NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

PERSISTENT PERIMETER SURVEILLANCE USING MULTIPLE SWAPPING MULTI-ROTOR UAS

by

Choon Pei Jeremy Teo

September 2018

Thesis Advisor: Oleg A. Yakimenko
Second Reader: Fotis A. Papoulias

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** September 2018	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE
PERSISTENT PERIMETER SURVEILLANCE USING MULTIPLE SWAPPING MULTI-ROTOR UAS

5. FUNDING NUMBERS

6. AUTHOR(S) Choon Pei Jeremy Teo

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
N/A

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release. Distribution is unlimited.

12b. DISTRIBUTION CODE
A

13. ABSTRACT (maximum 200 words)

This study evaluated the feasibility of swapping a system of small multi-rotor unmanned aerial systems during flight for persistent perimeter surveillance. The systems engineering processes, such as requirement analysis and functional analysis, were followed by designing the overall concept of system and identifying high-level requirements for it. The research also included system prototyping. The developed system prototype consisted of three COTS products: multiple quadcopters, a single router, and a laptop running Python code. Quadcopters were programmed to fly in different predefined patterns over the Military Operation at Urban Environment Training site at Impossible City, CA, and McMillan Field at Camp Roberts, CA. Videos were recorded during the tests. While one quadcopter was flying, the remaining two stayed on stand-by. Once the airborne quadcopter depleted its battery life to the predetermined level, one of the standby quadcopters was activated as a replacement. The process can be repeated continuously, assuring an uninterrupted video stream. By monitoring the battery level and autonomously swapping quadcopters, the system showed that endurance can exceed the capability of a single quadcopter and possibly perform 24/7 surveillance, or until the system fails mechanically.

14. SUBJECT TERMS
UAV, persistent surveillance, perimeter surveillance

15. NUMBER OF PAGES
81

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

PERSISTENT PERIMETER SURVEILLANCE USING MULTIPLE SWAPPING MULTI-ROTOR UAS

Choon Pei Jeremy Teo
Civilian, Defense Science and Technology Agency, Singapore
BS, Nanyang Technological University, 2008

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN SYSTEMS ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 2018

Approved by: Oleg A. Yakimenko
Advisor

Fotis A. Papoulias
Second Reader

Ronald E. Giachetti
Chair, Department of Systems Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This study evaluated the feasibility of swapping a system of small multi-rotor unmanned aerial systems during flight for persistent perimeter surveillance. The systems engineering processes, such as requirement analysis and functional analysis, were followed by designing the overall concept of system and identifying high-level requirements for it. The research also included system prototyping. The developed system prototype consisted of three COTS products: multiple quadcopters, a single router, and a laptop running Python code. Quadcopters were programmed to fly in different predefined patterns over the Military Operation at Urban Environment Training site at Impossible City, CA, and McMillan Field at Camp Roberts, CA. Videos were recorded during the tests. While one quadcopter was flying, the remaining two stayed on stand-by. Once the airborne quadcopter depleted its battery life to the predetermined level, one of the standby quadcopters was activated as a replacement. The process can be repeated continuously, assuring an uninterrupted video stream. By monitoring the battery level and autonomously swapping quadcopters, the system showed that endurance can exceed the capability of a single quadcopter and possibly perform 24/7 surveillance, or until the system fails mechanically.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

API	Application Programming Interface
CCTV	Closed Circuit Television
COTS	commercial off-the-shelf
DARPA	Defense Advanced Research Projects Agency
DoD	Department of Defense
DODAF	Department of Defense Architecture Framework
FAA	Federal Aviation Administration
FFBD	Functional Flow Block Diagram
FY	fiscal year
FV	field of view
GCS	ground control system
GHz	Gigahertz
GPS	global positioning system
ISR	Intelligence, Surveillance, and Reconnaissance
MAVLink	micro air vehicle link
MOE	Measure of Effectiveness
MOP	Measures of Performance
NPS	Naval Postgraduate School
SV	System View
SUAS	small multi-rotor UAS
TCP/IP	Transmission Control Protocol/Internet Protocol
UAS	unmanned aerial systems
UAV	unmanned aerial vehicle
UDP	User Datagram Protocol
USB	Universal Serial Bus

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

Unmanned systems are an effective tool in military and commercial applications, such as surveying. As unmanned systems get more autonomous, they promise to offer distinct advantages for persistent perimeter patrol, as these systems can be deployed more rapidly and extensively than humans. Thus, the focus of this thesis is to explore the feasibility and limitations of using a small multi-rotor unmanned aerial system (SUAS) to provide a persistent perimeter patrol by swapping SUAS autonomously.

The research involves the prototyping of a system composed of commercial off-the-shelf (COTS) products, including 3DR Solo quadcopters, a single router, and a laptop running Python software. The objective of the proof-of-concept (POC) is to show that an SUAS on patrol can be swapped seamlessly with another SUAS while maintaining a consistent field of view (FOV) during the swap. Figure 1 shows the swapping process in this POC. The first SUAS, shown in red, is replaced by the green SUAS, while the yellow SUAS is replaced by the green SUAS when its battery is depleted below the threshold.

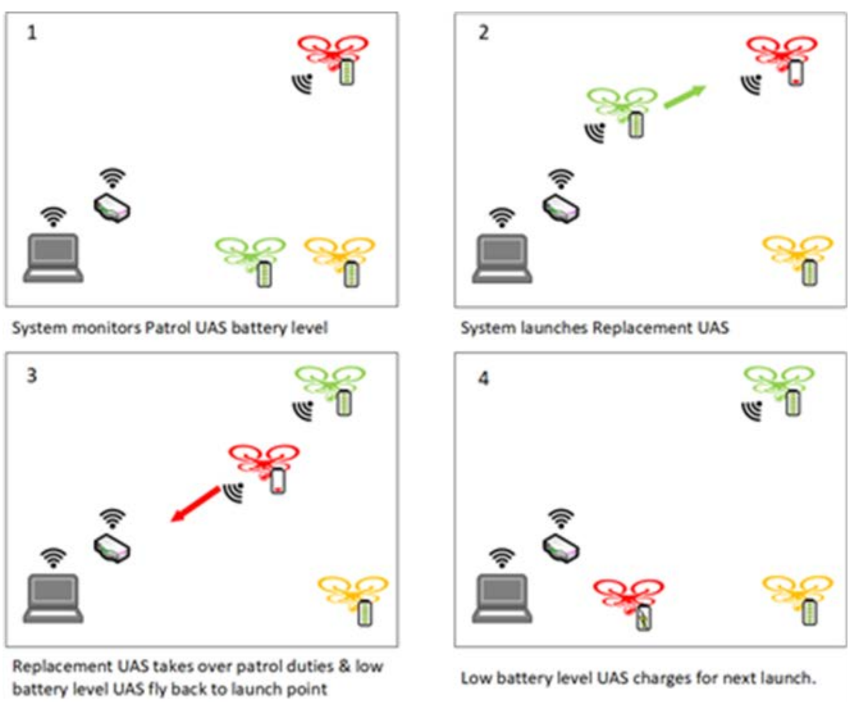


Figure 1. SUAS Swapping Process

The results show that nine out of ten trials at Impossible City, California, and at Camp Roberts, California, were successful, with one failure due to inconsistent FOV. This occurred because the SUAS being replaced was performing a 180-degree turn at the time of the swap. The FOV, however, recovered within half a second.

The system of UAS proved to be a feasible solution to provide persistent surveillance. By monitoring the battery level and autonomously replacing it with a ground control station, the system showed that its endurance can exceed the capability of a single UAS, and the SUAS can possibly perform 24/7 persistent surveillance, or until the system mechanically fails.

The solution proposed can be extended to other surveillance or target tracking tasks where persistent surveillance or targeting is required. The autonomous operations would also relieve security personnel from mundane tasks and so they could focus on more high value work. With further development on the operation effectiveness and suitability of this system, the military forces would benefit in terms of mission effectiveness and manpower savings from this technology.

ACKNOWLEDGMENTS

I would like to thank my lovely wife, Zhai Yuan, and daughter, Melody. Thank you for your understanding and patience when I was working on this thesis. I am grateful that you have come into my life, and I will love you always.

To my thesis advisor, Dr. Yakimenko, thank you for providing me the guidance and resources to complete this thesis on time. Your door was always open when I needed help, and I had fun working under you.

I would also like to acknowledge Sondre Engebraten for his help in resolving some of the network and software issues I have encountered.

And finally, I am grateful to university staff members Rushen Dal and Albert Jorden for their unfailing support and assistance in my testing.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. BACKGROUND

The number of applications utilizing unmanned aerial systems (UAS) has been growing rapidly in the past few years (Joshi 2017). With Federal Aviation Administration (FAA) help in granting exemptions for industries like agriculture, insurance, and construction to operate UAS, a wide range of applications for these industries has opened. The FAA reported that there were 110,000 registered commercial, non-model[1] UAS in 2017 and expects the average annual growth rate to be 32.5 percent (Office of Aviation Policy and Plans 2017). The usage of UAS for military applications has also increased at a tremendous rate. According to a study conducted by Bard College, the Department of Defense (DoD) has requested $9.39 billion in UAS-related funding for fiscal year (FY) 2019, which is 26 percent more than for FY 2018 (Gettinger 2017).

From facilitating the delivery of goods during traffic congestion to surveying the most remote areas, UAS have proven to be an effective tool to support operations, with little to no manpower. In its unmanned system integration roadmap report, the DoD explained that UAS have not only provided improved situational awareness, they have proven capable of reducing human workload, as well as risk to personnel and cost (Undersecretary of Defense Acquisition 2014).

Dr. Graham Drozeski, program manager in the Tactical Technology Office of the Defense Advanced Research Projects Agency (DARPA), has shared that effective 21st century warfare requires persistent collection of airborne intelligence, surveillance, and reconnaissance not possible for current technologies like helicopters, which are limited by distance and flight time, nor fixed wing aircraft, which require runways (DARPA 2017).

There has been new interest in many organizations to explore the use of small and inexpensive electric propulsion UAS. A small UAS provides lower capital cost, reduced carbon footprint, is quieter to operate and faster to deploy as compared to a larger UAS

[1] Non-Model refers to commercial, government, or other non-hobby purpose UAS.

like the Predator. While the Predator class has a longer endurance period than a small UAS, it is expensive to both procure and operate. Furthermore, a small UAS can operate at a lower altitude, which eliminates possible airspace sharing conflicts in the controlled airspace. LTC Brett Clark of the United States Army discusses the advantages of the Department of Homeland Security using a small UAS over the Predator UAS for southern border Intelligence, Surveillance, and Reconnaissance (ISR) requirements (Clark 2013).

The U.S. Marine Corps has also begun fielding small multi-rotor UAS (SUAS), by InstantEye Robotics, to its infantry units and will be ordering 800 InstantEye systems (Fuentes 2018; Kohlhepp 2018). The SUAS could help squads to see over buildings or hills to assess the risks and threats ahead (Fuentes 2018).

In the area of perimeter surveillance, Lee et al. (2015) discussed the potential of using SUAS for perimeter surveillance, which would address the disadvantages of human patrols that are limited in numbers available to patrol and in duration due to human fatigue. Apart from this, SUAS can be deployed more rapidly and extensively than humans, and are not restricted by any obstacles on the ground. Lee et al. also shared how using SUAS is a better option compared to installing Closed Circuit Television (CCTV) cameras that are typically mounted on fixed infrastructure with limited view. It is also cheaper and faster to implement an SUAS solution as compared to installing CCTVs for perimeter security. This is because CCTVs require the installation of long power and network cables as well as ground trenching for the cables, which would take more labor, money, and time. SUAS can overcome these constraints, which translates to a substantial cost savings. SUAS could also provide flexibility in surveying any area of interest and have the ability to respond to any incident site faster than a security patrol could to assess the risk before deploying any security personnel. SUAS could also provide more points of view of the situation so that more vulnerabilities could be identified than would normally be seen.

While there are many advantages in using a small SUAS, there is still no wide adoption of SUAS for perimeter surveillance. This could be due to the endurance of SUAS, which is typically around 30 minutes. This limits the type of mission that the UAS could carry out. Therefore, many research efforts have been done to extend the UAS’s duration and range (Lance 2011; Gudmundsson 2016).

The advantages of using small UAS were discussed in the preceding paragraphs. Nonetheless, small UAS are still limited by battery life which is typically around 15 to 30 minutes. This limits the type of mission that the UAS could carry out. Therefore, many research efforts have attempted to extend the UAS's duration and range (Zou et al. 2015; Chin 2011; Koumadi et al. 2017; Gudmundsson 2016).

A lightweight SUAS capable of hovering for 40 minutes was reported by Zou et al. (2015). Their research focused on extending the duration and target position accuracy of the UAS for a search and rescue mission. Carbon fiber materials were used to keep the weight to around five pounds. This may not be a sustainable solution as there is a limit to how much the weight can be reduced using carbon fiber material. Furthermore, the main challenge in extending the flight duration, as highlighted by Zou et al., is still limited by the current battery technology.

Another method to extend UAS duration and range is to use atmospheric wind as an energy source, which was proposed by Gudmundsson (2016). Gudmundsson showed that almost 50 percent of the energy lost could be recovered in a complex, multi-heading, multi-altitude mission. The proposed method, however, may only be applicable to areas with windy conditions, which may not be practical in actual deployment. A similar method of harvesting natural energy is by using solar cells, as discussed by Chin (2011). This technology also faces a similar limitation in that it is dependent on the availability of an energy source, in this case, the sun.

An alternative to Gudmundsson's and Chin's methods is to use a rotational energy harvesting device installed on the UAS. In the research of Koumadi et al. (2017), a brushless DC generator is used as a rotor to fly the UAS and to recharge the batteries on the UAS. The Koumadi report claims that the generator was able to extend flight duration by at least 42 percent. While this solution has greater potential than the previous methods, the improvement in the flight duration still poses a limitation to longer missions like perimeter surveillance.

More recent methods have included hybrid fuel cell technology and laser beam charging technology. The hybrid fuel cell technology uses fuel to convert electricity, which is used as the energy source for the UAS. In an experiment conducted by Rees (2017), the UAS was able to achieve 4.5 hours of flight duration and was claimed that it would continue to be improved. Laser-beam charging uses a laser beam to wirelessly charge the UAS while it is flying. But this approach is still in its infancy and could only transfer 20 percent of electricity from the ground to the UAS. Furthermore, it may also require the UAS to be stationary or to move at a slower speed to charge (Whittle 2012).

