Spring 2015

Ultra-Fast, Autonomous, Reconfigurable Communication System

Paul Bupe Jr
Georgia Southern University

ULTRA-FAST, AUTONOMOUS, RECONFIGURABLE UAV COMMUNICATION SYSTEM

by

PAUL BUPE JR

(Under the Direction of Rami Haddad and Fernando Rios-Gutierrez)

ABSTRACT

The recent years have witnessed an increase in natural disasters in which the destruction of essential communication infrastructure has significantly affected the number of casualties. In 2005, Hurricane Katrina in the United States resulted in over 1,900 deaths, three million land-line phones disconnections, and more than 2000 cell sites going out of service. This incident highlighted an urgent need for a quick-deployment, efficient communication network for emergency relief purposes. In this research, a fully autonomous system to deploy Unmanned Aerial Vehicles (UAVs) as the first phase disaster recovery communication network for wide-area relief is presented. As part of this system, an automation algorithm has been developed to control the deployment and positioning of the UAVs based on a traditional cell network structure utilizing 7-cell clusters in a hexagonal pattern. In addition to the software algorithm, a fully functional control interface was developed which allowed for full control of the system both locally and over an internet connection. This system represents a novel approach for handling a large-scale autonomous deployment of a UAV communications networks.

INDEX WORDS: Unmanned aerial vehicle, Emergency services, Telecommunication network, Global Positioning System, Telemetry, Emergency disaster response

ULTRA-FAST, AUTONOMOUS, RECONFIGURABLE UAV COMMUNICATION SYSTEM

by

PAUL BUPE JR

B.S. Electrical Engineering, Georgia Southern University, 2013

A Thesis Submitted to the Graduate Faculty of Georgia Southern University in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE IN APPLIED ENGINEERING

STATESBORO, GEORGIA

ULTRA-FAST, AUTONOMOUS, RECONFIGURABLE UAV COMMUNICATION SYSTEM

by

PAUL BUPE JR

Major Professor:	Rami Haddad
Committee:	Fernando Rios-Gutierrez
	Frank Goforth

Electronic Version Approved:
May 2015

DEDICATION

This thesis is dedicated to my family, who have always been by my side through life and as I have been in school. A special thank you to my parents, Paul and Catherine Bupe, who have always been a great example for me and have sacrificed so much to ensure my success. I would also like to thank my sisters, Faith and Hope Bupe, and brother David Bupe for always being there for me.

ACKNOWLEDGMENTS

I would first like to thank my thesis advisors, Dr. Rami Haddad and Dr. Fernando Rios-Gutierrez, for their help and guidance over the past year and a half — this work would not be possible without them. I would also like to thank my thesis committee, Dr. Rami Haddad, Dr. Fernando Rios-Gutierrez, and Dr. Frank Goforth, for their time and input through the thesis process. I would especially like to thank Dr. Rami Haddad for his very meticulous and thorough editing of my thesis.

I would also like to acknowledge and thank Georgia Southern University and specifically the Electrical Engineering department for allowing me to conduct my research and providing the resources I needed to successfully complete my work, which was funded by the Georgia Southern University College of Engineering and Information Technology 2014 Seed Grant.

I would especially like to thank Andrew Michaud for his technical advice and helpful instruction and also for allowing me to take over his workspace while I was building thirteen unmanned aerial vehicles.

My sincere thanks to my friends and church family who have helped and invested so much in me.

Finally, I would like to thank God for giving me so many great opportunities, granting me the knowledge and wisdom to be successful in my studies, and for keeping me alive and well all these years.

Table of Contents

List of Figures

Chapter 1

INTRODUCTION

Instant communication, whether wired or wireless, has come to be a vital and integral part of everyday life. Enabling this communication is a wide network of towers and transmission lines covering most of the inhabited areas of the globe. During normal operating conditions, this interconnected system provides the necessary capability to support the daily communication needs of the population, encompassing personal, business, or emergency situations. One of the prominent features, and subsequently one of the main problems with this type of network is that it relies heavily on infrastructure. While advanced "wireless" devices like cellphones or other WiFi devices may give the illusion of being wireless and independent, they still rely on a wired infrastructure-based network; all of those cellphone towers are connected to a backhaul trunk. This creates a very vulnerable and critical point of failure.

Infrastructure-based networks tend to be susceptible to major damage by natural disasters and other catastrophic situations. One example of such a situation was Hurricane Katrina that hit the Gulf Coast in 2005. This hurricane was so destructive that it caused catastrophic damage to an area the size of Great Britain and was, in some respects, the equivalent of a weapon of mass destruction (Miller 2006). Hurricane Katrina delivered the most widespread critical infrastructure collapse of any advanced country since World War II (Miller 2006). This infrastructure collapse lead to a cascading of other failures which eventually lead to mass confusion and an inability for emergency personnel to respond effectively to the problem at hand. According to Miller (2006, 192), widespread infrastructure collapse is one of the marker elements that help differentiate "catastrophes" from "disasters", which was exactly the case in this situation. Hurricane Katrina is really a perfect example of the devastation that natural disasters can have on infrastructure on a very wide scale. This

infrastructure collapse occurred over a wide area rapidly and almost simultaneously – causing damage across multiple states (Miller 2006, 193).

The White House report (*The federal response to Hurricane Katrina : lessons learned.* 2006) on Katrina stated that 911 emergency services were debilitated, nearly three million customers lost phone service, and over 50,000 utility poles were destroyed in Mississippi alone. In addition, over 50 percent of area radio stations and 44 percent of television stations were put out of service (*The federal response to Hurricane Katrina : lessons learned.* 2006). According to the Federal Communications Commission (FCC) panel on Katrina, much of the backbone network for land lines was flooded out and cell towers were put out of commission (Miller 2006). The magnitude of the storm, Miller (2006, 193) states, was such that the local communications system wasn't simply degraded; it was, at least for a period of time, destroyed. Miller (2006, 194) goes on to cite the "Independent Panel Reviewing the Impact of Hurricane Katrina on Communications Networks" stating that Hurricane Katrina knocked out more than three million customer phone lines in Alabama, Louisiana, and Mississippi. Local wireless networks also sustained considerable damage – more than a thousand cell sites were knocked out of service by the hurricane.

In addition to the communications infrastructure, destruction of the transportation infrastructure was also prevalent. Destruction of roads and bridges meant that repair crews were delayed and the replenishment of supplies and fuel for generators was almost impossible.

A second example of a catastrophic disruption of telecommunications networks was the World Trade Center attack in 2001. It took just minutes for the telecommunications network to be overloaded. The attacks caused an overload of a phone switch with over 200,000 lines, 20 cellphone towers, and 9 TV broadcast stations (Noam 2004). All of the aforementioned issues lend evidence to the fact that

essential telecommunications systems need to be decentralized in order to prevent them from having a single point of failure.

Disaster Emergency Communications, according to the Federal Emergency Management Agency (FEMA), is a specialized field within the broader field of emergency communications (FEMA 2015). These emergency communications encompass all the technical means that emergency personnel use to conduct their daily communications. Disaster emergency communications, on the other hand, refers only to the technical means that will allow for "operable and interoperable communication before, during, and after presidentially declared emergencies, disasters, or planned National Special Security Events" (FEMA 2015). FEMA's Disaster Emergency Communications Division oversees Mobile Emergency Response Support (MERS) detachments. According to FEMA (2015), the MERS teams are used to:

- Meet the needs of the government emergency managers in their efforts to save lives, protect property, and coordinate disaster and all-hazard operations.
- Provide prompt and rapid multi-media communications, information processing, logistics, and operational support to Federal, State, and Local agencies during catastrophic emergencies and disasters for government response and recovery operations.

The MERS teams during Hurricane Katrina, though, had little impact during the first few days after the hurricane landed. *The federal response to Hurricane Katrina : lessons learned.* (2006) stated that "The complete devastation of the communications infrastructure left responders without a reliable network to use for coordinating emergency response operations."

1.1 Communication

There are many phases in emergency disaster relief encompassing activities such as search and rescue, construction of temporary shelter, and distribution of food supplies. At the core of all the stages of emergency disaster relief is a communications backbone. The United Nation's report on the 2010 Haiti disaster reaffirms the importance

of communication in disaster relief when it states that according to humanitarian organizations, good communication has proven to be essential after each major disaster of the modern era (Crowlely and Chan 2011). In addition to providing a very critical relief path in disaster scenarios, communications also help to connect and move logistical, rescue, and first responder resources (Fultron 2007, 1). Typically one of the very first steps during an emergency response is to set up a reliable means of communication. This typically involves the deployment of wireless communications. As earlier mentioned in the introduction, most wireless networks are actually based around a wired infrastructure backbone network. This means they are easily susceptible to damage by disasters. In many disasters this infrastructure is destroyed, such as during Hurricane Katrina, or is not available before the disaster. This leaves infrastructure-independent communications as the only viable option.

1.1.1 Satellite Communications

Satellite communication is currently the best infrastructure-independent communication system available for use anywhere on the planet. On a basic level, satellite communication works by relaying messages through an orbiting satellite between any number of devices, as shown in fig 1.1. There are two primary satellite types available for use in communications: geostationary satellites (GEO) and low Earth orbit (LEO) satellites (Fultron 2007, 1). These satellite orbits are illustrated in figure 1.2.

GEO Satellites

Located at an altitude of approximately $36,000km$, geostationary (GEO) satellites stay at a fixed position in relation to the earth. These satellites can cover up to a third of the earth at a time and are typically positioned above the equator. Infrastructure used with these GEO satellites ranges from the typical large satellite dish antennae to mobile satellite phones.

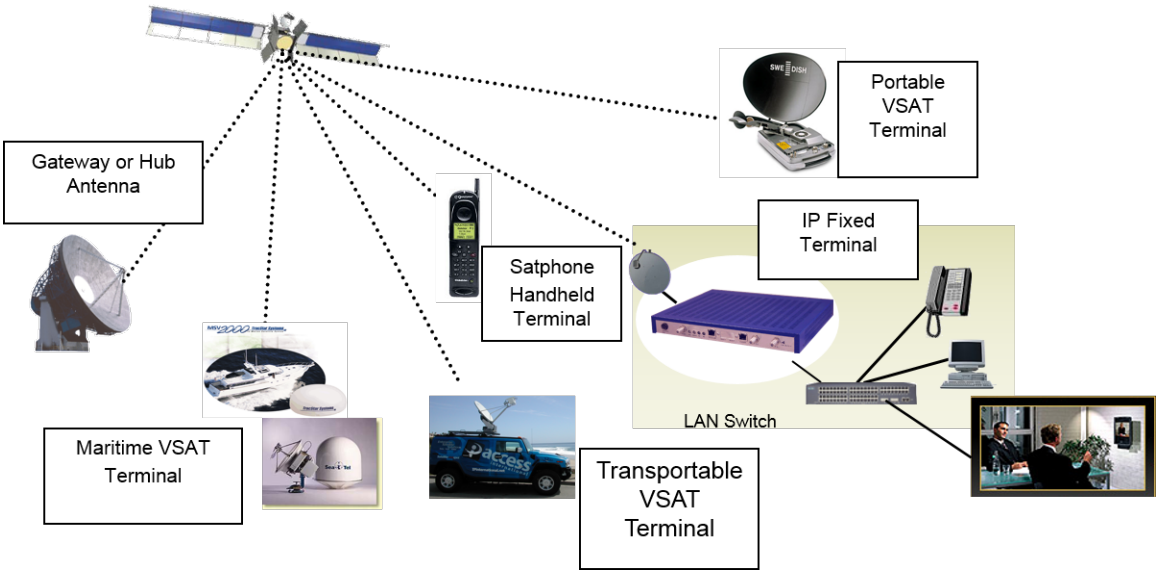


Figure 1.1. Satellite system. Source: Fultron 2007

GEO satellites are susceptible to an event called a sun outage, which is also referred to as sun fade or sun transit. Since the equator is offset by 22.5 degrees, the sun aligns with GEO satellites once in the spring and again in the autumn (Intelsat 2015). This phenomenon is displayed in figure 1.3. The thermal noise emitted by the sun covers all frequencies, including those utilized by the satellites. Interference occurs when the sun's thermal noise perfectly aligns with a GEO satellite and its corresponding receiver on earth. The noise created by this interaction is significant enough to cause a temporary loss of reception due to the fact that the receiver cannot distinguish between the sun's energy and the communication signal (Intelsat 2015). This intermittently over the course of two weeks starting with signal loss of a few minutes to a maximum of 24 minutes, depending on position, during peak outage time when the sun, earth, and satellite are perfectly aligned (Intelsat 2015).

Currently, there are almost 300 commercial GEO satellites in use (Fultron 2007, 1). These satellite networks are used constantly in many countries for providing data sensor to government agencies regarding flooding and seismic activity, broadcasting disaster warning notices, and facilitating communication between government

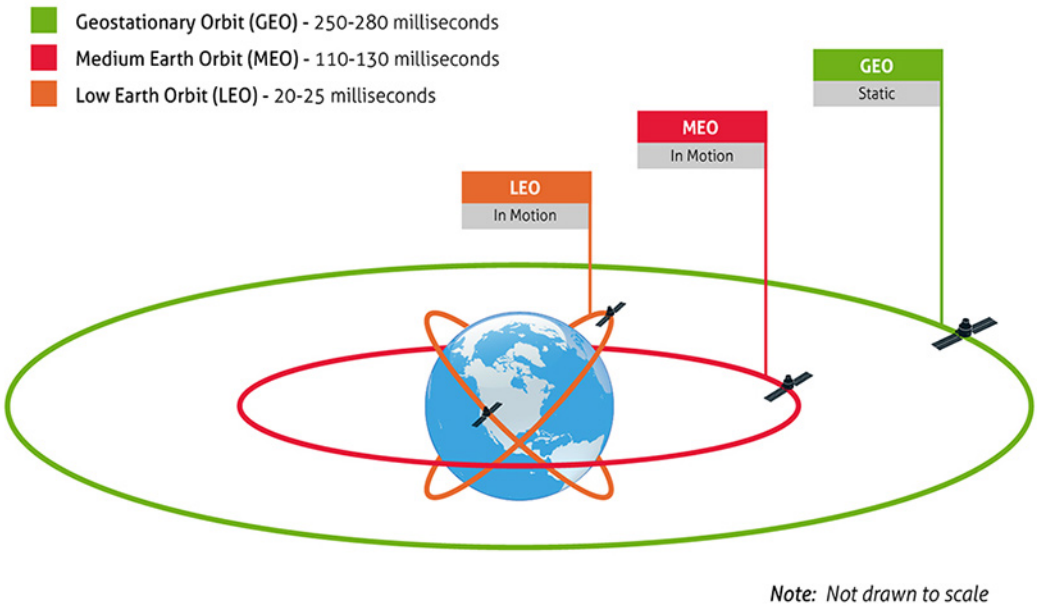


Figure 1.2. Satellite orbits. Source: Harris CapRock 2014

agencies, relief organizations, and the public. The main drawbacks of GEO satellites is that they have a very high cost of launching them into orbit, they have a very long link distance ($36,000km$), and as a result have large propagation delays (Fultron 2007, 2).

