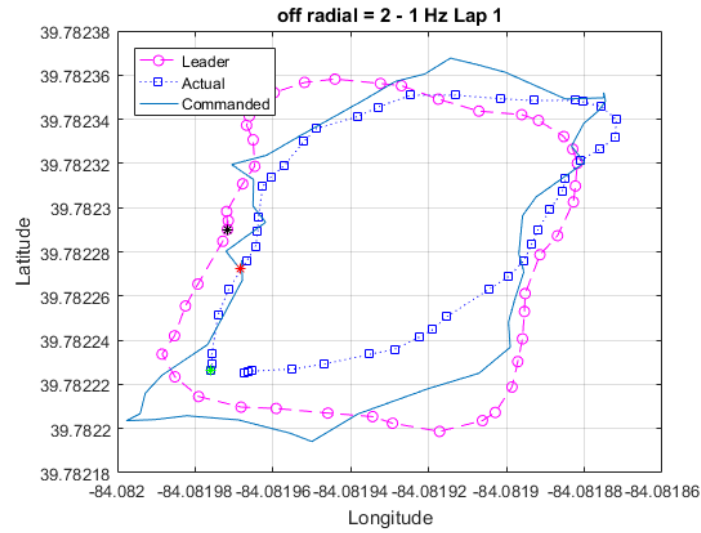
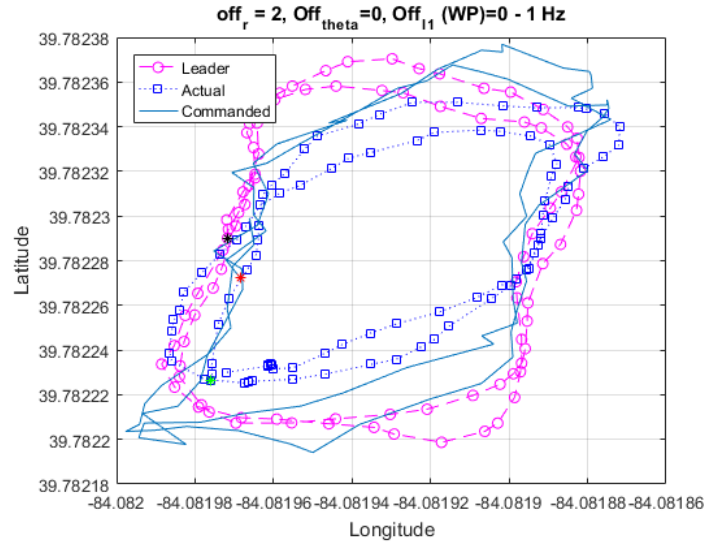


Figure 26: Test Plan 2 Test 1

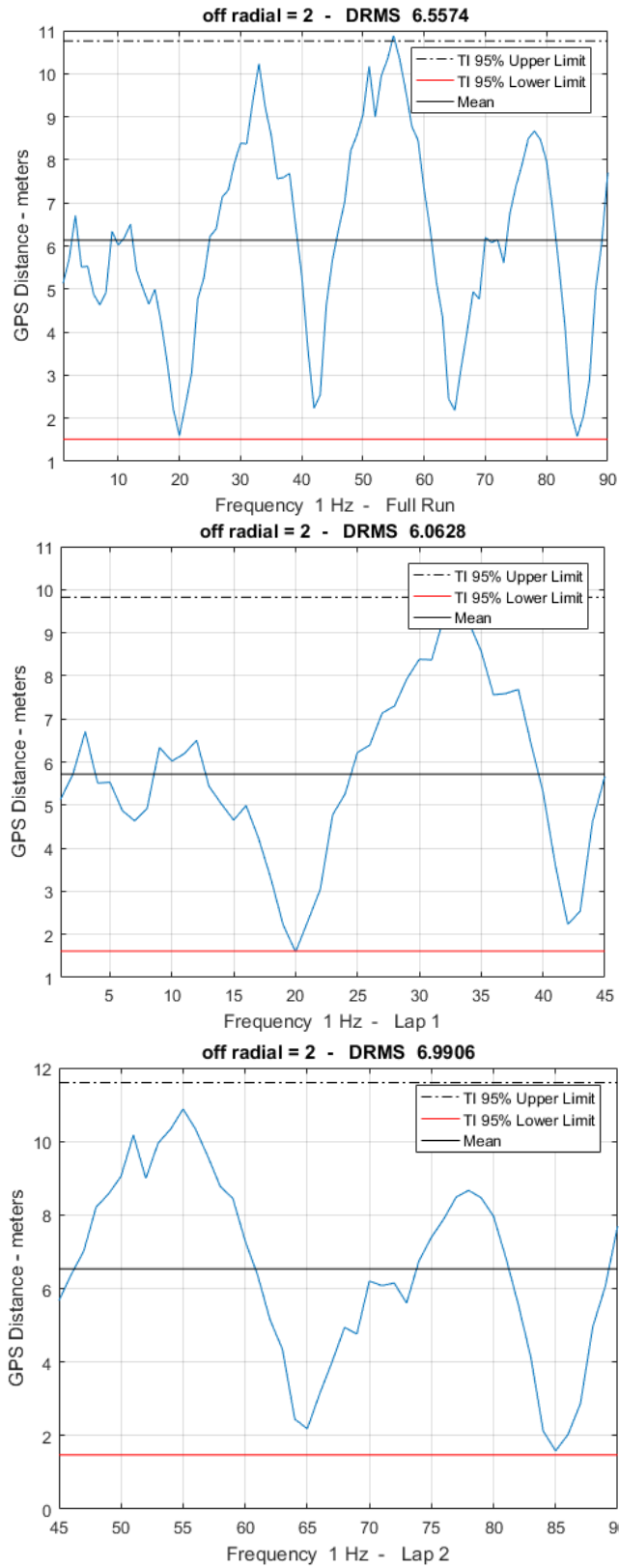


Figure 27: Test Plan 2 Test 1 DRMS Graphs

Table 6: Test Plan 2 Test 1 Results Summary

Test Plan 2-Test 1				
Test	DRMS (m)	STD (m)	95% TI High (m)	5% TI Low (m)
Full Run	6.56	2.87	11.37	1.51
Lap1	6.06	2.04	9.82	1.61
Lap 2	6.99	2.51	11.60	1.47

Test Plan 2 - Test 2 shows a run of five laps. In lap one, the leader was forming a box pattern; however, as can be seen on the bottom left of lap one, the leader strayed to the left but quickly fell back into the box pattern. The commanded WP shot out in a sharp corner (see Figure 28). The follower did not stray and completed a box with rounded corners; however, it did not adhere as closely to the commanded WP as desired. Its path is on the inside of the box even though the setting for the Off_{θ} was zero. Lap two had more of a heart shape than a box. In lap three the leader and the commanded WP have the closest shape to a box in this run, but the follower took a very noticeable shortcut at the bottom of the box. As mentioned, this may be due to poor GPS signal due to close proximity to the building or to the metal enclosure near the testing area also causing poor GPS signal. In laps four and five the leader had a tendency to overrun the corners but it did not cross over like it did on lap 1 while turning (see Figure 29). The follower was able to correct its path and flow back into the pattern.

Figures 30 and 31 illustrate the calculated DRMS. The first graph shows the DRMS for the entire run to be 7.18 m with a standard deviation of 2.48 m while the DRMS for the individual laps went as low as 6.49 m and as high as 7.71 m. The most erratic behavior was on lap 2 where the DRMS peaked at a GPS distance above 14 meters

(see Figure 30). Lap three appears to have the most erratic behavior swinging back and forth between 3 and 10 meters (see Table 7).

The safe distance provided by the TI indicates a 1.84 m distance away from the leader for close formation.