There are many companies like Skyfront and Quaternium that use hybrid-electric technology, which uses fuel to produce electricity to extend the endurance of the UAS to four hours. Even so, the UAS will still need to be recharged, leading to surveillance gaps. This may be unacceptable in a situation where the UAS is tracking an intruder. Another company, Elistair, uses tethering of the UAS to extend the power source from the ground to the UAS, which could provide 24/7 continuous surveillance. This is, however, limited in range and mobility. In one case, the U.S. military has also used an unmanned robotic vehicle, shown in Figure 1, to patrol its camp in Horn of Africa to overcome the limitation of flight endurance in UAS (Reid 2016). Nevertheless, the unmanned vehicle is limited in its maneuverability, especially in built-up areas or in challenging terrain. These challenges also limit the response time of the vehicle to incident areas.

Figure 1. Unmanned Surveillance Vehicle. Source: Reid (2016).

In experiment by Bethke (2007), multiple UAS were used to track a single moving target. The replacement of UAS is done sequentially, which means that one UAS would fly back before the swapping UAS is launched. Since there are multiple UAS, the swapping of one UAS would not impact the current tracking as the tracking is still being done by the remaining UAS. While this method would ensure the tracking of target is not lost, it may not be a cost effective solution to have multiple UAS track a target. Furthermore, it would add complexity to UAS management as there is a need to ensure all UAS do not fully deplete their battery life at the same time.

B. THE NEED FOR PERSISTENT PERIMETER SURVEILLANCE

The intent of the system being developed and described in this thesis is to provide persistent surveillance for perimeter security using a commercial off-the-shelf (COTS) system. Persistent surveillance is achieved by swapping small multi-rotor UAS autonomously and continuously without having any surveillance gaps or interruption. The monitoring and management of the SUAS is done by a ground control system (GCS). The GCS is used to monitor the system health and perform air traffic control during the SUAS swapping.

The focus of this thesis is not to prove that a specific SUAS used in this thesis can be deployed for perimeter surveillance, but rather to show the feasibility of using an aerial platform for perimeter surveillance. Furthermore, the research does not focus on changing the battery automatically or having wireless charging, but rather looks at what is currently available on the market.

The proposed system is intended to show how SUAS could complement the existing perimeter patrol tasks in a military installation. The system would be launched by a user to aid in 24/7 patrolling tasks. This is done by replacing the patrolling SUAS with a new one when its battery level is low. The solution could also be used as a first responder to any incident area within the perimeter and provide persistent tracking of the intruder.

C. RESEARCH APPLICATION

The thesis focuses on determining the feasibility of using a system of SUAS for persistent surveillance of a given perimeter. Using low-cost COTS quadcopters as a proof of concept, this thesis evaluates the capabilities and challenges of swapping SUAS while SUAS is moving. Exploring such persistent technologies would allow the military to minimize the number of patrollers and the installation of new surveillance infrastructure while also minimizing any coverage gaps. The solution would therefore be very applicable to DoD needs as an economical and effective method to provide persistent detection, identification, and monitoring of potential threats capability while protecting DoD assets such as forward operating bases and petroleum facilities. The solution also gives the military the capability to do early risk assessment and provides more reaction time by flying the SUAS to the incident site faster than any patroller likely could without any harm.

D. THESIS OBJECTIVES AND RESEARCH APPROACH

The objective of this thesis is to determine whether a system of SUAS can be operated autonomously providing persistent surveillance of a given perimeter, and what the limitations, if any, there are to the proposed solution. The system engineering process is followed in designing the overall concept of operations and identifying high-level requirements for the system.

The research involves prototyping of a system composed of COTS products, including 3DR Solo quadcopters, a single router, a laptop, and Python software. A quadcopter was selected because of its ability to hover and do abrupt maneuvers, which were deemed essential for perimeter surveillance. Quadcopters are programmed to fly in different predefined patterns over the Military Operation at Urban Environment Training site at Impossible City and McMillan Field, located at Camp Roberts in Monterey, California, with recording videos. One quadcopter is in flight while the remaining two are on standby. Once the airborne quadcopter has depleted its battery life to 30 percent, one of the standby quadcopters is activated as a replacement. The process is repeated a couple of times to verify that video consistency can be achieved during the swapping process. The battery is changed manually as battery swapping is not part of the scope of this thesis.

The system developed is not deployable in the field as it was developed only as a proof of concept. In order for the system to be used in the field, further analysis of the user's maintenance and other suitability requirements would be required.

E. THESIS ORGANIZATION

This thesis is organized as follows:

Chapter II describes the perceived operational concept of the system by looking at a scenario and vignettes of the system. It then follows a system engineering approach in order to derive the system architecture of the system.

Chapter III describes the developed prototype. It introduces the components that were used and describes the developed software.

Chapter IV discusses the test and evaluation approach of the system. The chapter includes a description of the different phases of the test conducted and the environmental conditions of the test scenario. It highlights the results and analysis.

Chapter V describes two examples on how the system can be implemented and discusses the survivability of the system.

Chapter VI provides conclusions and recommendations.

THIS PAGE INTENTIONALLY LEFT BLANK

II. CONCEPTUAL DESIGN

A. OPERATIONAL CONCEPT DESIGN

This section describes the perceived operational concept of operating the SUAS for camp surveillance. On a daily basis, the operator would send out the SUAS to do routine patrols within the camp perimeters. The capability of pursuing a suspect outside the camp perimeter is outside the scope of this thesis. In the next section, the operational scenario and vignettes are discussed. These would help to identify and provide insights about the type of responses or output required by the system.

1. Operational Scenario

Based on the problem statement discussed in Chapter I, a scenario has been generated to describe the process from the operator starting the system until the operation is terminated. In this scenario, the operator first starts the system and the system performs health checks on the SUAS and reports the number of SUAS available for the operation. The operator then decides whether there are sufficient SUAS for the operation. If the operation is started, the system then monitors the battery level of the SUAS that was launched. The system will swap a UAS that has a low battery level with a new one autonomously. The system informs the operator on the SUAS's status and how many SUAS are available for swapping. During the operation, live video feeds are sent to the operator for situation awareness. The operator can also choose to terminate the patrol at any time and the system will provide a summary of the patrol route and battery usage patterns to detect any anomalies in the system. The collected data could be used for maintenance planning.

2. Vignette 1: No SUAS Available for Swapping

A possible vignette would be that there is no available SUAS to take over the patrolling duties. In this scenario, the operator has started the operation and the system has detected that the available SUAS are not capable of flying. The system informs the operator of the issue and brings back the SUAS that was patrolling when its battery has reached a critical level.

3. Vignette 2: Error in the System

Another possible vignette would be the occurrence of an error in the system. When an error occurs in the system, the system issues return to launch point commands to the any SUAS that is flying and reports that the SUAS has landed safely to the operator.

B. FUNCTIONAL ANALYSIS

This section covers the functional decomposition for the system. The purpose of the functional decomposition is to identify functions and sub-functions of the system by decomposing them in a logical and well-defined manner. Figure 2 illustrates the functional hierarchy diagram, which shows the breakdown of key functions and sub-functions that the system must perform. Table 1 provides the description of each function.

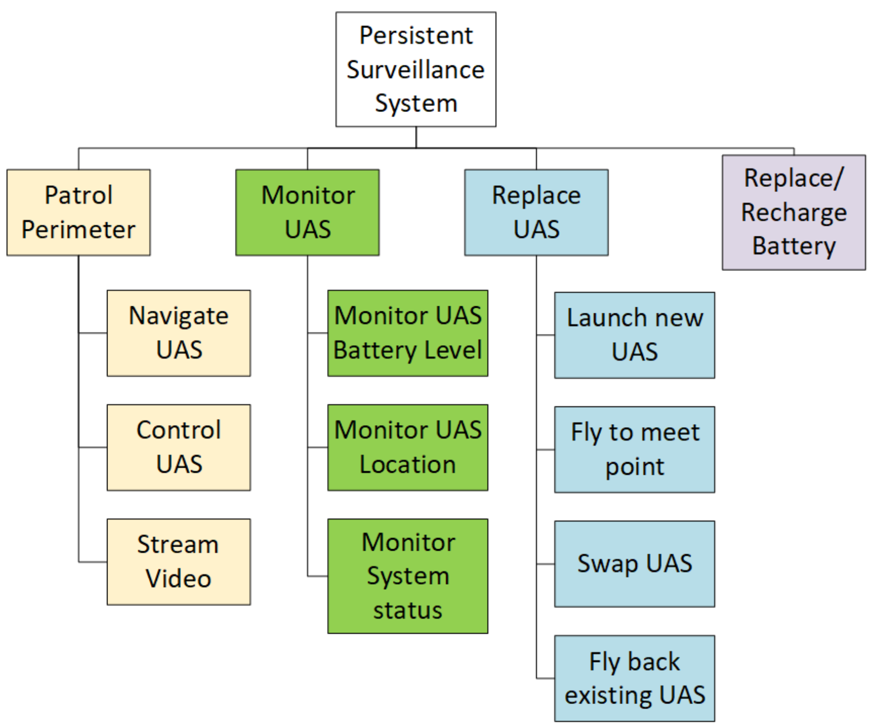


Figure 2. Functional Hierarchy Diagram.

Table 1. Description of System Functions and Sub-functions.

Key Function	Sub-functions	Description
1.0 Patrol Perimeter This function addresses the patrolling of the UAS around the perimeter.	1.1 Navigate UAS	Navigates the UAS to designated spots around the perimeter.
	1.2 Control UAS	Controls the UAS to designated spots around the perimeter.
	1.3 Stream Video	Streams the video back to a central system for viewing and recording.
2.0 Monitor UAS This function addresses the monitoring of various parameters of the UAS to ensure that the system is operating normally.	2.1Monitor Battery Level	Monitors the battery level of the UAS while it is flying.
	2.2 Monitor UAS Location	Monitors the UAS location while UAS is flying.
	2.3 Monitor UAS system status	Monitors the UAS response while UAS is flying.
3.0 Replace UAS This function addresses the replacement of the airborne UAS that has depleted its battery level to the critical limit or lower.	3.1 Launch New UAS	Launches a replacement UAS to appropriate height.
	3.2 Fly to meet point	Flies replacement UAS to existing UAS location.
	3.3 Swap UAS	Directs swapping UAS to take over existing UAS patrol tasks.
	3.4 Fly back existing UAS	Flies existing UAS back to launch location.
4.0 Replace / Recharge Battery This function addresses the replacement or charging of the UAS battery that has been depleted to or below the critical limit.		

C. FUNCTIONAL FLOW BLOCK DIAGRAM

The functional flow of the SUAS system is documented in the Functional Flow Block Diagram (FFBD) shown in Figure 3. It is used to determine the required actions and their required sequence. The FFBD takes into account the operational scenario that was discussed previously, which starts from checking the SUAS until swapping the SUAS.

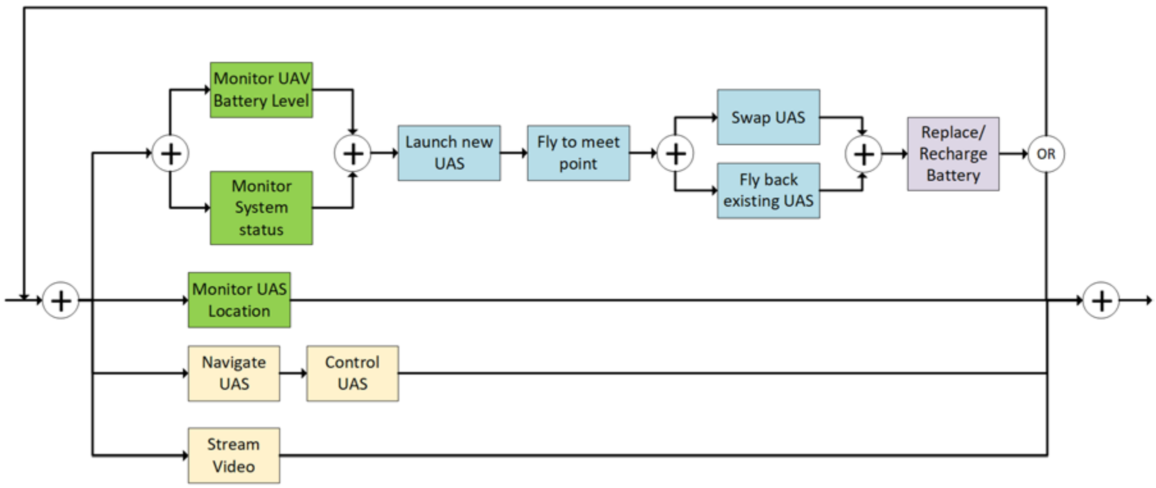


Figure 3. Functional Flow Block Diagram of the UAS.

D. FUNCTIONAL ALLOCATION

This section covers the mapping of functions described in the previous section into physical elements. The purpose of the functional allocation is to ensure that all required functions are addressed by the system and its components. Table 2 shows the components allocated to the functions.

Table 2. Functional Allocation.

System / Component	Sub-functions	Description
Ground Control Station (GCS), UAS and Global Positioning System (GPS)	1.1 Navigate UAS	Navigates the UAS to designated spots around the perimeter.
UAS	1.2 Control UAS	Controls the UAS to designated spots around the perimeter.
Camera and Wireless System	1.3 Stream Video	Streams the video back to a central system for viewing and recording.
GCS	2.1Monitor Battery Level	Monitors the battery level of the UAS while it is flying.
GCS	2.2 Monitor UAS Location	Monitors the UAS location while the UAS is flying.
GCS	2.3 Monitor UAS system status	Monitors the UAS response while the UAS is flying.
Ground Control System and UAS	3.1 Launch New UAS	Launches replacement UAS to appropriate height.
GCS, UAS, and GPS	3.2 Fly to meet point	Flies replacement UAS to existing UAS location.

System / Component	Sub-functions	Description
UAS	3.3 Swap UAS	Directs swapping UAS to take over existing UAS patrol tasks.
Ground Control System and UAS	3.4 Fly back existing UAS	Flies existing UAS back to launch location.
Charging system	4. Replace/ Recharge Battery	Replaces or charges the UAS battery that has been depleted to the critical limit or lower.

E. SYSTEM ARCHITECTURE

The Department of Defense Architecture Framework's (DODAF) System View-1 (SV-1) diagram shows the interfaces between the different system components. Figure 4 shows the SV-1 diagram for the system. The GCS consists of three software modules, namely the main controller module, the Patrol module, and the Replace module. The GCS sends and receives information through a wireless system to the SUAS. The SUAS have a camera gimbal attached with a camera on board the SUAS. Each SUAS is also connected to a charging system when it is not flying.

