CS

COC251

B526376

Flying, Autonomous WiFi

by

Gareth J. Nunns

Supervisor: Dr P. Tso

Department of Computer Science

Loughborough University

May/June 2018

Abstract

This report covers the research, steps necessary and an initial proof of concept, for producing a flying drone which dynamically positions itself to provide the best WiFi signal to the users connected. This technology could be employed at outdoor gatherings, search and rescue efforts or in flooded areas.

In this report an interactive functional prototype is developed using WebGL, as well as a simplified algorithm on a constructed drone. This drone is controlled by a Raspberry Pi, to which users can connect via the WiFi access point it creates and allows the operator to retrieve a web dashboard, where the key diagnostics and controls for the aircraft are found.

The users' positions are localised by trilateration from the Received Signal Strength (RSS) values from the connected devices. Using these values, the drone aims to maximise the WiFi signal provided to all users. From testing of the drone, flight controlled by the Pi was achieved and a location was calculated for the user, allowing for future work to build on this.

Acknowledgements

I wish to acknowledge my project supervisor, Posco Tso, for his guidance and feedback during the project. Many thanks to my father for supporting me throughout the project, especially with proof-reading, altimeter measurements and soldering. Also a special thank you to Fraser Stockley for proof-reading and filming aspects of the project.

Index terms— Autonomous, Raspberry Pi, Drone, Trilateration, Received Signal Strength, WiFi

Contents

List of Algorithms

List of Figures

List of Tables

Program Listings

Chapter 1

Introduction

1.1 Scope

1.1.1 Purpose

This report aims to summarise and extend current research into the viability of an autonomous drone to provide the best WiFi signal to users below, producing a proof of concept. This has real-world potential applications, such as: search and rescue, concerts or generically the internet of things; furthermore, the autonomous flight could be applied to package delivery or camera tracking, to name a couple of examples.

1.1.2 Background

With the recent increase in drone popularity, due to lower prices & ease of control (Liu et al. 2015), they have become more common-place. Furthermore, with a greater demand for wireless networks (Soh & Kim 2003), this project looks to address the growing need for wireless network in ad hoc, temporary situations (EE 2017). Traditionally this is solved by installing semi-permanent base-stations (Bicheno 2017), which are costly and are based on the foreseen location of where users will be. Floreano & Wood (2015) predict the advent of greater autonomy in the small drone market, under the premise that regulations are lowered. This report looks to explore this emerging technology to solve the demand for an internet connection in various situations.

1.1.3 Applications

This would be ideal for relief efforts in inhospitable areas, be it after flooding, hurricanes or landslides, when there is no simple way to provide an internet connection to those stranded. The drones would be a quickly deployable solution that only require a single source of power; ideally this would be developed into a node based solution which would mean a continuous service could be provided and it would react to the movements of the users below.

For medium to large scale outdoor events, this works as a possible solution to provide a network to those attending. Traditionally one would have to erect multiple masts but this is a much more flexible product. It could be extended to mobile networks as well.

More generically drone manufacturers will be interested in the methods used and potential applications for aerial access points (APs). Whilst drones are still an emerging market, consumers are yet to see the full capabilities of these devices and companies will be looking for innovative solutions - at the core of any solution there will always be a need to avoid obstacles through sensors.

Another audience will simply be drone enthusiasts interested in what is possible with their drone and the problems it can solve.

1.2 Motivation

The motivation of the project is to better current solutions and expand understanding of this field - dynamically moving the location of wireless APs to meet the users needs hasn't been explored in depth as previously it wasn't feasible, which is what makes the project intriguing.

On a personal level, it is interesting to learn and expand my knowledge of the various aspects of this report, including drones, embedded systems, WiFi technology and Python programming. It's exciting to be on the forefront of a new technology which could bring about improvement and change to the world, providing a service where before it was less practical.

1.3 Aims

The aim of this project is to develop a drone controlled by a Raspberry Pi with a wireless AP, so that it can fly autonomously to provide the best coverage to the devices connected. The drone should be safe, with well documented code so that

it is easy to replicate and continue development. The hope is that this project will be further progressed in future years by either myself, or other students.

1.4 Objectives

The objectives of the project will be:

1. Simple flight controlled by the Pi - pitch, roll, yaw (figure 1.1) and altitude commands sent from the Pi which result in the correct movement of the drone
2. Creating a project that is accessible for others to progress - clear, concise code and a user interface that explains what the drone is doing.
3. Maintaining horizontal position - naturally a drone won't hover in a static position, due to the wind or differences in the motors: commonly this is achieved through the use of Global Positioning System (GPS) and correcting for these problems.
4. Maintaining vertical position - similar to the maintaining horizontal position issue, this can be solved by GPS (to a limited extent), radar or barometer.
5. Forward obstacle avoidance - sense an object in front of the drone and take action to avoid a collision. This may be beyond the scope and time frame of this project.
6. A functioning wireless access point with the ability to measure connected devices signal strength - allowing the drone to make a decision of where to move to deliver the best signal.
7. Movement based on the proximity to users connected to the access point, involving ascertaining the location of the connected users from trilateration and then providing the best signal.

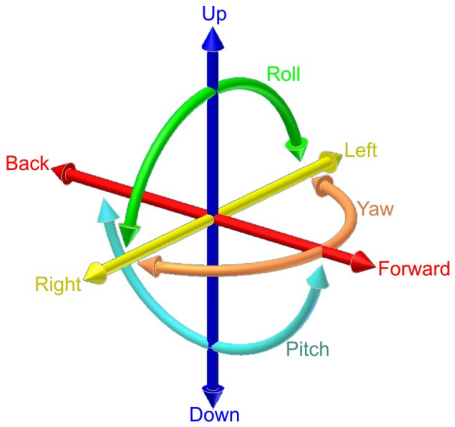


Figure 1.1: Pitch, roll & yaw - from Nigel (2017)

1.5 Constraints

The main constraints for this project were identified as cost and time. These constraints guided the planning of the project, influencing the breadth of what is covered.

1.5.1 Time

The project submission deadline is 30th April 2018 and is not flexible. This limits the total scope of the project: not all of the avenues of this project can be explored in this time frame, but this report looks to provide a founding for knowledge on the topic and show a proof of concept.

1.5.2 Cost

The main expenses of the drone components were approved and paid by Loughborough University. Other small costs were paid for by myself, including the microSD card for the Pi, the battery charger and LiPo-safe bag for batteries, as well as other tools which will be used for future projects.

1.6 Report Outline

This section summarises the contents of each chapter.

Chapter 1, the introduction, gives an overview of why the project has been

undertaken and what it aims to achieve. The aims & objectives provide some metrics for the project and are further explored in the State of the Art, chapter 2.

The literature review is essential to create a firm foundation for advancing knowledge and summarises similar research to this report, with the way it affects the project (Webster & Watson 2002). State of the Art, chapter 2, covers some of the related hardware and software concepts used in the project, drawing from various up-to-date sources, including conference proceedings, websites and articles.

Chapter 3 provides a brief summary of the methodology employed whilst completing tasks, such as the research, design and project management, containing steps used to carry out the project.

The requirements, chapter 4, give SMART points of reference for the project, which can be drawn upon for the design of the project. These are ranked using the MoSCoW system to give input to the Test-Driven Development (TDD).

Chapter 5, Design, draws upon the state of the art and the requirements, chapters 2 & 4 respectively, to formulate the main flight algorithm, considering how the drone will fly in response to the Received Signal Strength from users. This helps ensure the "must-have" features from the requirements are incorporated.