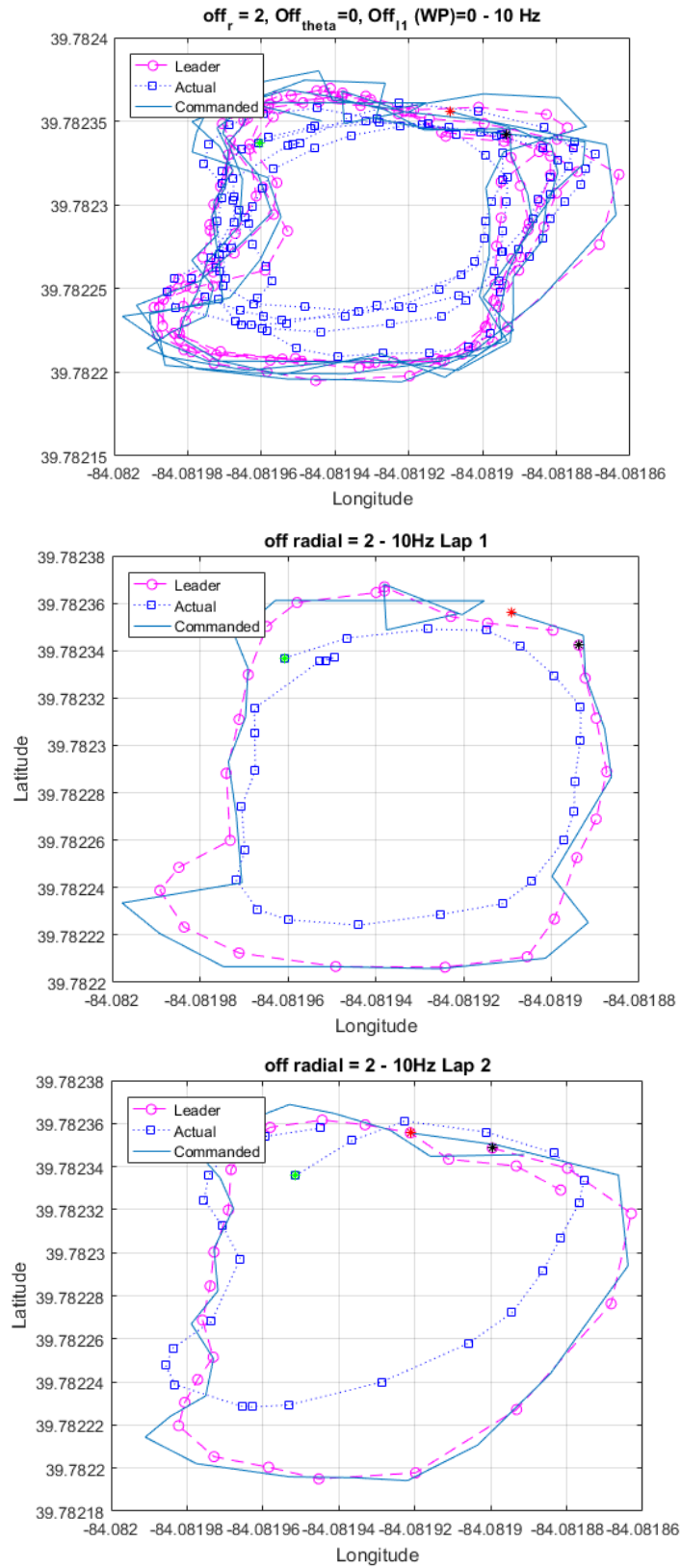


Figure 28: Test Plan 2 Test 2 part a

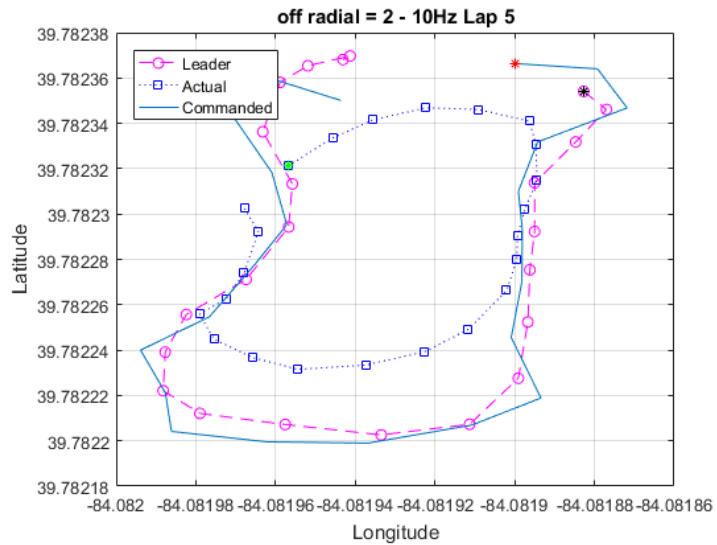
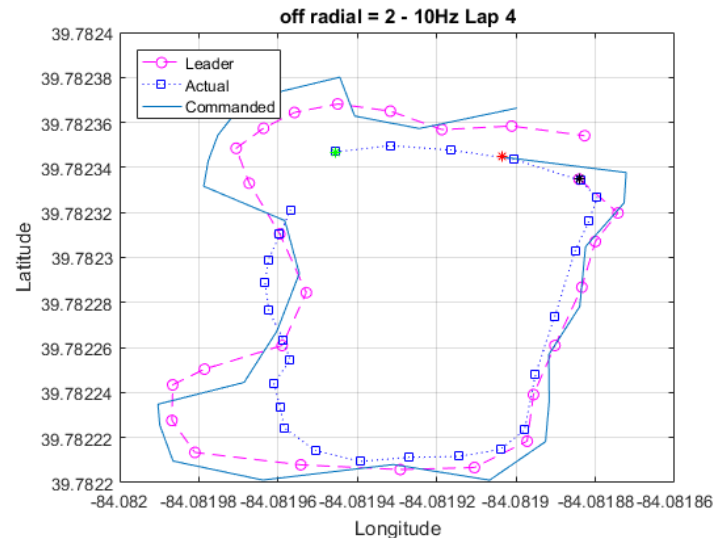
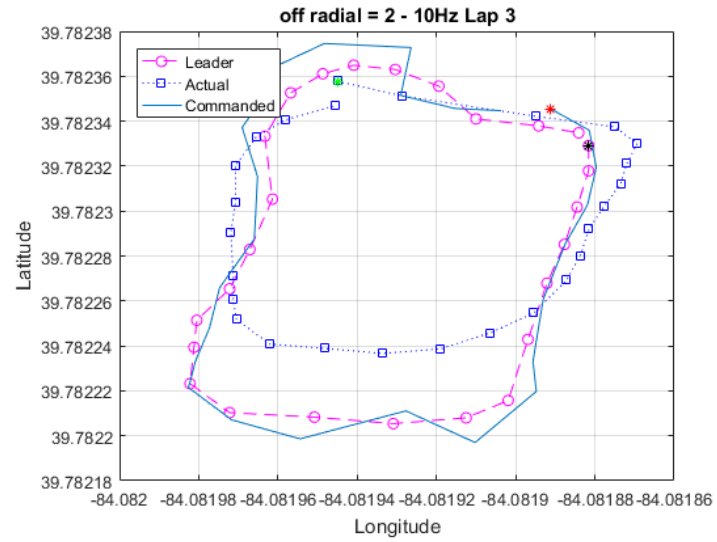


Figure 29: Test Plan 2 Test 2 b

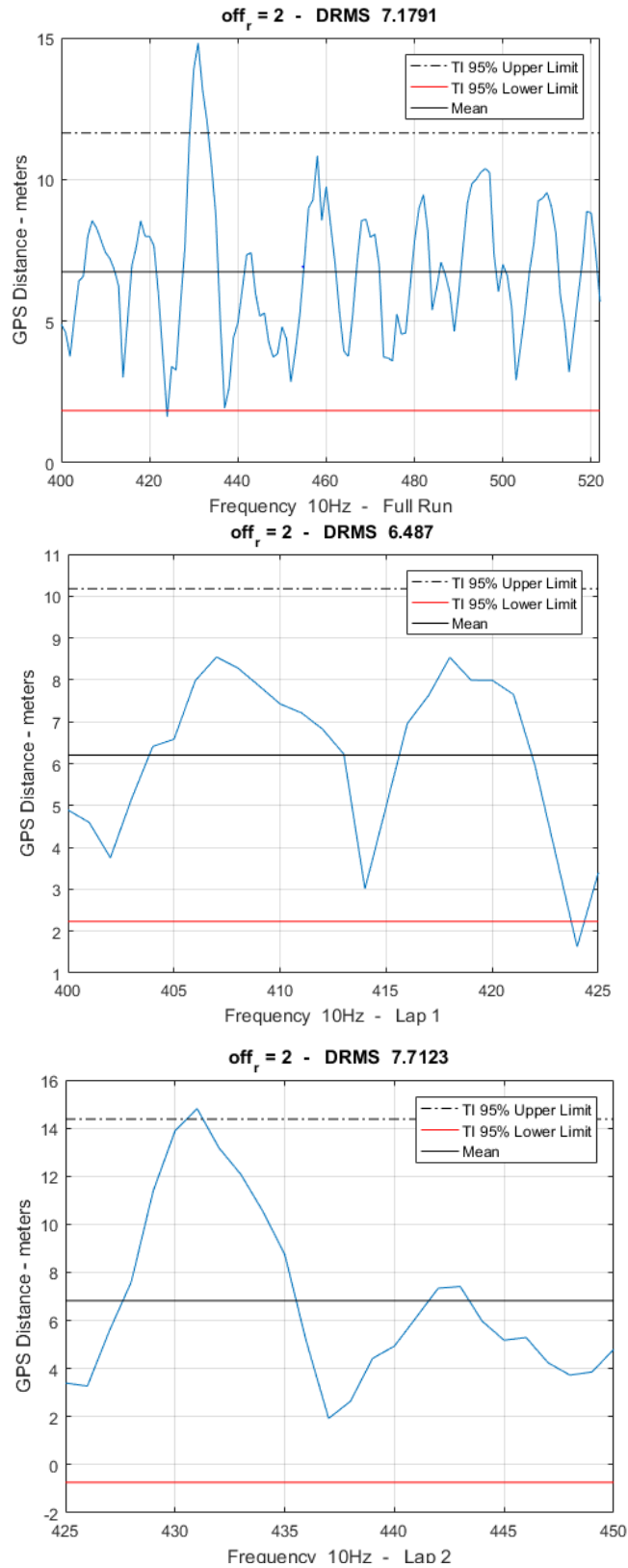


Figure 30: Test Plan 2 Test 2 DRMS Graphs part a

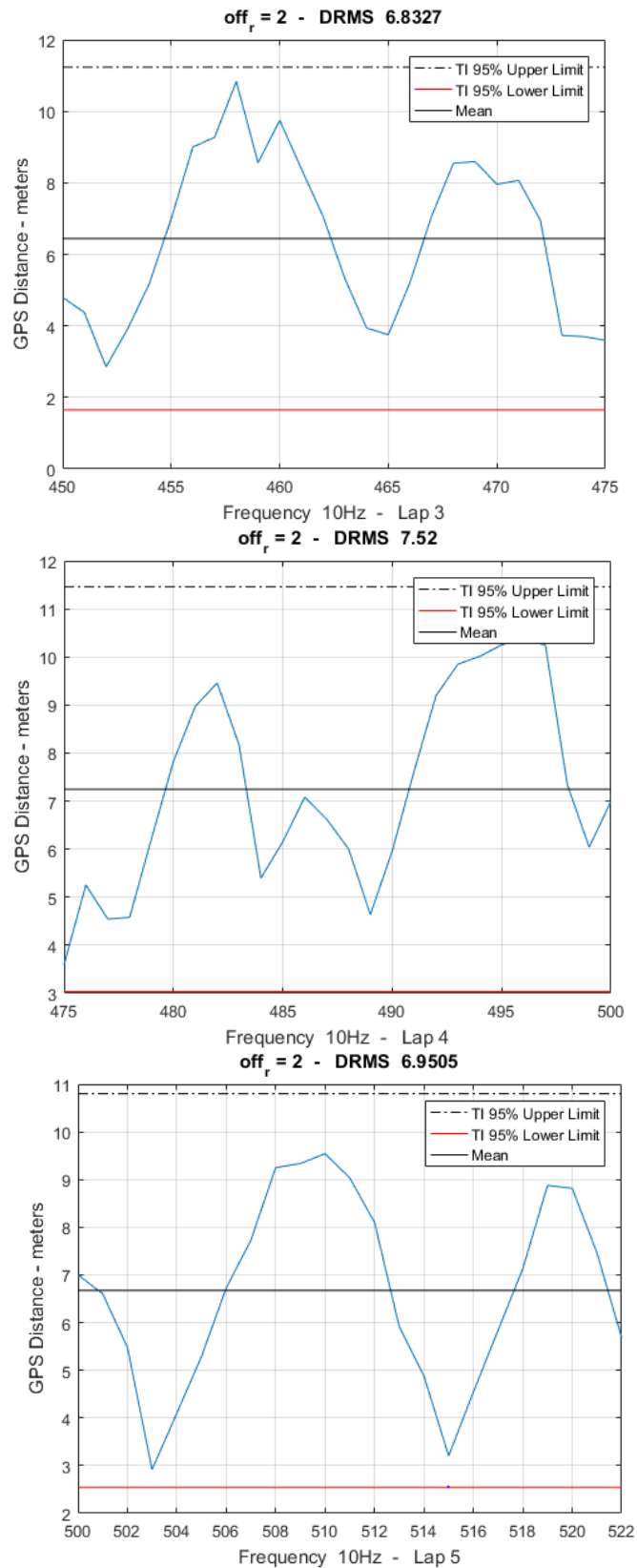


Figure 31: Test Plan 2 Test 2 DRMS Graphs part b

Table 7: Test Plan 2 Test 2 Results Summary

Test Plan 2 - Test 2				
Test	DRMS (m)	STD (m)	95% TI High (m)	5% TI Low (m)
Full Run	7.18	2.48	11.64	1.84
Lap1	6.49	1.93	10.18	2.23
Lap 2	7.71	3.67	14.38	-0.73
Lap 3	6.83	2.33	11.23	1.65
Lap 4	7.52	2.05	11.46	3.04
Lap 5	6.95	1.99	10.80	2.54

Test Plan 2 - Test 6 was slightly longer; it consisted of six laps. The runs have been broken into individual laps. In this example, the follower overshoots on the first two laps then settles into a box pattern following the commanded WP in a tight formation on laps three and four. Notice on these laps that the leader and the commanded WP overlap and the follower is navigating very close to the commanded WP (see Figures 32, 33, & 34). The follower breaks the pattern again on the fifth lap, overshooting on the top right corner of the run. Even though the follower overshoot three of the runs, overall it navigated in close formation. Test 6 was the best run of the three examples in terms the box pattern formation, with the follower adhering closely to the commanded WP. Lap two yielded the smallest DRMS (see Figure 35), the smallest STD, and a Tolerance Interval (TI) as shown in Table 8. Figures 36 and 37 show the DRMS results for the rest of the laps.

The DRMS graphs for example three range from 4.0 m to 5.1 m. For the TI, only the closest, or lowest, tolerance is of interest. Example two had the best DRMS of 4.0 m and provided a 95% confidence that the safe distance for the follower is approximately 2.6 m behind the leader.