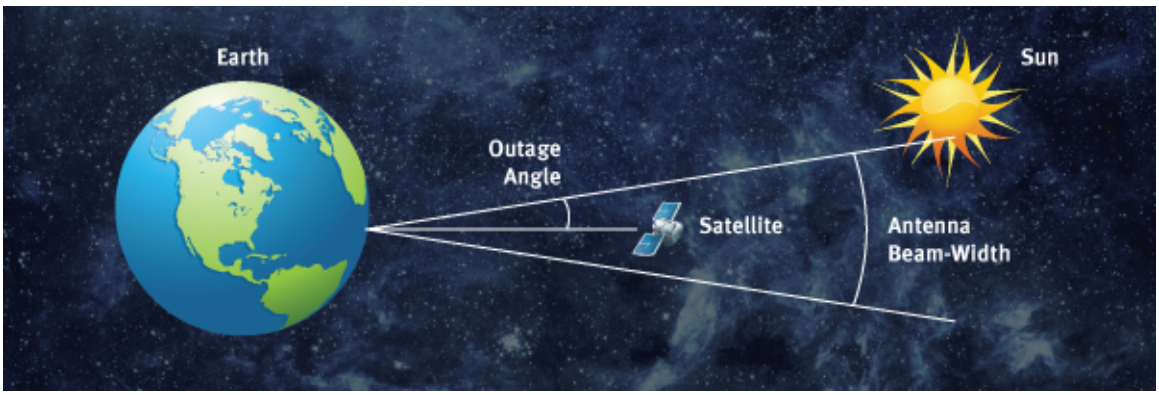


Figure 1.3. Sun outage scenario. Source: Adams 2015

Low Earth orbit (LEO) satellites operate at a much lower altitude in comparison to GEO satellites, between $780km$ and $1,500km$. LEO satellites are generally

free of the aforementioned issues of GEO satellites, having high bandwidth and very short propagation delay times. The primary disadvantage of LEO satellites is that unlike GEO satellites, they are not visible at all times from any single point on earth and thus are usually not utilized in "critical" applications.

Despite the aforementioned disadvantages, satellite communication is still an important part of emergency communication. The primary advantage of satellite communication is that it does not require any type of fixed infrastructure on earth to function. After Hurricane Katrina, satellite phones worked and were effective but they were very limited in quantity and eventually many ran out of battery (Miller 2006, 195). Satellite equipment is also not readily available and, for the more robust systems, takes a significant amount of time to set up and configure.

1.2 Multirotor Unmanned Aerial Vehicles

Multirotor UAVs are comparatively very simple devices in the aeronautics world. Their structure and dynamics are much simpler than conventional helicopters, making them less complex to control (Gyou et al. 2013, 1). The name of a multirotor UAV is typically related to the number of motors the UAV has. A quadrotor, for example, has four motors while a hexrotor has six motors. This nomenclature is used for as many motors as designed on the craft.

A quadrotor, the most basic multirotor UAV, has four rotors that run at very high speeds in opposing pairs. Figure 1.4 demonstrates this principle. In order for the UAV to achieve stable flight, the two motors on the x-plane have to spin in the opposite direction of the two motors on the y-plane. This creates vertical lift which then powers the craft.

The most basic quadrotor consists of a frame and propulsion system as aforementioned, a flight control computer, and a ground control system. The flight control computer handles all the in-flight logic including controlling the motors and the

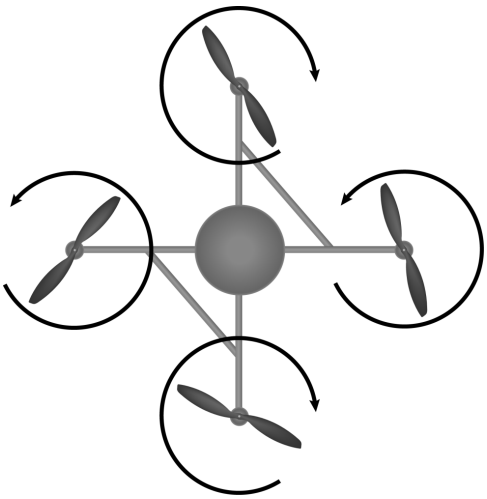


Figure 1.4. Quadrotor motor rotation direction for basic flight.

ground control systems is in constant communication with the quadrotor, sending commands and receiving feedback (Gyou et al. 2013, 2). Navigation of a quadrotor is done by varying the speed of the motors either individually or in pairs.

1.2.1 Stability and Efficiency

A common misconception around multirotor UAVs is that they are stable systems. While simpler in design, multirotors are less stable and less efficient than regular helicopters. Helicopters typically have two rotors, the main rotor and the tail rotor. The main rotor of a helicopter adjusts its pitch as it spins in order to vector its thrust, as illustrated in figure 1.5. This in combination with the trail rotor is what allows helicopters to maneuver (Gao 2013). Multirotors, on the other hand, use fixed-pitch blades that are directly coupled to motors. There are no rotor pitch adjustments that have to be made. The controls that go into operating the rotors on regular helicopters, though, are extremely complex —lending credence to the simplicity of multirotors.

This raises the issue of stability. Multirotors, quadcopters in particular, are inherently unstable and are less stable than regular helicopters. Due to this instability, they require a significant amount of electronic stabilization to fly. Flying an equiv-

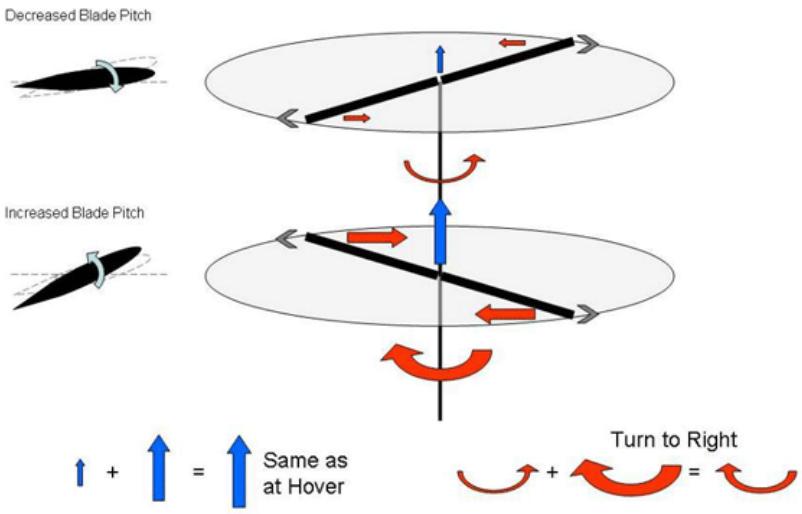


Figure 1.5. Helicopter rotor pitch adjustments. Source: Sim 2009

alently sized helicopter with the same electronic stabilization system of a multirotor would quickly prove that a helicopter is in fact more stable (Gao 2013). A quadrotor stays airborne by individually adjusting the thrust of each propeller (speeding up or slowing down the motor) in response to the calculations performed by the stabilization system. Since speeding up and slowing down each propeller takes time and energy, there is a time delay to how fast a multirotor can react. The bigger the propeller, the more energy it takes to change speed. On the other hand, helicopters simply need to change the pitch of their blades to change thrust, which requires significantly less energy (Gao 2013). This is one of the main reasons that multirotors are only feasible in relatively small sizes.

Efficiency

The biggest point of inefficiency for multirotors is having to continuously speed up and slow down the propellers to maintain stability, which is not a factor in regular helicopters. Larger and slower spinning propellers are in fact more aerodynamically efficient than small fast-spinning propellers (Gao 2013). This is true because, according to the formula for kinetic energy $\frac{1}{2}mv^2$, it takes four times more energy to move

a mass of air at twice the speed (Gao 2013). This means that a larger slower moving propeller such as that in a helicopter is more efficient than the smaller faster moving helicopter in a multirotor.

1.2.2 Conclusions

While it is true that helicopters are inherently more stable and efficient than quadrotors, it's important to note that this only applies when compared on a full-size scale. On a small scale, as multirotors are typically built, the mechanical simplicity of multirotors makes them much more desirable and useful than helicopters. The point of stability is also not an issue since multirotors are always equipped with stabilization. The simplicity of design of multirotors makes them ideal for swarm operations and also field operations due to the ease of maintenance.

1.3 Contributions

This work aims to provide a robust, quick to deploy GPS-based UAV platform that is able to serve as an Ad Hoc temporary emergency communications network. Imagine an emergency or disaster situation such as the aforementioned where the communications infrastructure is either completely destroyed or severely overloaded to the point of uselessness. With this Ad Hoc, any number of UAVs could be brought in to the affected area, connected to the Ad Hoc network via radio telemetry, and then after passing automatic flight checks would be deployed over the area with a single click of a button. The system would then control everything from positioning for maximum coverage, re-configuring if a UAV is added or removed, and also responding to any manual input from the operator via the provided interface. The main advantage of this system is that it is extremely fast to deploy. This is quite an important factor in emergency situations because it enables emergency responders to quickly establish reliable communication in an infrastructure-independent environment or heavily congested environments. The main contributions of this work are:

- The presentation of a quadcopter UAV platform upon which an Ad Hoc communications network can be deployed, regardless of protocol.
- A cross-platform application and algorithm to autonomously deploy and control any number of UAVs in a hexagonal cell pattern.
- An interface to effectively monitor and control a swarm of UAVs.

Chapter 2

LITERATURE REVIEW

2.1 Current Solutions

To date there have been very few practical solutions offered, outside of military applications, to the issue of immediate emergency disaster relief communication. When dealing with heavy congestion of cellular networks, the current approach is to bring in portable cell phone towers on trucks typically called Cell on Wheels (COW). These trucks, with telescoping poles and backup generators, can usually be found at venues and events that draw thousands of people, such as football games and other sporting events.

In addition to COWs, cell phone carriers also have trucks that carry emergency equipment that allow for satellite communication. While these trucks satisfy the need for infrastructure-free communication, they cost hundreds of thousands of dollars and are few in number (Richtel 2009). The main theme among all these applications involving COWs is that they are planned events. Most sporting events are planned months ahead of time and provisions are made to handle such special situations. Bringing in a COW to a disaster area could prove to be difficult due to the random nature of such occurrences, damage to the transportation infrastructure, or just the inability to navigate a large truck to a specific area.

There has been a recent effort to develop more robust and near instantaneous disaster relief communications systems. Two examples in particular have offered some solutions to this issue utilizing various wireless protocols.

2.2 Portable Satellite Backhauling Solution

A portable satellite backhauling solution for emergency communications was proposed by Estrem and Werner (2010). This particular system was to be used as part of the

e-Triage project. The e-Triage project is sponsored by the German Federal Ministry of Education and Research and is a system meant to electronically register victims and distribute information among the medical and control centers in the case of an emergency (Estrem and Werner 2010, 262). The overall goal was to make the triage process easier and much more reliable for emergency responders (Estrem and Werner 2010, 262).

The e-Triage communication system, displayed in figure 2.1, primarily consists of 3 parts which are of the actual triage end device, the distributed database containing victim information, and the communication infrastructure (Estrem and Werner 2010, 262). The end device can be something like a tablet or any device used by triage personnel to interface with the system. Esteem and Werner's paper concentrated on the technical aspects of this system and designing a portable solution that would encompass enabling global system for mobile communication with general packet radio service (GSM/GPRS) and wireless local area network (WLAN) coverage for first responders. The system was also designed to fit in a hand-luggage-size suitcase (Estrem and Werner 2010, 262). Requirements for the e-Triage communication system as described by Estrem and Werner (2010) were:

- Reliable and high bandwidth links with large coverage for voice and data in the disaster area.
- High bandwidth links between disaster-safe and disaster areas.
- Portable by single person, also in rough terrain, to almost inaccessible places.
- Fast deployment and reliable.
- Scalable in terms of the dimensions of the MCI.

To accomplish this task, WLAN, GSM/GPRS, and Terrestrial Trunked Radio (TETRA) were chosen as the communication protocols for voice and data. Voice was set to be transmitted over the User Datagram Protocol (UDP) due to transmission time being a higher factor over reliability and database traffic was transmitted over TCP (Estrem and Werner 2010, 262). UDP is typically used for audio and video

applications because it does not have transmission overhead. It doesn't have any connection overhead or need to retransmit lost packets with error checking.

Due to the broad nature of the system, two solutions were developed. The first of these was a portable, rapid deployment unit. This unit, though, did not offer all technologies and a high data rate due to its small form factor. The unit was designed to fit in a suitcase and as a result could not utilize full sized antenna and other more advanced communications equipment. The second solution was a unit installed in a command truck. This unit offered the full suite of technologies including a bigger coverage area and a higher bandwidth satellite link (Estrem and Werner 2010, 263). The first solution was, as earlier mentioned, portable and easy to deploy while the second solution was not portable and was slower to deploy. The order of deployment would be the portable unit first and then the command truck second.

The elements that comprised the communications equipment were the Emergency Communications Suitcase (ECS) and the Coordination Point Communications Equipment (CPCE). The ECS offered GSM/GPRS and WLAN and was connected to the disaster-safe area though Inmarsat-BGAN networks [Estrem 263]. Inmarsat is a British satellite telecommunications company that operates a satellite network. The CPCE offered GSM/GPRS, WLAN and TETRA as was connected through either proprietary VSAT or UMTS networks if available. This needed more time to be deployed but had more coverage and robustness (Estrem and Werner 2010, 262). WLAN and TETRA are more advantageous over GSM because they are still available when there is no external connectivity while GSM requires infrastructure with external connectivity.