Chapter 6 covers the prototyping, setup of the Raspberry Pi & Flight Controller (FC) and the programming of the Pi, along with any issues that occurred.

Where possible, tests were written prior to or during coding as part of the Test-Driven Development process, however chapter 7 covers a more comprehensive test plan, as well as outlining physical tests due to the outdoor nature of this project. These relate directly to the requirements, where feasible.

Chapter 8 encompasses any issues that were encountered within the various sections of the project, e.g. the design, implementation and testing. Furthermore it covers future work on the subject that could bring about new understanding and benefits to users of a similar system.

The discussion in chapter 9 reconsiders the project's constraints and methods used, reflecting upon the lessons learnt from the project.

The conclusion, chapter 10, ties the report together, evaluating the success of the objectives, coverage of the requirements, and deviations from the work plan, taking into consideration the constraints.

Chapter 2

State of the Art

2.1 Basic Definitions

2.1.1 Drone

Simply put:

"A drone, in a technological context, is an unmanned aircraft. [...] The aircrafts may be remotely controlled or can fly autonomously through software-controlled flight plans in their embedded systems working in conjunction with onboard sensors and GPS" (Rouse et al. 2016)

2.1.2 Raspberry Pi

The Raspberry Pi is an Arm based computer, which typically runs a distribution of the Linux operating system (Cellan-Jones 2011). It is an immensely popular device, now in it's third large iteration (Miller 2017), gaining its popularity from the low price and small size, only slightly larger than a credit card (Hern 2016). The combination of being lightweight, computationally powerful and ease of deployment, make it a good fit for this project (Lu et al. 2017).

2.2 Drones as a Service

Drones as a commercial service have increased in recent years and this increase is forecast to continue (Joshi 2017).

The most widely popularised application of drones currently is aerial cine-

matography (Torres-González et al. 2017). Initially, it was more independent film that made use of drones as most hobbyist drones have a gimbal-stabilised high definition camera (Howell 2015, Rao et al. 2016), however larger production companies have since embraced them for use in feature films (Thomas 2015). This shift from the use of helicopters is largely down to much lower costs; furthermore, there are much shorter setup times, greater safety, and in general they are more efficient for productions (Kulkarni 2017).

Another application is crop monitoring (Valente et al. 2011) and spraying, which DJI (2017*a*) have been at the forefront of. The benefits of using drones include that they save a lot of time, compared to farmers walking the acres on foot, plus the health of the crops can be ensured by spraying the correct volume of pesticide and monitoring the crop colour (Tiwari & Dixit 2015). Similarly to this project, drones are able to provide a service in a remote location to replace expensive traditional methods.

Internet connectivity to the masses is being pursued by many, including by Facebook and Google, however they are able to avoid the traditional, wired Western telecoms model to develop more innovative methods (Edwards 2015):

- Facebook, through its charitable internet.org project, is building a fleet of Aquila drones - lightweight, unmanned flying wings which will fly slowly at 60,000 - 90,000 feet for up to three months, using efficient electric motors powered by batteries fed by large solar panels on the wing (Internet.org 2016, Cellan-Jones 2016). These drones will provide internet by communicating to base stations via infrared lasers. *Figure 2.1a*

- Project Loon by X (formerly Google X and a subsidiary of Alphabet) are developing solar-powered balloons that will float at approximately 65,000 feet for more than three months, with data transmitted from base stations and received by users on their LTE devices in the 3000 miles^2 coverage area (Project Loon 2017). The clusters of balloons will maintain their position by ascending or descending to take advantage of different wind directions (Waters 2017). *Figure 2.1b*

- Space X's Starlink plan is to deploy a network of 800 miniature internet satellites, orbiting 680 miles above the earth to provide a service similar to broadband; however launches have only recently begun and not all of the ground technologies have been confirmed (Pasztor 2018). Users will access the internet via satellite dishes, with Elon Musk aiming for fibre speeds (Meyer 2018, Pasztor 2018). *Figure 2.1c*

(a)

(b)

(c)

Figure 2.1: *A set of three figures:*
Figure 2.1a: Facebook Aquila - from Internet.org (2016)
Figure 2.1b: Project Loon - from Project Loon (2014)
Figure 2.1c: SpaceX's Starlink - from Heathman (2018)

These projects, in comparison to this one, are focused much more on providing connectivity to much larger areas, where the time taken to set up and deploy these solutions is much greater - especially putting satellites in space.

2.3 Hardware

There are many potential options for which drone to use for this project: an off-the-shelf commercially produced drone or a custom-built drone from a large selection of parts available to hobbyists and manufacturers.

2.3.1 Commercial Drone

The research started with commercial drones, looking at the Parrot AR Drone 2.0 (Parrot 2017), however to use this drone for the project would mean deconstructing it and heavily modifying it (Ogden 2012). Also assessed was the PiCopter (PiCopter 2017), which was promising although the project is still in its infancy with little to no distribution chain, also has difficulties adding a distance sensor.

At the time of research, there did not appear to be a drone on the market that was, or could be, controlled by a Raspberry Pi that:

- was open-source
- had a WiFi chip that could be accessed
- had *(or could be modified to add)* a proximity sensor for obstacle avoidance

2.3.2 Constructing a Drone

There are many options for a building a drone which many have summarised, a helpful guide is provided by Rufus (2016). This offers plenty of advice including opting for a Ready to Fly (RTF) kit. As the background of this project is not from an electrical engineering perspective, nor do I have extensive experience in this field, I heeded this advice, exploring ARF (a.k.a. ATF - Almost-Ready-to-Fly/Assemble-to-Fly) kits. The benefit being that one doesn't have to find individual components (motors, Electronic Speed Controllers (ESCs), frame) that match each other.

Both Rufus (2016) and Gonzalez (2017) highly recommend a quadcopter style frame for first time builders and it is the most common style. Notably with quadcopters, if one rotor fails, the others will not be able to compensate for this, which means the drone will crash Rufus (2016). If this project were to be used above crowds, such as at large outdoor events, a hexacopter or octocopter would be beneficial for safety, however to reduce weight, battery use and costs, these will not be considered for this project, focusing instead on quadcopter ARF kits.

Rufus (2016) recommends a frame "size range of 400mm to 500mm" for beginners and as DJI is a leader in drone technology (Glaser 2017), a DJI Flame Wheel F450 ARF Kit has been used for this project. This includes the frame itself, motors, ESCs and propellers; furthermore, it has space to house the Pi, battery and other peripherals.

All that is left is to control the motors from the Pi: this is handled by the Flight Controller (FC), which sends the speed controls to the ESCs; they commonly also contain other useful peripherals like barometers and compasses (le Bon 2015). It is possible to control a motor through an ESC with a Raspberry Pi (Ahmed 2017), however there are plenty of dangers with attempting this and it is already a solved problem. A MultiWii controller was chosen for this project as it's a well-known controller and has open-source resources for controlling it (le Bon 2015, MultiWii 2014*b*).

2.4 Software

In order to localise users connected to the drone's access point and provide the best WiFi signal to them, the following methods were researched.

