Relief and Emergency Communication Network Based on an Autonomous Decentralized UAV Clustering Network

Paul Bupe Jr, Rami Haddad, Fernando Rios-Gutierrez
College of Engineering and Information Technology
Georgia Southern University
Statesboro, GA, United States
Email: {pb00702, rhaddad, frios}@georgiasouthern.edu

***Abstract*—The recent years have witnessed an increase in the natural disasters in which the destruction of the essential communication infrastructure has significantly affected the number of casualties. In 2005, Hurricane Katrina in the United States resulted in over 1,900 deaths, 3 million land-line phones disconnections, and more than 2,000 cell sites going out of service. This incident highlighted an urgent need for a quick-deployment efficient communication network for relief purposes. In this paper, we propose a fully autonomous system to deploy Unmanned Aerial Vehicles (UAV) as the first phase disaster recovery communication network for wide-area relief. An automation algorithm has been developed to control the deployment and positioning of UAVs based on a traditional cell network structure utilizing 7-cell clusters in a hexagonal pattern using MAVLink. The distributed execution of the algorithm is based on a centralized management of UAV cells through assigning higher ranked UAVs referred to as supernodes. The algorithm autonomously elects supernodes based on weighted variables and dynamically handles any changes in total number of UAVs in the system. This system represents a novel approach for handling a large-scale autonomous deployment of a UAV communications network. The proposed autonomous communication network was verified and validated using software simulation and physical demonstration using identical quadrotor UAVs.**

***Keywords*—Unmanned aerial vehicles, Emergency services, Telecommunication network, Global Positioning System, Telemetry**

I. 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 [1]. Hurricane Katrina delivered the most widespread critical infrastructure collapse of any advanced country since World War II [1]. 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 [1], widespread infrastructure collapse is one of the marker elements that help differentiate catastrophes from disasters, which was exactly the case in this situation. The White House report [2] 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 [2]. Lastly, 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 [1].

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 the disruption of a phone switch with over 200,000 lines, 20 cellphone towers, and 9 TV broadcast stations [3]. All of the aforementioned issues lend evidence to the fact that essential telecommunications systems need to be decentralized in order to prevent them having a single point of failure.

A. Current Solutions

To date there have been no practical solutions offered, outside of military applications, to the issue of immediate emergency disaster relief communication. When dealing with

978-1-4673-7300-5/15/$31.00 ©2015 IEEE

heavy congestion of cellular networks, the current approach is to bring in portable cellphone towers on trucks 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 [4]. 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 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.

II. Contributions

What this work aims to do is provide a robust, quick to deploy, and scalable GPS-based UAV platform that is able to serve as the backbone of a temporary 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 system, any number of UAVs would be brought in to the area, connected to the system 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 infrastructure-free environments or environments with severe blockage.

The main contributions of this work are:

- The presentation of a quadcopter UAV platform upon which a temporary 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.

III. Methodology

The system as a whole contains three distinct layers: the hardware layer, the application layer, and finally the presentation layer as shown in fig. 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.

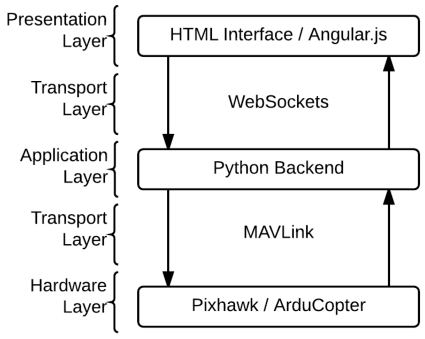


Fig. 1. System Overview

A. 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.

For the purpose of this research, 13 identical UAVs were assembled using parts from various manufacturers. Each UAV was assembled to be as minimalistic as possible, having only the features essential for proper flight. The frame for the UAVs is a standard X-frame quadcopter design as displayed in fig. 2. This means each UAV has two pairs of counter-rotating blades. The UAVs are also equipped with a battery monitor, telemetry radio, RC receiver, and a GPS module. The primary advantage of this hardware is being very modular. Most parts on the UAV, such as motors and blades, can be replaced quickly and easily in the field, adding a layer of robustness to the hardware and system.