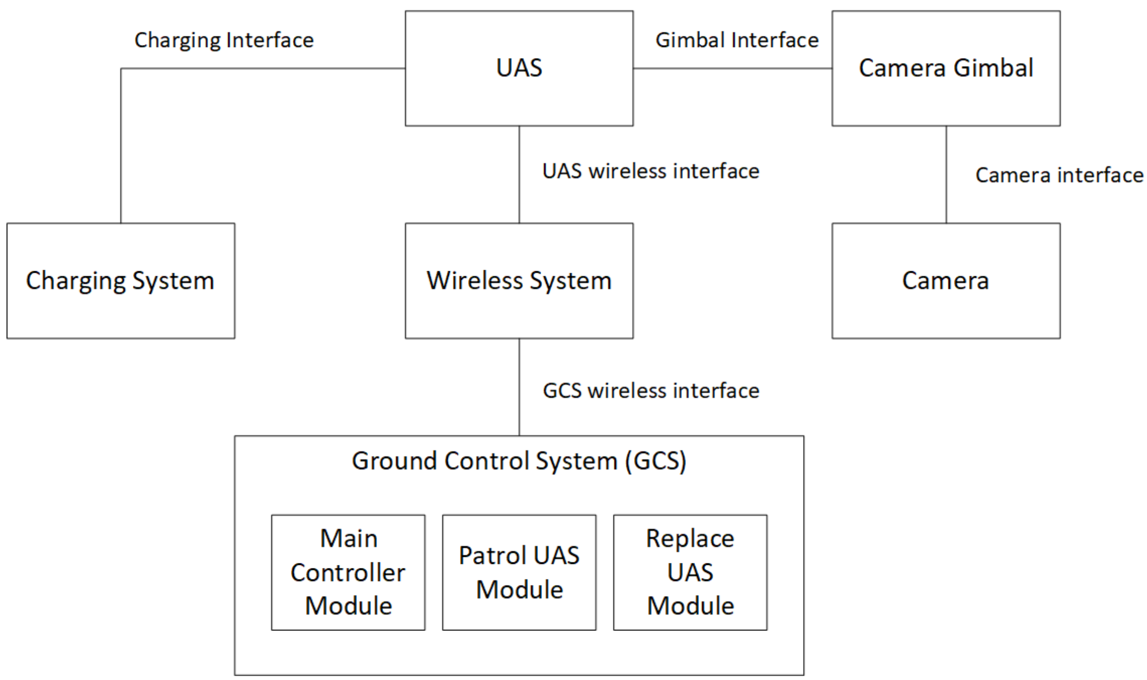


Figure 4. SV-1 Diagram.

The System View-2 (SV-2) diagram is shown in Figure 5, where the means of communication between the components or system are depicted. The wireless exchanges are based on Transmission Control Protocol and Internet Protocol (TCP/IP) network communications. The gimbal control is communicated through a micro air vehicle Link (MAVLink) between the SUAS controller and gimbal by a serial connection. The gimbal is connected to the camera through a Universal Serial Bus (USB). The charging system can be based on electromagnetic wave to wireless charging technology, which is not within the scope of this thesis.

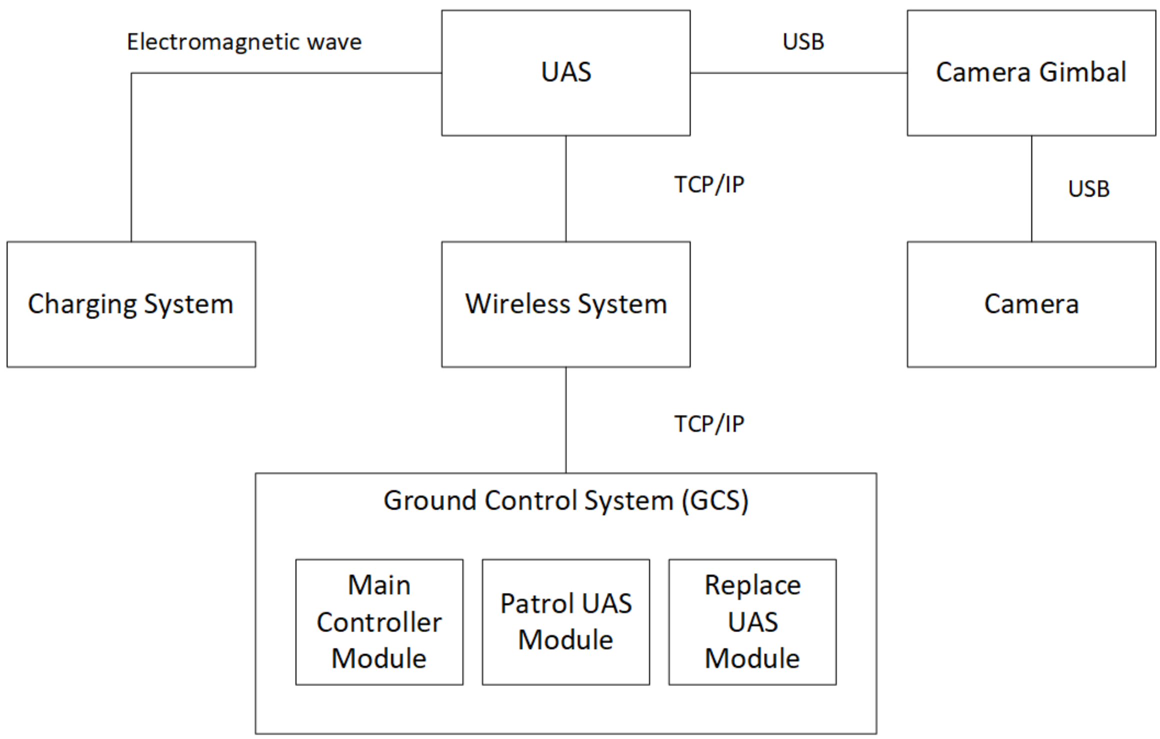


Figure 5. SV-2 Diagram.

The next chapter discusses the configuration and development process of the system prototype. The system prototype is based on the system architecture developed in this chapter. It is important to note that the prototype is developed as a proof of concept for this thesis and is not representative of an operational system.

III. SYSTEM PROTOTYPING

This chapter describes the process and the system used for the proof-of-concept of the persistent tracking system.

A. SYSTEM OVERVIEW

The system under development builds upon the static persistent surveillance system developed by Williams (2017). Figure 6 shows the overall setup of the system. It consists of a laptop running on the Linux operating system, a WIFI router, and three SUAS (or unmanned aerial vehicles, UAV) with cameras attached.

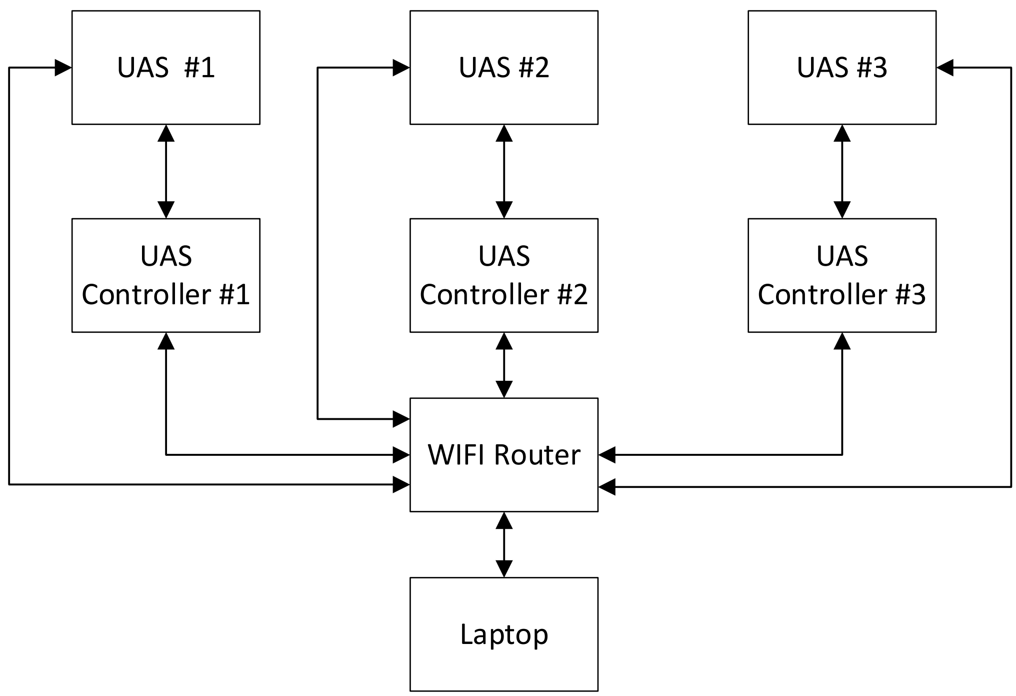


Figure 6. Overall System Setup.

The laptop is running Python software to control and manage the SUAS. The WIFI router is used as the communication medium between the laptop and the SUAS to send control commands from the laptop and receive status from the SUAS. The SUAS controllers can also be used to control the SUAS manually in case of emergency.

The objective of the proof-of-concept is to show that one SUAS on patrol can be swapped seamlessly with another SUAS. In this setup, the criterion on which to swap a SUAS is based on battery level of the SUAS. The system will first conduct an initialization to determine how many SUAS are available for the mission. Once it is determined, the first SUAS will take off to perform a patrol on a predefined route. The system will monitor the battery level of the airborne SUAS, and when the battery level is depleted to a user-defined threshold, a replacement SUAS will be launched. The system will then monitor the distance between the existing and replacement SUAS. Once the replacement SUAS has reached within one meter of the existing SUAS, the replacement SUAS will take over the patrolling duties while the existing SUAS returns to where it was launched for battery replacement or charging. The system will cycle through all the SUAS until there are no SUAS capable of replacement or the user terminates the mission. Figure 7 shows the swapping process of the SUAS.

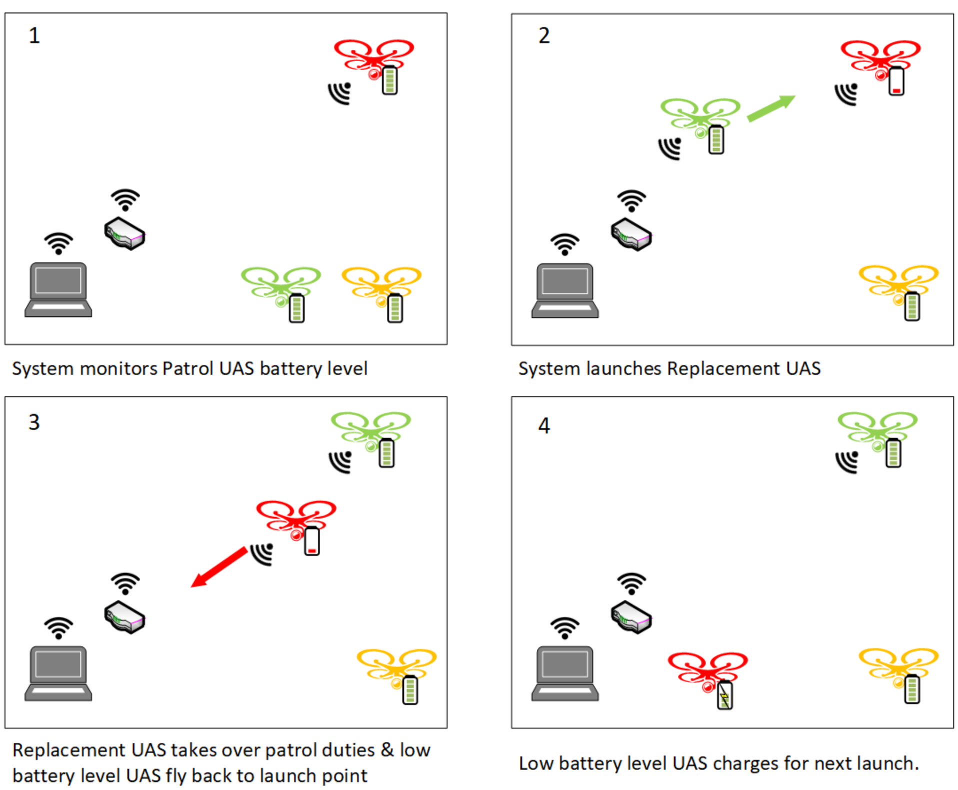


Figure 7. SUAS Swapping Process. Source: Herzog (2017).

B. HARDWARE SOLUTION

The hardware consists of a laptop, WIFI router, and multiple SUAS. Configurations were done on each of the hardware components to ensure that information was send and received at the appropriate time to achieve the mission objective.

1. Laptop

The laptop runs the Ubuntu 16.04 operating system with the technical specifications shown in Table 3.

Table 3. Ground Control Station Specifications of Laptop.

Components	Description
Operating System	Ubuntu 16.04 LTS
Processor	Intel Core i7-7700HQ-CPU @ 2.8GHz x 8
Random Access Memory	32 GB DDR4
Graphic Card	Gefore GTX 1060/PCIe/SSE2
Memory Disk	1 TB 2.5" Drive

The development software used was Python, which is described later under "Software Solution." In addition to the development software, third-party software was used to test and communicate with the SUAS. The list of required software is shown in Table 4.

Table 4. Software Required.

Software Name	Description
Solo CLI	A command-line tool to control, update, or connect with Solo for development
Virtualenv	A tool to create isolated Python environments
Dronekit	The Application Programming Interface used to communicate with UAS over MAVLink

Software Name	Description
Dronekit-Software In The Loop	The simulator used to test Dronekit applications without using real UAS
APM Planner 2.0	An open-source ground station application for MAVLink based UAS
Geany	A text editor used to debug the software program
VLC Media Player	An open-source cross-platform multimedia player used to record video streams from UAS

2. Unmanned Aerial Vehicle

The SUAS used for this thesis is from 3D Robotics, which is a commercially available low-cost quadcopter SUAS. The SUAS can be easily programmed to fly autonomously by either downloading a script to the SUAS or sending commands wirelessly through a WIFI to a ground control station like a laptop. The SUAS is connected to a three-axis gimbal with a GoPro Hero 4 Silver camera to provide stable shots automatically. Figure 8 shows the picture of the SUAS.

Figure 8. 3DR Solo Drone.

The specifications of the SUAS are shown in Table 5. The SUAS is powered by two 1-gigahertz (GHz) processor flight control systems. The flight controller uses a Global Positioning System (GPS) to navigate and uses gyros, accelerators, and other sensors as safety features for flying. The SUAS comes in a ready-to-fly bundle with a radio controller and mobile applications to control both the SUAS and the camera.

Table 5. 3DR Solo Drone Specifications.

Components	**Description**
Dimensions	18 x 18 x 10 inches
Weight	3.9 lbs. with GoPro camera and Solo Gimbal
Maximum Speed	55 miles per hour
Flight time	20–25 minutes
Battery	Lithium Polymer 5200mAh@14.8Vdc
Communication	2.4GHz WIFI network
Motors	880 kilovolts (kV)

3. WIFI ROUTER

The WIFI Router is used to connect the ground control system with the SUAS in the same network. In order to achieve this, the SUAS are each configured within the same Internet Protocol (IP) address class and operate on the 2.4GHz frequency bandwidth. Unique User Datagram Protocol (UDP) ports were also assigned to each SUAS that uses the MAVLink protocol to communicate with the laptop. Figure 9 shows the wireless network configuration diagram of the SUAS and the laptop.