2.4.1 Localisation

Rozyyev et al. (2011) defines triangulation as such:

"Triangulation does not require knowing the distance between the target node and the neighboring nodes to calculate the distance. It measures the angles between the intersection lines of the target and the reference nodes. The distance can be calculated with those angles. Triangulation only requires two reference points to calculate the location of the target." *(shown in figure 2.2)*

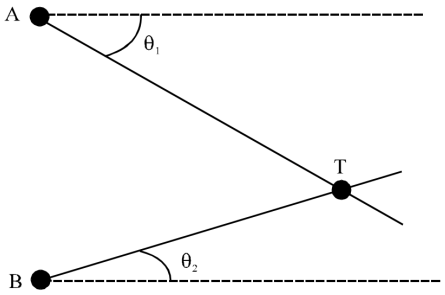


Figure 2.2: Triangulation method - from Rozyyev et al. (2011)

This of course requires knowledge of the angle from the drone to the user. The Pi has a single WiFi antenna (B. Benchoff 2016), however there may be some directionality to it - there is little documentation on this and it was necessary to perform tests, *chapter 5*. If it were very directional, the drone would simply rotate in two locations and, using trivial geometry, pinpoint the user.

If this is not feasible, another method is trilateration, defined as:

"Lateration algorithms can locate a target by using the distance information between the target and several reference points [...]. The distance between the target and a reference point can be calculated either by measuring the time delay between the target and the receiver and then converting it into a distance value, or by measuring the RSS values at the target and then converting them into distance values." (Dag & Arsan 2017, p. 116) *(shown in figure 2.3)*

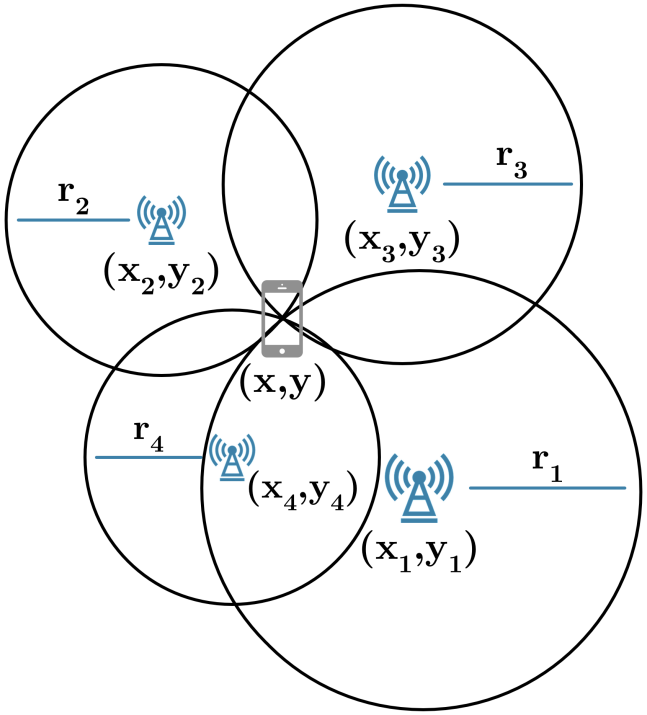


Figure 2.3: Lateration method

The formula for lateration is based on Pythagoras' theorem (Dag & Arsan 2017, p. 118) and can be generically expressed as equation 2.1 (from Dag & Arsan (2017), equation 11), where N is the number of readings; x_i, y_i & r_i are the X & Y coordinates and distance away from drone at point i respectively; similarly x & y are the two unknowns: the X & Y coordinates of the device being localised - the same notation as figure 2.3.

$$\begin{bmatrix} r_1^2 - r_2^2 + x_2^2 + y_2^2 - x_1^2 - y_1^2 \\ r_1^2 - r_3^2 + x_3^2 + y_3^2 - x_1^2 - y_1^2 \\ \cdots \\ r_1^2 - r_N^2 + x_N^2 + y_N^2 - x_1^2 - y_1^2 \end{bmatrix} = \begin{bmatrix} 2 \cdot (x_2 - x_1) & 2 \cdot (y_2 - y_1) \\ 2 \cdot (x_3 - x_1) & 2 \cdot (y_3 - y_1) \\ \cdots & \cdots \\ 2 \cdot (x_N - x_1) & 2 \cdot (y_N - y_1) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \quad (2.1)$$

For example, applying equation 2.1 to figure 2.3, where there are four readings ($N = 4$), gives equation 2.2.

$$\begin{bmatrix} r_1^2 - r_2^2 + x_2^2 + y_2^2 - x_1^2 - y_1^2 \\ r_1^2 - r_3^2 + x_3^2 + y_3^2 - x_1^2 - y_1^2 \\ r_1^2 - r_4^2 + x_4^2 + y_4^2 - x_1^2 - y_1^2 \end{bmatrix} = \begin{bmatrix} 2 \cdot (x_2 - x_1) & 2 \cdot (y_2 - y_1) \\ 2 \cdot (x_3 - x_1) & 2 \cdot (y_3 - y_1) \\ 2 \cdot (x_4 - x_1) & 2 \cdot (y_4 - y_1) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{2.2}$$

In equation 2.1, let C = $\begin{bmatrix} 2 \cdot (x_2 - x_1) & 2 \cdot (y_2 - y_1) \\ 2 \cdot (x_3 - x_1) & 2 \cdot (y_3 - y_1) \\ \cdots & \cdots \\ 2 \cdot (x_N - x_1) & 2 \cdot (y_N - y_1) \end{bmatrix}$

and let D = $\begin{bmatrix} r_1^2 - r_2^2 + x_2^2 + y_2^2 - x_1^2 - y_1^2 \\ r_1^2 - r_3^2 + x_3^2 + y_3^2 - x_1^2 - y_1^2 \\ \cdots \\ r_1^2 - r_N^2 + x_N^2 + y_N^2 - x_1^2 - y_1^2 \end{bmatrix}$

Using these substitutions, equation 2.1 *(and equation 2.2)* can be rearranged to give a unique solution, as shown in equation 2.3 (Dag & Arsan 2017, p.119).

$$D = C \begin{bmatrix} x \\ y \end{bmatrix} \Leftrightarrow \begin{bmatrix} x \\ y \end{bmatrix} = (C^T \cdot C)^{-1} C^T D \tag{2.3}$$

This provides a means to localise the user from a number of readings taken by the drone.

2.4.2 AP Signal Levels

In order to determine the location of the connected devices beneath the drone on the ground, the Pi, behaving as an AP, must be able to establish their relative distances away (Lim et al. 2007).

Received Signal Strength (RSS) is a measurement of the power of the received signal at a device (Benkic et al. 2008), which is typically measured between

0dBm and -100dBm (Dag & Arsan 2017, p.119); for WiFi, typical values range between -30dBm to -90dBm (Metageek 2017). Elnahrawy et al. (2004) found that one could expect a median error of approximately 10ft for localising WiFi signal, with a good algorithm and much sampling. This could provide a metric to use for trilateration, similar to the use of time of arrival with mobile phones and GPS (Zhao 2000), as shown in figure 2.3.

RSS is the simplest and cheapest method for localisation, compared to systems based off the precise time of arrival of the signal or the angle at which the signal arrives to an array of antennas (Tahat et al. 2016, p. 6657). With a large fleet of these drones, it may be possible to use time of arrival in conjunction with a more expensive synchronised clock system, which is outside of the scope of this project. Conversely, RSS values could be stored inexpensively in a shared database between the drones. With respect to the disadvantages of RSS discussed by Tahat et al. (2016), for this application, there will most likely be maintained line-of-sight between the client and drone, furthermore only a low level of accuracy is required, such as within approximately 10 metres. It was necessary to test the fidelity and reliability of the the WiFi antenna on the Pi, *chapter 5*, as there is a lack of literature on this topic.