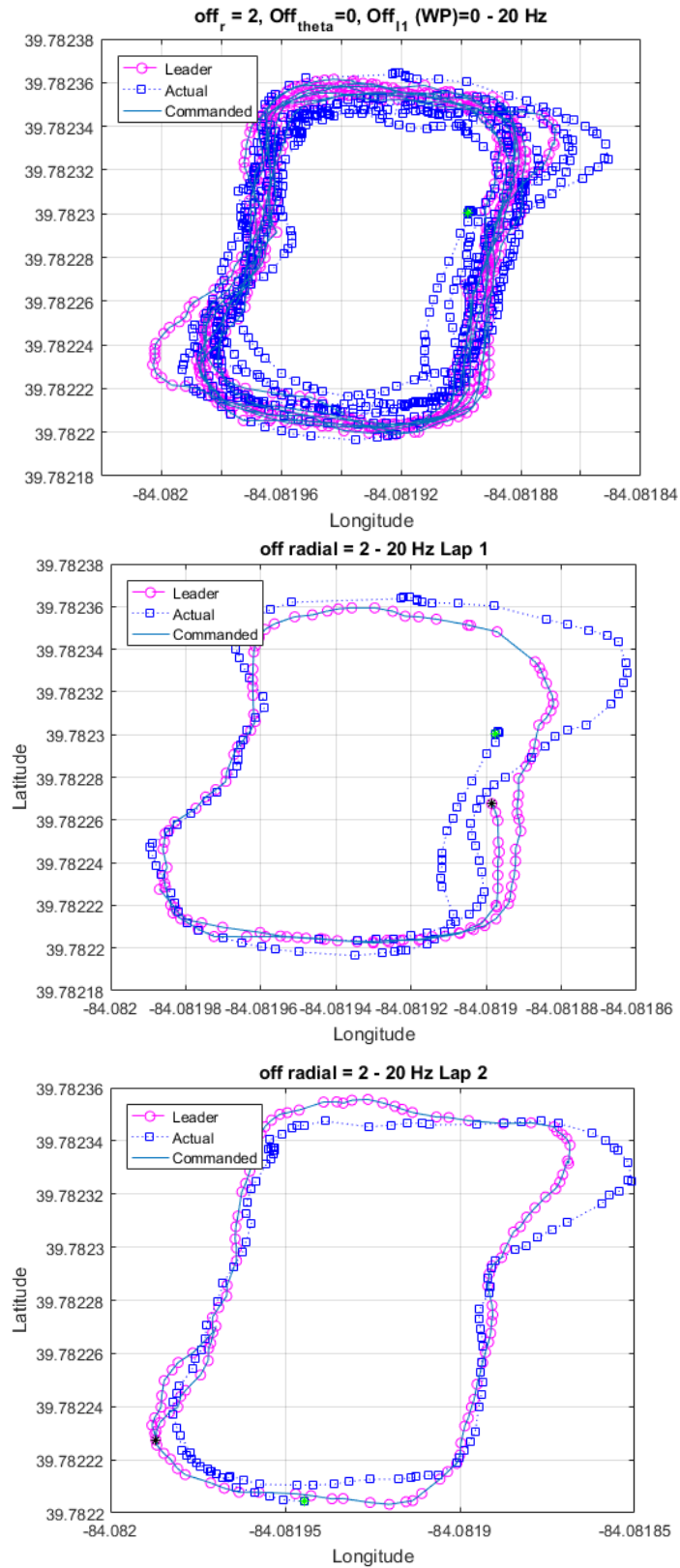


Figure 32: Test Plan 2 Test 6 part a

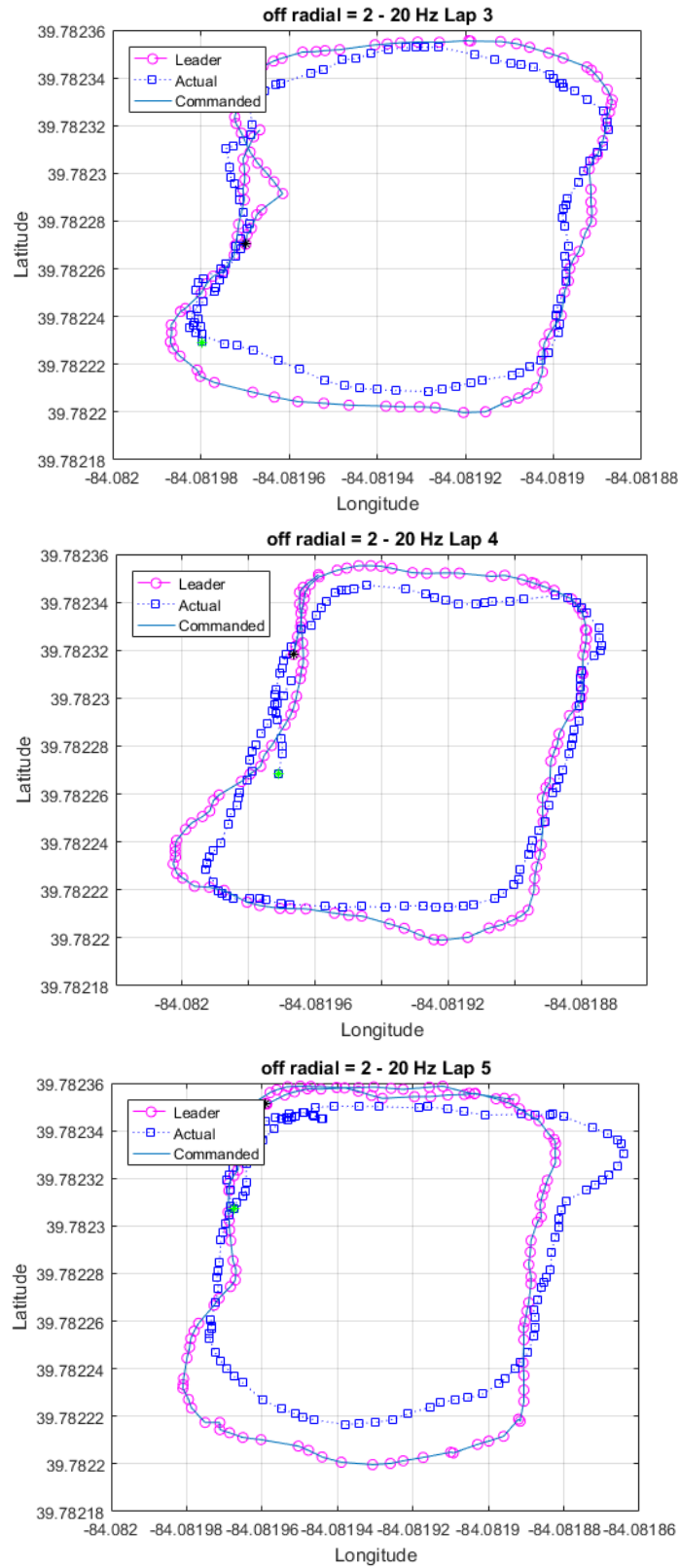


Figure 33: Test Plan 2 Test 6 part b

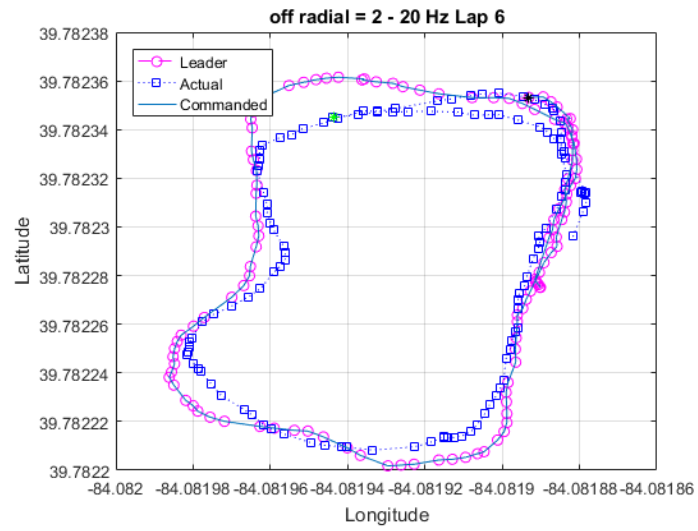


Figure 34: Test Plan 2 Test 6 part c

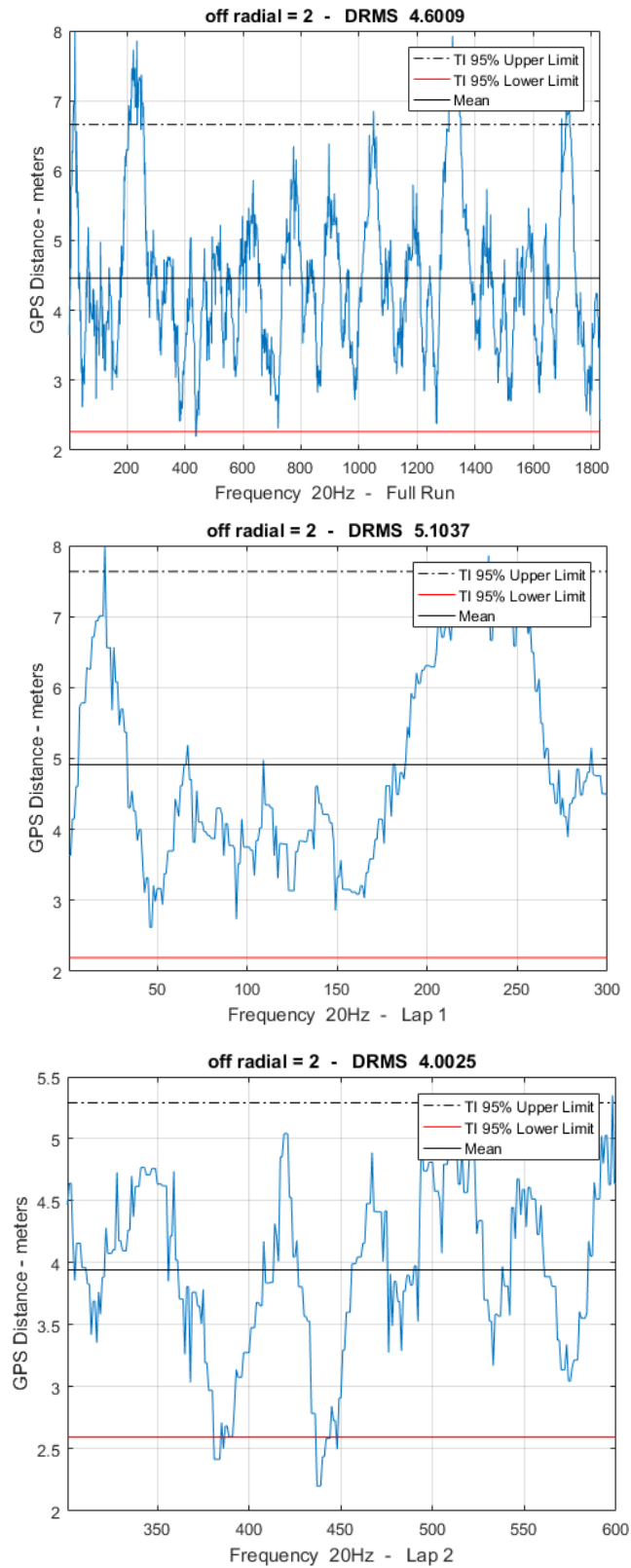


Figure 35: Test Plan 2 Test 6 DRMS Graphs part a

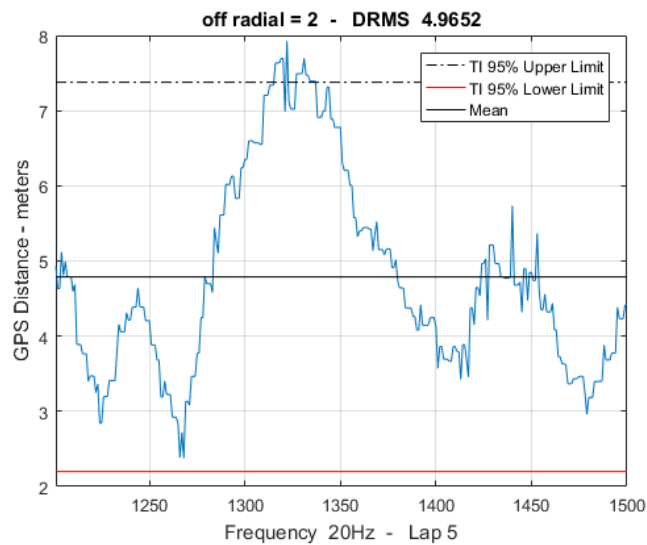
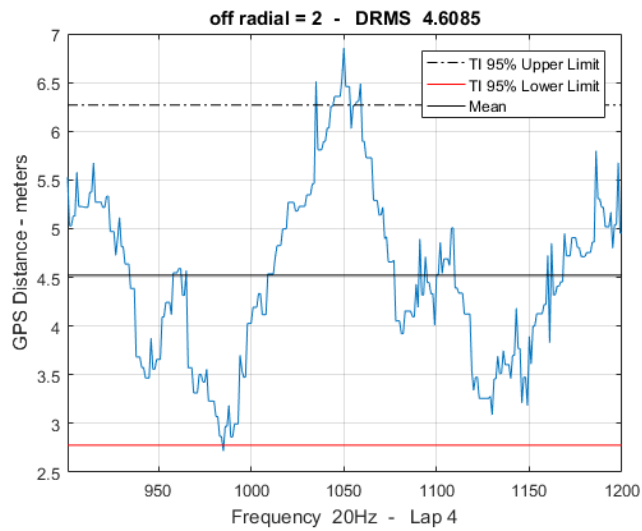
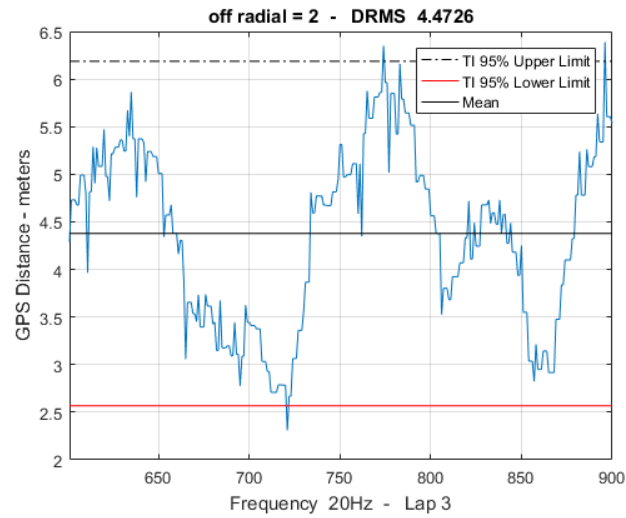