The hardware components used in the UAV design are discussed as follows:

1) Pixhawk Autopilot: Each UAV is equipped with a Pixhawk autopilot system. This system offers a complete flight stack running on powerful hardware. This includes a 32-bit ARM Cortex M4 processor with a 32-bit STM32F103 failsafe co-processor, a 3-axis 16-bit gyroscope, a 3-axis 14-bit accelerometer / magnetometer, and a barometer, among other sensors [5]. The Pixhawk also has a host of ports, buses, and inputs / outputs.

Running on top of the Pixhawk hardware is the open-source ArduCopter flight controller. ArduCopter is the most popular and mature open-source flight software in use today. In addition to the standard manual controls found in flight control software,

Fig. 2. UAVs in testing configuration.

ArduCopter also has a number of useful autonomous features which make it ideal for this application. Namely, it has a number of important modes such as $Loiter$, which allows the UAV to hold its position using GPS and altitude data, $ReturntoLaunch$, which recalls the UAV back to its launch coordinates, and it is able to handle landing and taking off autonomously using various sensors. These features, and the extremely thorough documentation, made ArduCopter the ideal flight controller for the system.

2) Telemetry: The MAVLink protocol facilitates communication between the UAVs and the ground station in this application. MAVLink was first released in 2009 by Lorenz Meier [6]. This protocol is a very lightweight header protocol that has only 8 bytes overhead per packet and has built-in packet-drop detection, among other features. The low overhead makes it ideal for User Datagram Protocol (UDP) and UART/radio communication [7]. MAVLink utilizes an XML-based common message library which can be compiled to other languages to facilitate compatibility among multiple platforms. While it has a standard library of messages, distributed as a header-only C library, custom messages can be implemented using a generator and XML. MAVLink is licensed under LGPL, meaning it can be used royalty-free in open and closed-source applications.

Radio telemetry for this system is facilitated by 915MHz radios developed by 3D Robotics (3DR). These radios are made specifically for use with drones, both land and air-based, and come optimized for handling MAVLink packets. In line with the underlying theme of this work, these radios also utilize open source firmware. Each UAV is equipped with a 915 MHz radio paired to another radio connected to the ground station. There were a number of factors considered in choosing a suitable system for telemetry, mainly focused on interference and packet loss. The fact that MAVLink has built-in error checking alleviated the concern about packet loss, which left interference as the main concern. 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, the radios utilize Gaussian frequency-shift keying (GFSK) [8]. 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 uses a central processing hub, 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. This means that the transmitting power of the radios is an important factor in this application. Adjusting for range with these radios is a simple matter of configuring the duty cycle, data rate, and power level, which can range from $1dBm$ to $20dBm$ [8]. Adjusting these parameters can allow for a range of a few kilometers without external amplifiers.

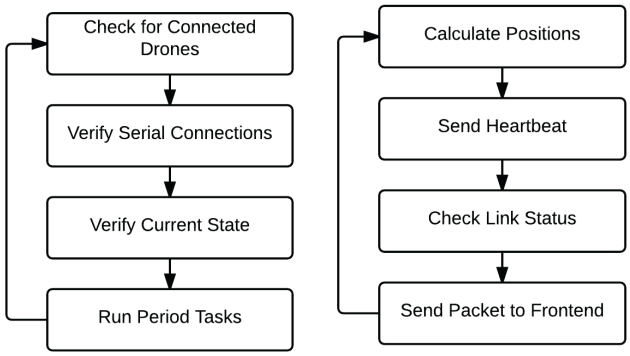
3) GPS: Having a reliable GPS unit is extremely critical in this application since GPS is the only means of localization available to the system. The 3DR ublox LEA-6 GPS module provides a fast high performance GPS with high accuracy and a compass, which is especially useful for position-hold situations [9]. A few important features of this GPS module are that is utilizes an active ceramic patch antenna, has a backup battery for warm starts, and operates at a $5Hz$ refresh rate. Lastly, this 3DR 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 percent of the time [10].

IV. System Implementation

A. Software

The most important part of the system is the software-based algorithm. One of the overarching design ideas for this system is that it should be very portable and quick to deploy. This implies that the operating software needs to be able to run on many platforms with minimal hassle and compatibility issues. With that in mind, the traditional approach to a problem such as this had to be abandoned and re-imagined when approaching the initial design of this software. Typically an algorithm such as this is implemented in a compiled language such as C or C++ due to their fast nature. The issue with using such a language is that it is heavily system dependent, not very cross-platform and is difficult to quickly modify due to it being a compiled language. Depending upon the system they are running on, compiled languages can also have hard to resolve dependencies. This shifted the focus to an interpreted language such as Perl or Python, from which Python was chosen as the primary language of choice for the algorithm.