The operation of the system begins in the disaster area with the communications equipment (ECS or CPCE) being deployed and positioned for maximum coverage. This is in conjunction with the end devices being provided to the rescue

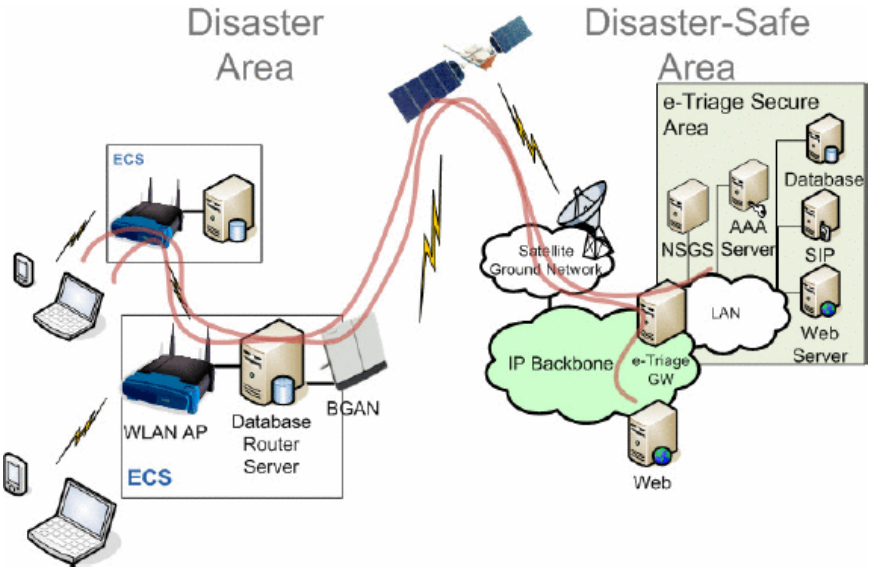


Figure 2.1. e-Triage system overview. Source: Estrem and Werner 2010

teams. The end devices communicate to the ECS or CPCE which then transmits the data via satellite to an offsite location which distributes the information to hospitals.

Overall Estrem and Werner (2010) presented a robust and useful system for medical personnel to more effectively register victims in disaster areas. The system demonstrated the importance of satellite technology in infrastructure-free environments. Esteem and Werner noted that the integration of satellites with terrestrial technologies is necessary to organize the re-establishment of a telecommunication infrastructure in a post-disaster situation (Estrem and Werner 2010, 269).

2.3 Emergency Communication System Based on IP and Airship

Another solution to disaster relief emergency communication was presented in by Huang, Yu, and Hu (2010) that consisted of a WLAN enabled airship as shown in figure 2.2. This system was capable of providing real-time audio and video services over Wi-Fi. Characteristics of this system included high bandwidth, large-scale coverage, rapid and flexible deployment, and certain immunity to most disasters due to flying at high altitudes (Huang, Yu, and Hu 2010, 1008).

For this system, a remote controlled airship (a blimp) was launched and used as a relay or wireless bridge. This was a low altitude airship that cruised at approximately 400m (Huang, Yu, and Hu 2010, 1009). The airship was acting as a bridge between a transmitter source and a mobile monitoring car. For testing, the airship was equipped with an outdoor carrier-grade wireless bridge, a power amplifier with 2 watts of output power, and an omnidirectional antenna (Huang, Yu, and Hu 2010, 1009). The airship antenna was mounted to the bottom of the airship. The wireless bridge itself operated in the 2.3-2.4 GHZ frequency band, utilized Orthogonal Frequency Division Multiplexing (OFDM) and supported point-to-point and point-multipoint links (Huang, Yu, and Hu 2010, 1009).

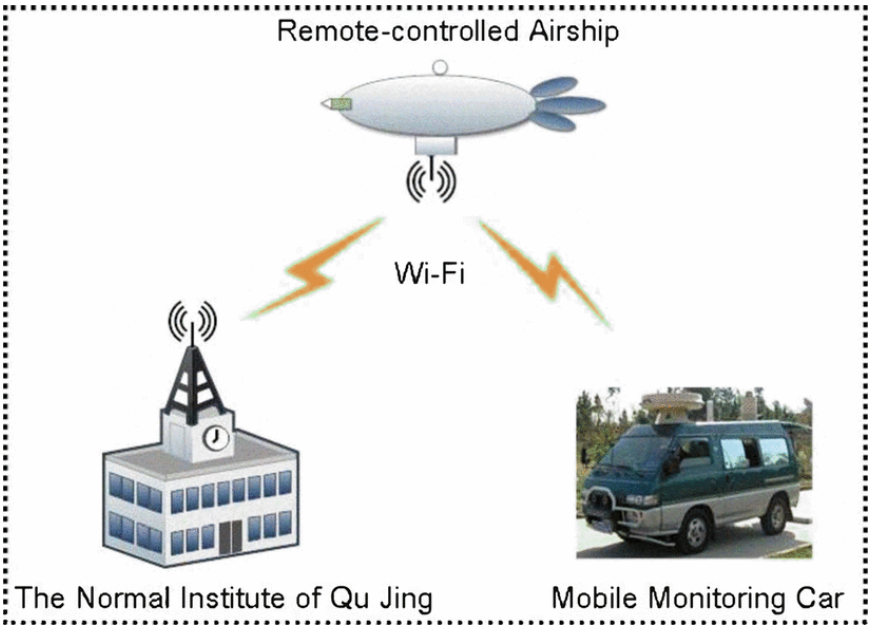


Figure 2.2. Airship communication system. Source: Huang, Yu, and Hu 2010

On the ground, another wireless bridge of similar specifications with a directional antenna was mounted on top of a building and connected to a local Wi-Fi network. Also on the ground was a vehicle that was equipped with monitoring and measurement equipment, another wireless bridge with a 2 watt amplifier, a broadband directional antenna, and access to global positioning system (GPS).

Huang, Yu, and Hu (2010) concluded that this type of system is very feasible. The main issue that was discovered was the fact that it was difficult to keep the airship stationary. This along with the use of directional antennas created communication issues that increased with distance.

2.4 Feasibility

The two aforementioned papers offer some intriguing solutions to the problem of infrastructure independent communication. The solution offered by Estrem and Werner (2010) for use with the e-Triage system comes built in with options for a number of communication protocols for use in the disaster area. These protocols all feed to a central satellite link which then sends the data off-site. One of the drawbacks of this system stems from it being a very specialized system designed for a specific application in disaster relief, dealing with medical personnel cataloging victims. The solution is tied to end devices which means that interfacing with the system from any other communication device would be difficult. The e-Triage system is also more focused on transmitting data one way into a database than with real time full duplex communication.

Huang, Yu, and Hu (2010)'s work falls more in line with the proposed UAV communications platform. Huang, Yu, and Hu (2010) essentially creates a roving Wi-Fi network using an airship. While going to the air is an improvement over the proposal by Estrem and Werner (2010), this solution falls short because it has no provisions for load balancing and does not scale. The airship acts as a stationary Wi-Fi repeater in the sky whose position greatly affects signal strength for both the transmitter and receiver. Also as mentioned in the paper, maintaining a fixed position with an airship proved to be an issue since the use of directional antenna meant that the position of the airship was an important factor in maintaining connectivity.

The proposed UAV communications platform addresses the outlined shortfalls of the aforementioned solutions. This UAV system is simply a communications platform. Unlike the works by Huang, Yu, and Hu (2010) and Estrem and Werner (2010), this system leaves the choice of protocol to the operator meaning that it can be used for almost any application. Since this system is based around quadrotor UAVs, positioning and load balancing is as trivial as modifying the control algorithm based on the protocol of choice. Unlike the airship solution, these UAVs are able to hold a position accurately and can quickly move to respond to changing variables. Lastly, the UAV platform proposed in this work scales very easily because the control algorithm can seamlessly add or remove UAVs to the system on the fly.

Chapter 3

PROPOSED UAV COMMUNICATION SYSTEM

The UAV communication platform as a whole contains three distinct layers: the hardware layer, the application layer, and finally the presentation layer as shown in figure 3.1. Between these three layers are two transport layers that utilize the MAVLink and WebSockets protocols. The hardware layer consists of the actual UAV, autopilot, and all associated hardware. The application layer consists of the control logic and algorithm. Finally, the presentation layer consists of the user-facing interface.

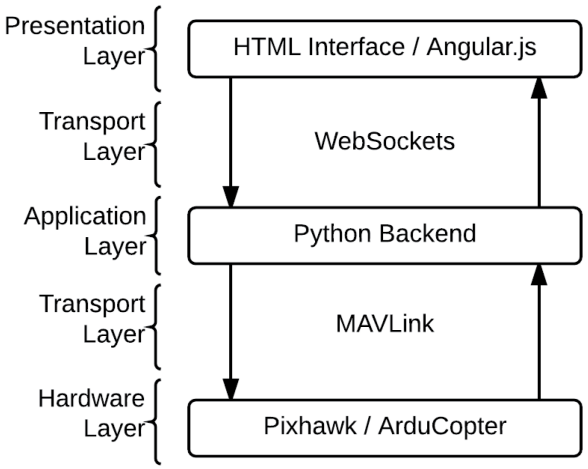


Figure 3.1. System Overview

3.1 Hardware

One of the important points to note about this work is that, barring a few compatibility criteria, the hardware is to be viewed as a black box. The hardware simply has to be compatible with the standard MAVLink protocol message library and have a working GPS. Compatibility with MAVLink in this case also means that the hardware has the standard sensors required in a UAV including an inertial measurement

unit (IMU), compass, and basic voltage monitoring. This hardware includes the actual UAV and any communication equipment that will be utilized for the required application.

The hardware layer of this system is the physical platform upon which a communications network would be built. The primary hardware used for this system consists of identical quadrotor UAVs assembled using custom parts from various manufactures, such as 3D Robotics, as opposed to being bought as a single product. In addition, they were assembled to be as minimalistic as possible, having only components and features essential for safe operation and flight. Each UAV was configured in a standard X-frame quadcopter design shown in figure 3.2. This type of configuration features two sets of brushless DC motors with counter-rotating propellers for a total of four motors. The fully assembled hardware for each UAV consisted of, in addition to the frame: a Pixhawk autopilot, 915 MHz telemetry radios by 3D Robotics (3DR), an RC receiver for manual control, a GPS unit by 3DR, and power monitoring circuitry.

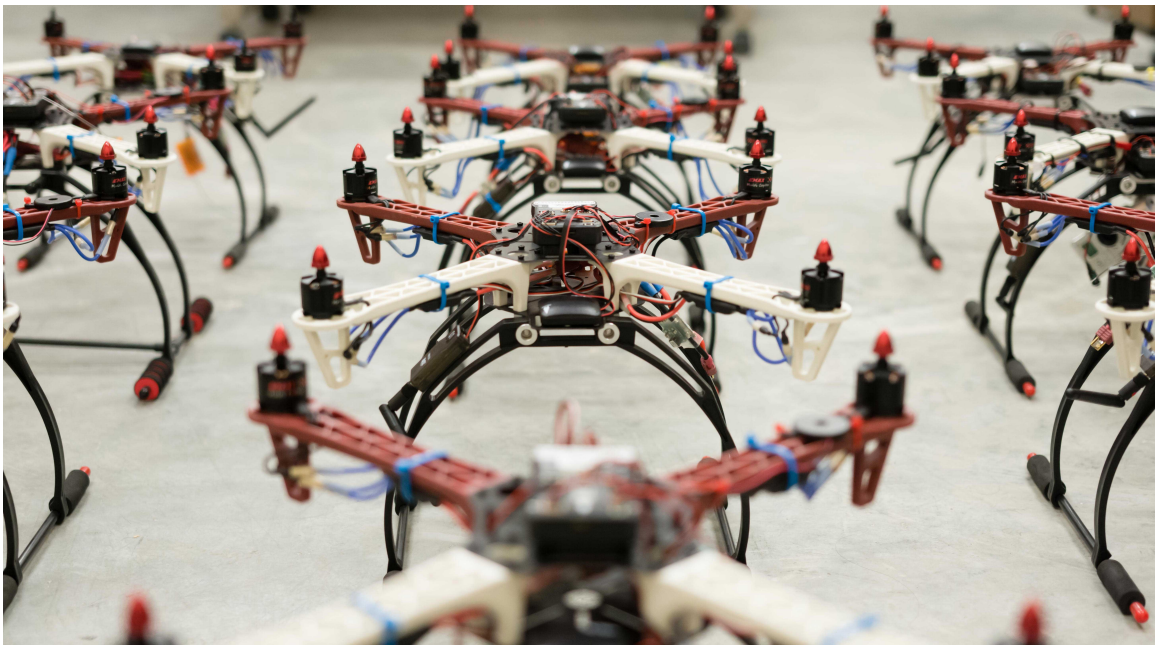


Figure 3.2. Assembled UAVs

3.1.1 Autopilot

The primary controller of each UAV was the Pixhawk autopilot system. This autopilot module, shown in figure 3.3, offers a complete flight stack running on powerful hardware. Partial specifications for the Pixhawk autopilot include a 32-bit ARM Cortex M4 processor equipped with a 32-bit STM32F103 fail-safe co-processor, a 3-axis 16-bit gyroscope, a 3-axis 14-bit accelerometer and magnetometer, a barometer, and a number of other redundant sensors and systems (*Pixhawk Autopilot System* 2015). In addition to these sensors, the Pixhawk also provides a number of ports, buses, and inputs / outputs. Running on top of the Pixhawk hardware is the open source ArduCopter flight controller (ArduPilot 2015). The ArduCopter flight controller was used because it is one of the most mature open source flight controllers and has a very active community.

Figure 3.3. 3DR Autopilot

In addition to the standard stabilization and manual controls found in most flight control software, ArduCopter also has a number of very useful autonomous features which make it ideal for this specific application. ArduCopter works primarily

through the use of modes. Modes, as the name suggest, put the flight controller into a number of predetermined operating conditions. The "Altitude Hold" mode, for example, allows the UAV to hold its current altitude regardless of external forces or manual inputs. ArduCopter has a number of important modes which are utilized in this system such as, *Loiter*, which allows the UAV to hold its position using GPS and altitude data, and *Return to Launch (RTL)*, which recalls the UAV to its launch coordinates and either hovers at a predetermined altitude or lands. ArduCopter also allows the UAV to take off and land autonomously. All these features, including the extremely thorough documentation, made ArduCopter the ideal flight control software to use in this system.