Equation 2.4 illustrates the relationship between RSS, S, and distance from the access point, r, where A is the RSS at a distance of 1m and n is the path loss exponent (Xu et al. 2010).

$$r = 10^{\frac{S-A}{-10n}} \tag{2.4}$$

The value for A will depend on the strength of the transmitter in the device and hence may well vary between different devices *(tested in chapter 5)*, nonetheless, with a large data set in laboratory conditions it is possible to approximate the value for A (Dag & Arsan 2017, p.120). However, as that would not be possible with the drone in flight and with a user at an unknown location, a simpler heuristic could use a series of readings to approximate the value for A - e.g. using the minimum value or an average of all values.

The path loss exponent refers to attenuation in power of an electromagnetic wave as it disperses, where 2 is in free space and 4 - 6 is obstructed in a building (Mathuranathan 2013); this is affected by a range of factors, including reflection, diffraction, shadowing, etc. (Miranda et al. 2013). As the drone is likely to be in a similar open-air setting for most uses with line-of-sight to the users, one would expect a value close to 2 for the majority of flights - the true value can be ascertained through testing, *chapter 5*.

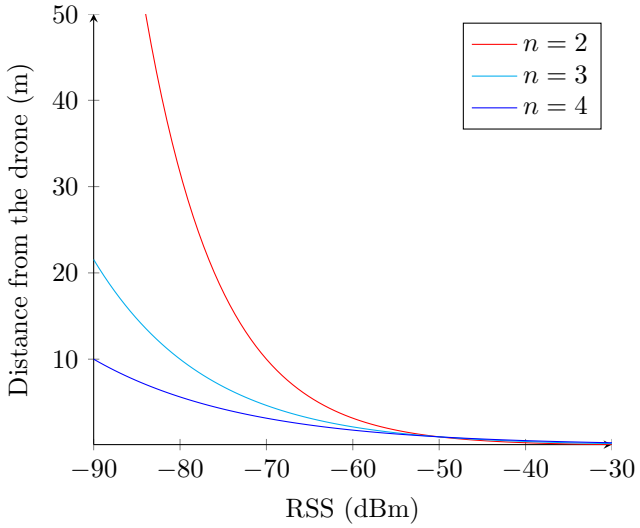


Figure 2.4: The relation between RSS and distance, using equation 2.4, where $A = -50$ and varying the value for n

Figure 2.4 shows the effect of changing the path loss exponent on the RSS at typical distances. If incorrectly approximated, the distance will be magnitudes away from their true value - plotting test data was essential, *chapter 5*. The path loss exponent determines the range of the WiFi, as -80dBm is a minimum for basic connectivity (Metageek 2017).

The caveat is that when referring to RSS in the sense of WiFi, it is typically a client-side measurement and for this project it needs to be monitored on the server side *(at the AP)*, however it is possible to obtain on the server side (Lim et al. 2007, p. 619).

Vasisht et al. (2016) has developed Chronos, a means to localise WiFi users with decimetre accuracy by measuring the time of flight of packets to the multiple antennas of a single WiFi chip. This was applied to an indoor drone using motion capture to track the precise movement of the drone and maintained a steady distance away from the user (Vasisht et al. 2016, p. 175); however, this has not been tested outdoors and may lack accuracy in flight and localisation. This relies on the MIMO antenna (Multiple Input Multiple Output) which unfortunately the Pi does not have built in (B. Benchoff 2016).

2.4.3 Mobility Prediction

As the users on the ground will not be static, it will be important to predict their movements. “Location prediction is the process of predicting a roaming

user's next location given their current location and prior movement history." There are three main options for this (Song et al. 2006, p. 2):

1. Store the transmitter which a user is connected to and when they connect to a new transmitter, so that their path through the transmitter nodes can be predicted, which is commonly used for mobile roaming (Liu et al. 1998). For the proof of concept presented in this research, there will be one, possibly two wireless access points, and hence this method can not be explored fully - however at a large music festival this would be important to deploy.

2. Periodically store the location and velocity of each user, so as to predict their future movements (Aljadhai & Znati 2001, p. 3). These values can be obtained in various ways, including signal-triangulation - with a drone this traditional concept (Lim et al. 2007) can be turned on its head, by moving the drone in a triangular path in the horizontal plane and measuring the varying signal levels to obtain the user's location, e.g. the four sites in figure 2.3 could be the reading taken by the drone on its flight path. Alternatively, one could use GPS data from the user's device (Song et al. 2006, p. 2): this would involve producing a client-side application to send the locations to the drone - this would mean all connected devices must have a GPS chip, uncommon in laptops, hence making it less applicable to this project.

3. Using high-level techniques, such as a map (Soh & Kim 2004) or an office layout (Lu & Bharghavan 1996) to predict users' movements along pre-defined paths. This is not as relevant to my area of research because the drone is likely to be used in situations such as earthquake relief or large outdoor events where these paths are not currently in use.

2.4.4 Location Optimisation

There is a large field of network planning, covering the best place for transmitter locations (Wu et al. 2005). However, there are many challenges locating a single user so this will be an interesting area of future work. A simple heuristic would be to average the predicted positions of the users, shown in equation 2.5, where (x_i, y_i) is the predicted location of user i and there are n users.

$$optimal\ location = \left(\frac{1}{n}\sum_{i=1}^{n} x_i \quad , \quad \frac{1}{n}\sum_{i=1}^{n} y_i \right) \tag{2.5}$$

It will be important to place limits on where the drone can fly; for example, at a festival, the drone should not stray away from the land on which the festival is taking place and only be considering users within that area.

2.4.5 Collision Avoidance

Before a collision is avoided, the obstacle must first be detected (Gageik et al. 2012). Gageik et al. (2015) state:

"Infrared (IR) sensors, like all optical sensors, fail under poor lighting conditions such as smoke or fog and cannot detect diaphanous obstacles; contrary to Ultrasonic (US) sensors, which do not pose such drawbacks. However US sensors cannot detect sound absorbing surfaces like clothes or curtains properly. US sensors are therefore not reliable to detect people or the available distance from them, which is no challenge for IR sensors."

Hence, as in the outdoors most objects the drone is likely to encounter will be sound-reflective and may be translucent, the focus of this project will be on Ultrasonic sensors.

For the purposes of this project, the drone will only be flying forwards in the horizontal plane, so only a simple forward-facing sensor is required, similar to the use by (Wagster & Rose 2012, p. 10), as opposed to the more complex arrays employed by Gageik et al. (2012). In likeness to the way the DJI (2017*b*) *Return To Home Obstacle Check* works, if an obstacle is detected the drone will attempt to fly over/around it.

2.5 Summary

I have presented the state of the art in the field, covering drone usage in industry, potential drones to use for the project and how users' locations will be predicted, including research towards all objectives. The information gathered in this section guides the research and design presented in chapter 5, as well as the requirements and implementation, chapters 4 & 6 respectively.

Chapter 3

Methodology

3.1 Project Management

3.1.1 Project Plan

Figure 3.1 shows the initial project plan drawn up at the start of the project, which laid out at idealised plan, provisioning a fair amount of time to the key aspects of the report: the development and report, with plenty of contingency. The plan is revisited and discussed in chapter 10.

Figure 3.1: Initial project plan

3.1.2 Software Development Life Cycle

The waterfall is the traditional sequential model for software development, where each phase of development is completed before another can proceed. It does not suit this project as a change in requirements or a problem in one of the phases would lead to a badly structured system, or reworking the previous sections (Balaji 2012, p. 27).