Figure 36: Test Plan 2 Test 6 DRMS Graphs part b

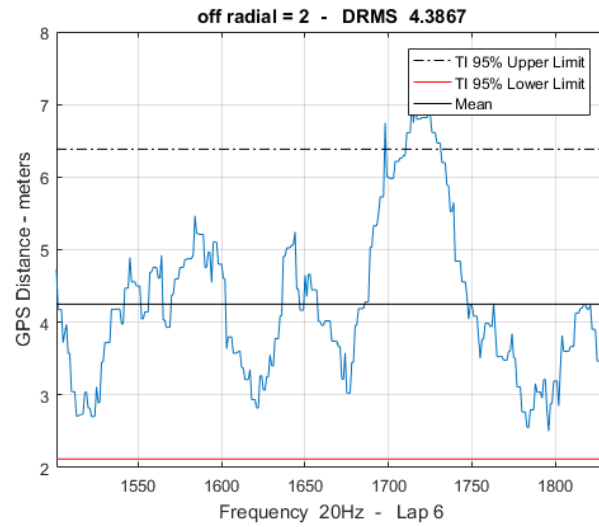


Figure 37: Test Plan 2 Test 6 DRMS Graphs part c

Table 8: Test Plan 2 Test 6 Results Summary

Test Plan 2 - Test 6				
Test	DRMS (m)	STD (m)	95% TI High (m)	5% TI Low (m)
Full Run	4.61	1.12	6.66	2.27
Lap1	5.10	1.38	7.63	2.19
Lap 2	4.00	0.68	5.29	2.60
Lap 3	4.47	0.92	6.19	2.57
Lap 4	4.61	0.89	6.27	2.78
Lap 5	4.97	1.32	7.38	2.20
Lap 6	4.39	1.08	6.38	2.12

Figure 38 compares three frequencies: 1 Hz, 10 Hz, and 20 Hz and the full run DRMS results for each. Comparing the frequencies and the DRMS obtained, it was expected that the 1 Hz would have resulted in a higher DRMS; however, that was not what the data indicated since the 1 Hz frequency is lower than the 10 Hz frequency. Overall, the 20 Hz frequency resulted in smooth runs in which the follower adhered to the

commanded WP in close formation and yielded a DRMS of 4.0 m and a TI of 2.6 m in the second lap.

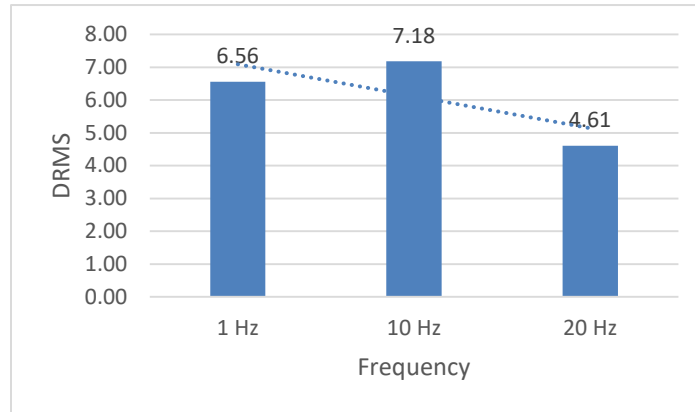


Figure 38: Frequency DRMS Comparison

Comparison with previous UGV research

This section compares the previous application of cooperative communication by Gray (2015) in which he used a UGV to follow a Multi-Rotor UAS. His test is inherently different by virtue of a UGV following a Multi-Rotor vehicle while this project used two UGVs to prototype UAV application of on-board cooperative communication. Also, due to the fact that Gray used a Multi-Rotor vehicle, his testing took place in a paved runway whereas the testing for this project was performed on a grass field. The box pattern used by Gray was 15 x 15 m, the box pattern for this research was 15 x 12 m. In the tests being compared, the followers were both UGVs. Gray's algorithm was used to calculate the commanded WP distance in both cases.

Gray's DRMS were lower than the ones obtained in this research. There are three possible explanations for the difference in DRMS:

- 1) the difference in testing fields: the asphalt runway tends to be level resulting in smooth continuous navigation while an uneven grassy field adds friction and resistance resulting in slower reaction.
- 2) the location of the testing area. This research was conducted immediately outside the ANT lab. The nearness to the building may have caused the GPS signal to be of poor quality and strength, thus causing irregularities when vehicle navigation took place on the side closest to the building.
- 3) the enclosure near the testing area: test 1 exhibits a type of distortion that appears to pull towards the general structure.

The parameters for Gray's UGV following a Multi-Rotor UAS test were set as follows: $Off_r = 2$ m, $Off_{\theta} = 0$, and $L_{11} = 0$. On the upper left corner of Figure 39, the UGV broke out of the pattern and continued to move forward, driving out of formation.

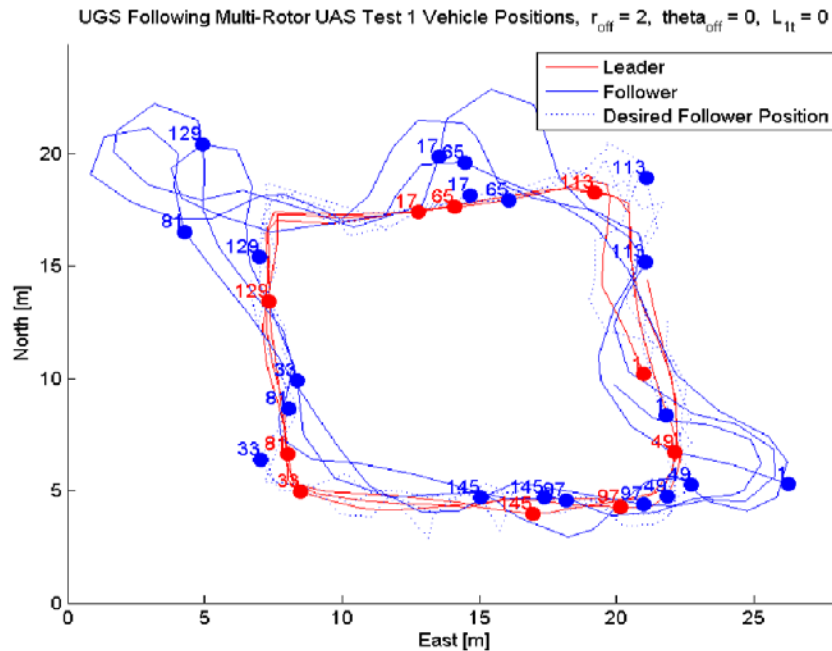


Figure 39: UGV Following Multi-Rotor UAS

The UGV then turned left, making a counter-clockwise circle to get back into a desired position. The top middle of the box path indicates it was having some difficulty correcting its path while attempting to follow the leader. It abandoned the path and made a big arc, cutting the top right corner and overcompensating towards the inside of the box. At the bottom right corner, the follower UGV again drove out of the box, but this time it recovered without making the counter-clockwise circle performed at the top right corner. In his analysis of the results, Gray attributes these aberrations in the pattern to the UGV's failure to reduce its speed enough to be able to handle a turn without over-running it. Another factor he mentions is the way the UGV performs a change in heading direction (Gray, 2015, p. 67). The results he obtained for his test were a DRMS of 3.46 m and a standard deviation of 2.23 m.

The results for this research, which tested two UGVs on-board cooperative communication using a BBB, produced smooth results when tested in a similar box pattern and applying the same settings as in Gray's test: Off_r set = 2 m, $Off_{\theta} = 0$, the $L_{11} = 0$; however, the frequency was set to 20 Hz. On the top right corner, the follower drove out of the pattern for a few meters but made a recovery in a nice continuous flow of movement. There was one slight divergence from the box pattern at the bottom left corner, but it was rather small. The follower again flowed into sync without much problem. Since the leader's path and the commanded WP were the same in this case, the follower's path was right in line with the pattern established by the leader. It does not appear that the follower had trouble rounding the corners, but it did drive out slightly before correcting its path on laps 1, 2, and 5. This test example resulted in a full run DRMS of 4.61 m at a frequency of 20 Hz and a standard deviation of 1.12 m. Figure 40

shows the leader, actual, and commanded WP paths in relation to each other, it also illustrates the smooth roundness turning corners and the nice straight line pattern of the follower navigating in close formation. Due to the limited data collection ability for this project, an Off_{θ} analysis was not feasible. Table 9 shows the test comparison results summary.

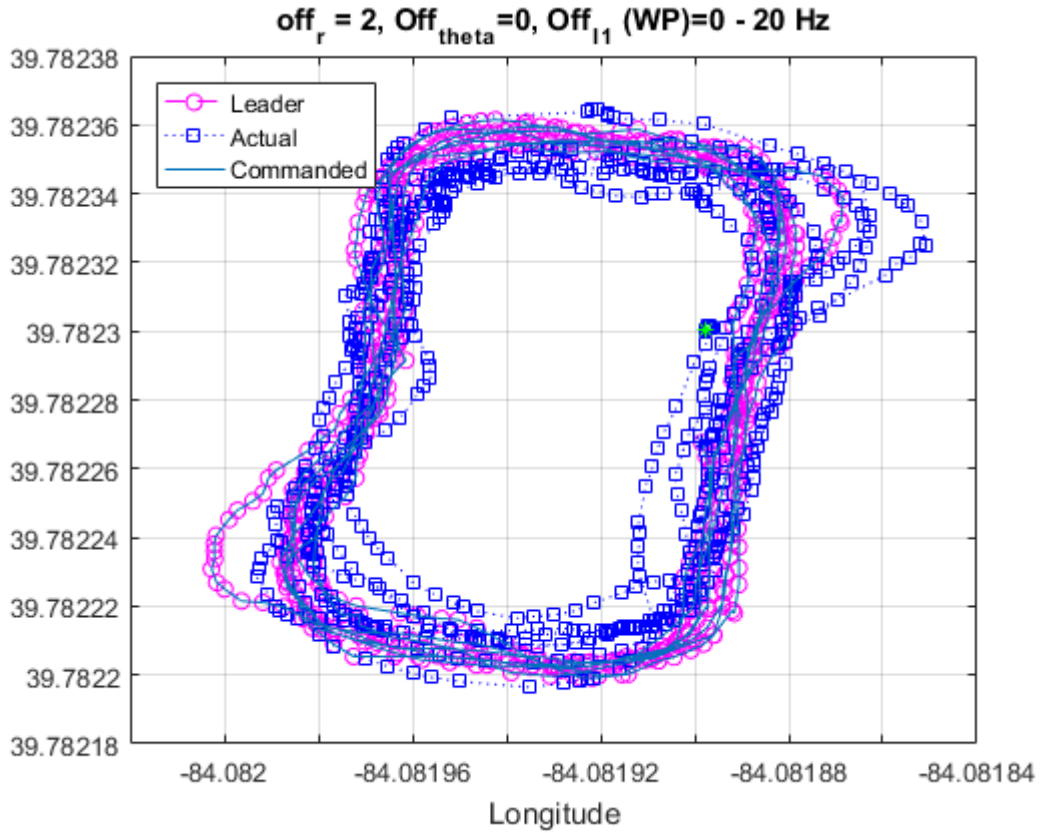


Figure 40: Test Plan 2 Test 6 Comparison

Table 9: Gray - Toscano Comparison

2015 - Gray		2017 - Toscano	
UGV	Multi-rotor	UGV	UGV
Box Pattern	15x15	Box pattern	15x12
Test Field	Asphalt	Test Field	Grass
TPMs		TPMs	
Off _r	2	Off _r	2
Off _{theta}	0	Off _{theta}	0
Off _{l1}	0	Off _{l1}	0
Update rate (Hz)	4 Hz	Update rate (Hz)	1 Hz
	N/A		10 Hz
	N/A		20 Hz

Integration Anomalies

One of the assumptions at the beginning of this research was that MP would be used as the GCS GUI to monitor the activity of the UGVs as they navigated through the different WPs of the mission plan. During the integration of the BBB and DroneKit, it was discovered that MP and DroneKit did not work together as anticipated. DroneKit is a relatively new platform by 3D Robotics; it is still in the development state and is not as robust as their previous version of Drone API. It was mentioned earlier that OSS has an advantage of having a large user community to support it. Such a community contributes to the knowledge base by identifying bugs in the software and, in some cases, suggesting solutions.