3.1.2 Telemetry

MAVLink is a very lightweight header protocol first released by Lorenz Meier in 2009 (*MAVLink Micro Air Vehicle Communication Protocol - QGroundControl GCS* 2015). MAVLink is the most used protocol among the open source UAV community. This protocol has only 8 bytes overhead per packet and comes with built-in packet-drop detection. The low overhead makes it quite idea for User Datagram Protocol (UDP) and UART/radio communication (Meier et al. 2011, 2994). The MAVLink protocol utilizes an XML-based common message library which can be compiled to other languages in order to facilitate compatibility among multiple platforms. While MAVLink does have a standard message library primarily distributed as a header-only C library, custom messages can be implemented through XML using a generator. The MAVLink protocol is licensed under LGPL, meaning it can be used royalty-free in both open and closed source applications. Radio telemetry operated in the 915 MHz ISM band through radio pairs developed by 3DR. These radios, shown in figure 3.4 have been specially developed for use with drones, both land and air-based, and come already optimized for handling MAVLink packets. These 3DR radios also utilize open

Figure 3.4. 3DR Telemetry Radio

source firmware and are open to modification. Each UAV was equipped with one of these radios which was then paired to another radio connected to the ground station. A number of factors guided the selection of a suitable system for radio telemetry, mainly regarding the handling of cross-radio interference and packet loss detection. The fact that MAVLink, as earlier mentioned, has built-in error checking alleviated the concern about packet loss, which left interference as the main consideration. These radios utilize frequency hopping spread spectrum (FHSS) as well as adaptive time division multiplexing (TDM), allowing for full duplex communication. This also allows for each radio set to transmit only to its paired counterpart within the 915 MHz range, giving up to 50 available channels in this industrial, scientific, and medical (ISM) band. While operating on a particular frequency within the band, the radios utilize Gaussian frequency-shift keying (GFSK) (ArduPilot 2015). While the firmware on the radios supports a multi-point setup (a many-to-one configuration) using individual radios allowed for more bandwidth for each UAV.

Since this system was developed around a central processing hub, the main limitation is that each UAV needs to be within radio range of the ground station, or receiving antenna, as opposed to a more traditional mesh network which requires each node to only be within range of another node. The primary implication of this

scenario is that the transmitting power of the radio has to be significant for long operating distances. Adjusting for range with the 3DR radios was a simple matter of configuring the duty cycle, data rate, and power level, which can range from $1dBm$ to $20dBm$ (ArduPilot 2015). Adjusting these parameters can allow for a range of a few kilometers without major hardware changes.

3.1.3 GPS

Since this is an outdoor system, having a reliable GPS was crucial; the system fully relied on GPS for localization. The 3DR ublox LEA-6H GPS module, displayed in figure 3.5, was used in this application. This GPS module, with a built-in compass, provides a fast high performance GPS with fairly high accuracy for a civilian system, which is especially useful for position-hold situations. A few important features of

Figure 3.5. 3DR GPS Module

this GPS are that it utilizes an active ceramic patch antenna, has a backup battery, extremely fast warm starts, and operates at a $5Hz$ refresh rate. Finally, this GPS module has an accuracy of $2.5m$ Circular Error Probability - meaning that it will be within a $2.5m$ radius of the true measurement at least 50% of the time (U-blox 2012, 2).

3.2 Software

The most important part of this system is the software-based control application. Portability and fast deployment are the overarching design ideas for this system. This implies that the operating software needs to be able to run on a multiple of platforms with minimal setup and compatibility issues. With that idea in mind, the traditional approach to such a problem had to be abandoned and re-imagined in a more tactical sense. Typically, this type of system would be implemented using a compiled language such as C or C++ due to their fast nature and overall time-tested reliability. The issue with using such a language, though, is that it is very system-dependent, not very cross-platform, and quite cumbersome to quickly modify due to it being a compiled language. Depending on which system they are running, compiled languages can also have very difficult to resolve dependency issues. These factors contributed to the shift of focus unto an interpreted language such as Perl or Python, from which Python was chosen as the primary language of choice for the system.

Python is an interactive object-oriented language that provides high-level data structures including lists and associative arrays (referred to as dictionaries), modules, classes, automatic memory management, and a host of other features (Sanner 1999, 3). One of the biggest draws to Python was that it features an extremely simple and friendly syntax. The primary factor for the choice of Python for the system, though, was that it can run on almost any modern computer, regardless of the operating system.

3.2.1 Python Application

The Python control application, also referred to as the server in this system, utilized a number of Python libraries (called packages). The most important package used in the development of the application was the *pymavlink* package. This package handled the very important task of implementing the MAVLink protocol in the Python

environment. Having this package allowed for the control application to communicate fully with the hardware through the serial port and the 3DR telemetry radios. The MAVProxy package, developed by Andrew Tridgell, was also important to the development of this application (*MAVProxy* 2015). MAVProxy provided very useful functions that assisted with interfacing the control application with the UAVs more efficiently.

The control application has three parallel main threads. The first thread is a hardware communication thread that listens for data coming from all the connected UAVs and sends commands to them as initiated by the operator or the control algorithm. Once a UAV is connected to the system by the operator, an instance of the *Drone()* class is instantiated and at the same time stored in a global *drones* list. This class encapsulates all the required data and also prepares a packet of that data ready to be sent to the HTML user interface (UI). This packet includes information such as GPS coordinates, altitude, estimated distances, and mode as shown:

```
    def packet(self):
        obj = {}
        obj["id"] = self.id
        obj["gps"] = [self.lat, self.lon]
        obj["gps_fix"] = "No GPS Fix" if self.fix_type != 3 else "GPS Fix"
        obj["sats"] = self.satellites_visible
        obj["home"] = self.home
        obj["alt"] = self.alt
        obj["alt2"] = self.config_alt
        obj["mode"] = self.flightmode
        obj["armed"] = "Disarmed" if self.isArmed is False else "Armed"
        obj["dest"] = self.destination
        obj["dist_to_home"] = self.dist_to_home
        obj["dist_to_dest"] = self.dist_to_dest
        obj["voltage"] = self.battery_voltage
        obj["waypoint"] = self.last_waypoint
        obj["batt"] = self.battery_remaining
        obj["port"] = self.port
        obj["state"] = self.state
        obj["link"] = "Link" if self.link_error is False else "Link Error"
        return obj
```

The *Drone()* class also subclasses the *Commands()* class which houses, as methods, all the system commands that are sent to the UAVs.

Within each command method is an implementation of the MAVLink protocol through *pymavlink*. Included in the command set are the three essential commands *takeoff(), goto()*, and *land()*, with the *takeoff()* and *goto()* commands shown below:

```
    def takeoff(self):
        self.mode("Stabilize")
        """Take off to the provided altitude"""
        alt = self.module.config_alt
        if alt is not None:
            altitude = float(alt)
            self.master.mav.command_long_send(self.master.target_system,
                                              self.master.target_component,
                                              mavutil.mavlink.MAV_CMD_NAV_TAKEOFF,
                                              0, 0, 0, 0, 0, 0, 0,
                                              altitude)
            logging.debug("Taking Off...")

    def goto(self, l):
        if l.is_relative:
            frame = mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT
        else:
            frame = mavutil.mavlink.MAV_FRAME_GLOBAL
        self.master.mav.mission_item_send(self.master.target_system,
                                          self.master.target_component,
                                          0,
                                          frame,
                                          mavutil.mavlink.MAV_CMD_NAV_WAYPOINT,
                                          2, 0, 0, 0, 0, 0,
                                          l.lat, l.lon, l.alt)
```

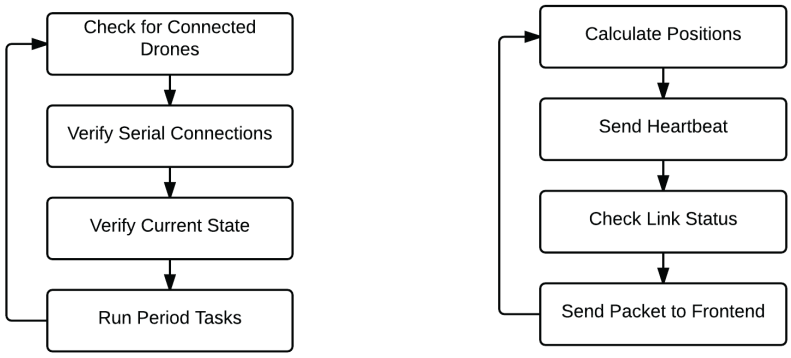
The second thread of the control application listens for commands from the HTML interface. Two Python packages were used for this communication. The first package was called *Flask*, and serves the important purpose of exposing the control application as a server over the HTTP protocol. The second Python package utilized was the *Flask-SocketIO* package. *Flask-SocketIO* worked in tandem with *Flask* to enable the control application to use WebSockets. This allowed the application to be accessible through a standard URL and thus could be accessed via WebSockets

— allowing for real-time full-duplex communication between the control application and the HTML interface (Bupe, Haddad, and Rios-Gutierrez 2015, 4). The third and central thread of the control application was the main loop thread that handled the autonomous deployment and positioning of all the UAVs. Within this thread was another loop that ran at $1Hz$. The main thread ran the primary control loop, illustrated in figure 3.6a. This loop handled all the administrative tasks and ran various system checks and tests. Within this loop was the second periodic loop, illustrated in figure 3.6b and shown below, whose purpose was to calculate the desired positions of the UAVs based on the number of connected UAVs and the system center.

```
def periodic_tasks(drones):
    """Periodic tasks"""
    global center
    if center is not None:
        if cluster_period.trigger(): # calculates destinations
            swarm.clusters(separation, center, drones)

    if heartbeat_send.trigger():
        for n in drones:
            n.commands.send_heartbeat()

    if heartbeat_check_period.trigger():
        for n in drones:
            n.check_link()

    if sendPeriod.trigger():
        packet = []
        for n in drones:
            packet.append(n.packet())
        socketio.emit('status', {'data': packet})
```

This loop also handled sending heartbeat messages to the connected UAVs and checking link status.

Autonomous control of the system was achieved primarily by the use of a finite-state machine (FSM). Using an FSM allowed for the control of the entire system to be broken down into smaller and more manageable states, which simplified the



(a) Drone Control Loop (b) Periodic Control Loop

Figure 3.6. Main Control Loops

overall codebase and made it more modular. Each connected UAV resulted in the instantiation of an FSM, which worked independently from the other FSMs. All the FSMs were monitored by a central process that then controlled the fleet based on the states of all the FSMs. The FSM was implemented in the Python application in an object-oriented manner using classes and objects as shown in figure 3.7.

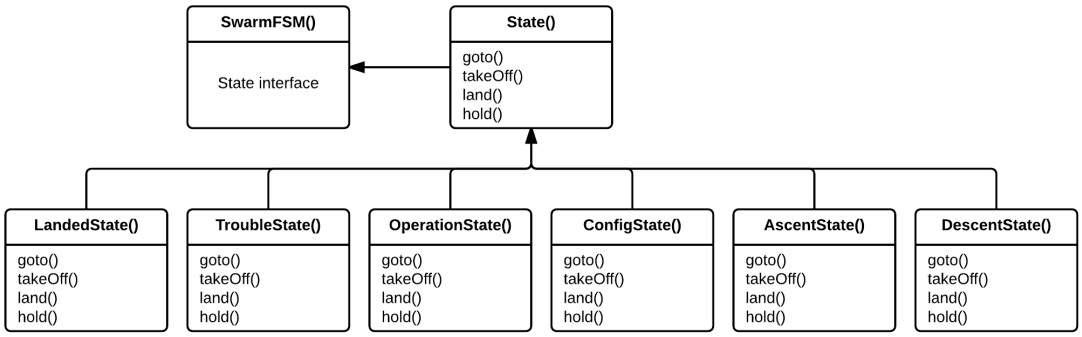


Figure 3.7. Finite-state Machine Implementation

On a high level, an FSM allows an object to alter its behavior when it's internal state changes. The object will then appear to change its class (Gamma et al. 1994, 338). The control FSM contained a total of six states:

- Landed

- Trouble
- Operation
- Configuration
- Ascent
- Descent

These states encompass all the operating conditions of the system. Each one of these states is represented by a unique class which inherits properties from a main state class. Because of that, these state classes all share the same methods even though the behavior of the methods is unique for each class. The class names are *LandedState(), TroubleState(), OperationState(), ConfigState(), AscentState(), and DescentState()*. Each of these classes has four common methods: *goto(), takeOff(), land(), and hold()*. Depending on the containing class, each one of these methods performs a different task in accordance with the state diagram in 3.8. For example calling the *takeOff()* method while in the *LandedState()* class will cause the UAV to begin the takeoff sequence and then transition the FSM to the Ascent state. On the other hand, a call to the same *takeOff()* method while in the *OperationState()* class will not have any effect since the UAV is already in the air and does not need to take off — no transition happens in this case.