The Agile development method is based around the idea of a quickly moving team, where working software is delivered regularly (in weeks, as opposed to months) - the benefit being that the customer can give continuous feedback to ensure satisfaction (Balaji 2012, p. 28). The tasks that the team works on are prioritised in the backlog as to what they will work on next (Highsmith & Cockburn 2001, p. 121). As there is no customer in this project, the Agile model is not well-suited to this project.

"The Test-Driven Development strategy requires writing automated tests prior to developing functional code in small, rapid iterations" (Janzen & Saiedian 2005). These unit tests based off the requirements can then be rerun every time a new version of the code is produced to ensure validity, which makes it ideal for this project. Using Test-Driven Development (TDD) means that as long as the test is designed correctly and the code successfully completes the tests, the code is guaranteed to be correct; however, there is an onus on the developer to know the purpose of the code before it is written.

3.1.3 MoSCoW

The MoSCOW technique is a means of prioritising requirements and is usually defined as Brennan (2009):

- **Must have**: essential to the project and if it is not included the project is deemed a failure.
- **Should have**: these requirements are still important to the project, however are not functionally required.
- **Could have**: will improve the project but are not necessary.
- **Won't have**: least critical requirements that may not be feasible at this time.

It is worth noting that there are other methods such as bubble-sort, 1-10 ranking & planning poker (Kukreja et al. 2012). As there are few requirements

(sub 300), a simple system is ideal and Kukreja et al. (2012) found through survey that there is indifference between bucketed (must have, should have, etc.) and ordinal (a ranked system). Therefore as MoSCoW is clear and unambiguous, it will be used for this project to rank the requirements.

3.2 Research Methodology

The scope of the research in this project was to collect quantitative data, in order to verify existing models and test the precision of sensors. This used applied research to recognise the effect of one variable upon another (Rajasekar et al. 2013). The techniques used and control measures taken are described in chapter 5.

Chapter 4

Requirements

The key requirements of the system are detailed in table 4.1, prioritised using the MoSCoW ranking system.

#	**Priority**	**Requirement**	**Description**
M.1	Must have	A constructed drone that is capable of flight	The project relies on a functioning drone to lift the Pi *acting as an AP*.
M.2	Must have	Drone is controlled by Pi	One of the foci of this project is to control the drone with a Pi. This covers the control of pitch, roll, yaw and throttle.
M.3	Must have	The drone maintains its position in the horizontal plane, with stable pitch, roll and yaw	It is important that the craft is stable throughout flight, so should automatically correct from the effects of the wind or other external factors. To ensure accuracy of the trilateration, there should be an accuracy of +/- 3m in each axis.
M.4	Must have	The drone maintains its position in the vertical axis, within +/- 1m	To ensure accuracy of the flight commands and the trilateration.

#	Priority	Requirement	Description
M.5	Must have	A means to force the drone to land	For the purpose of safety and to ensure the batteries do not become over-exhausted.
M.6	Must have	An emergency stop button which stops the rotors spinning	For the purpose of safety: the drone should stop within 2 seconds of the command being sent.
M.7	Must have	Measuring of the battery level	Ensures the batteries are not depleted to an unsafe level. The operator should be made aware clearly aware of this, either visually or audibly.
M.8	Must have	Pi acting as an AP	For devices to be able to connect to the drone.
M.9	Must have	Log the Received Signal Strength (RSS) of the devices connected to the Pi's AP	In order to trilaterate the locations.
M.10	Must have	Trilaterate the users' locations, based off the RSS	In order to compute where the best location is to provide the best signal to all connected devices.
M.11	Must have	Self-documenting code, which is efficient and well-structured	This will benefit further development of the project and aid debugging.
S.1	Should have	The drone should make movements based on the trilaterated location of multiple connected users	In order to provide the best signal to all connected devices.

#	**Priority**	**Requirement**	**Description**
C.1	Could have	Predict the future location of users, based on the prior predictions made	In order to provide the best signal to all connected devices.
W.1	Won't have	Obstacle avoidance, using US, IR or a similar technology	To protect the drone and nearby objects. This is beyond the scope of this project and it will be assumed that the drone is in a clear space; nonetheless, it still remains a desirable feature.

Table 4.1: Key Requirements

Chapter 5

Design

5.1 WiFi Research

The project relies upon estimating user locations from trilateration of Received Signal Strength (RSS) values, *chapter 2*, hence it was necessary to record data on the relation between distance and the Pi's RSS of connected devices to provide the basis for further decisions.

5.1.1 Test Setup

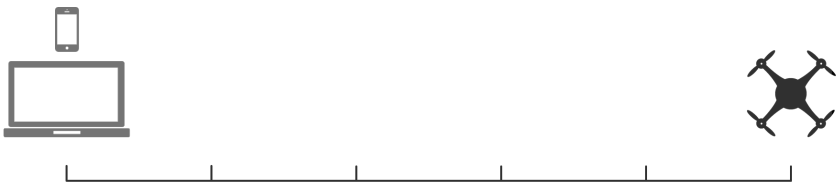


Figure 5.1: General setup used for measuring RSS from the Pi

Figure 5.1 shows the concept of the test setup: measurements on the floor with one metre increments, where the drone, with the Pi attached to it, is placed at one end and the laptop/phone moving away from the drone, along the measurements. The maximum distance was 40m as this was considered to be furthest away the drone would optimistically be from the user, in this proof of concept. In order for the tests to be accurate, these variable factors were

considered:

- The **same devices** were used throughout, as even similar models may have had stronger transmitters, more receptive receivers or had a different RSS scaling. The devices used were:
  - **Pi**: Raspberry Pi 3 Model B
  - **Laptop**: Apple MacBook Pro (13-inch, Mid 2012)
  - **Phone**: Apple iPhone 5 (64GB)

  These devices were considered to be typical for those connecting. During all of the tests where the laptop was involved, an SSH tunnel was running to the drone in order to start/stop/monitor logging; this would have made little difference.
- All of the research was performed at the **same location** - changing location could result in greater reflection/absorption from the surrounding environment. The location chosen was a remote, empty, outdoor car park with a concrete/gravel surface, walled by hedges. This was chosen so that markings could easily be made on the ground and there would be little reflection from the area. Importantly, the test was not performed indoors, as that is not the intended purpose for this project. The drone remained directly on the floor in the same place (facing forward for all range tests) - meaning the data may be slightly inaccurate as during flight it would be clear in all directions.
- Similarly, all testing was recorded on the **same day** and where possible recording comparable sets successively (e.g. when rotating the drone at a fixed distance). The weather was fairly consistent throughout and of course was not controllable - especially the sudden downpour after the final test.
- The logging of the data was all performed internally on the Pi using the **same code** throughout.
- During the tests to ensure consistency, safety and distance accuracy **the drone was not flying** during the tests, however the other boards on the drone were powered on - meaning there would be approximately the same interference, *if any.*
- The Pi was set to a **static WiFi channel**, however there were a couple of other networks in range which may automatically switch to the channel

with the least interference so may have affected the results differently. They were measured to be below -70dBm with the laptop so would have most likely caused little interference.

- All devices were left connected to the drone for at least a minute to ensure any initial **handshaking protocols were avoided** and did not affect the results.
- I attempted to minimise **the human element**: the devices were all held in a similar fashion and stepping between the markers was kept as uniform as practicable.