Python is an interactive object-oriented language that provides high-level data structures including lists and associative arrays (dictionaries), modules, classes, automatic memory management, and a host of other features [11]. One of the biggest draws to Python is its extremely simple and friendly syntax. The biggest advantage of Python for this system is the fact that it can run on almost any modern computer, regardless of the operating system.



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

Fig. 3. Main Control Loops

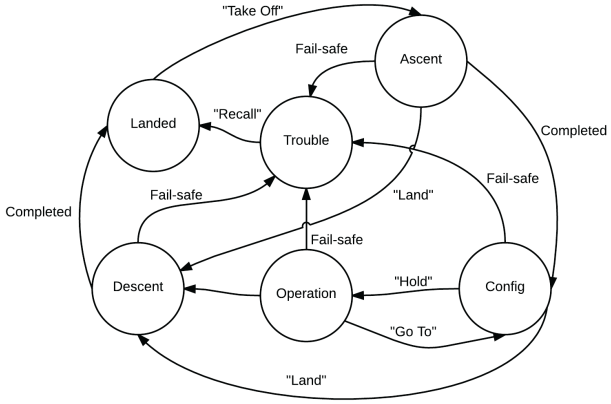


Fig. 4. State Diagram

B. Python Server

Critical to the design of the Python application was the pymavlink Python package. This package implements the MAVLink protocol in Python. Having this package allows the Python application to communicate fully with the hardware through the serial port or telemetry radios. Also important was the MAVProxy package developed by Andrew Tridgell [12]. This package provided useful helper functions that assisted with interfacing with the UAVs through the MAVLink protocol.

C. Control Algorithm

The Python application has three main parallel threads. The first thread listens for data from the connected UAVs and send commands to them as initiated by the operator or the application itself. Once a UAV has been connected by the operator, an instance of the $Drone()$ class is created and at the same time stored in a global $drones$ list. The $Drone()$ class encapsulates all the required data and also prepares a packet of that data ready to send to the HTML user interface (UI). This packet includes information such as GPS coordinates, altitude, estimated distances, and mode. The $Drone()$ class also subclasses a $Commands()$ class which houses all the commands that are sent to the UAVs. Each command is a method of the $Commands()$ class and communicates directly to the UAV via pymavlink and the MAVLink protocol. The principle commands for this application are $takeoff()$, $goto()$, and $land()$. The second thread of the Python application listens for commands from the UI and also sends status packets to the UI, as earlier described. Two Python packages are utilized for this task, of which the first is Flask. What Flask does, among other things, is expose the Python script as a server, accessible through a standard web URL. The second Python package, Flask-SocketIO, works in tandem with Flask to enable the now exposed server to use WebSockets, a protocol allowing for real-time full-duplex communication between the Python script and the UI.

The third and most important thread of the Python back-end application runs the main control loop that handles the autonomous deployment and positioning of the UAVs. This thread has two distinct tasks. The first is to calculate the desired positions of the UAVs based on the number of connected UAVs and the center. The main control loop, illustrated in fig. 3a, handles all the administrative tasks and continuously runs various system checks and tests. The periodic loop shown in fig. 3b runs various repetitive tasks such as calculating destination coordinates and sending heartbeat messages to the connected UAVs.

Autonomous control of the system is accomplished by the use of a finite-state machine (FSM) illustrated in fig. 4. Using an FSM allows for the control of the entire system to be broken down into smaller and more manageable states, thus simplifying the overall program and making it more modular. For each UAV connected to the system, a new FSM is instantiated, working independently from the other FSMs. These FSMs are all monitored by a central process that then controls the fleet based on the states of the FSMs.