Finite-State Machine Class

The main controlling class for the FSM is called *SwarmFSM*. The *SwarmFSM* class does all the heavy lifting in regard to the operation of the FSM. This class is initiated with instance variables that are actually objects of the corresponding state classes as shown:

```
1     def __init__(self, drone):
2         # State Machine Variables
3         self.landedState = LandedState(self, drone)
4         self.troubleState = TroubleState(self, drone)
```

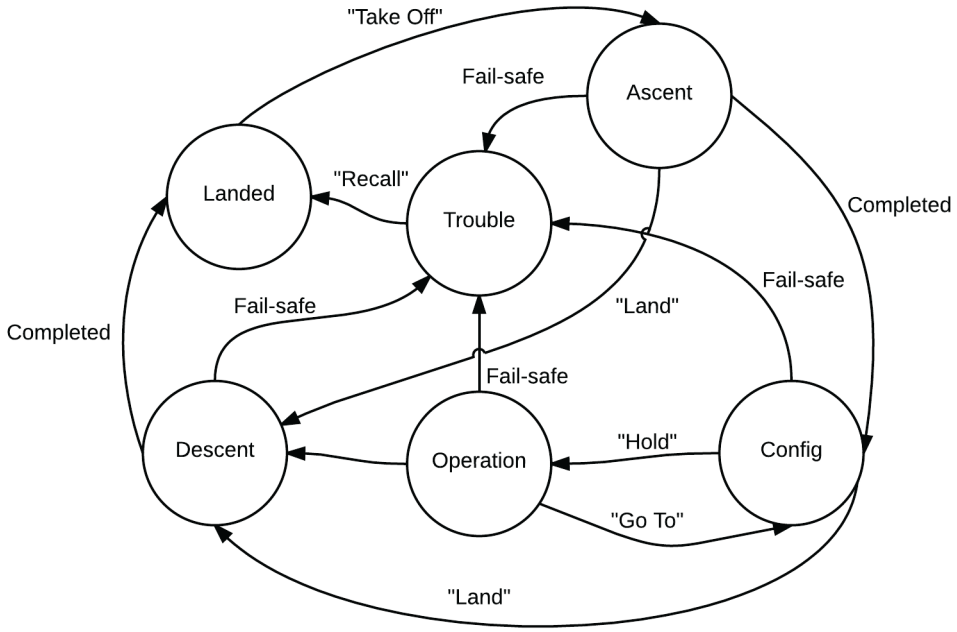


Figure 3.8. State Diagram

```
5         self.operationState = OperationState(self, drone)
6         self.configState = ConfigState(self, drone)
7         self.ascentState = AscentState(self, drone)
8         self.descentState = DescentState(self, drone)
9
10        self.state = self.landedState # Initial State
```

The *SwarmFSM()* class includes an important instance variable called *state* which represents the current state of the FSM and holds an object of the currently active state class.

The *SwarmFSM()* class also holds a number of important getter and setter methods. The purpose of these methods is to handle getting and setting states in the main FSM class from the state classes. Each state in the FSM has a corresponding getter method, such as *getLandedState()*, which returns the corresponding instance variable and thus the state object. There is only one setter method, *setState()*, which receives a state object and then assigns that state object to the state instance variable, thus changing the state of the FSM. The getter and setter methods are called only

from the state classes. For example, changing the state of the FSM to the *Ascent* state would be achieved by calling the *setState()* method with the *getAscentState()* method as an argument, such as:

```
1 def takeOff(self):
2     self.drone.commands.takeoff()
3     self.fsm.setState(self.fsm.getAscentState())
```

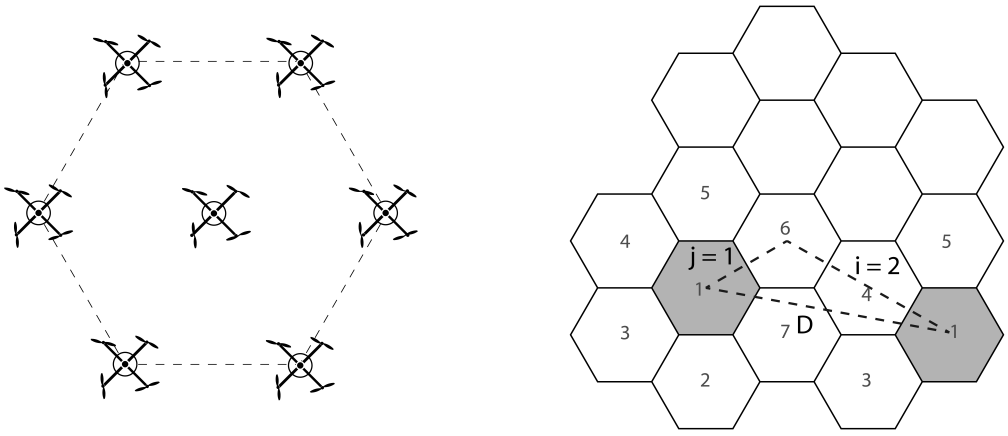
Clustering

The positioning algorithm is based on the traditional cellular reuse concept that utilizes hexagonal cells in a honeycomb patter. This cellular concept was first introduced by V.H. MacDonald in 1979 (MacDonald 1979). What MacDonald and his team at Bell Systems sought to accomplish was figuring out a means of more effectively using the allocated bandwidth and spectrum to handle a larger subscriber base and overall improve the quality of service to customers using the limited available resources. The solution to this issue was to arrange cellular base stations in a honeycombed hexagon pattern, such that frequencies could be reused in an effective and more predictable manner by making sure that no two adjacent cells use the same frequencies. The hexagon was used as a base shape because in terms of coverage, it costs less than triangular or square cells due to the hexagon having a larger area when the shapes all have the same center-to-vertex distance (MacDonald 1979, 20).

There are a number of equations that govern this cellular concept, with the first being the determination of the cluster size, N. The cluster size is defined as

$$N = i^2 + ij + j^2 \tag{3.1}$$

with i moving along any chain of hexagons and j moving along any chain of hexagons at an angle 60 degrees counter-clockwise to i. As shown in figure 3.9b and using 3.1, $N = 7$ when $i = 2$ and $j = 1$.



(a) Hexagonal Cell

(b) Cell layout with cluster size $N = 7$

Figure 3.9. Cellular Concept

Normalized to the size of each hexagon, the reuse distance, which is the shortest distance the same frequency can be used, is illustrated in figure 3.9 and is defined by:

$$D = \sqrt{3N}R \tag{3.2}$$

Using Eq. 3.2 the radius of each cluster can be defined as:

$$R_C = \frac{D}{\sqrt{3}} \tag{3.3}$$

This concept is the principle idea behind how the positioning algorithm works since this application is a platform for a communications system. Initially, the algorithm begins by receiving a list of all the currently active UAVs in the system, the operating center point of the system, and the radius of each cell as variables. The system then calculates the locations for the superposed of each cluster of 7 by calculating corner points on a hexagon in a clockwise fashion around the system center, increasing the radius after 6 iterations. These points are based on the total number

of UAVs in operation. After calculating these points, each of the nodes then becomes the center of another cluster of 7.

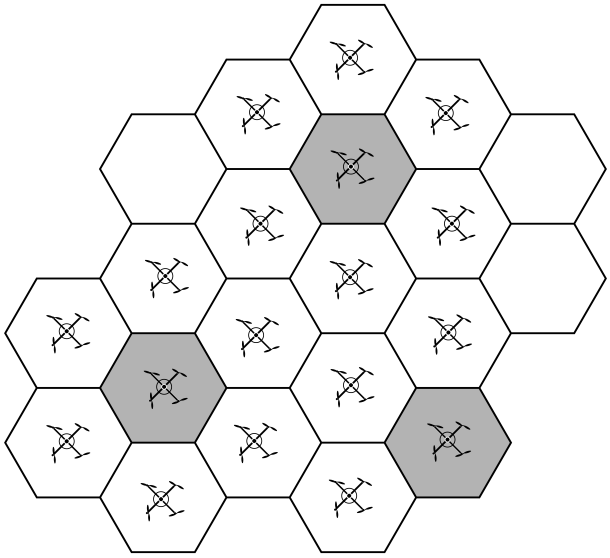


Figure 3.10. UAV formation with $N = 7$

The actual calculation of the position is achieved by extrapolating the new coordinated based on the current point, distance to travel, and bearing, as shown in Eq. 3.4 and Eq. 3.5 where φ is the latitude, λ is the longitude, θ is the bearing, and δ is angular distance calculated by $\frac{d}{R}$ with d being the distance to travel and R the radius of the earth:

$$\varphi_2 = \arcsin\left(\sin(\varphi_1)\cos(\delta) + \cos(\varphi_1)\sin(\delta)\cos(\theta)\right) \tag{3.4}$$

$$\lambda_2 = \lambda_1 + \arctan 2(\sin(\theta)\sin(\delta)\cos(\varphi_1), \cos(\delta) - \sin(\varphi_1)\sin(\varphi_2)) \tag{3.5}$$

These two equations were implemented in Python as:

```
def gps_newpos(lat, lon, bearing, distance):
    '''extrapolate latitude/longitude given a heading and distance
```

```
    thanks to http://www.movable-type.co.uk/scripts/latlong.html
    '''
    lat1 = math.radians(lat)
    lon1 = math.radians(lon)
    brng = math.radians(bearing)
    dr = distance/radius_of_earth

    lat2 = math.asin(math.sin(lat1)*math.cos(dr) +
                   math.cos(lat1)*math.sin(dr)*math.cos(brng))
    lon2 = lon1 + math.atan2(math.sin(brng)*math.sin(dr)*math.cos(lat1),
                            math.cos(dr)-math.sin(lat1)*math.sin(lat2))
    return (math.degrees(lat2), wrap_valid_longitude(math.degrees(lon2)))
```

By simply repeating this process, an infinite number of nodes can be generated by the algorithm that conforms to this hexagonal pattern. If a UAV drops out of the system, the pattern is simply shifted by one at the point closest to where the UAV dropped out. Adding a UAV to the system sends the UAV to the next available position in the system. Each UAV is assigned a unique incremental numeric node ID when connected to the system which the algorithm uses for identifying the UAV in the pattern. By changing the ID of interest, the entire system can move and rearrange itself dynamically. If, for instance, a UAV at a particular node ID has a low battery, the UAV would be recalled and a different UAV would be assigned to its node ID position, thus replacing it in the system. The algorithm can also shift all the node IDs by one, moving all the affected UAVs together as a system.

3.2.2 User Interface

The UI for the system was developed as a web browser-based system utilizing JavaScript as a programming language, Hypertext Markup Language (HTML) as a markup language, and Cascading Style Sheets (CSS) as a descriptive/stylistic language. These three languages and descriptors comprise the three main parts of any web application: the functional logic layer (JavaScript), the visual markup/presentation layer (HTML), and the styling/aesthetic layer (CSS).

The HTML code was used for laying out the visual structure of the interface. HTML consists of various elements called tags which feature words enclosed in angle brackets, usually appearing in pairs. The text appearing between these tag pairs is then interpreted by the browser based on the meaning of those tags. A paragraph of text, for example, would be enclosed within a starting tag `<p>` and then finished with a closing tag of `</p>`. Closing tags are always have a back slash. These tags are never actually displayed on a webpage – the browser just uses them to formulate the content on the webpage. In the development of this system, HTML was used to mark out the areas of the interface that displayed the map, the individual controls for each UAV, as well as the system-wide controls and overrides.

A CSS script was used to handle the styling of all the HTML elements on the page. CSS provides an important separation between the content, HTML, of the web page and the formatting. At its core, CSS is simply a list of rules that govern how all the HTML elements are styled on a web page. A typical rule for the `<body>` tag looks like:

```
1 body {
2   font-family: "Helvetica Neue",Helvetica,Arial,sans-serif;
3   font-size: 14px;
4   line-height: 1.42857143;
5   color: #333333;
6   background-color: #ffffff;
7 }
```

This rule does a number of things. First, it targets the body element which represents the entire web page. The second line sets the fonts used on the page. The third line sets the size of the font on the page. The fourth line dictates the line-height, or the space the between lines, of the page. The fifth line sets the color of the text on the page using a hexadecimal value. The final line sets the background color of the page, once again using a hexadecimal value. By creating more rules like this, the entire page can be easily styled independent of the actual text content on the page.

The logic behind the UI was handled by the JavaScript programming language. Unlike HTML and CSS, JavaScript is a dynamic language, as opposed to a compiled language, and thus executes at runtime. The JavaScript for the UI had the tasks of receiving information from the Python application, handling all events from the HTML frontend, updating the map, and finally sending commands back to the Python application. A JavaScript library called AngularJS was used to help dynamically load content in the HTML frontend with the incoming data from the Python application. The AngularJS code listened for any incoming data and upon receiving new data would update the content on the HTML frontend without reloading the page. This provided for the real-time display of data. Communication between the JavaScript script and the Python application was facilitated by WebSockets.

Communication over the World Wide Web, or simply the web, most commonly relies on the Hypertext Transfer Protocol (HTTP). HTTP by definition is an application-level protocol for distributed, collaborative, hypermedia information systems (W3 2015). This protocol has been one of the well-recognized protocols on the web due to the fact that it begins every uniform resource locator (URL), such as in http://www.google.com. HTTP has been extremely effective in most usage scenarios on the web with the exception of real time full-duplex communication, which is needed in applications such as real-time data feeds, group communication, and teleconferencing (Rakhunde 2014, 15). In fact, HTTP was not designed for real-time full-duplex communication due to the limited technological applications at the time the standard was created (Pimentel and Nickerson 2012, 45). Until recently, there were two popular ways of having real-time or near real-time HTTP communication: HTTP polling and HTTP long polling.

HTTP polling can best be described as a "call and response" in which one side initiates communication and the other side responds. This communication happens at a regular time interval Δ called the polling interval (Pimentel and Nickerson 2012, 46).

The workflow then becomes the client sending a request to the server for information and the server responding with the information if it exists or returning an empty request otherwise. Sometimes the server can be set to not respond if there is no new data. The main issue with this technique is that it initiates a large number of requests regardless of whether there is data. If the polling interval, Δ , is really small, this process can generate such a large amount of traffic that it would slow down communication for other parts of the application, and lead to increased latency. Long polling comes as a solution to the aforementioned issues with polling. In long polling, the server does not return anything if there is no new data on the server. Instead, the server simply keeps the connection open until it gets new data to send to the client; the server usually also has a timeout period after which it closes the connection. This technique greatly reduces the number of requests as compared to the interval-based polling technique.

HTML5 WebSockets

The HTML5 WebSockets protocol has emerged as the clear solution to the lack of real-time full-duplex communication. This protocol allows for fast real-time full duplex communication over the web on a single socket. WebSockets has been standardized by the Word Wide Web Consortium (W3C) as well as the Internet Engineering Task Force (IETF) and is supported by all the major web browsers including Chrome, Firefox, Safari, and Opera (Rakhunde 2014, 15). Communicating over WebSockets consists of two parts: the handshake and then the transfer of data. The most important event to happen during the handshake is that the server upgrades the connection from the HTTP protocol to the WebSockets protocol. After the handshake is complete, data then begins to flow in full-duplex using either text or binary frames (Rakhunde 2014, 16). As shown in figure 3.11, the WebSockets protocol is significantly more

efficient than HTTP Polling. This protocol greatly simplifies real-time full-duplex communication in a web browser.