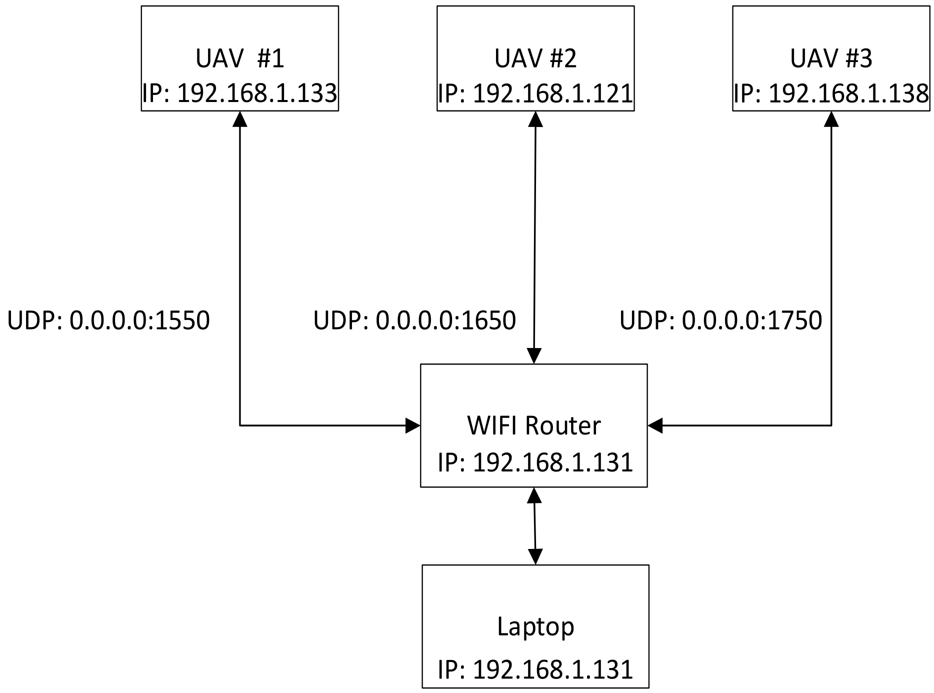


Figure 9. System Network Configuration Diagram

C. SOFTWARE SOLUTION

The developed software is written in the Python programming language. The Dronekit-Python Application Programming Interface (API) was used to communicate with the ArduPilot Flight Controller using a low latency link. The API provides a library of ready-to-use scripts to connect, control, and monitor the SUAS. Various functions could be called simply by importing the Dronekit-Python Library and referencing the function name. These functions were used to develop the autonomous functions of the system.

Software was based on having the GCS as a central control. This means that all SUAS will take commands from the GCS. Three software functions were run in parallel in order to maintain persistent tracking, air traffic control, and flight of the replacement SUAS when the airborne SUAS's battery level is depleted. Figure 10 shows the flow diagram for the developed software.

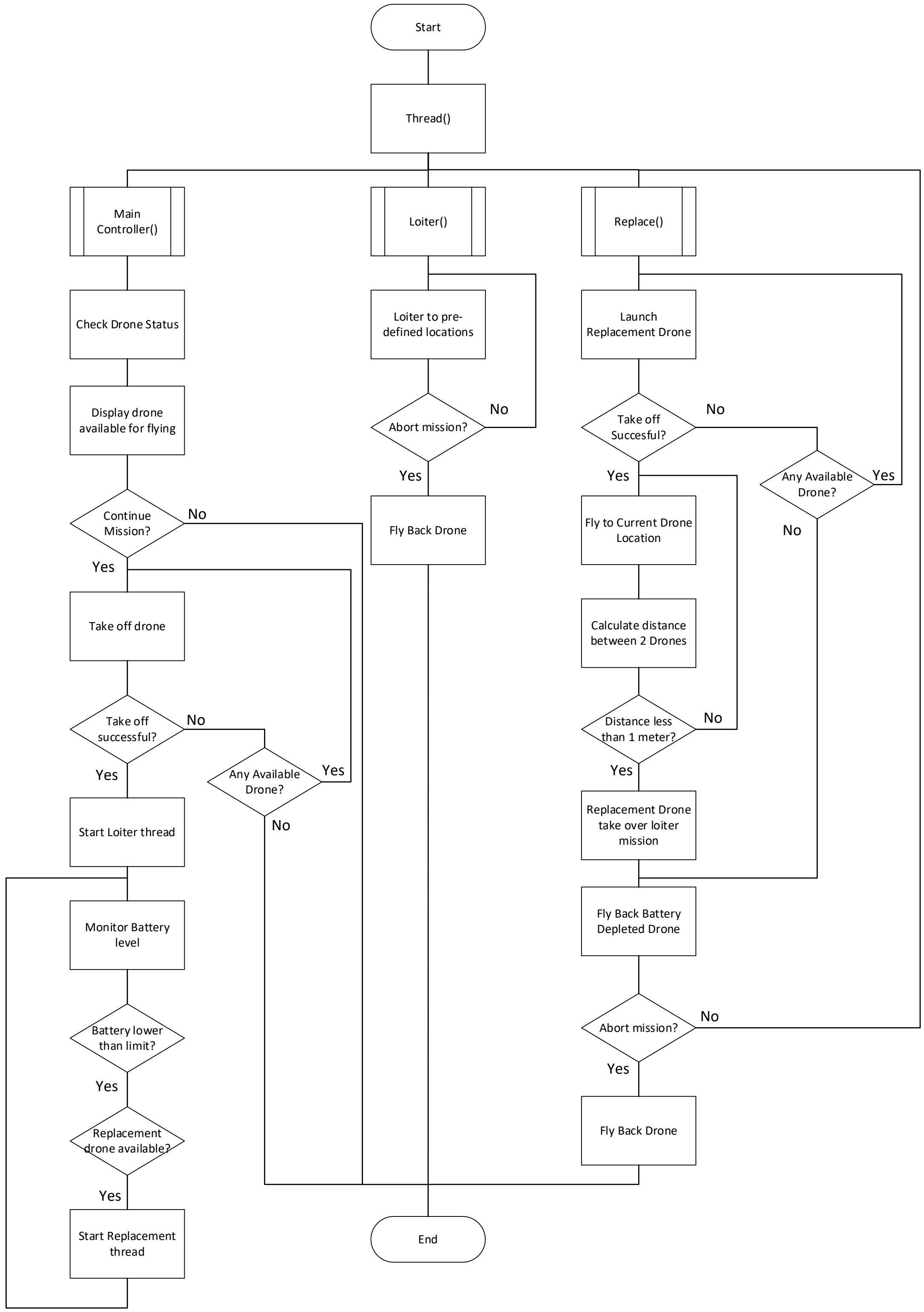


Figure 10. Software Flow Diagram.

In the next chapter, the test and evaluation process and environment for the prototype described in this chapter are discussed. This discussion includes the different phases of test that were done in order to arrive at the final phase of the test to validate the feasibility of the solution.

IV. SYSTEM TEST AND EVALUATION

This chapter describes the test and evaluation procedures and test environment to evaluate the feasibility of having 24/7 persistent surveillance while the SUAS is moving. The main focus of this experiment is to determine whether a similar FOV could be achieved during SUAS swapping. The results of the test and evaluation are then presented and discussed in the next chapter.

A. TEST PROCEDURES

The test was broken into many phases to ensure that the SUAS would operate in a safe manner as intended. Figure 11 shows the different phases of testing. The test began in March 2018 and ended in July 2018.

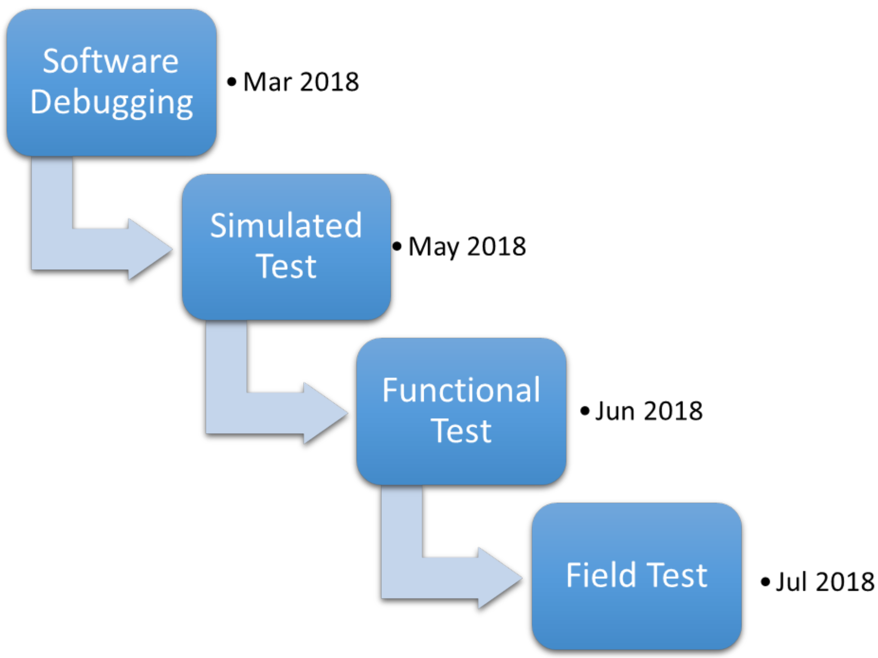


Figure 11. Different Phases of Testing.

Apart from using software debugging tools to check for syntax errors, a software-in-the-loop simulator was used to ensure the logic of the code was correct. As multi-threading was used as part of the solution, simple commands to fly two SUAS simultaneously were tested to observe the behavior of the SUAS before performing more complex commands. Different software functionality tests with the SUAS were done to facilitate faster troubleshooting and debugging of the system. The software codes contain a safety feature to ensure that the SUAS is not damaged if there is an error. This safety feature includes a command for the SUAS to return to the launching point when there is an error and also to have the software monitor the distances between two SUAS to prevent collision.

B. TEST ENVIRONMENT

The experiment was conducted at the Military Urban Environment Training site at Impossible City, California, and McMillan Field, at Camp Roberts, California. The test was conducted during the day and at about 30 meters and 15 meters above ground at Impossible City and Camp Roberts, respectively. The flying height was above the urban buildup in that area as maneuvering within buildings was not part of the scope of this experiment. Figure 12 shows a picture of the test sites.

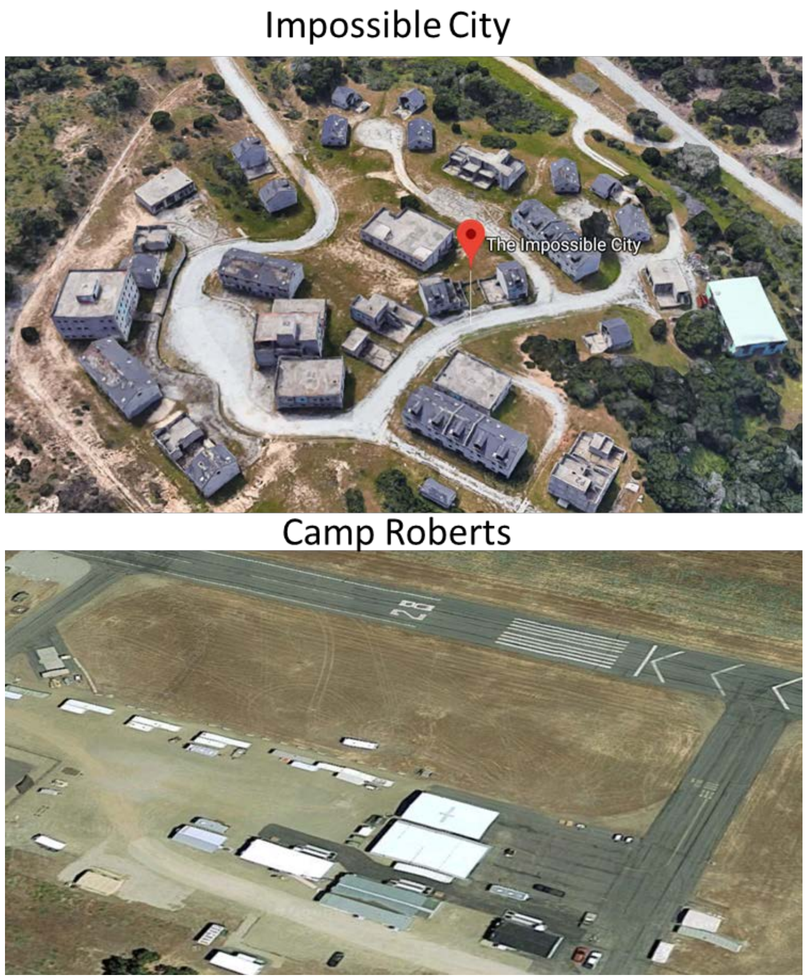
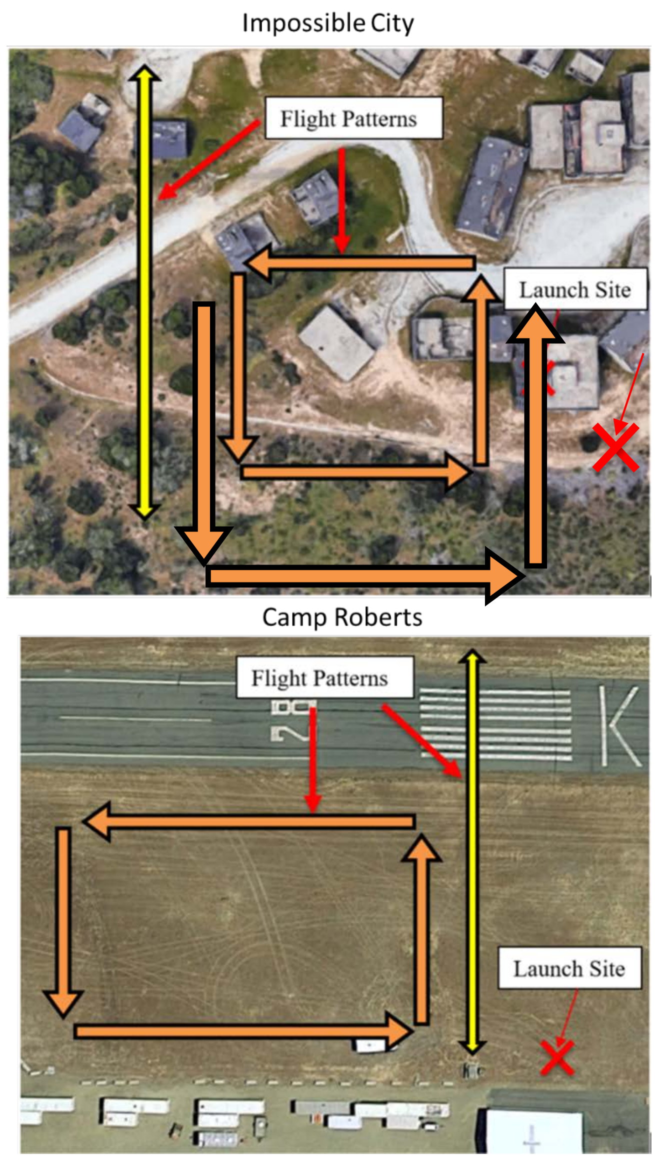


Figure 12. Experiment Conducted at Impossible City and Camp Roberts. Source: Google Maps (2018a, 2018b).