The FSM is implemented in the Python programming language in an object-oriented manner. On a high level, an FSM allows an object to alter its behavior when its internal state changes. The object will then appear to change its class [13]. This FSM in particular has a total of six states:

- Landed
- Trouble
- Operation
- Configuration
- Ascent
- Descent

These states encompass all the operating parameters of the system. Each one of these six states is represented by a unique class which inherits a standard state class. These state classes all share the same methods, even though the behavior of those methods is unique for each class. These 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 of these methods perform

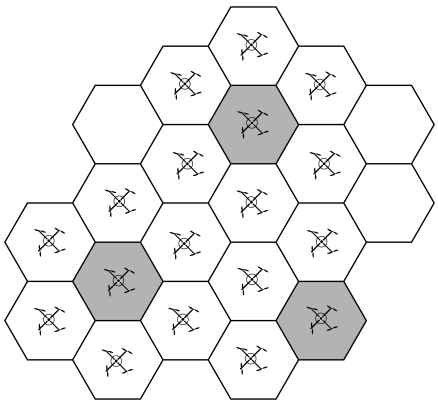


Fig. 5. UAV formation with $N = 7$

a different task in accordance with the state diagram in fig. 4. For example, calling the $takeOff()$ method in the $LandedState()$ class will cause the UAV to begin the take off sequence and then transition the FSM to the $Ascent$ state. On the other hand, calling the same $takeOff()$ method 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.

1) FSM Class: The main controlling class for the FSM is called $SwarmFSM()$. The $SwarmFSM()$ class is initiated with instance variables that are actually objects of the corresponding state classes. For example, the $landedState$ instance variable holds an instance of the $LandedState()$ class. The $SwarmFSM()$ class includes an instance variable called $state$ which represents the current state of the FSM and holds the object of the corresponding state class.

The $SwarmFSM()$ class also has a number of getter and setter methods. 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 setter and getter methods are called 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

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

2) Clustering: The positioning algorithm is based on the traditional cellular reuse concept utilizing hexagonal cells in a honeycomb pattern. This cellular concept was first introduced by V.H. MacDonald in 1978 [14]. MacDonald and his team at Bell Systems sought to figure out a means to more effectively use the allocated bandwidth and spectrum to handle a larger subscriber base and overall improve quality of service using the available resources. The solution to this problem, on a very basic level, was to arrange cellphone base stations in a honeycombed hexagonal pattern such that frequencies could be reused in an effective and predictable manner by making sure no two adjacent cells use the same frequencies. The hexagon was used as 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 [14]. 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 can be defined as

$$N = i^2 + ij + j^2 \tag{1}$$

with i moving along any chain of hexagons and j moving along any chain of hexagons at an angle 60 degrees counterclockwise to i, as shown in fig. 5. Normalized to the size of each hexagon, the reuse distance is illustrated in fig. 6 and is defined by:

$$D = \sqrt{3N}R \tag{2}$$

Using eq. 2 the radius of each cluster can be defined as:

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

This concept is the principle idea behind how the positioning algorithm works since this application is a framework for a communications system. Initially, the algorithm begins by receiving a list of all the UAVs currently active in the system, the operating center point of the system, and finally the radius of each cell are set as variables. The system then calculates the locations for the supernodes of each cluster of seven by calculating points in a larger hexagon in a clockwise fashion around the center, increasing the radius after six iterations. These points are based on the total number of UAVs in operation. After calculating these points, each of these nodes then becomes the center of a cluster of seven. The actual calculation of the position is achieved by extrapolating the new coordinate based on the current point, distance to travel, and bearing as shown below in eq. 4 and eq. 5 where φ is latitude, λ is longtitude, θ is 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:

$$\varphi_2 = \arcsin\big(\sin(\varphi_1)\cos(\delta) + \cos(\varphi_1)\sin(\delta)\cos(\theta)\big) \tag{4}$$

$$\lambda_2 = \lambda_1 + \arctan 2\big(\sin(\theta)\sin(\delta)\cos(\varphi_1), \cos(\delta)\sin(\varphi_1)\sin(\varphi_2)\big) \tag{5}$$

In this fashion, an infinite number of nodes can be generated by the algorithm that conform to this hexagonal pattern. If a UAV drops out of the system due to a failsafe, the pattern is decremented by one at the point closest to where the UAV dropped out. Adding a UAV to the system will send it to the next available position in the pattern.