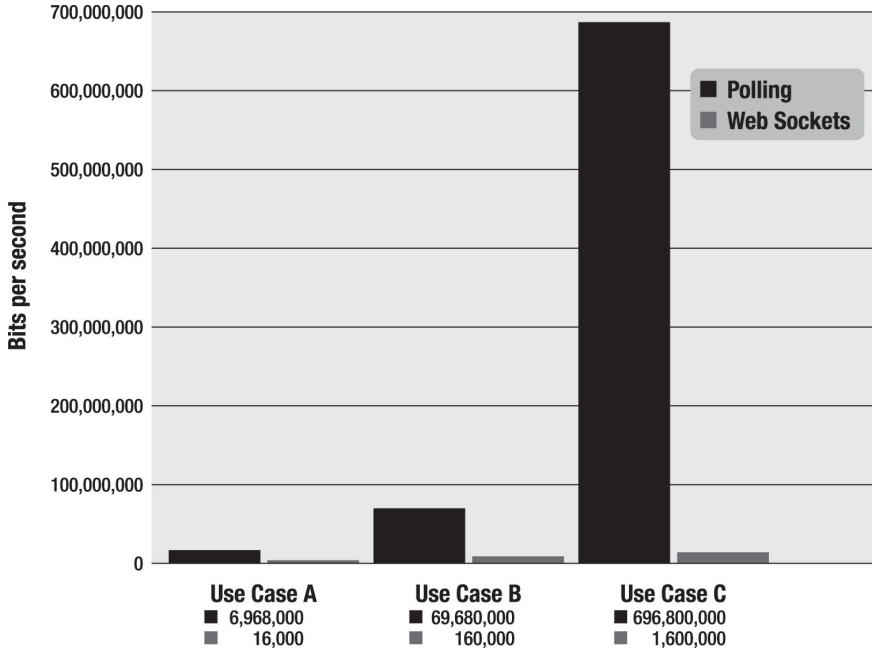


Figure 3.11. HTTP Polling vs WebSockets Bandwidth usage. Source: Lubbers, Albers, and Salim 2011

3.3 System Operation

The process control for the system begins with standard pre-arm hardware checks. This consists of checking the UAVs for any loose wires, broken or cracked frames and propellers, and insuring that the batteries are properly secured. Upon completion of these checks, a physical safety switch must be engaged before the UAV can arm. This safety is built in the Pixhawk autopilot to ensure that the UAV has been physically attended to before flight. On the software-side, pre-arm checks are conducted automatically and include ensuring that a good GPS lock is acquired, all electronic speed controllers (ESCs) are calibrated, and all essential sensors are functional and calibrated. Most of these checks happen as part of the ArduCopter flight control software and can be configured to be checked automatically.

3.3.1 Launch Sequence

The launch command for the system can only be initiated from the interface after the required configuration parameters have been set and verified. The configuration parameters that are set from the control interface are the central operating coordinates for the system, the GPS fence radius, the GPS height ceiling, the separation distance between the UAVs in the formation (hexagonal cell radius), and the operating altitude as depicted in figure 3.12. Once the configuration parameters are set then each connected UAV is then ready and added to the master UAV list for use in the swarming formation.

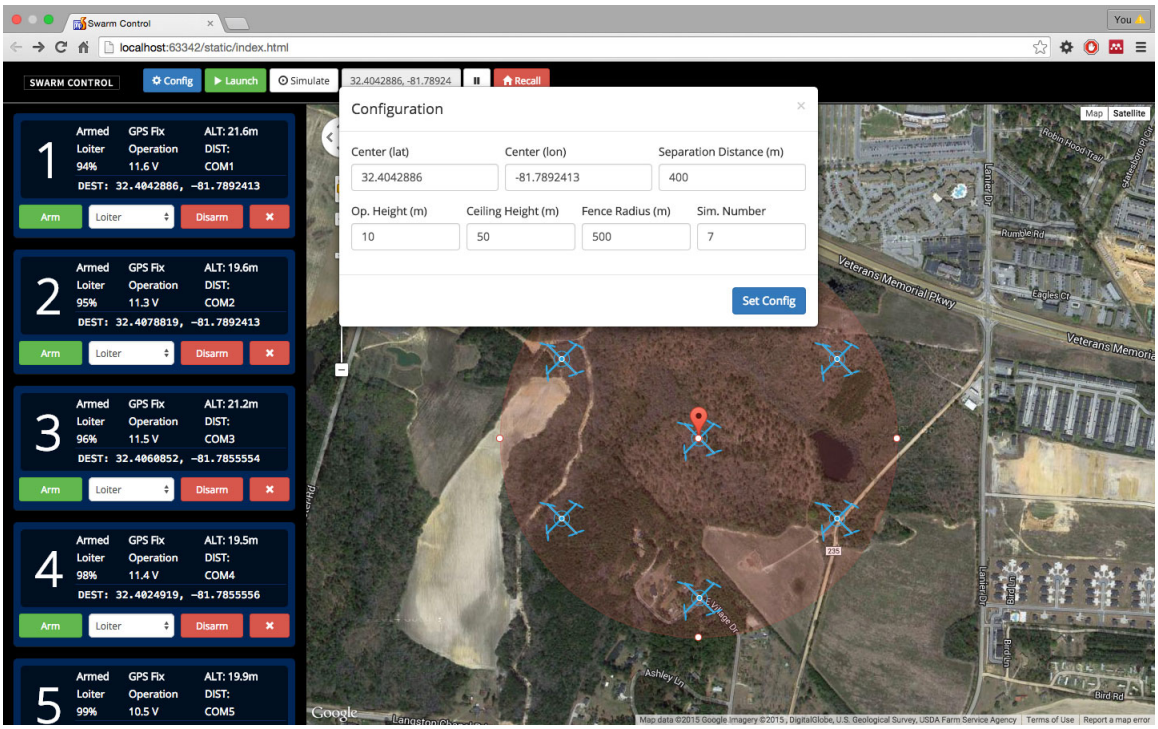
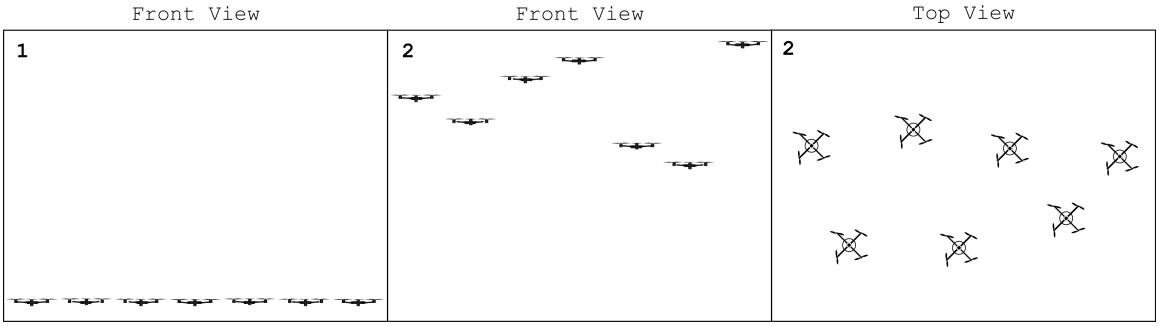


Figure 3.12. Configuration Parameters

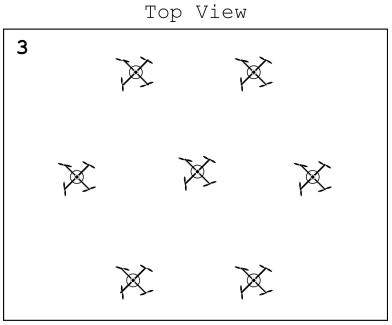
Upon launch, each UAV will fly straight up to its configuration altitude as shown in figure 3.13b. The configuration altitude is a height unique to each UAV that is used whenever the UAV has to travel from one position to another and is automatically assigned by the system. Since the height is unique and the UAVs travel laterally, there is no risk of collision when the UAVs are in the air. Upon reach-



(a) UAVs ready for launch.

(b) UAVs at configuration altitudes.

(c) UAVs at configuration altitudes.



(d) UAVs in final formation.

Figure 3.13. UAV launch sequence and configuration procedure.

ing the configuration altitude each UAV then travels to its destination coordinate, maintaining the configuration altitude shown in figure 3.13d. Once each UAV is at its destination coordinate, it will then settle to the global operating altitude for the system and remain there until otherwise commanded. For any time that a UAV in the system needs to change its position, it first climbs or descends to its configuration altitude before traveling to the new destination in order to avoid collisions.

3.3.2 Fail-safes

The system has been developed to incorporate a number of redundant fail-safes that are loaded to each UAV upon initiation. Most of these fail-safes are already built in the ArduCopter flight control software. Included fail-safe triggers are:

- Radio telemetry loss
- GPS signal loss
- Low battery
- Height ceiling breach
- GPS fence radius breach

The radio telemetry loss fail-safe is activated when the UAV does not receive a heartbeat packet from the groundstation for a set period of time. The battery fail-safe is triggered when the current and voltage sensors connected to the battery measure and estimate the remaining power to be below a set threshold. The GPS fail-safe simply checks if the current position is within a specified circle. With the exception of the GPS fail-safe, when these fail-safes are triggered they will recall the UAV back to its launch coordinates and land for evaluation. The loss of GPS, though, will trigger a wait period to try to regain a GPS lock and then will force the UAV to land if a GPS lock cannot be acquired.

Chapter 4

RESULTS

4.1 System Performance

Software performance was the primary concern when dealing with the Python programming language since it is one of the slower programming languages, especially when compared to compiled languages such as C and C++. Since Python supports CPU threading, that technique was utilized in optimizing the performance of the system. The initial approach to dealing with the multiple UAVs connected to the same host was to create a new thread (thread of execution) for each connected UAV. This would effectively solve the issue of trying to multiplex serial connections. In essence, each new object of the *Drone()* class is created in a separate thread. Testing this approach on a 24-core HP Z820 workstation showed no drop in performance nor any increase in CPU usage. On the other hand, running the same application on a laptop with limited resources started showing signs of degraded performance and a very noticeable increase in CPU usage over time as more threads were created; this lead to the abandonment of the individual threading idea. The more optimized approach was to use a total of only four threads for the entire system, representing the top level responsibilities of the application. The first and main thread handled all interactions between the HTML interface and the Python application. The second thread handled all the miscellaneous tasks and system checks that monitored the application. The third thread's responsibility was to handle the FSM that autonomously controlled the system. The fourth and final thread handled all the interactions with the UAVs, including sending commands and receiving data from the telemetry radios.

4.2 User Interface

The user interface, illustrated in figure 4.1, was a very efficient way of managing multiple UAVs. Built into the system was a simulator that was able to show the final position of any number of UAVs based on a coordinate of the central location.

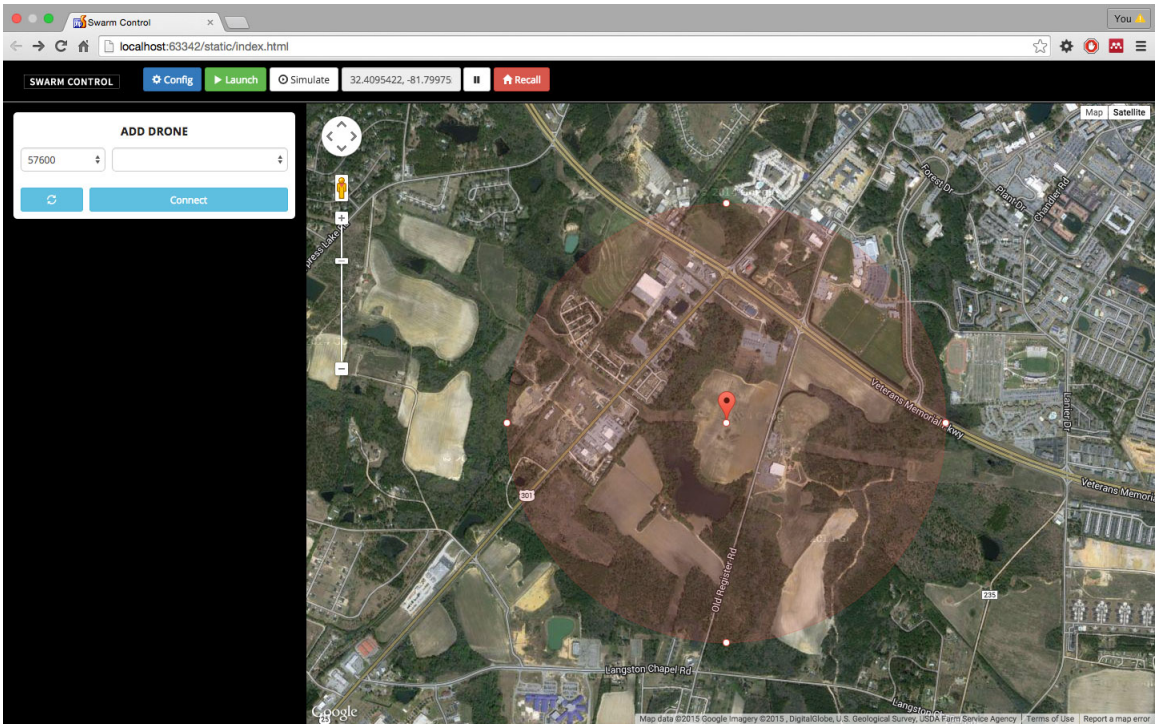


Figure 4.1. HTML User Interface

Included in the HTML interface were individual controls for each UAV as well as the buttons to start the launch sequence. The interface is made up of three main parts: the header, the sidebar, and the map area. The header, located at the top, contains the system-wide controls and configuration options. These controls include buttons for setting the system configurations, running a simulation, launching the UAVs, and recalling the UAVs. The sidebar displays a list of all connected UAVs with important status information about each UAV such as altitudes and battery levels. In addition, the sidebar controls allow for the mode of each UAV to be changed as well as arming and disarming the UAV. The map area displays the UAVs as icons on

the map and allows the position of any UAV to be adjusted by simply dragging and dropping it on the map.

The process of adding a new UAV to the system simple consisted of connecting the telemetry radio to a USB port on the computer and then selecting the new device in the dropdown menu as shown in figure 4.1. This menu automatically generates a list of all the occupied serial ports on the computer and updates the list upon clicking of the refresh button. The main area of the interface is occupied by a map overlaid with an icon of each connected UAV. Each of these UAVs can be manually positioned by dragging its icon and dropping it to a new position. Initiating the launch sequence is just a matter of clicking the *Launch* button as shown in figure 4.1.