C. SCENARIO

A perimeter patrol scenario was selected to evaluate the feasibility of the system. The SUAS flies in a square loop and on a straight-line through and flow at an altitude of 15 and 30 meters. Figure 13 shows the location of the launch and flight patterns at the respective test sites. The camera on board the SUAS will record the whole duration of the flight until the SUAS has landed. The SUAS will continue to provide perimeter surveillance until the user terminates the operation or battery level is low. This simulates the real-world environment where the SUAS operates around a perimeter providing airborne surveillance.



Impossible city

Figure 13. Launch Site and Flight Patterns at Impossible City and Camp Roberts. Source: Google Maps (2018a, 2018b).

The test scenario is deemed successful when coherent video can be seen during the SUAS airborne swapping and the developed SUAS can provide a persistent surveillance that is longer than the capability of a single SUAS.

D. SCOPE

The focus of the development and testing is on the SUAS' ability to perform on-the-move replacement when flying autonomously. The autonomous charging and target tracking functions were not included but should be developed in the future.

E. OBJECTIVE

The Measure of Effectiveness (MOE) and Measures of Performance (MOP) were used to quantify the results of the test and evaluation. This is shown in Figure 14. The MOE used for the SUAS is the persistent surveillance capability of the system. The MOPs used to evaluate the system were success rate of swapping and success rate of having a consistent FOV during the swap. A consistent FOV is achieved when the swapping SUAS field of view could cover at least 80 percent of the original SUAS field of view.

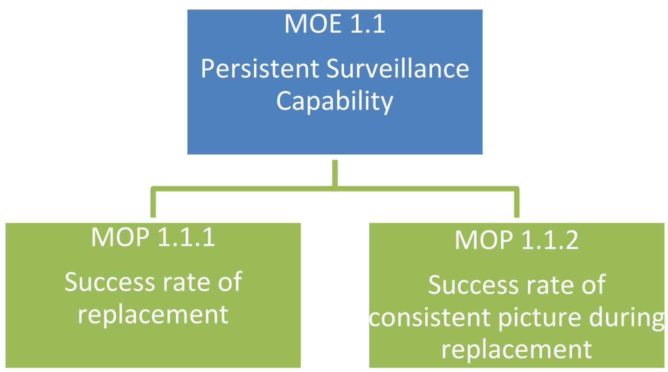


Figure 14. System MOE and MOPs.

F. SETUP

The experiment setup consists of the GCS, which is a laptop, a router, 3x SUAS controllers and 3x SUAS, as shown in Figure 15. The SUAS controllers were used to take over control from the GCS when any error occurs.

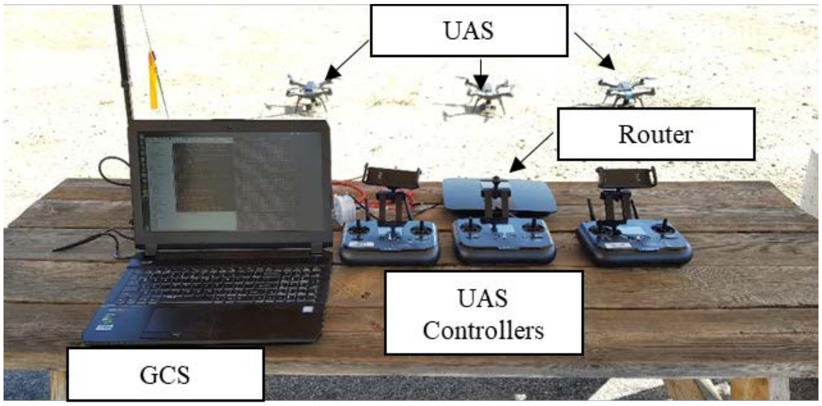


Figure 15. Experimental Setup.

Due to the inaccuracies of the GPS sensor, the replacement distance between the patrolling SUAS and replacing SUAS was set to be 10 meters. The update rate of the patrolling SUAS location to the replacing SUAS was every second.

G. TEST RESULTS

A total of ten samples were calculated during the test. Due to communication and GPS errors in the SUAS, most of the test sample consists of only two SUAS. Nevertheless, this does not affect the results of the test since the focus is on the successful swapping of the SUAS. Figure 16 shows the photos of the swapping process during the test.

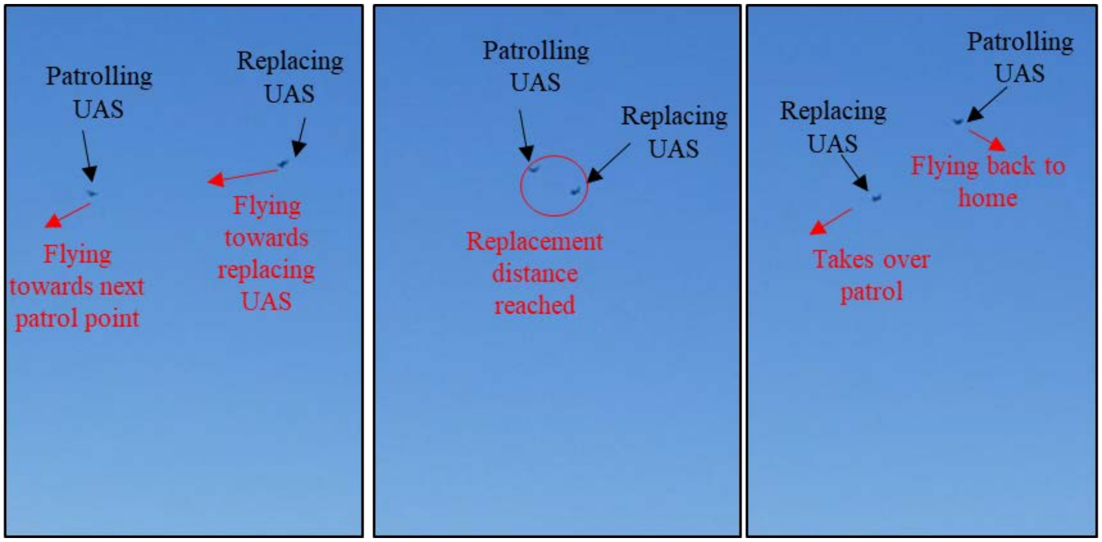


Figure 16. SUAS Swap.

Figure 17 shows the three-dimensional (3D) plot of the swapping process. From the results, it can be seen that the system successfully operated multiple SUAS to patrol in a square pattern and swapped the vehicles autonomously.

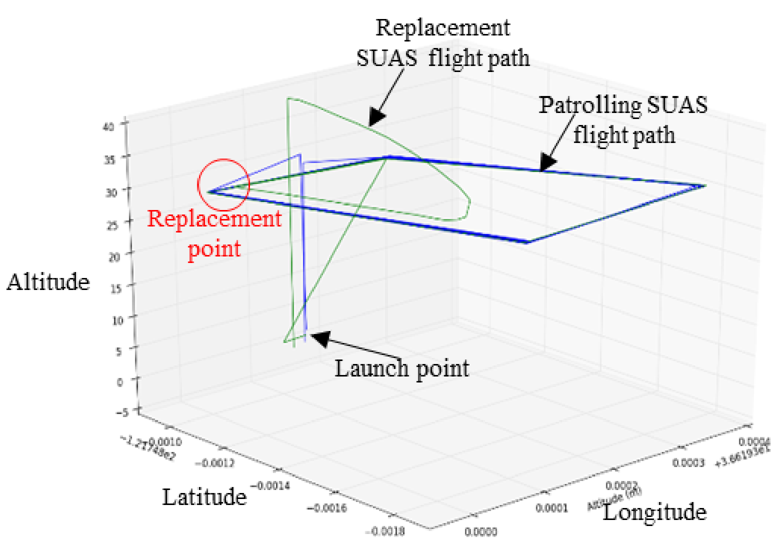


Figure 17. SUAS Flight Path.

The two-vehicle swap occurred during the field testing. From the results, it can be seen that the system is more capable of supporting a single SUAS that can only last for an average duration of 11–15 minutes in the test. With additional batteries, the system could provide a 24/7 perimeter surveillance. Due to limited resources, the 24/7 operation test was not conducted but should be done in the future to determine its feasibility.

Figures 18 and 19 show some screenshots of the camera videos from both SUAS during patrolling and swapping. All of the test screenshots can be found in Appendix B.

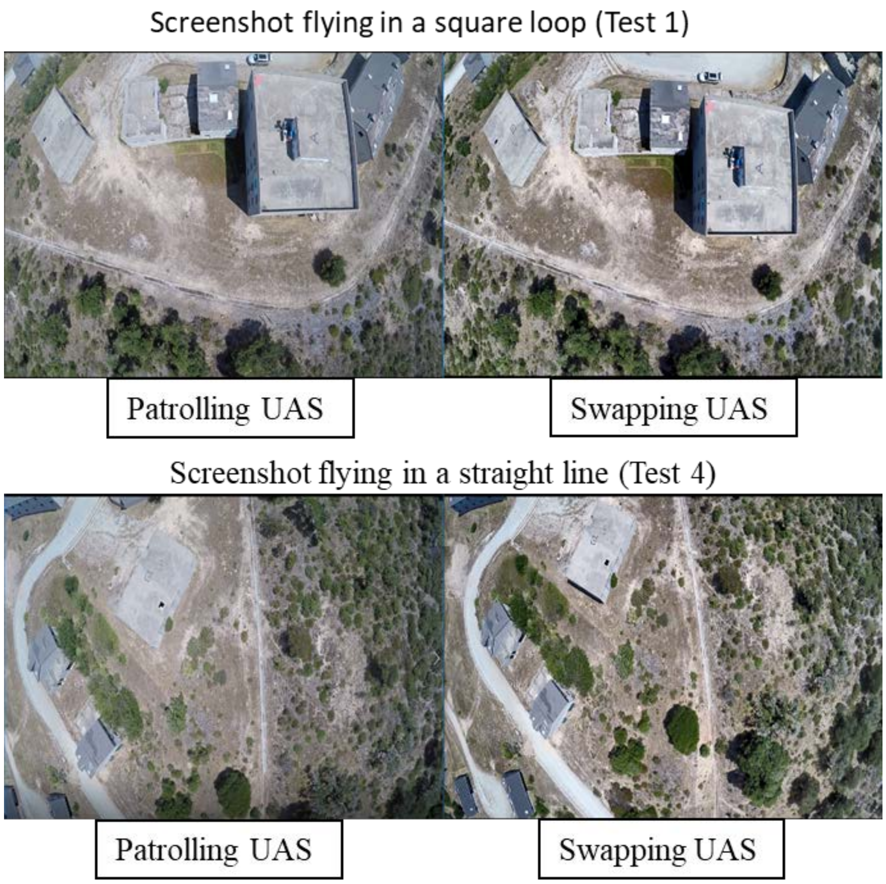


Figure 18. Screenshots of Videos during SUAS Swapping at Impossible City.

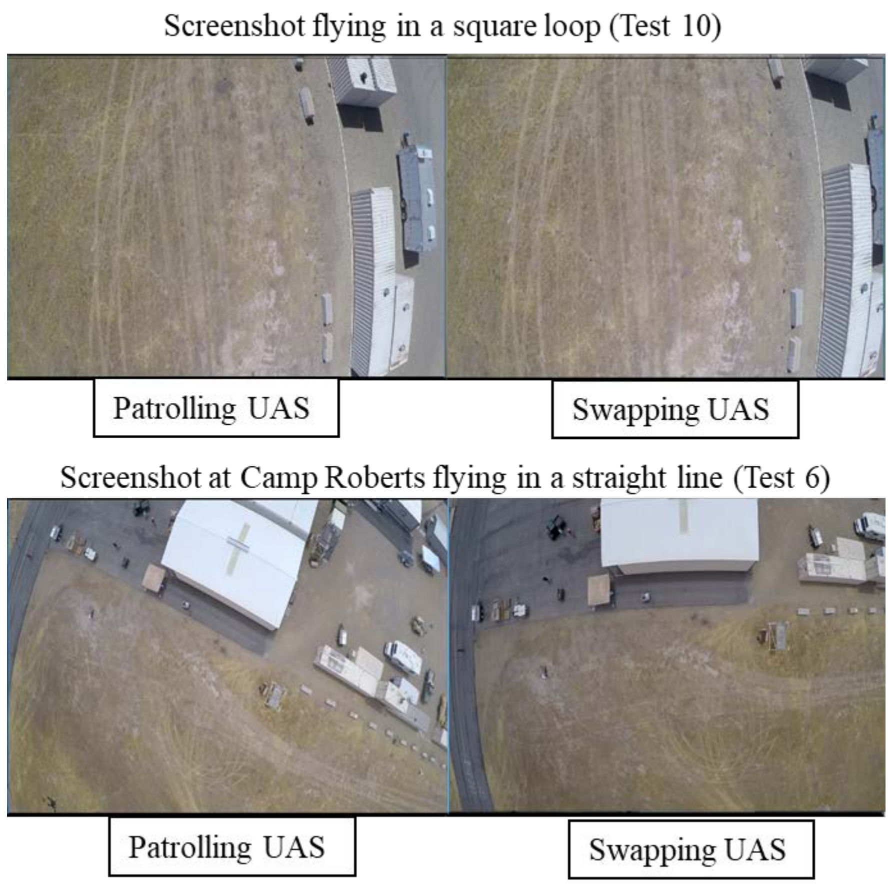


Figure 19. Screenshots of Videos during SUAS Swapping at Camp Roberts.

Table 6 shows the summary of the results for the ten test trials.

Table 6. Results Summary.

Test Number	Location	Patrol Pattern	Successful Swapping (Pass/Fail)	FOV Consistency (Pass/ Fail)
Test 1	Camp Roberts	Square Loop	Pass	Pass
Test 2		Square Loop	Pass	Pass
Test 3		Square Loop	Pass	Pass
Test 4		Straight Line	Pass	Pass
Test 5		Straight Line	Pass	Pass
Test 6	Impossible City	Straight Line	Pass	Fail
Test 7		Straight Line	Pass	Pass
Test 8		Straight Line	Pass	Pass

Test Number	Location	Patrol Pattern	Successful Swapping (Pass/Fail)	FOV Consistency (Pass/ Fail)
Test 9		Straight Line	Pass	Pass
Test 10		Square Loop	Pass	Pass
Success rate			100%	90%

From the results, it can be seen that the SUAS was able to take over the patrol task with consistent FOV at Impossible City. For test 6 that was conducted at Camp Roberts, the swapping was done when the patrolling SUAS was about to turn back and therefore a significant portion of the FOV was lost. Nonetheless, the replacing SUAS was able to cover the surveillance gap in half a second. The SUAS could also be configured to fly forward and backward without turning; this would prevent such inconsistency in the field of view.