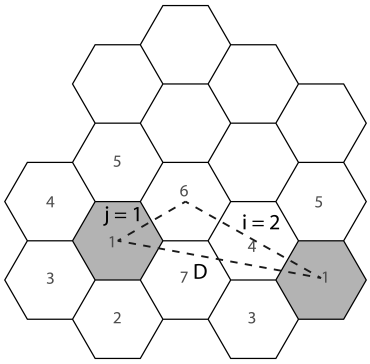


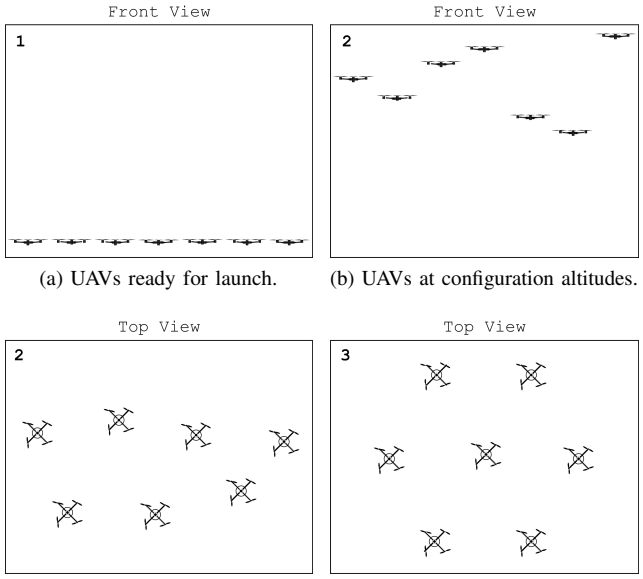
Fig. 6. Cell layout with cluster size $N = 7$

Each UAV is assigned a unique incremental numeric node ID which the algorithm uses for identifying the UAV in the pattern. By changing the ID of interest, the entire network 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, thus replacing it in the system. The algorithm can also shift all node IDs by one, moving all the affected UAVs together as a system.

D. Frontend Interface

The interface layer of this system is built upon a number of modern technologies that all lend to this being a cross-platform and very adaptable application. The first design decision was to make the interface a web browser based interface using HTML. A browser based application has several distinct advantages over native desktop application – one of the bigger advantages is that such an interface will work on any device that has a web browser, which is virtually every computer and most phones and tablets. This means that this system can be operated without creating separate applications for each system.

The HTML interface is built on top of the Twitter Bootstrap Cascading Style Sheets (CSS) framework. This framework can be seen as a set of tools that govern the display of basic UI elements like buttons and inputs. Because HTML is used mainly for generating static documents, creating dynamic interactions requires the use of a helper language to drive those interactions. The main client-side programming language utilized for the interface is JavaScript. The JavaScript program handled the incoming communication from the Python application and also relayed any relevant user commands back to the Python application through WebSockets. AngularJS, a JavaScript library, was used to handle the dynamically changing data and refresh the content on the interface. AngularJS listens for any changes to the data in the script, such as the change in position of a UAV, and as soon as that change occur, updates the view (user interface) with that new data in real-time. Since the WebSockets transport layer also occurs in real-time, there is near instantaneous feedback from changes in the application



(a) UAVs ready for launch. (b) UAVs at configuration altitudes.

(c) UAVs at configuration altitudes. (d) UAVs in final formation.

Fig. 7. UAV launch sequence and configuration procedure.

layer to the UI. The display of the UAVs on a map was handled by the Google Maps JavaScript application program interface (API). This API allowed for the plotting and tracking of the UAV locations on a map.

E. Interface Controls

Included in the HTML interface are individual controls for each UAV as well as the buttons to start the launch sequence. Included also is the ability to automatically connect a UAV through the listed serial ports. For each UAV, there are options to arm and disarm, take off, land, hold altitude, and change modes. In addition, important status information such as altitude and state is also displayed for each UAV. While the position of the UAVs on the map is controlled by the clustering algorithm, it can also be adjusted by simply dragging and dropping an active UAV on the map. The algorithm center can also be selected in this fashion.

F. Process Control

The process control for the system starts with pre-arm hardware checks. This consists of checking for any loose wires, broken or cracked frames and propellers, and ensuring that the battery is properly secured. Upon completion, a physical safety on the UAV is then engaged, indicating that it is flightworthy. On the software-side, pre-arm checks are conducted to ensuring that a good GPS lock is acquired, electronic speed controllers (ESCs) are calibrated, and all essential sensors are functional and calibrated. Most of these internal checks are conducted automatically by the flight control software.