In this particular case, DroneKit API has a bug with the way the XML definitions in PyMAVLink, a DroneKit utility, are loaded. The files don't load properly and are not reachable by the code when executed. These definitions are necessary in order for DroneKit to interact with MP. Whenever the leader/follower script was running and a

connection to MP was made, DroneKit would go into an “exception in message handler for HEARTBEAT mode 0 not available on MAVLink definition” error. This error would disappear from the running script as soon as MP was disconnected. According to the online forums, many Dronekit users were experiencing the same error. When the researcher posted a question in the forum, she received an email from a 3D Robotics developer, saying, “OK, congratulations, you’re living on the bleeding edge” (Barker, 2016, n.p.).

The success of correcting this error seemed to be random. Some users reported the suggested “fixes” worked for them, while others reported the fixes didn’t work at all. Part of the problem may be that there were several tools used within the community for apparently the same purpose (such as package-managers, pip, and python setup.py), but they were not consistently interacting properly.

It was decided that MP would only be used to load the ground plan onto the leader autopilot. Attempts to use it as a GUI to monitor leader/follower activity in conjunction with DroneKit were abandoned. As a result, MP was not used to monitor UAV activity and the ability to collect telemetry data was hindered.

Another challenge encountered was in the behavior of the UGVs. The make and model of the UGV vehicles was the same, yet variability in performance persisted. The follower always started at a higher throttle speed than the leader. At the beginning of the field tests, the UGVs had different versions of operating software loaded, the follower having the newer 3.0 version. An attempt was made to enforce configuration management in the vehicle software by reloading the follower UGV with version 2.0 and loading both vehicles with the same list of parameters. Since the leader’s behavior was

desired, the leader's parameters were loaded onto the follower. Unfortunately, the changes did not make any difference in the behavior of the follower. The follower always took off rapidly and ran quickly for a few seconds before establishing a constant speed behind the leader. One of the files logged the follower going from zero to 4.1 m/s in just a couple of seconds. The difference in the speeds was never reconciled.

Summary

In this chapter, procedures, equations, and results of the two test plans introduced in Chapter III were analyzed. Integration anomalies encountered and a comparison of this research with work previously done was delineated. Lastly, answers to the investigative questions introduced in Chapter I were discussed. Chapter V will provide the conclusions and recommendations for future research.

V. Conclusions and Recommendations

Overview

This chapter presents the conclusions drawn from the architecture design and test results. It also provides answers to the investigative questions that were presented at the beginning of this project and concludes with recommendations for future research.

Conclusions of Research

The integration of the BBB with DroneKit has significant application to future UAV utilization. This research has successfully demonstrated that one BBB employed cooperative communication with at least two UGVs; however, the objective is for cooperative communication to be employed with several UGVs and ultimately with multiple UAVs flying in formation.

It has also been demonstrated that more than one UGV can be controlled using only one software script where previously it was not possible to do so. As a result of this research, multiple UGVs can engage in cooperative communication using a single GCS, thus reducing the amount of equipment necessary to control various vehicles.

Tests performed demonstrated successful inter-vehicle communications and on-board processing of telemetry data using the COTS components: the BBB processor, the Pixhawk autopilot, and the OSS DroneKit Python.

Investigative Questions Answered

1. What is the architecture that supports on board processors and inter-vehicle communication?

This research has shown that the three COTS components chosen, BBB, Pixhawk, and DroneKit-Python, have successfully been integrated to provide inter-vehicle communication on-board UGVs to navigate in close formation. Architecture design using these three items streamlined the amount of equipment and the software algorithms necessary to deploy at least two UGVs. More than two UGVs could be easily added to the configuration design implemented in this research. There were three main architecture hardware components: the leader, the follower, and the GCS. Use of the Wave Relay enabled the network configuration connectivity. The software configuration made use of DroneKit Python as the key for algorithm development. Due to its capability to control several autopilots from within one script, the software was also streamlined into only one algorithm to successfully support inter-vehicle communication between the leader and the follower UGVs.

Future testing possibilities could include a flock of UGVs on the ground in formation. The ANT center has availability of a number of UGVs that can be equipped with autopilots and BBBs. One scenario would be to form a flock of UGVs with one BBB controlling the flock. Another scenario would be to have a flock of UGVs all equipped with BBBs and have each of the members of the flock interchange formation leadership, interchanging not just position in the flock but leadership roles.

2. What is the Distance Root Mean Square (DRMS) between the follower's navigation path and the commanded WP?

This question was answered with a series of tests performed in Test Plan 2. The tests generated both good and bad data. Three of good were shown as examples in the

results chapter of this document. The best results obtained indicated a DRMS of 4.0 m at 20 Hz (see Figure 41) and a standard deviation of 0.68 m.

3. What is the expected safe distance between UGVs using this architecture?

The 95% TI provides an upper and a lower tolerance. The lower tolerance affords the

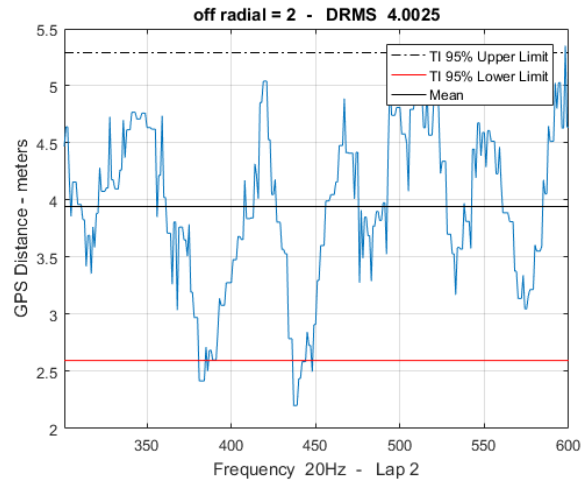


Figure 41: Test Plan 2 Test 6 Lap 2 DRMS

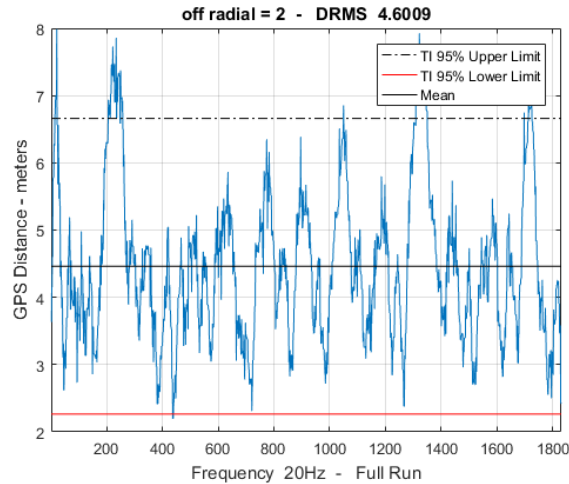


Figure 42: Test Plan 2 Test 6 FULL Run DRMS

confidence that a UGV can safely navigate 2.3 m behind the leader (see Figure 42).

There may be a relationship between distance and error, further investigation would need to be performed in order to analyze what that relationship is and how it affected the results.

Study Limitations

This project was limited by the inability to use MP. As a general mode of operation, MP records a telemetry log (tlog) file that contains a set of over 30 parameters that are available for data analysis and navigation playback. Since DroneKit is not at this time compatible with the operation of MP, use of the tlog file was not an option.

Recommendations for Future Research

Further research to investigate if there exists a DroneKit function to record the time telemetry data is sent from one vehicle and the time it is received by the other vehicle in order take accurate timing measurements to perform a quantitative analysis is recommended. If a function is not available, development of a software script that can perform this task could advance any follow-on work in this area.

As a result of the inability to use MP in conjunction with DroneKit, it is highly recommended that future research be conducted to find out if the anomalies with the XML definitions in PyMAVLink have been resolved in order to implement MP in future UAV cooperative communication research. It cannot be stressed enough that having a tool like MP to be used as part of a GCS is desirable. The telemetry data that it logs is an enormous contribution to data analysis.

Further study of the differences in UGV performance is recommended in order to fine-tune the quantitative measurement of lagging latency response in the system. The

results obtained were not of high quality, nor were they dependable due to the fact that the follower was so much faster at the beginning of the tests and it took it a period of time before it settled down to an expected, normal speed. Figuring out why the vehicles behaved differently despite the fact that they were both of the same make and model, had the same parameters list loaded, and had the same version of software would enable a reasonable comparison in the analyses. However, since the third phase is to fly this architecture, a similar performance of two or more UAVs should be examined.

While this project prototyped cooperative communication using UGVs, the ultimate purpose was to implement cooperative communication between two UAVs. The UGVs used for this project were not designed with precision work in mind, and the type of research needed for this project surpasses their ability; therefore it is not recommended to spend a lot of time characterizing the vehicles. Effort needs to be made to accomplish functionality in UGVs and the bulk of the effort should be spent developing and improving cooperative communication and on-board processing.

All of the IP addresses were hard-coded in. Further research could design plug-and-play modularity to work without manual reconfiguration. Also, performance variables could be changed only before the run started. Once the vehicles started the run, the only way to change variables was to bring the UGVs to a complete stop, change variables and start again. Design of a software module that enables the operator to change variables during a run would be advantageous.

This research focused on controlling two UGVs with one BBB. To take this concept further, three or more vehicles could be tested to examine how many vehicles can be successfully be controlled with one BBB. A variation of this same concept can be

explored with DroneKit Python. This research has proven that at least two UGVs can be successfully controlled within one DroneKit Python script; however; it is not known if there is a limit and what it might be.

Velocity, wind, atmosphere, time delays, GPS signal, and vehicle are factors that affect the distribution of the follower position about the commanded waypoint. The influence of these factors, and their relative importance, needs to be studied in the future.

Significance of Research

The successful integration of COTS components like the BBB, Pixhawk, and DroneKit for cooperative communication and on-board processing has far-reaching implications for military operation of UAVs. Several outcomes have been delineated. The successful utilization of the BBB has added the capability of increased data processing power. This additional processing power enabled the development of intelligence implementing a higher level of autonomy and self-governance in UGVs. In conjunction with the BBB and DroneKit Python, the capability of controlling more than one UAV within a single software algorithm was integrated. Implementation of the BBB has added the capacity to control several Pixhawk autopilots to effectively reduce the footprint of equipment crucial to fly UAVs in flocking or swarming formations. The result is a reduction in personnel, and a higher level of autonomy in the deployment UAVs.

Appendix A: Manufactured Cables

Pixhawk to USB-TCP232-24 Cable pinout			
Pixhawk	Nomenclature	UART	Nomenclature
1	VCC	7	VDD
2	TX	2	RX
3	RX	3	TX
6	GND	5	GND



Figure 43: Pixhawk to USB-TCP232-24 Manufactured Cable

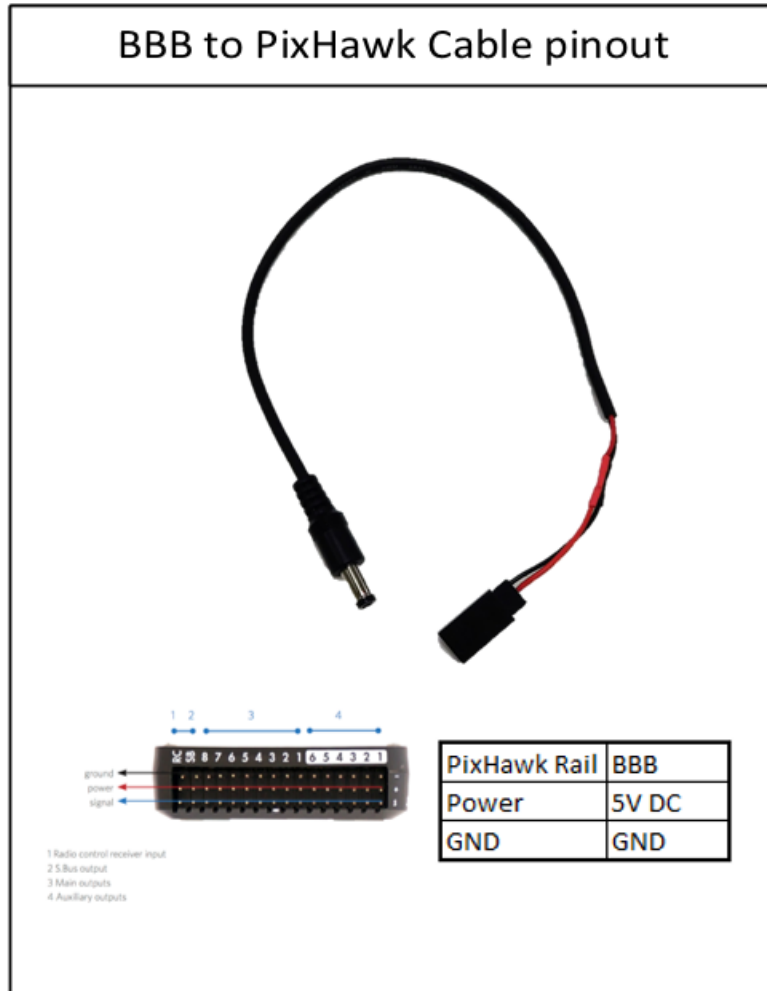


Figure 44: BBB to PixHawk Manufactured Cable

Appendix B: Use Case Specification

Use-Case Specification: Employ Cooperative Communication

Version 1.0

Revision History

Date	Version	Description	Author
12 DEC 16	1.0	Employ Cooperative Communication use case	Lidia Toscano
24 JAN 17	1.1	Updated document with latest details	Lidia Toscano
1 FEB 17	1.2	Corrected numbering and spacing	Lidia Toscano

Use-Case Specification: Employ Cooperative Communication

1. Employ Cooperative Communication

1.1 Brief Description

UAVs Operator wants to use two or more UAVs to fly through a basic set of waypoints using a formation flight pattern. An UAV is designated as the leader and a second UAV is designated the follower. Follower is configured with a BeagleBone Black microprocessor.