V. FEASIBILITY ANALYSIS

In the experiment conducted, only three SUAS were used for the perimeter patrol tasks. Depending on the size of the surveillance area and revisit rates requirement, more SUAS could be added to expand the surveillance capability and survivability of the system. The following sections give two examples of the quantities of SUAS needed given the distance of the perimeter and revisit rates requirements.

A. NAVAL POSTGRADUATE SCHOOL

The Naval Postgraduate School (NPS) has an approximate perimeter distance of three kilometers, as shown in Figure 20. Thus, a total of at least 21 SUAS would be required given that the SUAS is flying at 16 Kilometers per hour, the revisit rate is every five minutes, and charging takes 90 minutes.

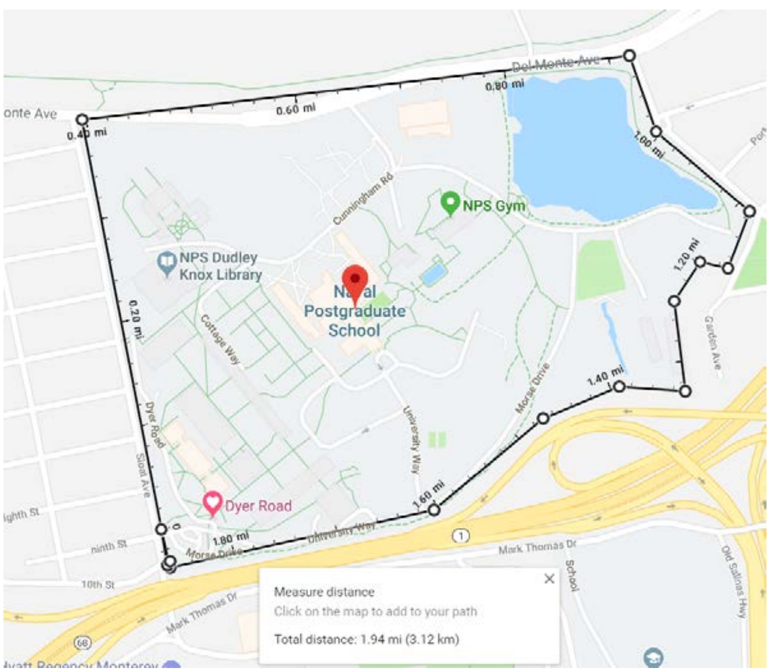


Figure 20. Map of Naval Postgraduate School. Source: Google Maps (2018c).

In order to meet the revisit rate of five minutes, three SUAS would have to patrol simultaneously at any given time. The total number of SUAS can then be calculated using the following formula.

Number of UAS required =   (1)

where is the number of UAS patrolling at any given time.

Specifically, for = 3
Number of UAS required = 21

Figure 21 shows the time schedule of the 21 SUAS for one continuous cycle. From the figure, it can be seen that a total of 18 SUAS are required to patrol the perimeter before the first three batteries are charged and can be used again.

	15 mins	30 mins	45 mins	60 mins	75 mins	90 mins	105 mins	120 mins	135 mins	150 mins	165 mins	180 mins	195 mins
UAS (1-3)	Patrol	Charging											
UAS (4-6)		Patrol	Charging										
UAS (7-9)			Patrol	Charging									
UAS (10-12)				Patrol	Charging								
UAS (13-15)					Patrol	Charging							
UAS (16-18)						Patrol	Charging						
UAS (19-21)							Patrol	Charging					

Figure 21. Time Schedule to Patrol the Naval Postgraduate School Perimeter.

Furthermore, the number of SUAS could be reduced if new batteries could be swapped with old ones. This would mean that a minimum of six SUAS with 21 batteries would be able to cycle through until the first set of batteries is fully charged. Figure 22 shows the time schedule of using swappable batteries.

	15 mins	30 mins	45 mins	60 mins	75 mins	90 mins	105 mins	120 mins
		Battery (1-3) Charging						
UAS (1-3)	Battery (1-3)		Battery (7-9)		Battery (13-15)		Battery (19-21)	
UAS (4-6)		Battery (4-6)		Battery (10-12)		Battery (16-18)		Battery (1-3)

Figure 22. Time Schedule when Using Swappable Batteries (NPS).

B. CORRECTIONAL TRAINING FACILITIES IN SOLEDAD

The Correctional Training Facilities in Soledad, California, provides another example of how the SUAS can be used to patrol the perimeter. Figure 23 shows the map of the training facility.

Figure 23. Map of Correctional Training Facilities at Soledad. Source: Google Maps (2018d).

The training facility has an approximate perimeter of 2 kilometers. It is assumed that the revisit rate for such facility would be 1 minute. This would require at least eight SUAS patrolling simultaneously. The total number of SUAS can be calculated using Equation (1):

Number of UAS required = 56

Given = 8, time to charge battery = 90 and time to deplete battery =15

Similar to the time schedule for patrolling the Naval Postgraduate School in Figure 21, the time schedule for patrolling the Correctional Training Facilities is shown in Figure 24. A total of 46 SUAS (from UAS 9 to 56) are required before the first eight SUAS are fully charged and ready to patrol again.

	15 mins	30 mins	45 mins	60 mins	75 mins	90 mins	105 mins	120 mins	135 mins	150 mins	165 mins	180 mins	195 mins
UAS (1-8)	Patrol	Charging											
UAS (9-16)		Patrol	Charging										
UAS (17-24)			Patrol	Charging									
UAS (25-32)				Patrol	Charging								
UAS (33-40)					Patrol	Charging							
UAS (41-48)						Patrol	Charging						
UAS (49-56)							Patrol	Charging					

Figure 24. Time Schedule to Patrol Correctional Training Facilities Perimeter.

The number of SUAS can be reduced to 16 if batteries are swapped with new ones when the batteries are depleted. This would mean that a total of 56 batteries is required to cycle through until the first sets of batteries were recharged. Figure 25 shows the time schedule using swappable batteries.

	15 mins	30 mins	45 mins	60 mins	75 mins	90 mins	105 mins	120 mins
		Battery (1-8) Charging						
UAS (1-8)	Battery (1-8)		Battery (17-24)		Battery (33-40)		Battery (49-56)	
UAS (9-16)		Battery (9-16)		Battery (25-32)		Battery (41-48)		Battery (1-8)

Figure 25. Time Schedule When Using Swappable Batteries (Correctional Training Facilities).

It can be observed that the number of SUAS or batteries requirement increases significantly when the revisit rates are reduced to a minute.

C. SUAS QUANTITY IMPLICATIONS

The previous calculations are based on the minimum number of UAS required for perimeter surveillance, but these calculations do not take into consideration of the operation suitability of the system. Operation suitability like reliability, availability, and maintainability requirements would have an impact on the number of SUAS required for the mission.

Increasing the number of SUAS in the system also increases the complexity of the software. The GCS needs to manage multiple UAS and the battery depletion rate in such cases may not be uniform. This also increases the bandwidth needed for control and to do video streaming.

An alternative solution is to have onboard computers on the SUAS such that the SUAS control is distributed to individual SUAS rather being performed by a central GCS. The SUAS could also communicate among themselves to request a replacement when their battery levels are low. This would simplify the software and reduce the bandwidth requirement by not having all communication data going through a central GCS. The bandwidth requirement could also be further reduced by having local recording on the SUAS and only streaming on demand. The videos could then be downloaded to a central storage location during charging. The limitation to this solution is that the endurance of the SUAS may be reduced due to added load and increased processing power that draws more electrical power.

D. SUAS SURVIVABILITY

In terms of system survivability, the SUAS is susceptible to hostile attacks either by kinetic weapons like small arms or by electronic weapons. It is assumed, however, that the intruder does not want to be detected and it may be challenging to attack a small and fast moving SUAS at a distance. Even if it is possible, a replacement drone could be launched to the last known location within minutes and security forces could also be deployed to the location to suppress the threat. While shielding on SUAS could potentially minimize the vulnerability of the SUAS, it may not be as cost effective as having more redundant SUAS.

Furthermore, the SUAS could also be subjected to bad weather conditions, which may impact its ability to fly. As such, weather resistant SUAS should be implemented to ensure that surveillance could be performed in all weather conditions.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSION AND RECOMMENDATIONS

A. CONCLUSION

In this thesis, the challenge of persistent surveillance in the area of perimeter security was discussed. A set of multi-rotor UAS to provide persistent surveillance through sequential swapping was proposed and a proof of concept was developed to evaluate the solution feasibility. The proof of concept was conducted at Impossible City and Camp Roberts in California. The results showed that the SUAS could be swapped autonomously and was able to maintain consistent FOV during the swap. The FOV during transition could be further improved if higher accuracy GPS sensors were used.

The system of SUAS proved to be a feasible solution to provide persistent surveillance. By monitoring the battery level and autonomously replacing the depleted SUAS via a GCS, the system showed that the endurance can exceed the capability of a single SUAS and possibly perform 24/7 persistent surveillance or until the system mechanically fails.

Certain limitations were identified in the system. One of which was that the number of SUAS increases 3.5 times if depleted batteries are not replaced with new ones. This is due to the 90-minute long charging time for a battery. During SUAS swap, the fields of view can also be inconsistent when the patrolling SUAS is turning, as seen in Figure 19. This can be improved by flying in a straight path (front, back, left, and right) without turning.

Furthermore, the SUAS is susceptible to attacks. This threat could be overcome, however, by having a redundant SUAS available to replace the airborne SUAS when it is compromised. A swarm of SUAS can also be deployed if a surveillance gap during downtime is an issue for the mission. The cost of having a swarm of SUAS versus having countermeasures to reduce the vulnerability of the SUAS should be analyzed to determine the more cost-effective solution.

The solution proposed can also be extended to other surveillance or target tracking tasks where persistent surveillance or targeting is required. The autonomous operations could relieve security personnel from such mundane tasks and allow them to focus on more high value work.

B. FUTURE WORK

In testing of the SUAS, a few potential challenges were identified, and thus, the following future work is recommended:

(1) Determine whether the SUAS should be launched centrally or distributed based on the flight pattern.

(2) Determine the feasibility of tracking moving targets during SUAS swapping.

(3) Develop a mechanism to replace batteries automatically.

(4) Conduct a 24-hour flight test to determine the feasibility of 24/7 operations.

(5) Develop an onboard computer for the SUAS to communicate among one another to perform mission task and swapping when required.

With further development on the operational effectiveness and suitability of this system, military forces could benefit in terms of mission effectiveness and manpower savings from this technology. Although the endurance of SUAS continues to improve, SUAS swapping to provide persistent surveillance will be crucial in providing the force deploying the system with a technological edge over its adversaries.