G. Launch Sequence

The launch is initiated from the HTML interface after a center point, cell radius and operating altitude has been selected. Fig. 7a depicts the initial pre-launch step. Initially each UAV

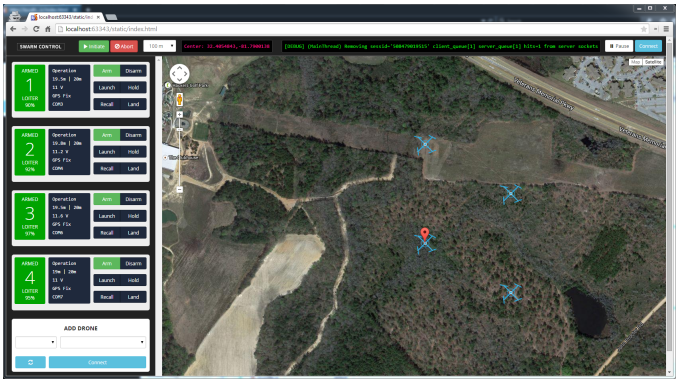


Fig. 8. HTML frontend interface.

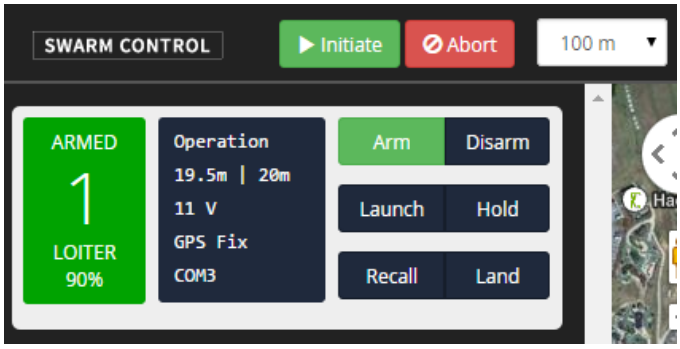


Fig. 9. Available manual overrides.

will fly straight up to its configuration altitude, which is automatically determined by the algorithm as depicted in fig. 7b and fig. 7c. No two UAVs will have the same configuration altitude; this is to ensure that there will be no collisions while moving and as a result removes the need for additional location tracking. After the UAVs reach configuration altitude, they then travel to their respective coordinates, illustrated in fig. 7d. Upon reaching the destination coordinates, the UAVs then settle down to their operating altitudes. For any time that a UAV needs to change position, it will first ascend or descend to its configuration altitude before moving in order to avoid collisions.

H. Fail-safes

This system features a number of fail-safes that are loaded upon initiation of the UAV. Most of these fail-safes are already part of the ArduCopter flight control software. Included failsafe triggers are:

- Radio telemetry loss
- GPS signal loss
- Low battery
- Maximum altitude ceiling
- GPS maximum radius breach

The radio telemetry loss failsafe is activated when the UAV does not receive a heartbeat packet from the groundstation for a set period of time. The battery failsafe 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 failsafe simply checks if the current position is within a specified circle. With the exception of the GPS failsafe, each of these when triggered 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 GPS signal and then will force the UAV to land after a timeout period.

V. Results

A. System Performance

Software performance was the primary concern when dealing with Python since it is one of the slower programming languages. The initial approach to dealing with 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 dealing with simultaneous 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 program on a laptop started showing signs of performance drops and a very noticeable increase in CPU usage over time; this lead to the abandonment of the individual threading concept. The more optimized approach was to use a total of four threads for the whole system, representing the top level responsibilities of the application. The first thread handled all interactions with the HTML frontend and the local "server." The second thread handled all the miscellaneous tasks and system checks that monitored the overall application. The third thread's responsibility was to handle the FSM. The final thread handled all interactions with the UAVs including sending commands and receiving data.

1) HTML Frontend Interface: The UI, shown in fig. 8, is a very efficient way of managing multiple UAVs. The simulation running in fig. 8 shows four UAVs covering an area with a radius of $100m$. The process of adding a new UAV to the system simply consists of connecting the telemetry radio to an open USB port on the computer and then selecting the new device in the dropdown menu shown in fig. 8. This menu automatically generates all the occupied ports and updates them in the list upon clicking the refresh button. The main area of the interface is occupied by a map overlayed with the position of each connected UAV. Each of those UAVs can be manually positioned by dragging their icon and dropping it to a new location. Initiating the sequence is just a matter of clicking the $Initiate$ button as shown in fig. 9. Also shown in fig. 9 are all the common controls available for each UAV.

2) Remote Access: The remote access feature of this system was tested by setting up a web-accessible HTTP tunnel to the local server. This was accomplished using a command line program called ngrok. ngrok works by opening up the local server on a computer and a specific port to one of its servers and then generates a link that can be accessed from anywhere in the world. The system was able to successfully connect to the Python program running on another computer through the

web. Because of the failsafes and the fact that the autopilot handles all the flight dynamics, latency due to remote control was not an issue.