5.1.2 Logging

The general algorithm used for logging the RSS is outlined in algorithm 1. It was necessary to ping the devices each time to refresh the RSS. The algorithm was implemented using native Unix commands and logged to CSV with Python.

Algorithm 1: Logging RSS of connected devices

```
1 while true do
2 |   devices[] ← getConnectedDevices();
3 |   for device in devices do
4 |   |   ping(device);
5 |   |   RSS ← getRSS(device);
6 |   |   log(device, RSS);
7 |   end
8 |   wait(5);
9 end
```

5.1.3 Range Tests

It was important to first perform the range tests in order to assess the viability of the project and guide the further tests - e.g. *is there a discernible difference in the RSS when the user moves away from the drone within a reasonable range of values?*

The setup for this test was as per figure 5.1. Every five seconds a metre step was taken along the line, with a camera setup to line up the data to when the device was at the various distances. The laptop was disconnected from the Pi's WiFi whilst the phone tests were executed.

Both figures 5.2 & 5.3 show the strong correlation between distance and RSS up until 5 - 10m away from the drone, however beyond that there is a lack of

fidelity - this means that ideally the drone needs to be as close as possible to devices to gain greater accuracy above the significant noise. Hence, the lower the height of the drone the better, but balanced with safety, a hovering height of approximately 5 - 6m would minimise error.

Notably they are not at the same level - e.g. when the drone receives an RSS value of -72 dBm from the phone it is at approximately 5m away from the drone; conversely receiving this level from the laptop, it is more likely to be 20 - 30m away. This difference is most likely caused by a stronger transmitter in the laptop and means that a model must be created on the drone, approximately matching the lines of best fit.

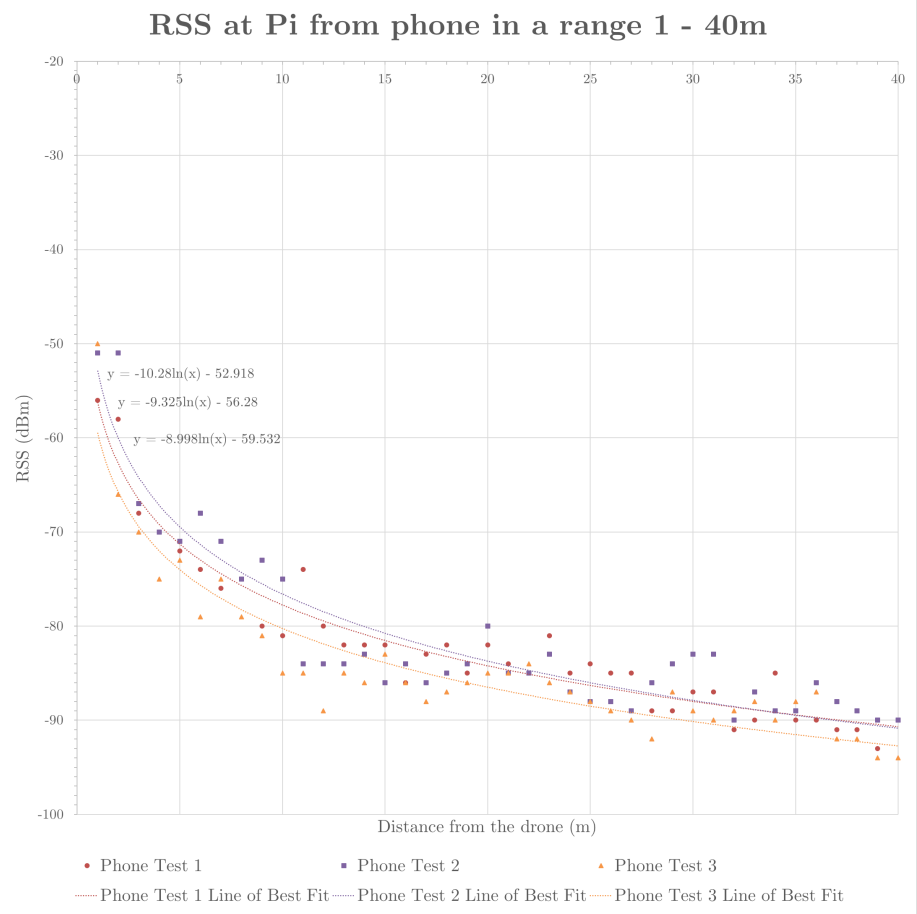


Figure 5.2: The effect on the phone's RSS received by the Pi, when varying the distance of the phone

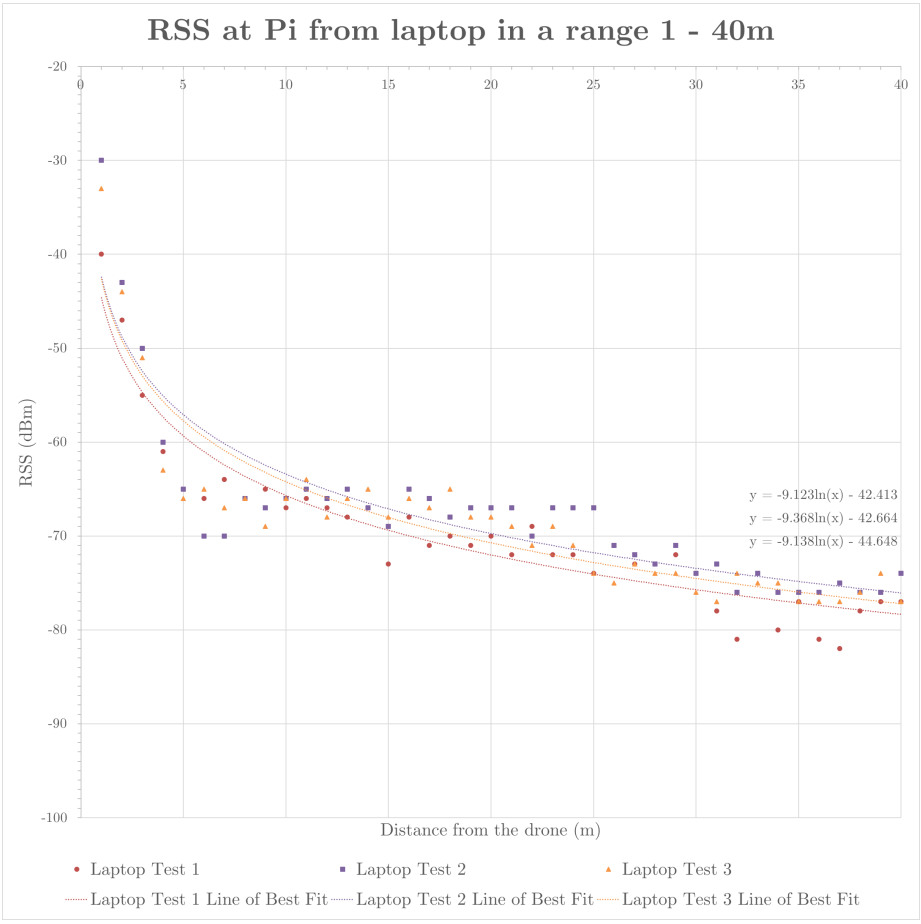


Figure 5.3: The effect on the laptop's RSS received by the Pi, when varying the distance of the laptop

5.1.4 Directional Tests

The aim of these was to see if the WiFi antenna on the Pi, when attached to the drone, was at all directional. If there was a strong correlation then the drone could simply rotate at a number of points, providing an angle at each and triangulation could be used to localise the user, *chapter 2*.