APPENDIX A. PYTHON SCRIPTS

```python
# Simplifying threading lock function
class Interface():
    def __init__(self):
        self._value = None
        self._lock = threading.Lock()

    def acquire(self):
        self._lock.acquire()

    def setandprint(self, newvalue):
        self._value = newvalue
        sleep(random()*0.1)
        print self._value
    def release(self):
        self._lock.release()

inter = Interface()

class SimpleLock(object):
    def __init__(self):
        self._lock = threading.Lock()

    def __enter__(self):
        self._lock.acquire()
        return self

    def __exit__(self, type, value, traceback):
        self._lock.release()
        return

inter_lock = SimpleLock()


# Function for arming and takeoff to set altitude
def arm_and_takeoff(vehicle,aTargetAltitude,vehiclename):

    print "Basic pre-arm checks"
    # Do not try to arm until autopilot is ready
    while not vehicle.is_armable:
        print " Waiting for vehicle to initialise..."
        time.sleep(1)


    print "Arming motors"
    # Copter should arm in GUIDED mode
    vehicle.mode = VehicleMode("GUIDED")
    vehicle.armed = True

    while not vehicle.armed:
        print " Waiting for arming..."
        time.sleep(1)

    print "Taking off! Heading to ",aTargetAltitude

    takeofftime = datetime.now()
    landedalt=vehicle.location.global_relative_frame.alt
    global sortie
    sortie+=1

    # For data logging
    storesortiedata(vehiclename,vehicle)

    # Wait until the vehicle reaches a safe height before
    # processing the goto (otherwise the command
    # after Vehicle.simple_takeoff will execute immediately).
```

```
    while True:
        vehicle.simple_takeoff(aTargetAltitude) # Take off to target altitude
        timeelapsed=((datetime.now()-takeofftime).total_seconds())
        print " Altitude: ", vehicle.location.global_relative_frame.alt, " Time Elapsed: ",timeelapsed, " sec"
        #Trigger just below target alt
        if vehicle.location.global_relative_frame.alt>=aTargetAltitude*0.95:
            print "Reached target altitude in ",timeelapsed," sec"
            status="success"
            break
        if timeelapsed>5 and vehicle.location.global_relative_frame.alt<landedalt+1:
            print "!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!Launch ERROR!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!"
            print "Land vehicle (%s)" %(vehiclename)
            vehicle.mode = VehicleMode("LAND")
            status="fail"
            break
        time.sleep(1)

    # For data logging
    storesortiedata(vehiclename,vehicle)

    return status



# Function to calculate the distance in meters between two position
def get_distance_metres(aLocation1, aLocation2):
    """
    Returns the ground distance in metres between two LocationGlobal objects.

    This method is an approximation, and will not be accurate over
    large distances and close to the earth's poles. It comes from the
    ArduPilot test code:
    https://github.com/diydrones/ardupilot/blob/master/Tools/autotest/common.py
    """
    dlat = aLocation2.lat - aLocation1.lat
    dlong = aLocation2.lon - aLocation1.lon
    return math.sqrt((dlat*dlat) + (dlong*dlong)) * 1.113195e5


# Function to store data during testing
def storesortiedata(vehiclename,vehicle):
    timenow = datetime.now()
    global sortie

    sortiename=vehiclename
    sortiedata[sortiename].append(
        (timenow,
            vehicle.location.global_relative_frame.lat,
            vehicle.location.global_relative_frame.lon,
            vehicle.location.global_relative_frame.alt,
            vehicle.battery.level,
            vehicle.battery.voltage,
            sortiename
            ))

    print "Stored data for %s in dictionary"%(vehiclename)


# Function to write data to csv file and plot data
def showplot():
    if len(sortiedata)>0:
        fig=plt.figure()
        fig.suptitle('Battery Management Testing')
```

```
    plt1=fig.add subplot(211)
    plt2=fig.add subplot(212)
    fig2=plt.figure()
    plt3=fig2.add subplot(111,projection='3d')

    # Open csv file and append
    f = open('data.csv','a')

    fulltime=[]
    fulllatitude=[]
    fulllongitude=[]
    fullaltitude=[]
    fullbatterypct=[]
    fullbatteryvolt=[]
    fullsolo=[]

    global starttime

    for key, value in sorted(sortiedata.iteritems()):
        soloid=key
        time=[]
        latitude=[]
        longitude=[]
        altitude=[]
        batterypct=[]
        batteryvolt=[]
        solo=[]

        for var in value:
            time.append((var[0]-starttime).total seconds())
            latitude.append(var[1])
            longitude.append(var[2])
            altitude.append(var[3])
            batterypct.append(var[4])
            batteryvolt.append(var[5])
            solo.append(var[6])

        plt1.plot(time,altitude,label=soloid)
        plt1.set xlabel('Time (sec)')
        plt1.set ylabel('Altitude (m)')
        plt2.plot(time,batterypct,label=soloid)
        plt2.set xlabel('Time (sec)')
        plt2.set ylabel('Battery Life (%)')
        plt3.plot(latitude,longitude,altitude)
        plt3.set xlabel('Latitude (m)')
        plt3.set xlabel('Longitude (m)')
        plt3.set xlabel('Altitude (m)')

        plt1.scatter(time[0],altitude[0],marker='o',color='g',s=50)
        plt1.scatter(time[-1],altitude[-1],marker='o',color='r',s=50)
        plt2.scatter(time[0],batterypct[0],marker='o',color='g',s=50)
        plt2.scatter(time[-1],batterypct[-1],marker='o',color='r',s=50)

        fulltime.extend(time)
        fulllatitude.extend(latitude)
        fulllongitude.extend(longitude)
        fullaltitude.extend(altitude)
        fullbatterypct.extend(batterypct)
        fullbatteryvolt.extend(batteryvolt)
        fullsolo.extend(solo)

    duration=math.ceil(fulltime[-1]/60)
    start=starttime.strftime('%Y%m%d %H%M')
    stoptime=var[0]
    stop=stoptime.strftime('%Y%m%d %H%M')
```

```
        flightdata=['Start: '+start,'Stop: '+stop,'Duration: '+`duration`+ ' minutes','Sortie Count: '+`len(sortiedata)`]

        # Add row titles
        fulltime.insert(0,'Time (sec)')
        fullsolo.insert(0,'Vehicle ID')
        fulllatitude.insert(0,'Latitude')
        fulllongitude.insert(0,'Longitude')
        fullaltitude.insert(0,'Altitude (m)')
        fullbatterypct.insert(0,'Battery (%)')
        fullbatteryvolt.insert(0,'Battery (V)')

        # Write to csv file
        w = csv.writer(f, delimiter=',')
        w.writerow(flightdata)
        w.writerow(fullsolo)
        w.writerow(fulltime)
        w.writerow(fulllatitude)
        w.writerow(fulllongitude)
        w.writerow(fullaltitude)
        w.writerow(fullbatterypct)
        w.writerow(fullbatteryvolt)

        print "############# Data stored to CSV #############"

        # Close csv file
        f.close()

        plt1.autoscale(enable=True, axis='both', tight=False)
        plt2.autoscale(enable=True, axis='both', tight=False)


# Initialize lists of available Solos (connect and names) set in
# connection (set empty here)
def connect_solo():
    global solo, soloname, soloid, i, vehicle

    i=0
    solo=[]
    soloname=[]

    print "======================= CONNECTING TO SOLOS ======================="



    #Loop through solo ports to attempt connection with each vehicle
    for soloport in soloports:

        index=soloports.index(soloport)
        soloid=soloids[index]

        print("Connecting to solo: %s (%s)" % (soloport,soloid))

        # Try to connect to the solo for XX heartbeat timeout seconds
        try:
            vehicle=connect(soloport, wait_ready=True, heartbeat_timeout=10)
            solo.append(vehicle)
            soloname.append(soloid)

            solos[soloid]=vehicle

            # Print some vehicle attributes
            print "Get some vehicle attribute values:"
            print " GPS: %s" % solo[i].gps_0
            print " Battery: %s" % solo[i].battery
```

```
            print " Last Heartbeat: %s" % solo[i].last_heartbeat
            print " Is Armable?: %s" % solo[i].is_armable
            print " System status: %s" % solo[i].system_status.state
            print " Mode: %s" % solo[i].mode.name

        # Increment 1 to look on the next solo port
            i+=1

        except:
            print (soloid+" vehicle not found.")


    print "////////////////////// END CONNECTING TO SOLOS //////////////////////"


    # Write which solos were found
    print "%s Solos available: " %len(solo)
    print soloname

# Go around perimeter
def loiter():
    global loiterindex, currentaltitude, loiteralt, loiterlat1, loiterlon1, vehicleindex, vehicle, loiterpoint, loitervehicle,loitername, shuttingdown, loitershuttingdown, battery_level_limit


    loiterpoint = LocationGlobalRelative(loiterlat1[loiterindex],loiterlon1[loiterindex], loiteralt)
    loitervehicle=vehicle
    loitervehicle.simple_goto(loiterpoint)
    fly='True'
    shuttingdown='n'
    loitershuttingdown='n'
    try:
        while True and fly=='True' and shuttingdown=='n':
            try:

                currentaltitude=loitervehicle.location.global_relative_frame.alt
                dist = get_distance_metres(loitervehicle.location.global_frame, loiterpoint)
                print "Loiter: Distance to go: %s Altitude goal: %s Current: %s"%(dist,loiteralt,currentaltitude)
                print 'Loiter: %s Battery limit: %s  Current battery: %s'%(soloname[vehicleindex],max(battery_level_limit,battery_level_critical_limit),loitervehicle.battery.level)
                time.sleep(1)
                if currentaltitude>loiteralt:
                    goalpct=1.02
                else:
                    goalpct=0.98
                # Check if UAS reached waypoint
                if dist < 2:
                    print 'Loiter: Within 2m of waypoint'

                    storesortiedata(soloname[vehicleindex],loitervehicle)

                    loiterindex += 1
                    if loiterindex==4:
                        loiterindex = 0
                        time.sleep(1)

                    if loitershuttingdown=='y':
                        print"Loiter: shutdown"

                        break
```

```python
                    loiterpoint = LocationGlobalRelative(loiterlat1[loiterindex],loiterlon1[loiterindex], loiteralt)
                    with inter_lock:
                        loitervehicle.simple_goto(loiterpoint)

                    print "Loiter: moving to Lat:%s, Lon:%s" %(loiterlat1[loiterindex], loiterlon1[loiterindex])

                if loitershuttingdown=='y':
                        break
            except KeyboardInterrupt:
                traceback.print_exc()
                fly='False'
                print "Exited"
    finally:
        go_home(loitervehicle, vehicleindex)
        print"Loiter shutting down..."

        time.sleep(1)
# Function to go back launch point and land
def go_home(vehiclehome, vehiclehomeindex):
    global homelat, homelon, home, loiteralt, soloname, vehicleindex, battery_level_limit, battery_level_critical_limit, vehicle

    print "Return to launch"
    currentaltitude=vehiclehome.location.global_relative_frame.alt
    home = LocationGlobalRelative(homelat[vehiclehomeindex],homelon[vehiclehomeindex], currentaltitude)
    vehiclehome.simple_goto(home)

    while True:
        currentaltitude=vehiclehome.location.global_relative_frame.alt
        dist = get_distance_metres(vehiclehome.location.global_frame, home)
        if vehiclehome.location.global_relative_frame.alt<=2:
            print"Home: Vehicle already landed"
            break
        print "Home: Distance to go: %s Altitude goal: %s Current: %s"%(dist,loiteralt,currentaltitude)
        print 'Home:  %s Battery limit: %s  Current battery: %s'%(soloname[vehiclehomeindex],max(battery_level_limit,battery_level_critical_limit),vehiclehome.battery.level)
        time.sleep(1)
        if vehiclehome.location.global_relative_frame.alt<=2:
            print"Home: Vehicle already landed"
            break
        if dist < 1:
            print "Home: Land vehicle (%s)" %(soloname[vehiclehomeindex])
            vehiclehome.mode = VehicleMode("LAND")
            while True:
                print "Home: Landing %s Altitude: %s" %(soloname[vehiclehomeindex],vehiclehome.location.global_relative_frame.alt)
                time.sleep(1)
                if vehiclehome.location.global_relative_frame.alt<=2: #Trigger just below target alt.
                    print "Home: Vehicle On Deck"
                    break



#Function to takeoff UAS
def take_off(vehicleto, vehicleindexto,takeoffaltto):
    global takeoffstatus, takeoffalt, soloname
```

```
    try:
        cmds = vehicleto.commands
        cmds.download()
        cmds.wait_ready()

        # Arm and take off - receive status
        takeoffstatus=arm_and_takeoff(vehicleto,takeoffaltto,soloname[vehicleindexto])

        # If launch fail
        if takeoffstatus=="fail":
            print "Launch failed, vehicleindex=%s" %(vehicleindexto)


    except:
        if takeoffstatus=="fail":
            print "Take off aborted, vehicleindex=%s" %(vehicleindex)
#Function to check UAS status
def check_vehicle(checkindex):
    global continuecheck, solo, takeoffstatus, soloname, unavailablevehicles, vehicle, sortienum, battery_level_limit, battery_level_critical_limit, check_status

    try:

        print "Check index=%s" %(checkindex)

        #check if the drone's battery is low
        if solo[checkindex].battery.voltage < battery_volt_limit or solo[checkindex].battery.level < battery_level_critical_limit:
            print 'Vehicle Check: %s battery too low for operation, skip to next' %(soloname[checkindex])
            unavailablevehicles+=1
            check_status='n'
        else:
            check_status='y'
            sortienum+=1
            return checkindex




    except KeyboardInterrupt:
        print"Keyboard interrupted"
        unavailablevehicles=0
        check_status='n'
#Function to fly standby UAS for swap
def replace():
    global vehicle, vehicle2, vehicleindex, vehicleindex2, solo, takeoffalt1, soloname, homelat, homelon, takeoffstatus, loitervehicle, shuttingdown, checkindex
    global replacementfails, replacestop, vehicle2status, replaceshutdown
    replaceshutdown='n'
    try:
        while True and shuttingdown=='n' and replaceshutdown=='n':
            if vehicle.battery.voltage < battery_volt_limit or vehicle.battery.level < battery_level_limit or vehicle.battery.level < battery_level_critical_limit:
                while True:
                    if len(solo)==1:
                        print "Replace: No standby available"
                        shuttingdown='y'
                        break

                    vehicleindex2=vehicleindex+1
                    if vehicleindex2>=len(solo)and vehicleindex!=0:
                        vehicleindex2=0
```

```
    check vehicle(vehicleindex2)
    if check status=='y':

        vehicle2=solo[vehicleindex2]
        unavailablevehicles=0
        break
    else:
        if vehicleindex2>=len(solo):
            print "Replace: No standby available"
            replaceshutdown='y'
            break
        vehicleindex2+=1
        unavailablevehicles+=1
        if vehicleindex2>=len(solo) and vehicleindex!=0:
            vehicleindex2=0
        if unavailablevehicles>len(solo)-1:
            print "Replace: No standby available"
            replaceshutdown='y'
            break
if shuttingdown=='y'or replaceshutdown=='y':
    break
print 'Replace: Launch replacement'
print 'Replace: Launch %s' %soloname[vehicleindex2]

replacestop='n'
while True and replacestop=='n':

    take off(vehicle2, vehicleindex2, takeoffalt1)
    takeoffcheckr=0
    if takeoffstatus=="fail":
        takeoffcheckr+=1
        vehicleindex2+=1
        if vehicleindex2>=len(solo) and vehicleindex!=0:
            vehicleindex2=0
        vehicle2=solo[vehicleindex2]
    elif takeoffcheckr==len(solo):
        print"Replace: No drones available for takeoff"
        shuttingdown='y'
        go home(loitervehicle, vehicleindex)
        break
    else:

        print "Replace: Set vehicle airspeed to 40"
        vehicle2.airspeed = 40
        vehicle2status=1
        if vehicle2.location.global relative frame.alt<=10:
            print 'Replace: Altitude too low, return to launch point'
            go home(vehicle2,vehicleindex2)
            replaceshutdown='y'
            break
        with inter lock:
            storesortiedata(soloname[vehicleindex],vehicle)
            storesortiedata(soloname[vehicleindex2],vehicle2)

        if vehicleindex2>len(homelat)-1:
            print "Adding home point for %s" %(soloname[vehicleindex2])
            with inter lock:
                cmds = vehicle2.commands
                cmds.download()
                cmds.wait ready()
            with inter lock:

                homelat[vehicleindex2]=(vehicle2.location.global frame
                .lat)