2. Flow of Events

2.1 Basic Flow – Employ Cooperative Communication

1. UAVs Operator pre-loads basic square mission pattern on the designated leader UAV's autopilot using Mission Planner.
2. UAV Operator loads mission parameters into the software algorithm at the ground control station.
3. UAV Operator loads software algorithm onto BeagleBone Black microprocessor residing on the follower UAV.
4. Safety pilots are standing by with Radio Controls (RCs) in manual mode
5. UAV Operator starts software algorithm to initiate cooperative communication between the leader and the follower.
 - a. Software algorithm
 - i. opens a telemetry file
 - ii. initializes leader and follower
 - iii. monitors safety pilots' RC manual/auto switches
 - iv. sets leader and follower to "GUIDED" mode
 - v. monitors leader and follower locations
 - vi. sends follower
 1. radial distance away from the follower
 2. angle away from the -x axis from the tail of the leader
 3. commanded location (way point)
 4. frequency control (update rate in Hz)
 - vii. appends telemetry data to file
6. Safety pilots flip MANUAL switch to AUTO as soon as they receive the command from the UAV Operator at the ground control station.
7. Mission is terminated when the UAV Operator determines mission is complete or when one or both of the safety pilots determine(s) it is appropriate to do so.
 - a. UAV Operator performs a keyboard interruption
 - b. Algorithm prints "Unexpected error" on GCS
 - c. Algorithm closes leader and follower objects
 - d. Algorithm is terminated

2.2 Alternative Flows

1. UAV Operator aborts the mission
 - a. UAV Operator enters a keyboard interruption
 - b. Algorithm prints “Unexpected error”
 - c. Leader and follower objects closed
 - d. Exit program
2. Safety Pilot aborts the mission
 - a. One or more of the safety pilots flips the switch from AUTO to MANUAL
 - b. Algorithm prints “mode set to MANUAL”
 - c. Leader and follower objects closed
 - d. Exit program

3. Special Requirements

3.1 BeagleBone Black

1. BBB must be pre-loaded with all open-source software necessary to run software algorithm (i.e. Python, DroneKit-Python, MAVLink MavProxy and all necessary modules)
2. BBB must be setup to work using an Ethernet format

4. Preconditions

4.1 Mission

1. UAV Operator has a mission to fly
2. Mission is pre-loaded on the leader’s autopilot
3. UAVs are available and functioning to complete mission
4. Space is available and secure to navigate
5. Safety pilots are available to control space

4.2 UAVs

1. UAV Operator has access to two or more UAVs
2. At least one of the UAVs has been configured with a BeagleBone Black microprocessor
3. Software algorithm has been configured to communicate with all UAVs involved in the mission

4.3 WaveRelay

1. In this project the WaveRelay is available to enable communication between UAVs, GCS, and the BBB

5. Post conditions

5.1 UAV Operator

1. UAV Operator has terminated the mission
2. UAV Operator is satisfied with flight formation during the mission
3. UAV Operator has collected satisfactory sample data

5.2 Mission

1. UAVs performed mission pattern correctly
2. UAVs stayed within boundaries of mission
3. UAV Operator has collected satisfactory sample data|

Appendix C: Leader-Follower.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# Lidia Toscano
# AFIT Winter quarter 2017

"""
Adding portions of Jeremy Gray's code
This is Follower Leader Code in RC or running preloaded waypoints -

Inputs specific lat, lon, alt values to test follower receipt of telemetry
"""

from dronekit import connect, VehicleMode, LocationGlobalRelative, LocationGlobal
from dronekit import sys, Command
import numpy as np
import math
import time
from datetime import datetime
from pymavlink import mavutil
from LLA_ECEF_Convert import LLA_ECEF_Convert
from multi_vehicle_toolbox_v3 import follower_pos

'''INIT PARAMS'''
freq_control=10.0      #frequency of control loop, how often to get leader telemetry,
                        #must be float (0.0)
freq_store=freq_control #frequency of data storage, must be float (0.0)
freq_print=2.0         #frequency of printed updates, must be float (0.0)

off_r = 0              #radial distance away from leader [m]meters
off_theta = 0          #angle (deg) from -x axis (out of tail), CCW is (+) rotation degrees
off_ll = 0             #second or meters prediction (in front of guided point)

print ' '
print '***** TEST 1
*****'
print 'Script to test cooperative communication between two rovers'
print 'Using the BBB to run a DroneKit Python Algorithm
print 'To QUIT press Control C'
print
print '*****
*'
print ' '

```

```
'''DATA FILE INIT'''
```

```
timestr = time.strftime("%m-%d-%Y_%H-%M-%S") #date-time for file name
```

```
file_name='leader_follower_tel_' + timestr #file name appended with date time
```

```
data_file = open(file_name, 'a') #create txt doc to append to
```

```
print 'Telemetry file open'
```

```
print''
```

```
'''LEADER CONNECT INIT'''
```

```
#Connect to the leader
```

```
print 'Connecting Leader' #TCP 232 T-24 IP 192.168.1.13 through 14551
```

```
leader_connection_string = '0.0.0.0:14551'
```

```
leader = connect(leader_connection_string, wait_ready=True)
```

```
LIP='0.0.0.0'
```

```
LPort=14551
```

```
leader_address=(LIP,LPort)
```

```
print leader_address
```

```
print ' '
```

```
'''FOLLOWER CONNECT INIT'''
```

```
#Connect to the follower commented out for bench testing
```

```
print 'Connecting Follower' #TCP 232 T-24 IP 192.168.1.11 through 14550
```

```
follower_connection_string = '0.0.0.0:14550'
```

```
follower = connect(follower_connection_string, wait_ready=True)
```

```
FIP='0.0.0.0'
```

```
FPort=14550
```

```
follower_address=(FIP,FPort)
```

```
print follower_address
```

```
print ' '
```

```
# Set LEADER mode to MANUAL to start mission
```

```
leader.mode = VehicleMode("GUIDED")
```

```
print 'Leader should already be in manual(RC) or auto mode'
```

```
# Set FOLLOWER mode to GUIDED to start mission
```

```
# Lidia, what happens when you put follower in GUIDED mode without setting a  
waypoint/loiter point???
```

```
#follower.mode = VehicleMode("GUIDED")
```

```
follower.mode = VehicleMode("MANUAL") #PREFER in MANUAL!!!!
```

```
print 'Follower in STABILIZE mode'
```

```
rc_ch=follower.channels #Get Channel 5 Position (Changed)
```

```
t_write=0 #forces first write to occur on start
```

```
t_print=0 #forces first print to occur on start
```

```
while True:
```

```

try:
    # get time for data file ...
    t1 = time.time()

    #The > 1400 came from looking at what the RC radio's numbers were in Mission
    Planner

    if rc_ch['8'] > 1400: # MANUAL MODE FAIL SAFE, will not store data
        follower.mode = VehicleMode("MANUAL") #PREFER in MANUAL!!!!
        print 'Channel 8: %s' %rc_ch['8']
        if time.time() - t_print > 1 / freq_print:
            print "Follower Mode Set to MANUAL " + str(datetime.now().time())
            t_print = time.time()
            time.sleep(0.5)

    else:
        # get LEADER telemetry information for the log file and to make follower Loiter
        point
        lat = float(leader.location.global_relative_frame.lat) # latitude (deg)
        lon = float(leader.location.global_relative_frame.lon) # longitude (deg)
        alt_asl = float(leader.location.global_relative_frame.alt) # altitude above sea
        level (m)
        p = float(np.deg2rad(leader.attitude.pitch)) # pitch (rad) of leader
        relative to NEU frame
        r = float(np.deg2rad(leader.attitude.roll)) # roll (rad) of leader relative
        to NEU frame
        y = float(np.deg2rad(leader.attitude.yaw)) # yaw (rad) of leader
        relative to NEU frame
        v_b = leader.velocity # velocity vector (m/s) relative to
        body
        t_tel = time.time() # time telemetry was received, time is
        in epoch & Unix timestamp

        # upload data to leader
        leader.commands.upload() # commands are sent to the leader only after calling:
        upload()

        # determine gc relative to leader frame (NED)
        v_b = np.array([v_b[0], v_b[1], v_b[2]])
        c_p = np.cos(p);
        s_p = np.sin(p)
        c_r = np.cos(r);
        s_r = np.sin(r)
        c_y = np.cos(y);
        s_y = np.sin(y)
        R_v_b = np.array([c_p * c_y, c_p * s_y, -s_p],

```

```

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

# if V is too slow use yaw (rad) as ground course
if v < 0.25: # Commented out, not sure it is needed on rovers
    gc = y

print 'Leader heading = ' + str(leader.heading)

gc = np.rad2deg(gc) # Keeping units the same
print 'Leader ground course = ' + str(gc)

# get FOLLOWER telemetry information for the log file only
f_lat = float(follower.location.global_relative_frame.lat) # follower latitude
(deg)
f_lon = float(follower.location.global_relative_frame.lon) # follower longitude
(deg)
f_alt_asl = float(follower.location.global_relative_frame.alt) # follower altitude
above sea level (m)
fp = float(np.deg2rad(follower.attitude.pitch)) # pitch (rad) of leader
relative to NEU frame
fr = float(np.deg2rad(follower.attitude.roll)) # roll (rad) of leader
relative to NEU frame
fy = float(np.deg2rad(follower.attitude.yaw)) # yaw (rad) of leader
relative to NEU frame
fv_b = follower.velocity # velocity vector (m/s) relative
to body
ft_tel = time.time() # time telemetry was recieved

# upload data to follower
follower.commands.upload() # commands are sent to the follower only after
calling: upload()

# determine gc relative to follower frame (NED)
fv_b = np.array([[fv_b[0], fv_b[1], fv_b[2]]])
fc_p = np.cos(p);
fs_p = np.sin(p)
fc_r = np.cos(r);
fs_r = np.sin(r)
fc_y = np.cos(y);
fs_y = np.sin(y)

```

```

    fR_v_b = np.array([[fc_p * fc_y, fc_p * fs_y, -fs_p],
                       [fs_r * fs_p * fc_y - fc_r * fs_y, fs_r * fs_p * fs_y + fc_r * fc_y, fs_r *
fc_p],
                       [fc_r * fs_p * fc_y + fs_r * fs_y, fc_r * fs_p * fs_y - fs_r * fc_y, fc_r *
fc_p]])

    # rotation transform from vehicle to body
    fv_v = np.dot(fR_v_b.T, fv_b.T) # velocity vector relative to vehicle (NED)
frame
    fgc = np.arctan2(fv_v[1], fv_v[0]) # ground course (rad) relative to North ...
    fv = np.linalg.norm(fv_v) # velocity of leader, used to calc L1
    print 'Follower ground course =' + str(fgc)