4.3 Simulator

A simulator was created to test the positioning algorithm as well as how the system responded to changes in the system center. The simulator was created as a part of the Python application codebase and was activated using the user interface. With the simulator active, the Python application ran as normal except for the fact that no physical UAVs were connected. All the UAV sensor data was seeded by random number generators which gave values within the acceptable range for each data type. The simulator responded normally to changes in these values and triggered fail-safes as normal. The finite-state machine was not initiated during the simulator run. With the simulator, it was possible to see exactly how the system would behave over an area and as such allow for the proper area of coverage to be determined. For example, running the simulator with a single UAV resulted in the UAV being centered around the system operating point, as shown in figure 4.2. Adding three more UAVs to the system as shown in figure 4.3 resulted in four UAVs following the hexagonal pattern. The seven UAVs, shown in figure 4.4, resulted in a full hexagon with a central UAV.

Finally, figure 4.5 demonstrates how the algorithm creates hexagons in a clockwise manner centering around the system center.

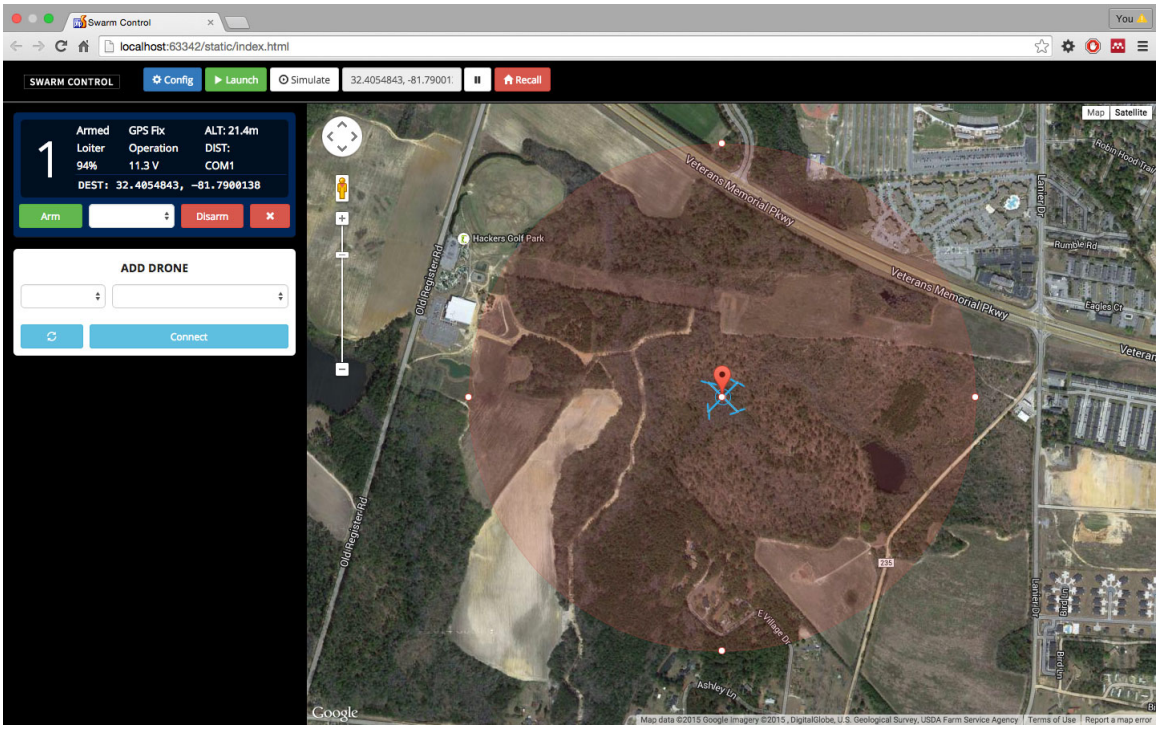


Figure 4.2. Simulation output for one UAV

4.4 Remote Access

The remote access feature of this system was tested by setting up a web-accessible HTTP tunnel to the Python application running on the system. This was achieved by using a command line program called ngrok. ngrok works by opening up the local server (Python application) on a specified port to one of its servers and then generates a URL that can be accessed from anywhere in the world. As shown in figure 4.6, the system was able to successfully connect to the application running on another computer through the web tunnel. The main issue with remote access is latency but since this system is fairly autonomous and can operate for significant periods of time without input, latency was not an issue. The autopilot handles flight

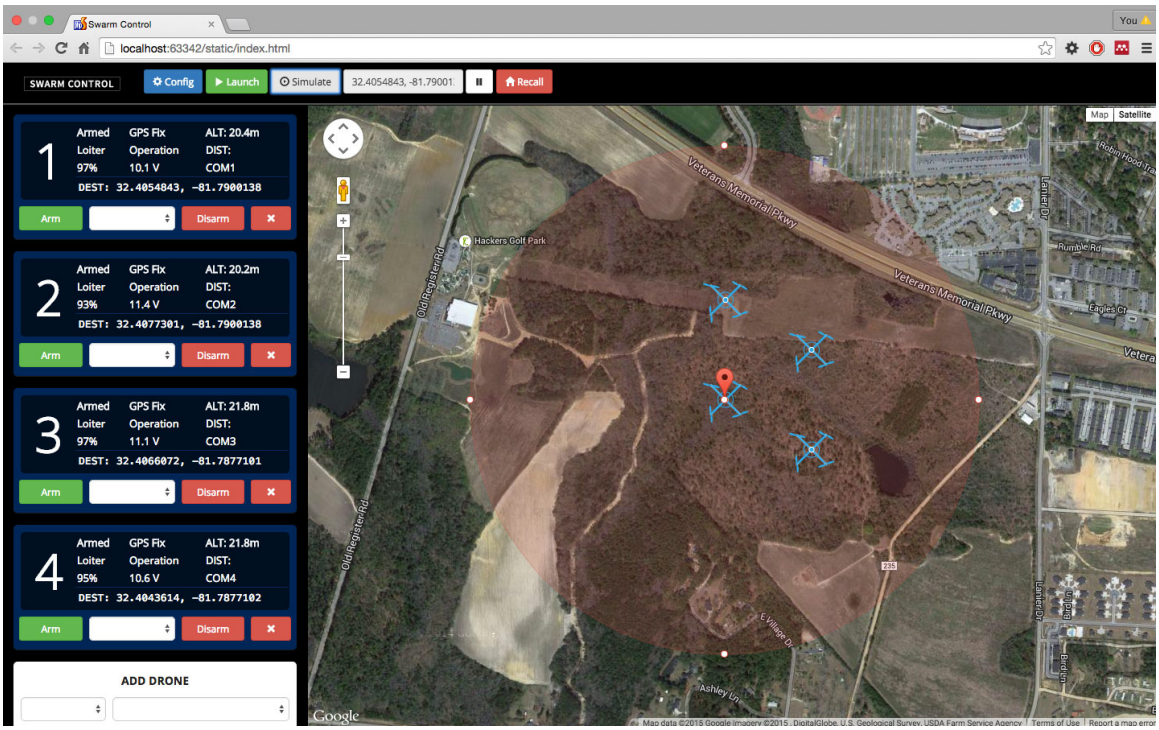


Figure 4.3. Simulation output for four UAVs

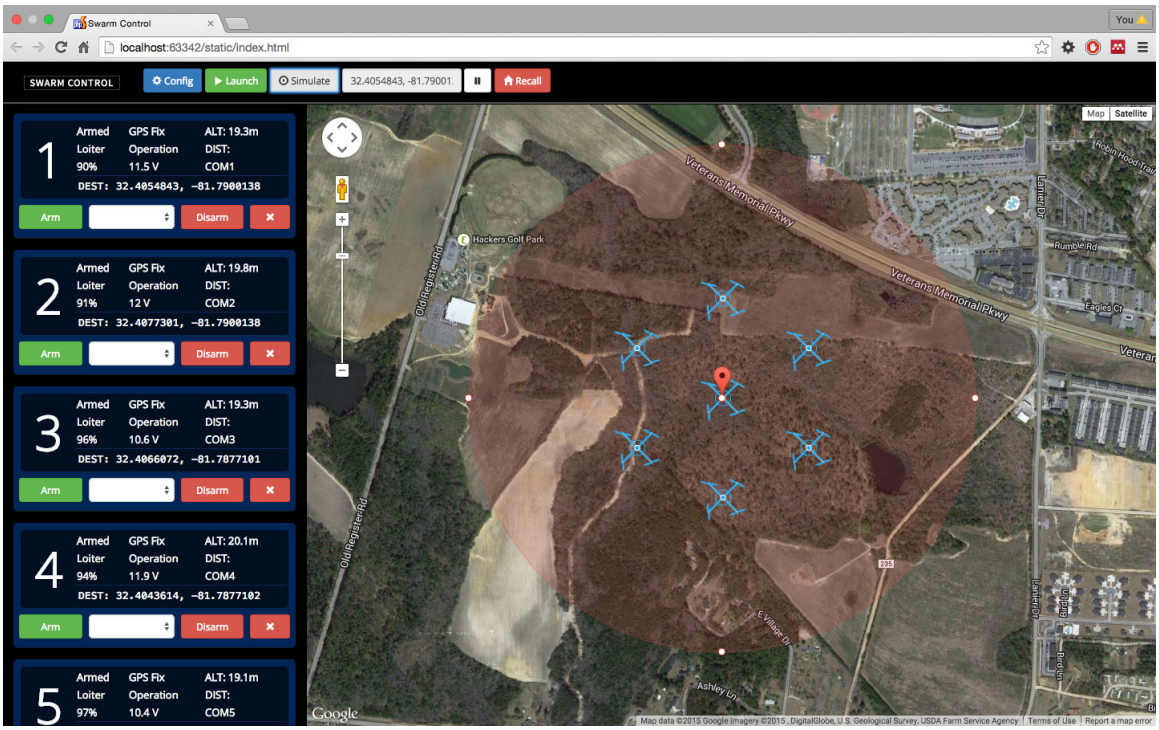


Figure 4.4. Simulation output for seven UAVs

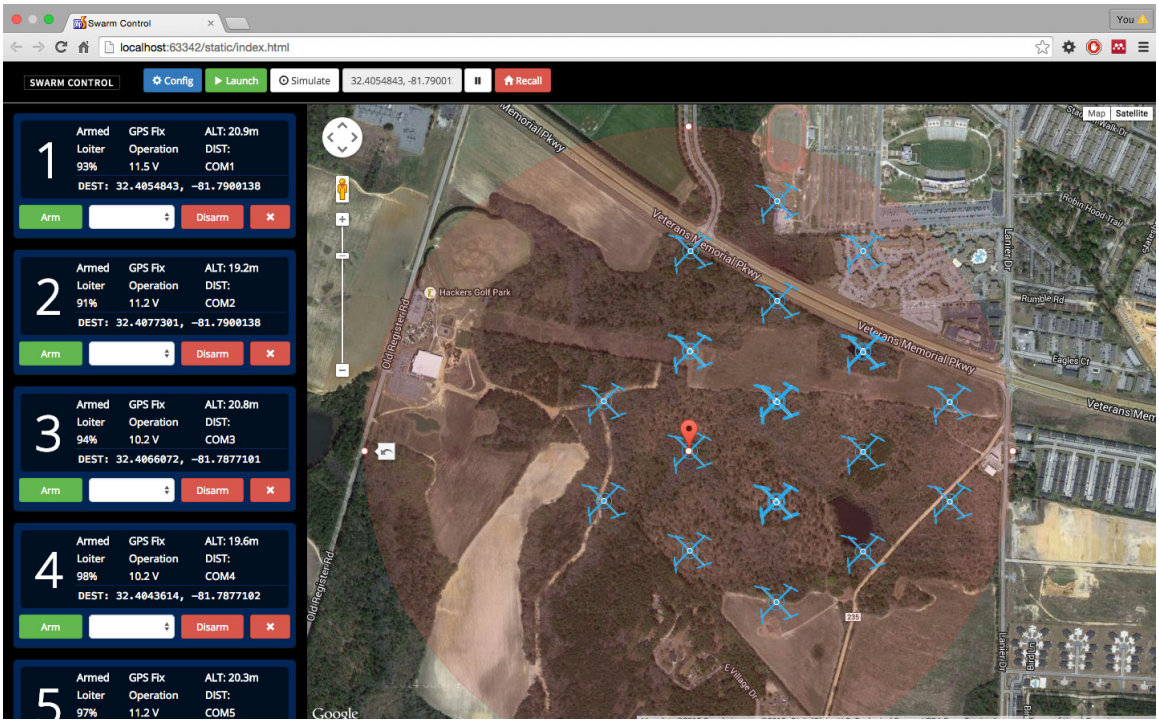


Figure 4.5. Simulation output for seventeen UAVs

and the fail-safes ensure that the system operates properly even without constant input.