In a similar vein to the previous tests, the directional tests were setup in the same manner as figure 5.1 shows. The routine for these tests was to leave the laptop and phone connected whilst rotating the drone 90 clockwise every 2:05 minutes - in order to log 50 points of data every 2.5 seconds for each orientation. The test was performed with the laptop and phone 5m, 10m and 20m away from the drone as these would be common distances in flight.

The tests were performed in the order: *forward, left, backward, right*, where left means the phone is to the left and vice-versa for the other directions.

The box and whisker plots shown in figures 5.5 - 5.12 show the results from these tests - with the box representing the 25^{th} - 75^{th} percentile data points, the central line as the median, the whiskers indicating the minimum and maximum values and the marker showing the mean of the values.

Figure 5.5 shows an ideal case of rotating the drone, where it should be possible to discern the direction to the user from recording a few data points in each orientation. Unfortunately, the findings from the laptop at the same range, figure 5.6, are slightly less conclusive as the mean values are less than 4dBm apart. There is a notable level of noise in the readings, meaning to draw meaningful conclusions a number of data points should be averaged.

The reason the best reception is found towards the back of the drone is possibly due to the location of the WiFi antenna on the Pi board (Eames 2016), hence receiving very little electromagnetic inference from other boards. In the context of the drone, shown in figure 5.4, the antenna has few obstructions when it is directly behind and this is best reflected in data from when the laptop was 10m away, figure 5.8 - the greater the angle from the back of the drone, the worse the signal, but this is inconsistent with the signal received from the laptop at 20m, figure 5.10.

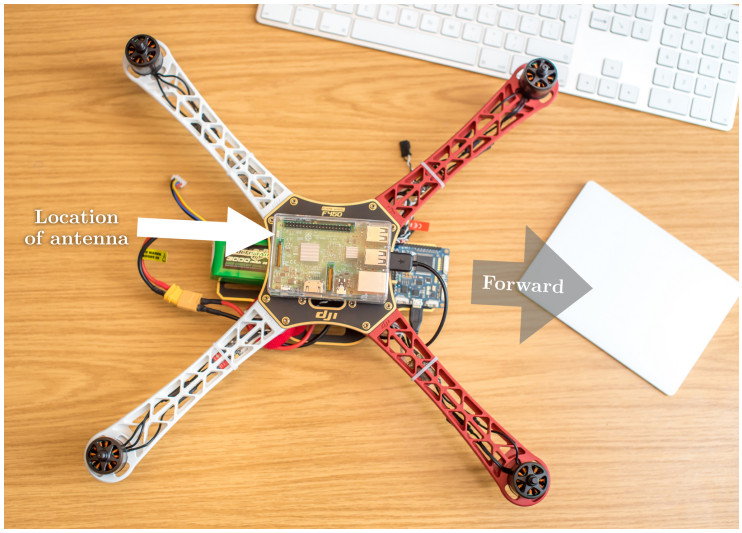


Figure 5.4: Location of the WiFi antenna on the Pi in relation to the drone

Comparing the data at 10m against 20m, figures 5.7 & 5.8 and 5.9 & 5.10 respectively, there is a greater overlap in data but there is still a distinctly

higher trend for the backward values. However, in figure 5.10, the forward and backward data is quite close - meaning that it would not be reliable to base the drone's movements solely from these readings.

Using the data from the aforementioned tests, figures 5.11 and 5.12 compare the forward facing values from the various distances for the phone and laptop. This determines the distance between the points used to trilaterate the user. Based on figure 5.11, within a horizontal plane it may be possible to use a distance between the points of 5m, however due to the sizeable overlap, for greater certainty and considering the larger distances when the drone is airborne, a distance of 10m or more would mean there is likely to be a more identifiable difference in the readings.

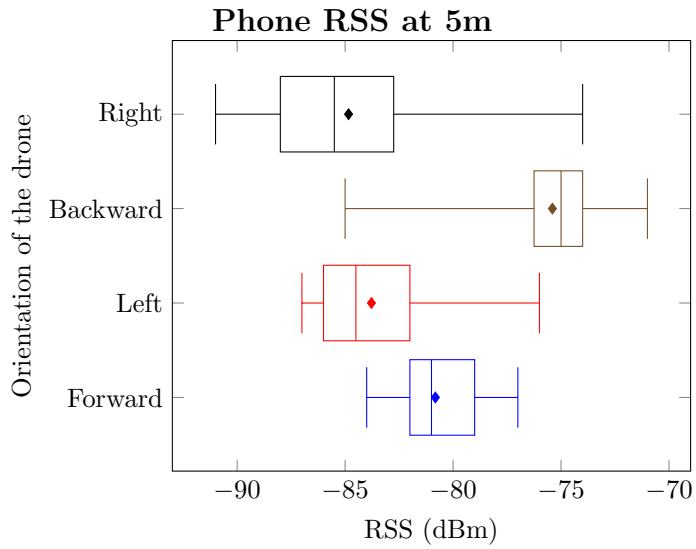


Figure 5.5: The effect of rotating the drone on the RSS received by the Pi from a phone 5m away

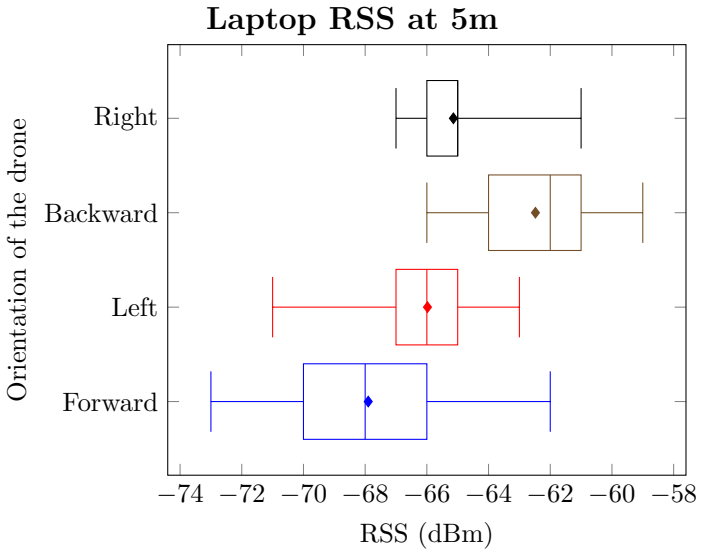


Figure 5.6: The effect of rotating the drone on the RSS received by the Pi from a laptop 5m away

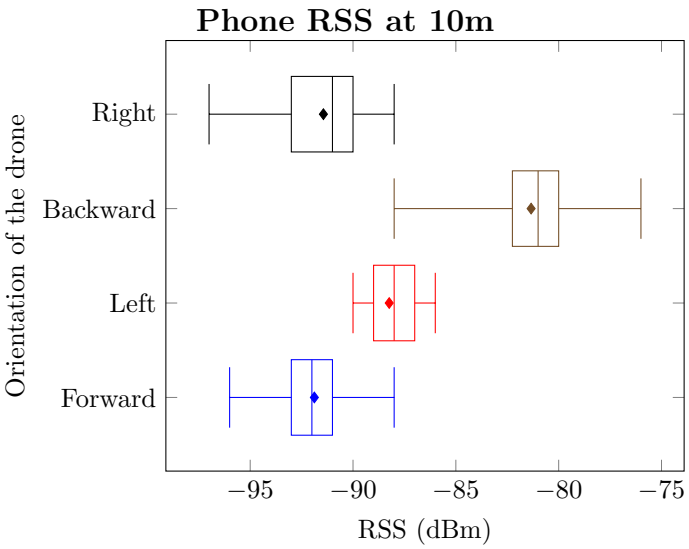


Figure 5.7: The effect of rotating the drone on the RSS received by the Pi from a phone 10m away