    # if V is too slow use yaw (rad) as ground course
    if fv < 0.25: # fv < 0.25
        fgc = fy # fgc = fy

    fgc = np.rad2deg(fgc)

    #Call Jeremy's calculation of safe distance to follow
    # Convert off_l1 from time to distance
    off_l1 = off_l1 * v
    leader_location = [lat, lon, alt_asl]
    follower_loiterpoint = follower_pos(off_r, off_theta, 3, leader_location, gc)
    point1 = [follower_loiterpoint[0], follower_loiterpoint[1] - 360.0]

    cmds = follower.commands
    #print''
    #print'Testing GUIDED mode code'
    #print''

    print''
    point1 = LocationGlobalRelative(follower_loiterpoint[0], follower_loiterpoint[1]-
360.0, alt_asl)
    follower.simple_goto(point1)
    follower.mode = VehicleMode("GUIDED")

    # build telemetry msg to be a known length (msg_size)
    tel_msg_raw = '%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t' %
(str(lat), str(lon), str(alt_asl), str(float(gc)), str(v),
    str(f_lat), str(f_lon), str(f_alt_asl), str(float(fgc)), str(fv) ) # build msg for leader
and follower

    msg_WP = '%s %s' % (str(follower_loiterpoint[0]), str(follower_loiterpoint[1]-
360)) # build msg for followerwp

```

```

print ''
print 'Leader location' '' + str(lat) + '' + str(lon) + '' + str(alt_asl)
print 'Follower location' '' + str(f_lat) + '' + str(f_lon) + '' + str(f_alt_asl)
print 'The follower commanded location ' + '\n' + str(point1)
print 'msg wp ' + str(msg_WP)

# append data w/ unix time on new line of data txt file, if 1/freq_store has passed
if time.time() - t_write > 1 / freq_store:
    msg_data = '%s\t %s\t %s' % (t_tel, tel_msg_raw, msg_WP)
    data_file.write(msg_data + '\n')
    t_write = time.time()

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

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

except KeyboardInterrupt: #This is the fail-safe to be able to get out of the program
    print "Unexpected error"
    data_file.close()
    break

print 'Ending back in MANUAL'
follower.mode = VehicleMode("MANUAL") #PREFER in MANUAL!!!!

#Close leader object before exiting script
print "Close Leader object"
leader.close()

#Close follower object before exiting script
print "Close Follower object"
follower.close()

```

Appendix D: IP Configuration Table

Table 10: IP Configuration

Network Connections					
	Switch	Laptop	USR-TCP232 (1)	USR-TCP232 (2)	BBB
			UDP Mode	UDP Mode	
IP address	192.168.1.1	192.168.1.2	192.168.1.11	192.168.1.13	192.168.1.12
Checksum	255.255.255.0	255.255.255.0	255.255.255.0	255.255.255.0	255.255.255.0
Gateway	192.168.0.1	192.168.0.201	192.168.1.1	192.168.1.1	192.168.1.1
Ethernet		195.168.1.99			
Baud rate		115200	57600	57600	115200
Destination IP			192.168.1.12	192.168.1.12	
Remote port			14550	14551	
Module Port			20101	20201	
	Wave Relay				
Truck 1 IP address	192.168.1.10				
Truck 2 IP address	192.168.1.20				
Laptop IP address	192.168.1.60				

Appendix E: Basic CodeVerify.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# Lidia Toscano
# AFIT Winter quarter 2017

from dronekit import connect, VehicleMode, LocationGlobalRelative, LocationGlobal
from dronekit import sys, Command
import numpy as np
import math
import time
from datetime import datetime
from pymavlink import mavutil
from LLA_ECEF_Convert import LLA_ECEF_Convert

'''INIT PARAMS'''
freq_control=10.0    # frequency of control loop, how often to get leader telemetry, must
                      # be float (0.0)
freq_store=2.0       # frequency of data storage, must be float (0.0)
freq_print=2.0       # frequency of printed updates, must be float (0.0)

off_r = 3            # radial distance away from leader [m]meters
off_theta = 0        # angle (deg) from -x axis (out of tail), CCW is (+) rotation degrees
off_l1 = 0           # second or meters prediction (in front of guided point)

print ' '
print '*****'
print 'This software tests the functionality of the '
print 'follower_pos(off_r,off_theta,off_l1,loc0_l,heading_l) '
print '*****'
print ' '

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

''' Jeremy's definition '''
def follower_pos(off_r,off_theta,off_l1,loc0_l,heading_l):
    #function description:    determines the next desired location of the    follower
    #                        vehicle in lat lon alt (LLA)
    #Inputs:    off_r:        radial distance away from leader [m]
```

```

#      off_theta:      angle (deg) from -x axis (out of tail), CCW is (+) rotation
#      off_l1:        distance the guided point is placed forward of the desired
#                      follower location
#      loc0_l:        location of the leader at current increment of time
#      heading_l:     heading of the leader at current increment of time
#Outputs: loc1_f:     current desired location of the follower

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

#Follower loc1 relative to Local Level Frame (L, North-East-Up) frame
cos_h=np.cos(np.deg2rad(heading_l))
sin_h=np.sin(np.deg2rad(heading_l))
R_BtoL=np.array([ [cos_h,  sin_h, 0],
                  [-sin_h, cos_h, 0],
                  [0,      0,   1] ]) # Rotation from body to local
loc1_f=np.dot(loc1_f,R_BtoL)

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

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

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

print '*****'
print 'loc1_f or follower location ECEF to lat lon alt'

```

```

print loc1_f
print '*****'

return loc1_f

# get time for data file ...
t1 = time.time()

'''Using a "known location" for the leader consisting of lat (39.782273) and lon (-
84.081874)
variables that are varied to test follower_pos calculation are:
y=np.deg2rad(180) #yaw (rad) of leader relative to NEU frame
off_r = 3        #radial distance away from leader [m]meters
off_theta = 0    #angle (deg) from -x axis (out of tail), CCW is (+) rotation degrees
off_l1 = 0       #second or meters prediction (in front of guided point)
'''

# get LEADER telemetry information for the log file and to make follower Loiter point
lat = float(39.782273)    # latitude (deg)
lon = float(-84.081874)   # longitude (deg)
alt_asl = float(10.00)    # altitude above sea level (m)
# p = float(np.deg2rad(leader.attitude.pitch)) # pitch (rad) of leader relative to NEU
frame
# r = float(np.deg2rad(leader.attitude.roll)) # roll (rad) of leader relative to NEU
frame
# y = float(np.deg2rad(leader.attitude.yaw)) # yaw (rad) of leader relative to NEU
frame
# v_b = leader.velocity # velocity vector (m/s) relative to body
# t_tel = time.time() # time telemetry was received, time is in epoch & Unix timestamp

y=np.deg2rad(180) #value has to be negative

v_b=[0, 10, 0]
p=0.0
r=0.0
# determine gc relative to leader frame (NED)
v_b = np.array([v_b[0], v_b[1], v_b[2]])
c_p = np.cos(p);
s_p = np.sin(p)
c_r = np.cos(r);
s_r = np.sin(r)
c_y = np.cos(y);
s_y = np.sin(y)
R_v_b = np.array([c_p * c_y, c_p * s_y, -s_p],
                  [s_r * s_p * c_y - c_r * s_y, s_r * s_p * s_y + c_r * c_y, s_r * c_p],

```

```

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

# if V is too slow use yaw (rad) as ground course
# if v < 0.25:
# gc = y
print 'Leader ground course in radians=' + str(gc)
gc = np.rad2deg(gc)
print 'Leader ground course in degrees =' + str(gc)

ft_tel = time.time() # time telemetry was recieved

# Call Jeremy's calculation of safe distance to follow
# convert off_l1 from time to distance
off_l1 = off_l1 * v
leader_location = [lat, lon, alt_asl]
follower_loiterpoint = follower_pos(off_r, off_theta, off_l1, leader_location, gc)

point0 = [follower_loiterpoint[0], follower_loiterpoint[1]]
print 'The follower commanded location degrees ' + '\n' + str(point0)

point1 = [follower_loiterpoint[0], follower_loiterpoint[1] - 360.0]
print 'The follower commanded location ' + '\n' + str(point1)

```

References

- 3DR Pixhawk. (n.d.). *3D Robotics*. Retrieved from <https://store.3dr.com/products/3dr-pixhawk>
- About DroneKit. (n.d.). *3D Robotics*. Retrieved from <http://python.dronekit.io/about/overview.html>
- Amazon Prime Air. (n.d.). *Amazon*. Retrieved from <https://www.amazon.com/Amazon-Prime-Air/b?ie=UTF8&node=8037720011>
- ArduPilot Mega (n.d.). *ArduPilot Mega*. Retrieved from <http://www.ardupilot.co.uk/>
- Arizona FPV's home surveillance drones. (n.d.). *Real World Survivor*. Retrieved from <http://www.realworldsurvivor.com/2014/07/22/arizona-fpvs-home-surveillance-drones/#arizona-fpvs-home-surveillance-drones-1>
- Autonomy Priority Steering Council. (2012, November 8). *Brief on Autonomy Initiatives in the US DoD*. Retrieved from http://www.defenseinnovationmarketplace.mil/resources/Autonomy-PSC_Briefing_DistroA_RE.pdf
- Barker, P. (2016, October 11). Re: [dronekit/dronekit-python] Exception in message handler for heartbeat (#610). Email communication sent to Lidia Toscano.
- BeagleBone Black. (n.d.). *Beagle Board*. Retrieved from <http://beagleboard.org/black>
- Calculate distance, bearing and more between Latitude/Longitude points (n.d.). *Movable Type Scripts*. Retrieved from <http://www.movable-type.co.uk/scripts/latlong.html>
- Chamberlain, B. (n.d.). Computing Distances. *NYU*. Retrieved from <http://www.cs.nyu.edu/visual/home/proj/tiger/gisfaq.html>

- Choosing a ground station. (n.d.). *Ardu Pilot*. Retrieved from <http://ardupilot.org/copter/docs/common-choosing-a-ground-station.html#common-choosing-a-ground-station>
- Companion computers. (n.d.). *3D Robotics*. Retrieved from <http://python.dronekit.io/develop/companion-computers.html#supported-companion-computers>
- Cox, T., Nagy, C., Skoog, M., Somers, I., & Warner, R. (2004, December). A report overview of the civil UAV capability assessment. *Semantic Scholar* Retrieved from <https://pdfs.semanticscholar.org/a4d4/f78fd94afcb2b68f3e13d7074add947b0d05.pdf>
- Digits. (n.d.). *MathWorks*. Retrieved from https://www.mathworks.com/help/symbolic/digits.html?s_tid=srchtitle
- Donley, M. B., & Schwartz, N. A. (2009). United States Air Force unmanned aircraft systems flight plan 2009-2047. *Training*, 1–82. Retrieved from https://fas.org/irp/program/collect/uas_2009.pdf
- Elston, J., Stachura, M., Dixon, C., Argrow, B., & Frew, E. W. (2015) Layered approach to networked command and control of complex UAS. In K. P. Valavanis & G. J. Vachtsevanos (Eds.), *Handbook of Unmanned Aerial Vehicles*, 782-808. Retrieved from https://link.springer.com/referenceworkentry/10.1007%2F978-90-481-9707-1_33#page-1
- Excel formula to calculate distance between 2 latitude, longitude (lat/lon) points (GPS positions). (n.d.). *BlueMM*. Retrieved from <http://bluemm.blogspot.com/2007/01/excel-formula-to-calculate-distance.html>

- Fabio, A. (2014, June 6). Droning on: Choosing a flight controller. *Hackaday*. Retrieved from <http://hackaday.com/2014/06/06/droning-on-flight-controller-round-up/>
- Gabbert, B. (2013, January 8). Unmanned aerial vehicles considered for monitoring fires in Australia. *Wildfire Today*. Retrieved from <http://wildfiretoday.com/tag/uav/page/2/>
- Gray, J. (2015). *Design and implementation of a unified command and control architecture for multiple cooperative unmanned vehicles utilizing commercial off the shelf components*. Thesis written for the Air Force Institute of Technology.
- Haddal, C. C., & Gertler, J. (2010, July 8). Homeland security: Unmanned aerial vehicles and border surveillance. *Congressional Research Service*. Retrieved from <https://www.fas.org/spg/crs/homsec/RS21698.pdf>
- Hern, A. (2016, Feb 29). Raspberry Pi 3: The credit card-sized 1.2GHz PC that costs \$35. *The Guardian*. Retrieved from <https://www.theguardian.com/technology/2016/feb/29/raspberry-pi-3-launch-computer-uk-bestselling>
- Huang, H., Messina, E., & Albus, J. (2007, December). Autonomy levels for unmanned systems (ALFUS) framework. *Framework*. NIST Special Publication 1011-II-1.0
- Kim, S. W., & Seo, S. W. (2012). Cooperative unmanned autonomous vehicle control for spatially secure group communications. *IEEE Journal on Selected Areas in Communications*. <http://doi.org/10.1109/JSAC.2012.120604>
- Kopeikin, A. N., Ponda, S. S., Johnson, L. B., Toupet, O., & How, J. P. (2011). Real-time dynamic planning to maintain network connectivity in a team of unmanned air vehicles. In *2011 IEEE GLOBECOM Workshops, GC Wkshps 2011*. <http://doi.org/10.1109/GLOCOMW.2011.6162396>