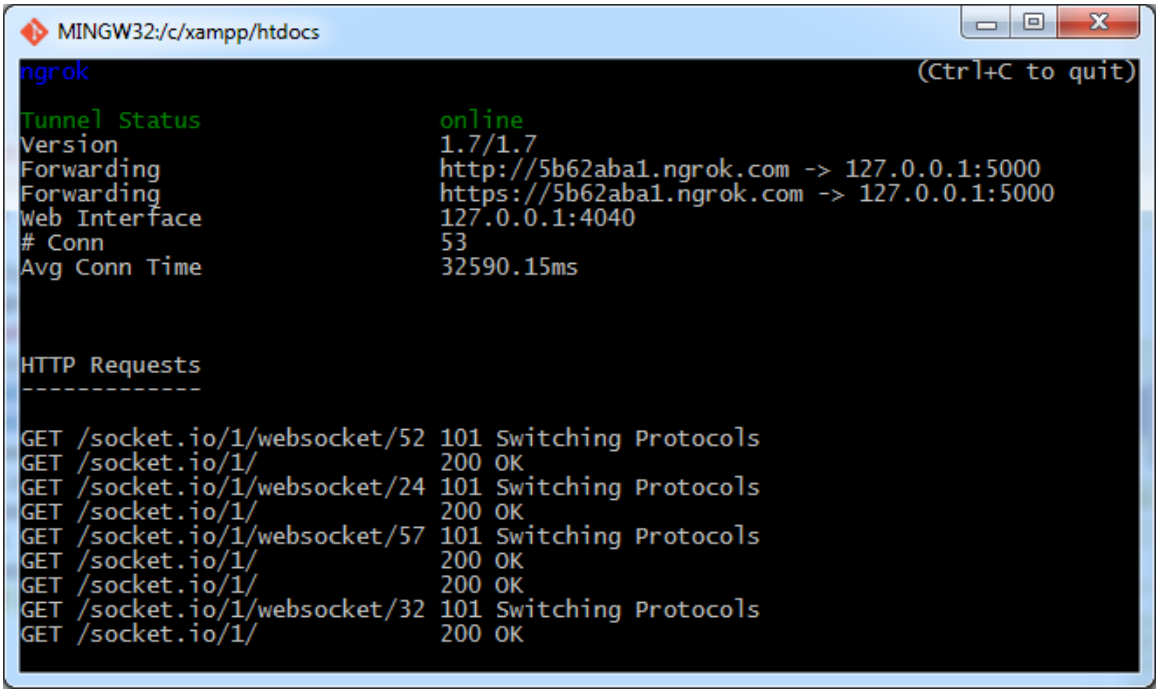


Figure 4.6. System control via HTTP

4.5 Hardware Performance

The system underwent limited hardware tests due to autonomous UAV flight restrictions instituted by the FAA during the course of this research. The UAVs were declared flight-worthy only after undergoing a thorough tuning of the autopilot PID and attitude controllers. Tuning was performed indoors on a custom tuning rig. The UAV was attached to the tuning rig at four points using Paracord as shown in figure 4.7. This allowed the UAV to fly up to a height of approximately 4 feet. The

Figure 4.7. UAV flying in PID tuning harness

positioning of the Paracord was such that the UAV could not turn at more than a 20 degree angle in any axis, thus preventing the blades from hitting the cords. Using $4500mAh$ batteries, flight time was measured to have an upper limit of 20 minutes.

4.5.1 Telemetry

Since each UAV had a corresponding telemetry radio that had to be connected to a computer, hardware was designed to better facilitate this process. A harness to hold multiple radios was designed in SolidWorks and then fabricated using a 3D printer.

This harness allowed the telemetry radios to be neatly arranged near the computer and made cable management easier. There was no noticeable drop in performance due to the class proximity of the radios to each other. From the harness, the radios were connected to a USB hub, which then fed to the computer with a single cable.

4.5.2 GPS

The GPS reliance of this system limits its operating conditions to outdoor settings with at most moderate cloud coverage. This actually is not much of a detracting factor because operating multirotor UAVs in potentially rainy conditions makes them susceptible to lightning and water damage. With moderate cloud coverage, 13 satellites on average were visible to the GPS module. In the case of signal loss due to a low number of satellites, the GPS module was always able to reconnect with a 4 second window. GPS accuracy was generally within a 4 to 5 meter radius.

4.6 The State of UAVs in the United States and the FAA

With the uptick of UAV usage in the United States during the last two years, there has been a great deal of concern from the FAA regarding safety to the national airspace system (NAS). This sudden advance in technology and the prevalence of UAVs capable of reaching high altitudes forced the FAA to take a very strong stance against anything but hobby use of UAVs. There are currently three different types of UAV operations classified by the FAA: Civil, Public, and Model; the first two of which require certification and authorization (*Unmanned Aircraft Systems* 2015).

Aside from recreational use, the operation of UAVs is currently very limited by the FAA in the United States. The FAA on February 15, 2015 released a proposal for new rules governing small unmanned aircraft systems (UAS). This covers UAS under 55 pounds. The proposed rules would limit flight to daylight and line-of-sight operations only (Dorr and Duquette 2015). The operator must be at least 17 years old, pass an aeronautical knowledge test and finally obtain an FAA UAS operator

certificate. Maintaining this certification will require passing the knowledge test every 24 months. Dorr and Duquette (2015) outlines more requirements as:

- A small UAS operator must always see and avoid manned aircraft. If there is a risk of collision, the UAS operator must be the first to maneuver away.
- The operator must discontinue the flight when continuing would pose a hazard to other aircraft, people or property.
- A small UAS operator must assess weather conditions, airspace restrictions and the location of people to lessen risks if he or she loses control of the UAS.
- A small UAS may not fly over people, except those directly involved with the flight.
- Flights should be limited to 500 feet altitude and no faster than 100 mph.
- Operators must stay out of airport flight paths and restricted airspace areas, and obey any FAA Temporary Flight Restrictions (TFRs).

The FAA is also considering creating a new class of micro UAS that weigh less than $4.4lbs$ and travel less than $35mph$. This class would have more lenient rules. It is important to note that these are only proposals. Aside from recreational use, the operation of UAVs is currently very limited by the FAA in the United States. The rules and exact requirements for licensure are still very vague at the moment and the fact that acquiring exemptions requires having a private pilot license greatly limits entry to this field. This fact is also causing innovators and companies to move their drone work outside the United States. Google, and Amazon, has actually moved some of the development of its drone program overseas — with Google testing their drone delivery in Australia (*America's clumsy regulation of drones stirs up frustration, confusion.* 2014). The new proposed rules will hopefully allow for greater innovation in the field of unmanned aircraft systems.

Chapter 5

CONCLUSION

In this work, a feasible platform for quick deployment and operation of a temporary emergency relief communications network has been proposed. The platform outlined in this work represents the unity of three otherwise individually functioning systems into one coherent platform that is both simple and safe in operation. Open source hardware and software has been utilized in the creation of this system, making it is easily reproducible and accessible. With the current popularity of UAVs and the continuing advancements in UAV technology, this type of system will continue to be a valuable and important asset in emergency response situations. The threat of major disasters is ever present and having a system such as this is an effective way of ensuring that emergency personnel have an easy, fast, and adaptable way of communicating in disaster situations.

5.1 FUTURE WORK

Upon full approval from the FAA to autonomously operate a swarm of UAVs, the first step will be to fully operate the system at varying heights and distances. While all the hardware and software have been verified and the system tested using a simulator, it still requires outdoor testing to verify the functionality of all the fail-safes.

Another point of improvement for this system will be making the algorithm decentralized. Currently all the control logic comes from a central point, which is the ground control computer. This type of control scheme limits the total operating range of the system due to each radio having to be within range of the ground control station. Making this change would require adding a radio to the UAV that supports the traditional mesh network. This would then allow the UAVs to communicate with each other. The control algorithm would then have to be integrated into the

ArduCopter flight control software. The current design of the control algorithm is such that this implementation would not require any significant changes to core logic, other than porting from one language to another.

The system will also need to be able to change the swarming algorithm pattern on the fly. This can be accomplished by surveying more coverage algorithms besides the cellular concept and then implementing those as options in the Python application. An option can then be added to the HTML interface for choosing the swarming algorithm to be used before launching.

Finally, work will have to be done to mitigate the effects of the limited battery life of multirotor UAVs. One solution that can be very easily implemented into this system is to have an automated battery changing system in which the UAV lands autonomously on a platform and then a battery changing mechanism switches the almost depleted battery with a full one. This would allow the system to function for an extended amount of time as long as the batteries can be recharged at an appropriate pace.

BIBLIOGRAPHY

Adams, Matthew. 2015. *Sun Outages.* Accessed March 10, 2015. http://www.orbitelcom.com/2015/02/sun-outages/.

America's clumsy regulation of drones stirs up frustration, confusion. 2014. Washington D.C., December. http://www.washingtonpost.com/blogs/innovations/wp/2014/12/09/americas-clumsy-regulation-of-drones-stirs-up-frustration-confusion/.

ArduPilot. 2015. *Using the 3DR Radio for Telemetry with APM and PX4.* Accessed January 20, 2015. http://copter.ardupilot.com/wiki/common-using-the-3dr-radio-for-telemetry-with-apm-and-px4/.

Bupe, Paul, Rami Haddad, and Fernando Rios-Gutierrez. 2015. Relief and Emergency Communication Network Based on an Autonomous Decentralized UAV Clustering Network. In *Southeastcon, 2015 proceedings of ieee,* 8. IEEE.

Crowlely, John, and Jennifer Chan. 2011. *United Nations Foundation - Disaster Relief 2.0: The Future of Information Sharing in Humanitarian Emergencies.* Accessed February 15, 2015. http://www.unfoundation.org/news-and-media/publications-and-speeches/disaster-relief-2-report.html.

Dorr, Les, and Alison Duquette. 2015. *DOT and FAA Propose New Rules for Small Unmanned Aircraft Systems.* Accessed March 15, 2015. http://www.faa.gov/news/fact%5C_sheets/news%5C_story.cfm?newsId=18297.

Estrem, Angels Via, and Markus Werner. 2010. Portable satellite backhauling solution for emergency communications. In *2010 5th advanced satellite multimedia systems conference and the 11th signal processing for space communications workshop,* 262–269. IEEE, September. doi:10.1109/ASMS-SPSC.2010.5586923. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5586923.

FEMA. 2015. *Disaster Emergency Communications Division — FEMA.gov.* Accessed February 14, 2015. https://www.fema.gov/disaster-emergency-communications-division.

Fultron. 2007. *WHY SATELLITE COMMUNICATIONS ARE AN ESSENTIAL TOOL FOR EMERGENCY MANAGEMENT AND DISASTER RECOVERY.* Accessed February 15, 2015. http://www.futron.com/upload/wysiwyg/Resources/Whitepapers/Why%5C_Satellite%5C_Communications%5C_Essential%5C_Tool%5C_1105.pdf.

Gamma, Erich, Richard Helm, Ralph Johnson, and John Vlissides. 1994. *Design Patterns: Elements of Reusable Object-Oriented Software.* Pearson Education. http://books.google.com/books/about/Design%5C_Patterns.html?id=6oHuKQe3TjQC%5C&pgis=1.

Gao, Yuan. 2013. *What Makes The Quadcopter Design So Great For Small Drones? - Forbes.* Accessed March 12, 2015. http://www.forbes.com/sites/quora/2013/12/23/what-makes-the-quadcopter-design-so-great-for-small-drones/.

Gyou, Beom Kim, Kien Nguyen Trung, Agus Budiyono, Keun Park Jung, Joon Yoon Kwang, and Jinok Shin. 2013. Design and Development of a Class of Rotorcraft-based UAV. *International Journal of Advanced Robotic Systems* 10:1–9. doi:10.5772/54885.

Harris CapRock. 2014. *High Throughput Satellite Communications Systems: MEO vs. LEO vs. GEO.* Accessed March 8, 2015. http://www.harriscaprock.com/blog/?p=68.

Huang, Ming, Jiang Yu, and Jingsong Hu. 2010. A pilot emergency communication system based on IP and airship. In *Proceedings of the 9th international symposium on antennas, propagation and em theory,* 1008–1011. IEEE, November. doi:10.1109/ISAPE.2010.5696645. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5696645.

Intelsat. 2015. *Satellite Sun Interference.* Accessed March 10, 2015. http://www.intelsat.com/tools-resources/satellite-basics/satellite-sun-interference/.

Lubbers, Peter, Brian Albers, and Frank Salim. 2011. *Pro HTML5 Programming.* 352. Apress. http://apress.jensimmons.com/v5/pro-html5-programming/ch7.html.

MacDonald, Verne H. 1979. Advanced Mobile Phone Service: the Cellular Concept. *Bell Syst Tech J* 58, no. 1 (January): 15–41. doi:10.1002/j.1538-7305.1979.tb02209.x. http://www.scopus.com/inward/record.url?eid=2-s2.0-0018294080%5C&partnerID=40%5C&md5=6e9661e7b1af38e4a833bb7ff6cd2ec5.

MAVLink Micro Air Vehicle Communication Protocol - QGroundControl GCS. 2015. Accessed January 20. http://qgroundcontrol.org/mavlink/start.

MAVProxy. 2015. Accessed January 21. http://tridge.github.io/MAVProxy/.

Meier, Lorenz, Petri Tanskanen, Friedrich Fraundorfer, and Marc Pollefeys. 2011. PIXHAWK: A system for autonomous flight using onboard computer vision. In *2011 ieee international conference on robotics and automation,* 2992–2997. IEEE, May. doi:10.1109/ICRA.2011.5980229. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5980229.

Miller, Robert. 2006. *Hurricane Katrina: Communications & Infrastructure Impacts.* Technical report. DTIC Document.

Noam, Eli. 2004. What the World Trade Center Attack has Shown us About our Communications Networks. Chap. 20 in *Global economy and digital society.* Amsterdam; Boston.

Pimentel, Victoria, and Bradford G. Nickerson. 2012. Communicating and Displaying Real-Time Data with WebSocket. *IEEE Internet Computing* 16, no. 4 (July): 45–53. doi:10.1109/MIC.2012.64. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6197172.

Pixhawk Autopilot System. 2015. Accessed January 20. http://3drobotics.com/pixhawk-autopilot-system/.

Rakhunde, Shruti M. 2014. Real Time Data Communication over Full Duplex Network Using Websocket. *IOSR Journal of Computer Science* 5:15–19. http://iosrjournals.org/iosr-jce/papers/ICAET-2014/volume-5/3.pdf?id=7557.

Richtel, Matt. 2009. *Inauguration Crowd Will Test Cellphone Networks,* January. http://www.nytimes.com/2009/01/19/technology/19cell.html?pagewanted=all.

Sanner, M F. 1999. Python: a programming language for software integration and development. *Journal of molecular graphics & modelling* 17, no. 1 (February): 57–61. http://www.ncbi.nlm.nih.gov/pubmed/10660911.

Sim. 2009. *DCS: Black Shark and Coaxial Rotor Aerodynamics — SimHQ.* Accessed March 12, 2015. http://www.simhq.com/%5C_air13/air%5C_427a.html.

The federal response to Hurricane Katrina : lessons learned. 2006. [Washington D.C.: White House.

U-blox. 2012. *"u-blox 6 GPS, QZSS, GLONASS and Galileo modules".*

Unmanned Aircraft Systems. 2015. Accessed January 22. https://www.faa.gov/uas/.

W3. 2015. *Hypertext Transfer Protocol – HTTP/1.1.* Accessed January 30. http://www.w3.org/Protocols/rfc2616/rfc2616.html.