```

```
                                homelon[vehicleindex2]=(vehicle2.location.global frame.lon)

                            print 'Home: (%s,%s)' %(homelat[vehicleindex2],homelon[vehicleindex2])

                        else:
                            print "Home already exists, ignoring!"


                        while True and replacestop=='n':
                            with inter lock:

                                vehicle2.simple goto(loitervehicle.location.global frame)

                                currentaltitude=vehicle2.location.global relative frame.alt

                            time.sleep(5)






            else:
                if len(solo)==1:
                    print "Replace: No standby available"
                    replaceshutdown='y'
                else:
                    vehicleindex2=vehicleindex+1
                    if vehicleindex2>=len(solo)and vehicleindex!=0:
                        vehicleindex2=0
                    check vehicle(vehicleindex2)
                    if check status=='y':

                        vehicle2=solo[vehicleindex2]
                        unavailablevehicles=0
                        print "Replace: %s standby for replacement" %(soloname[vehicleindex2])
                    else:
                        vehicleindex2+=1
                        unavailablevehicles+=1
                        if vehicleindex2>=len(solo)and vehicleindex!=0:
                            vehicleindex2=0
                        if unavailablevehicles>len(solo)-1:
                            print "Replace: No standby available"
                            replaceshutdown='y'
                            break
                time.sleep(5)
    finally:
        print"Shutting down replace"
#Function to coordinate swap and land back UAS
def main controller():
    global vehicle, continuecheck, battery level critical limit, unavailablevehicles, sortienum, takeoffstatus, loiterpoint, loiteralt, vehiclecheck, shuttingdown
    global loiterlat1, loiterlon1, solo, soloid, soloids, soloname, soloports, solos, sortie, sortiedata, sortienum, takeoffalt, takeoffalt1, loitername, loitervehicle
    global takeoffstatus, vehiclealtseparation, vehicleindex, vehicleindex2, vehicle2, index, loiterindex, currentaltitude, homelat, homelon, dist, dist2, homedist
    global battery_level_limit, home, loitershuttingdown, unavailablevehicles,
```

```
replacementfails, replacestop, vehicle2status, replaceshutdown
try:
    shuttingdown='n'
    while True and shuttingdown=='n':
        connect solo()
        # A check to see if the script should continue running in loop to follow
        continuecheck = 1
        # Ask user whether to continue where 'y' and 'n' are only acceptable
        answers
        while continuecheck != 'y' and continuecheck != 'n':
            continuecheck = raw input('Continue? (y/n):')
        if continuecheck =='n':
            shuttingdown='y'
            break
        # Create new dictionary variables for home lat and longs
        homelat=defaultdict(list)
        homelon=defaultdict(list)
        #initialize all variables
        vehicle=0
        sortienum=-1
        unavailablevehicles=0
        loiterindex=0
        takeoffstatus="true"
        battery level limit=0
        replacementfails=0
        vehicle2status=0
        replaceshutdown='n'
        try:
            # Initialize time for data collection
            starttime = datetime.now()
            vehicleindex=0
            vehicleindex2=1

            while True:

                check vehicle(vehicleindex)
                if check status=='y':
                    vehicle=solo[vehicleindex]
                    loitername=soloname[vehicleindex]
                    unavailablevehicles=0


                    # If vehicle is not already airborne (likely same vehicle)
                    if vehicle.location.global relative frame.alt <
                    takeoffalt-1:

                        takeoffcheck=0
                        while True:
                            take off(vehicle, vehicleindex, takeoffalt)
                            if takeoffstatus=="fail":
                                takeoffcheck+=1
                                vehicleindex+=1
                                if vehicleindex>=len(solo)-1:
                                    vehicleindex=0
                                vehicle=solo[vehicleindex]
                            elif takeoffcheck==len(solo):
                                print"Main controller: No drones available for
                                takeoff"
                                shuttingdown='y'
                                break
                            else:

                                vehicleindex2=vehicleindex+1
                                if vehicleindex2>=len(solo)-1:
                                    vehicleindex2=0
```

```
                    # Increasing vehicle airspeed to max (10)
                    print "Set vehicle airspeed to 10"
                    vehicle.airspeed = 1

                    # Set home location as the location above takeoff
                    # (for return later)

                    homelat[vehicleindex]=(vehicle.location.global fra
                    me.lat)

                    homelon[vehicleindex]=(vehicle.location.global fra
                    me.lon)

                    # Print for debug
                    print " Home Latitude: %s" % homelat[vehicleindex]
                    print " Home Longitude: %s" % homelon[vehicleindex]
                    b = threading.Thread(target=loiter)
                    b.start()
                    c = threading.Thread(target=replace)
                    c.start()
                    traceback.print exc()
                    break
        break
    else:
        vehicleindex+=1

        if vehicleindex>=len(solo)-1:
            vehicleindex=0
        if unavailablevehicles>len(solo)-1:
            print"No vehicle available"
            shuttingdown='y'
            break


while True and shuttingdown=='n':
    # If battery level below limit or voltage level below limit
    if loitervehicle.battery.voltage < battery volt limit or
    loitervehicle.battery.level < battery level limit or
    loitervehicle.battery.level < battery level critical limit:
        print 'Battery below limit'
        time.sleep(1)
        if replaceshutdown=='y':
            shuttingdown='y'

        if vehicle2status==1:


            currentaltitude=vehicle2.location.global relative frame.alt

            dist2 =
            get distance metres(vehicle2.location.global frame,
            loitervehicle.location.global frame)
            print " Home: Distance to replacement drone: %s Altitude
            goal: %s Current: %s"%(dist2,loiteralt,currentaltitude)

            if dist2 < 10:
                print 'Within replacement distance'
                with inter lock:
                    rvehicle=loitervehicle
                    loitervehicle=vehicle2

                    replacestop='y'
                    vehicleindex=vehicleindex2
                    rvehicleindex=vehicleindex2
```

```
                                        vehicleindex2+=1

                                    if vehicleindex2>len(solo) and vehicleindex!=0:
                                        vehicleindex2=0
                                    go home(rvehicle, rvehicleindex)




                        else:
                            with inter lock:
                                print"Main Controller monitoring battery status"
                                print ' %s Battery limit: %s  Current battery:
                                %s'%(soloname[vehicleindex],max(battery level limit,batter
                                y level critical limit),loitervehicle.battery.level)

                            time.sleep(5)


                    storesortiedata(soloname[vehicleindex],vehicle)

                except:

                    traceback.print exc()
                    # When loop is manually exited (CTRL+C)
                    print 'Main Controller terminated'
                    go home(loitervehicle, vehicleindex)
                    loitershuttingdown='y'
                    shuttingdown='y'



        finally:
            traceback.print exc()
            print"Main controller shutting down"
            loitershuttingdown='y'
            shuttingdown='y'
            replaceshutdown='y'

shuttingdown='n'
# Start the first thread in the beginning of program
a = threading.Thread(target=main controller)
a.start()
```

APPENDIX B. SCREENSHOTS OF REPLACEMENT FIELD OF VIEW

The pictures on the left are the patrolling SUAS field of view while those on the right are the swapping SUAS field of view.

Test 1: Square Loop Swap 1, Impossible City

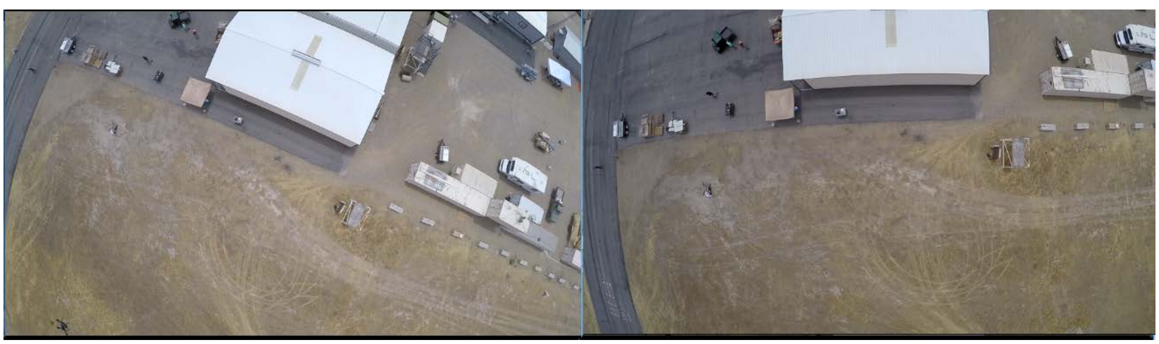
Test 2: Square Loop Swap 2, Impossible City

Test 3: Square Loop Swap 3, Impossible City

Test 4: Straight Line Swap 1, Impossible City

Test 5: Straight Line Swap 2, Impossible City

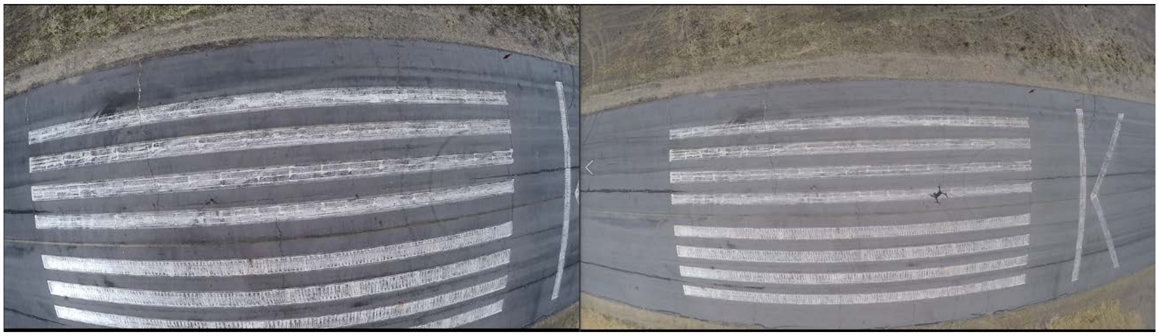
Test 6: Straight Line Swap 3, Camp Roberts



Test 7: Straight Line Swap 4, Camp Roberts

Test 8: Straight Line Swap 5, Camp Roberts

Test 9: Straight Line Swap 6, Camp Roberts



Test 10: Square Loop Swap 4, Impossible City

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

Bethke, Brett. 2007. "Persistent Vision-Based Search and Track Using Multiple UAVs." Master's thesis, Massachusetts Institute of Technology. http://hdl.handle.net/1721.1/40392.

Chin, Chee Keen. 2011. "Extending the Endurance, Missions and Capabilities of Most UAVS Using Advanced Flexible/Ridged Solar Cells and New High Power Density Batteries Technology." Master's thesis, Naval Postgraduate School. https://calhoun.nps.edu/bitstream/handle/10945/5824/11Mar_Chin.pdf?sequence=1&isAllowed=y

Clark, Brett M. 2013. "Small Unmanned Aerial Vehicles; DHS's Answer to Border Surveillance Requirements." Master's thesis, United States Army War College. http://www.dtic.mil/get-tr-doc/pdf?AD=ADA589119.

Defense Advanced Research Projects Agency. 2017. "Tern." Accessed January 5, 2018. https://www.darpa.mil/program/tern.

Fuentes, Gidget. 2018. "First Marine Battalion Gets 'Eyes in the Sky' Mini-UAS, Aviation." U.S Navy Institute. Last modified March 6, 2018. https://news.usni.org/2018/03/06/first-marine-battalion-gets-eyes-sky-mini-UAS

Gettinger, Dan. 2018. *Summary of UAS Spending in the FY 2019 Defense Budget Request*. Annandale-on-Hudson, NY: Center for the Study of the Drone at Bard College.

Gudmundsson, Snorri. 2016. "A Biomimetic, Energy-Harvesting, Obstacle-Avoiding, Path-Planning Algorithm for UAVs." PhD diss., Embry-Riddle Aeronautical University. https://commons.erau.edu/edt/306/.

Google Maps. 2018a. "Impossible City." Accessed Aug 21, 2018. https://www.google.com/maps/place/The+Impossible+City/@36.6201361,-121.7498285,170m/data=!3m1!1e3!4m5!3m4!1s0x0:0x903b429369d563ba!8m2!3d36.6203243!4d-121.7488801

Google Maps. 2018b. "McMillan Airfield / East Perimeter." Accessed Aug 21, 2018. https://www.google.com/maps/place/McMillan+Airfield+%2F+East+Perimeter/@35.7158858,-120.7648704,377m/data=!3m1!1e3!4m5!3m4!1s0x80ecd3ca0619c4f5:0x8ee6d45bd8f4929d!8m2!3d35.715083!4d-120.763032

Google Maps. 2018c. "Naval Postgraduate School." Accessed Aug 21, 2018. https://www.google.com/maps/place/Naval+Postgraduate+School/@36.5969368,-121.8764535,16.74z/data=!4m5!3m4!1s0x808e00e558298b37:0x972fe8117718bc7!8m2!3d36.5971462!4d-121.8740536

Google Maps. 2018d. "Correctional Training Facility." Accessed Aug 21, 2018. https://www.google.com/maps/place/Correctional+Training+Facility/@36.4711718,-121.3849725,654m/data=!3m1!1e3!4m5!3m4!1s0x54d106e56680d813:0x661d1de68c6de2e1!8m2!3d36.4708385!4d-121.3826394

Herzog, Julian. 2017. "Unmanned Aerial Vehicles, Plain Black SVG Icon." Last modified July 8, 2017. https://commons.wikimedia.org/wiki/File:Aerial_Photography_UAV_Icon.svg

Joshi, Divya. 2017. "Commercial Unmanned Aerial Vehicle (UAV) Market Analysis—Industry Trends, Companies and What You Should Know." Last modified August 8, 2017. http://www.businessinsider.com/commercial-uav-market-analysis-2017-8

Kohlhepp, Kimberly. 2018. "United States Marine Corps Orders 800 InstantEye Systems." InstantEye Robotics. Last modified June 2, 2018. https://instanteyerobotics.com/uncategorized/united-states-marine-corps-orders-800-instanteye-systems/

Koumadi, Koudjo M., Godfrey A. Mills, Abdul R. Ofoli, Moses Amoasi Acquah, and Robert A. Sowah. 2017. *Rotational Energy Harvesting to Prolong Flight Duration of Quadcopters.* (Technical report). Vol. 53. Institute of Electrical and Electronics Engineers, Inc. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7911303

Lee, Kian Seng, Mark Ovinis, T. Nagarajan, Ralph Seulin, and Olivier Morel. 2015. "Autonomous Patrol and Surveillance System Using Unmanned Aerial Vehicles." In *2015 IEEE 15th International Conference on Environment and Electrical Engineering (EEEIC)*, 1291–1297. IEEE.

Office of Aviation Policy and Plans. 2017. *FAA Aerospace Forecast: Fiscal Years 2018–2038*. Washington, DC: Federal Aviation Administration. https://www.faa.gov/data_research/aviation/aerospace_forecasts/media/fy2018-38_faa_aerospace_forecast.pdf

Rees, Caroline. 2017. "Hybrid Drone Sets Flight Endurance Record." Unmanned System News, Unmanned Systems Technology. December 28, 2017. http://www.unmannedsystemstechnology.com/2017/12/hybrid-drone-sets-flight-endurance-record/

Reid, David. 2016. “US Military Using ‘UAS Buggies’ to Patrol Africa Camps.” CNBC. Last modified July 29, 2016. https://www.cnbc.com/2016/07/29/us-military-using-UAS-buggies-to-patrol-africa-camps.html.

Under Secretary of Defense Acquisition. 2014. *Unmanned Systems Integrated Roadmap, FY2013-2038*. Washington, DC. http://www.dtic.mil/dtic/tr/fulltext/u2/a592015.pdf

Whittle, Richard. 2012. “How It Works: Laser Beaming Recharges UAV in Flight.” *Popular Mechanics*. July 28, 2012. https://www.popularmechanics.com/flight/drones/a7966/how-it-works-laser-beaming-recharges-uav-in-flight-11091133/

Williams, Alexander G. 2017. “Feasibility of an Extended-duration Aerial Platform Using Autonomous Multi-rotor Vehicle Swapping and Battery Management.” Master’s thesis, Naval Postgraduate School. https://calhoun.nps.edu/bitstream/handle/10945/56847/17Dec_Williams_Alexander.pdf?sequence=1&isAllowed=y.

Zou, Jie Tong, Zheng Yan Pan, Dong Lin Zhang, and Rui Feng Zheng. “Integration of the Target Position Correction Software with the High Endurance Quadcopter for Search and Rescue Mission.” *Applied Mechanics and Materials* 764–765*, Modern Design Technologies and Experiment for Advanced Manufacture and Industry* (May 1, 2015): 713–717. http://search.proquest.com/docview/1701026958/.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California