3) Hardware Performance: Flight tests, outside of software simulations, for this system were very limited at the time of this writing due to flight restrictions imposed by the The Federal Aviation Administration (FAA). While waiting for approval, basic operation was tested in an indoor lab and GPS operation was tested outdoors. The GPS reliance of this system limits its operating conditions to outdoor settings with at most moderate cloud coverage. This actually is not a significant detracting factor because operating multirotor UAVs in potentially rainy conditions makes them susceptible to lightning and water damage therefore this system would not be used in such conditions. 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 within a 4 second window. GPS accuracy was generally within a 4 to 5 meter radius.

B. 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 has 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 [15].

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 [16]. Overall this is still a developing scenario and more flexible rules by the FAA are needed.

C. Future Work

- Upon approval from the FAA, the goal is to fully operate the system at varying heights and distances. All the hardware and software has been verified up to this point therefore the system is ready for more thorough field tests.
- Another addition to the system will be the ability to change the swarming algorithm to other presets. This will be done through the HTML interface.
- Finally, the control process will be decentralized from the Python script and instead offloaded to the individual UAVs themselves. The Python program will then handle the monitoring of the system only. This will make the system more of a traditional swarm and make it more reliable.

VI. Conclusion

In this work, a platform for the 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 straightforward and safe in operation. Open-source hardware and software has been heavily utilized in the creation of this system, meaning that 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 easy way of ensuring that emergency personnel have an easy, fast, and adaptable way of communicating in disaster situations.

Acknowledgment

The research leading to these results was funded by the Georgia Southern University College of Engineering and Information Technology 2014 Seed Grant.

References

[1] R. Miller, "Hurricane Katrina: Communications & Infrastructure Impacts," DTIC Document, Tech. Rep., 2006.
[2] *The federal response to Hurricane Katrina : lessons learned.* [Washington D.C.: White House, 2006.
[3] E. Noam, "What the World Trade Center Attack has Shown us About our Communications Networks," in *Global economy and digital society*, Amsterdam; Boston, 2004, ch. 20.
[4] M. Richtel, "Inauguration Crowd Will Test Cellphone Networks," Jan. 2009. [Online]. Available: http://www.nytimes.com/2009/01/19/technology/19cell.html
[5] 3DR, "Pixhawk Autopilot System." [Online]. Available: http://3drobotics.com/pixhawk-autopilot-system/
[6] QGroundControl, "MAVLink Micro Air Vehicle Communication Protocol - QGroundControl GCS," 2014. [Online]. Available: http://qgroundcontrol.org/mavlink/start
[7] L. Meier, P. Tanskanen, F. Fraundorfer, and M. Pollefeys, "PIXHAWK: A system for autonomous flight using onboard computer vision," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, May 2011, pp. 2992–2997.
[8] ArduCopter, "Using the 3DR Radio for Telemetry with APM and PX4." [Online]. Available: http://copter.ardupilot.com/wiki/common-using-the-3dr-radio-for-telemetry-with-apm-and-px4/
[9] C. Horne, "Expandable Open Source Autonomous Quadcopter," *The UNSW Canberra at ADFA Journal of Undergraduate Engineering Research*, vol. 6, no. 1, 2013.
[10] U-blox, ""u-blox 6 GPS, QZSS, GLONASS and Galileo modules"," pp. 1–2, 2012.
[11] M. F. Sanner, "Python: a programming language for software integration and development." *Journal of molecular graphics & modelling*, vol. 17, no. 1, pp. 57–61, Feb. 1999.
[12] A. Tridgell, "MAVProxy." [Online]. Available: http://tridge.github.io/MAVProxy/
[13] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Pearson Education, 1994.
[14] V. H. MacDonald, "Advanced Mobile Phone Service: the Cellular Concept." *Bell Syst Tech J*, vol. 58, no. 1, pp. 15–41, Jan. 1979.
[15] FAA, "Unmanned Aircraft Systems." [Online]. Available: https://www.faa.gov/uas/
[16] "Americas clumsy regulation of drones stirs up frustration, confusion." Washington D.C., Dec. 2014. [Online]. Available: http://www.washingtonpost.com/blogs/innovations/wp/2014/12/09/americas-clumsy-regulation-of-drones-stirs-up-frustration-confusion/