- Lambach, J. L. (2014). *Integrating UAS flocking operations with formation drag reduction*. Thesis written for the Air Force Institute of Technology.
- Leonard, M. (2014, February 25). How to choose the right platform: Raspberry Pi or BeagleBone Black? *Make*. Retrieved from <http://makezine.com/2014/02/25/how-to-choose-the-right-platform-raspberry-pi-or-beaglebone-black/>
- Linci, P., & Mohanraj, V. (2015, April). Target identification and tracking using UAV. *International Journal of Engineering Research & Technology*, 4(4), 353-356.
- MAVLink micro air vehicle communication protocol. (n.d.). *QGroundControl GCS*. Retrieved from <http://qgroundcontrol.org/mavlink/start>
- MAVLink tutorial for absolute dummies (Part – I). (n.d.). Retrieved from http://api.ning.com/files/i*tFWQTF2R*7Mmw7hksAU-9IABKNDO9apguOiSOCfvi2znk1tXhur0Bt00jTOldFvob-Sczg3*IDcgChG26QaHZpzEcISM5/MAVLINK_FOR_DUMMIESPart1_v.1.1.pdf
- MAVProxy. (n.d.). *Q Ground Control*. Retrieved from http://www.qgroundcontrol.org/mavlink/mavproxy_startpage
- Meet the Beagles: Open source computing. (n.d.). *Beagleboard*. Retrieved from <http://beagleboard.org/>
- Mini PC Intel NUC kit NUC5PGYH. (n.d.). *Intel*. Retrieved from <http://www.intel.com/content/www/us/en/nuc/nuc-kit-nuc5pgyh.html>
- Mission planner overview. (n.d.). *ArduPilot*. Retrieved from <http://ardupilot.org/planner/docs/mission-planner-overview.html>
- Nayyar, A., & Puri, V. (2015). A review of Beaglebone smart board's – a Linux/android powered low cost development platform based on ARM technology. *2015 9th*

- International Conference on Future Generation Communication and Networking*.
Retrieved from <http://ieeexplore.ieee.org/document/7434875/?arnumber=7434875>
- An overview and comparison of today's single-board microcomputers. (2013, January 15). *IQ Jar*. Retrieved from <http://iqjar.com/jar/an-overview-and-comparison-of-todays-single-board-micro-computers/>
- Passino, K., Marios P., Jacques D., Pachter M., Liu Y., Yang Y., ... Baum, M. (2002). Cooperative Control for Autonomous Air Vehicles. In *Cooperative Control and Optimization* (pp. 233-268). Boston, MA: Kluwer.
- Pixhawk versus APM comparison review. (2015, July 20). *Blogodrone*. Retrieved from <http://www.blogodrone.com/2015/07/pixhawk-versus-apm-comparison-review.html>
- Police, border patrol, other agencies looking to expand drone use at home. (2012, February 26). *Fox News*. Retrieved from <http://www.foxnews.com/politics/2012/02/26/police-border-patrol-other-agencies-looking-to-expand-drone-use-at-home.html>
- PX4 autopilot. (n.d.). *Pixhawk*. Retrieved from <https://pixhawk.org/modules/pixhawk>
- PyPI - the Python Package Index. (n.d.) *Python*. Retrieved from <https://pypi.python.org/pypi>
- Reynolds, C. (1987). Flocks, herds, and schools. *Computer Graphics*. Retrieved from <http://dl.acm.org/citation.cfm?id=37406>
- RQ-11 Raven unmanned air vehicle, United States of America. (n.d.). *Army Technology*. Retrieved from <http://www.army-technology.com/projects/rq11-raven/>

RQ-11B Raven unmanned air vehicle (UAV), United States of America. (n.d.). *Air Force Technology*. Retrieved from <http://www.airforce-technology.com/projects/rq11braven/>

Sinnott, R. W. (1984). Virtues of the haversine. *Sky and Telescope*, 68(2), 158.

Spinetta, L. (2011, January). The rise of unmanned aircraft. *History Net*. retrieved from <http://www.historynet.com/the-rise-of-unmanned-aircraft.htm>

Stracker, M. (2007, September). *An operational manpower analysis of the RQ-8 Fire Scout vertical take-off unmanned aerial vehicle (VTUAV)*. Thesis written for the Naval Postgraduate School.

Teal Group predicts worldwide UAV market will total 91 billion in its 2014 UAV market profile and forecast. (2014, July 17). *Teal Group Corporation*. Retrieved from <http://www.tealgroup.com/index.php/about-teal-group-corporation/press-releases/118-2014-uav-press-release>

UAS Task Force Airspace Integration Integrated Product Team. (2011, March). *Unmanned Aircraft System airspace Integration Plan*. Retrieved from [http://www.acq.osd.mil/sts/docs/DoD_2011_UAS_Airspace_Integration_Plan_\(signed\).pdf](http://www.acq.osd.mil/sts/docs/DoD_2011_UAS_Airspace_Integration_Plan_(signed).pdf)

Verry, T. (2012, April 18). What is the Raspberry Pi? *Extreme Tech*. Retrieved from <http://www.extremetech.com/computing/124317-what-is-raspberry-pi-2>

Winnefeld, J., & Kendal, F. (2011). Unmanned systems integrated roadmap, FY2011-2036. *Department of Defense*. Retrieved from <http://www.acq.osd.mil/sts/docs/Unmanned%20Systems%20Integrated%20Roadmap%20FY2011-2036.pdf>

References for Figures

Figure 3: Drone mosquito. (2015, April 8). *One Cell One Light Radio*. Retrieved from

<https://onecellonelightradio.wordpress.com/category/mosquitoes/drone-mosquitoes/>

Figure 4: DoD Unmanned Aircraft Systems (2011). *Unmanned Systems Integrated*

Roadmap FY 2011-2036. Retrieved from [http://www.acq.osd.mil/sts/docs/](http://www.acq.osd.mil/sts/docs/Unmanned%20Systems%20Integrated%20Roadmap%20FY2011-2036.pdf)

[Unmanned%20Systems%20Integrated%20Roadmap%20FY2011-2036.pdf](http://www.acq.osd.mil/sts/docs/Unmanned%20Systems%20Integrated%20Roadmap%20FY2011-2036.pdf)

Figure 5: Autonomy Levels For Unmanned Systems (ALFUS). Winnefeld, J., & Kendal,

F. (2011). Unmanned systems integrated roadmap, FY2011-2036. *Department of*

Defense. Retrieved from [http://www.acq.osd.mil/sts/docs/Unmanned%20Systems](http://www.acq.osd.mil/sts/docs/Unmanned%20Systems%20Integrated%20Roadmap%20FY2011-2036.pdf)

[%20Integrated%20Roadmap%20FY2011-2036.pdf](http://www.acq.osd.mil/sts/docs/Unmanned%20Systems%20Integrated%20Roadmap%20FY2011-2036.pdf)

Figure 6: USAF Fighter Aircraft Formations. *Air Force manual 11-251, volume 1: Flying*

operations: T-38 Flying fundamentals (2013, August 1). Retrieved from

[http://static.epublishing.af.mil/production/1/af_a3_5/publication/afman11-](http://static.epublishing.af.mil/production/1/af_a3_5/publication/afman11-251v1/afman11-251v1.pdf)

[251v1/afman11-251v1.pdf](http://static.epublishing.af.mil/production/1/af_a3_5/publication/afman11-251v1/afman11-251v1.pdf)

Figure 7: NATO Heavy Aircraft Formation. *Air Force instruction 11-2C-17, volume 3:*

C-17 Operations procedures (2015, Mar 10). Retrieved from [http://static.e-](http://static.epublishing.af.mil/production/1/af_a3_5/publication/afi11-2c-17v3/afi11-2c-17v3.pdf)

[publishing.af.mil/production/1/af_a3_5/publication/afi11-2c-17v3/afi11-2c-](http://static.epublishing.af.mil/production/1/af_a3_5/publication/afi11-2c-17v3/afi11-2c-17v3.pdf)

[17v3.pdf](http://static.epublishing.af.mil/production/1/af_a3_5/publication/afi11-2c-17v3/afi11-2c-17v3.pdf) [*Standard SKE Enroute Formation* (p.206)]

Figure 7: NATO Heavy Aircraft Formation. ATP-56(B): Air-to-air refuelling (AJP

3.3.4.2). (2010, January 22). *Royal Air Force*. Retrieved from

http://www.raf.mod.uk/rafcms/mediafiles/849B8228_5056_A318_A82A98C9758

5EE00.pdf *Observation/Contact/Reform, En-Route, Echelon*. (pp. 1-5, 2D-1, 2D-2)

Figure 8: BeagleBone Black. (n.d.). *Embarcados*. Retrieved from http://www.embarcados.com.br/wp-content/uploads/2013/11/beaglebone_black.jpg

Figure 9: Pixhawk. (n.d.). *ArduPilot*. Retrieved from http://ardupilot.com/ardupilot/index.html#modules/pixhawk_connectors.png?w=700&tok=19e13f

Figure 43: BBB to Pixhawk Manufactured Cable. (n.d.). *PixHawk*. Retrieved from https://pixhawk.org/_media/

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 the 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 a 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) 23-03-2017		2. REPORT TYPE Master's Thesis		3. DATES COVERED (From – To) AUG 2015 – MAR 2017	
TITLE AND SUBTITLE Effectiveness of Inter-Vehicle Communications and On-Board Processing for Close Unmanned Autonomous Vehicle (UAV) Flight Formations				5a. CONTRACT NUMBER	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER	
				5d. PROJECT NUMBER	
6. AUTHOR(S) Lidia Toscano, Civilian, USAF				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 Hobson Way, Building 640 WPAFB OH 45433-7765				8. PERFORMING ORGANIZATION REPORT NUMBER AFIT-ENV-MS-17-M-230	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Intentionally left blank				10. SPONSOR/MONITOR'S ACRONYM(S) AFIT/ENV	
				11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION/AVAILABILITY STATEMENT DISTRUBTION STATEMENT A. APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.					
13. SUPPLEMENTARY NOTES This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.					
14. ABSTRACT The small size, full-featured autopilots, and on-board processing of UAVs make them highly versatile. They can be used to search in inaccessible areas, conduct Battle Damage Assessment (BDA), provide battle commanders with real-time intelligence, and, most importantly, protect our military force where loss of life is imminent. The goal of this research was to integrate two or more small Unmanned Airborne Vehicles (UAVs) to successfully employ cooperative communication using three Commercial-Off-The-Shelf (COTS) components: a BeagleBone Black on-board microprocessor, the Pixhawk autopilot, and Open Source Software (OSS) DroneKit-Python. These components are readily available and have a low-cost factor. To baseline functionality, this project prototyped and tested two Unmanned Ground Vehicles (UGVs). Successful integration of hardware and software components was demonstrated resulting in cooperative communication of two UGVs in formation. Experimentation found that the best root-mean-squared distance was 4.0 m that occurred at an update rate 20Hz. Confidence level calculations indicated a follower UGV can safely navigate at a 2.5 m distance in close formation. A variety of qualitative observations were captured indicating the maturity of this cooperative architecture and readiness to implement across multiple UAVs in military applications.					
15. SUBJECT TERMS Cooperative Communication, Inter-Vehicle, Autonomy, Self-Governing, Architecture, UAS, UGV, COTS, OSS, Formation, Leading, Following					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT UU	18. NUMBER OF PAGES 109	19a. NAME OF RESPONSIBLE PERSON Dr. John M. Colombi, AFIT/ENV
a. REPORT U	b. ABSTRACT U	c. THIS PAGE U			19b. TELEPHONE NUMBER (Include area code) (937) 255-3636, ext 4709 john.colombi@afit.edu

THIS PAGE INTENTIONALLY LEFT BLANK