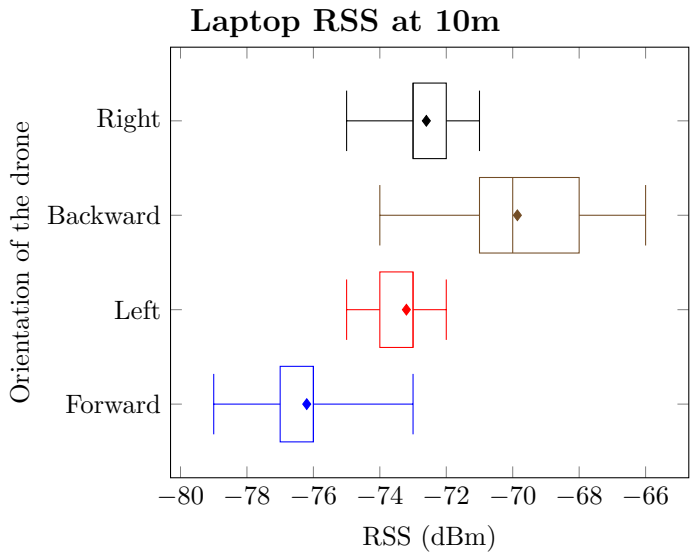


Figure 5.8: The effect of rotating the drone on the RSS received by the Pi from a laptop 10m away

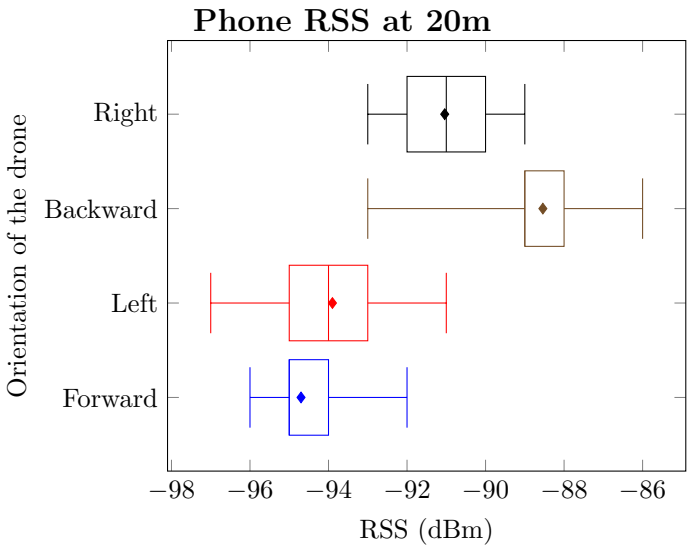


Figure 5.9: The effect of rotating the drone on the RSS received by the Pi from a phone 20m away

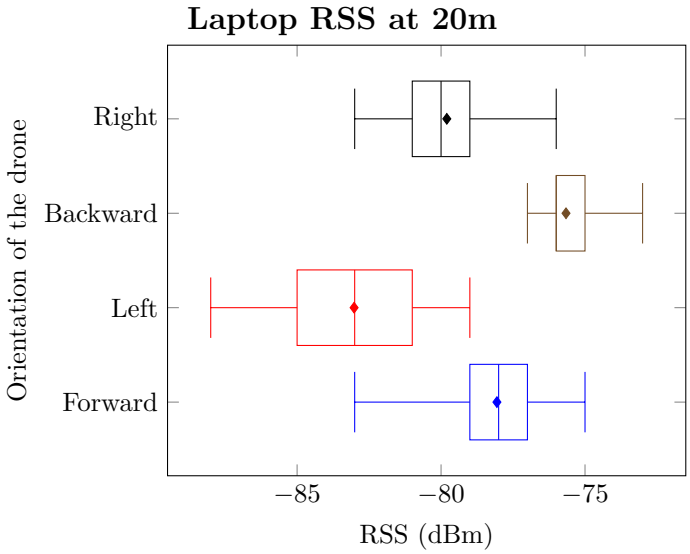


Figure 5.10: The effect of rotating the drone on the RSS received by the Pi from a laptop 20m away

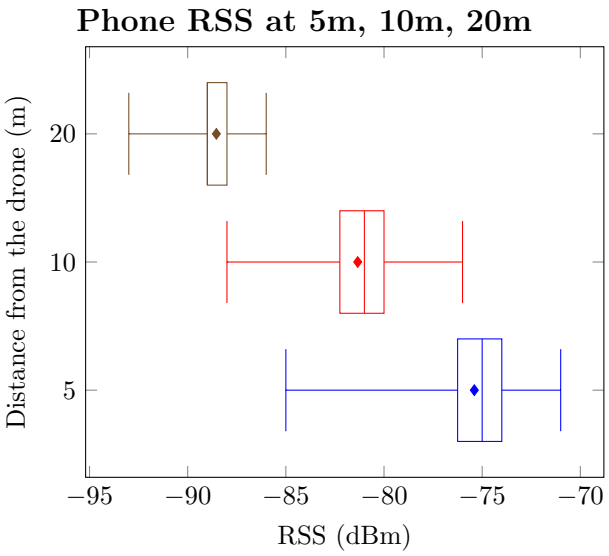


Figure 5.11: The effect of varying the distance on the RSS received by the Pi (facing forward) from a phone, with 50 readings in each range, logged 24/min

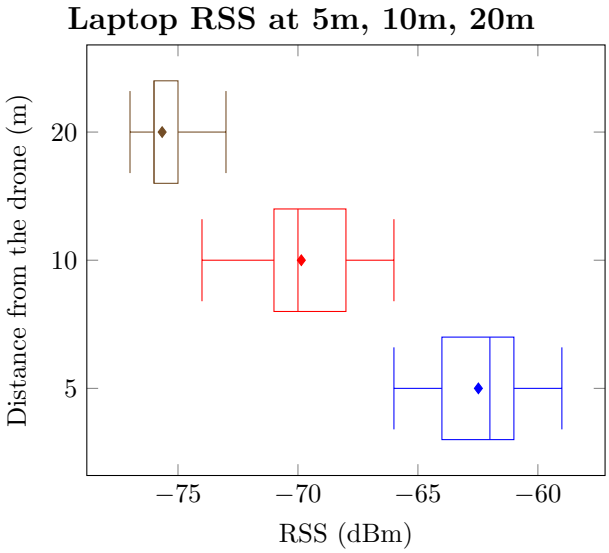


Figure 5.12: The effect of varying the distance on the RSS received by the Pi (facing forward) from a laptop, with 50 readings in each range, logged 24/min

5.2 Drone Sensor Research

It is necessary to ascertain the fidelity of the drone's sensors in order to guide the design of the project's algorithms, including the GPS, altimeter and compass.

5.2.1 Test Setup

As before, all data was logged successively using the same code, with all of the boards on the drone powered up, however not flying. The weather conditions remained foggy throughout, which could potentially have affected GPS and barometer readings.

5.2.2 GPS Tests

Knowing the position of the drone is obviously desirable, as this will be used for maintaining position and trilaterating the location of the users. Whilst the accuracy of GPS is well-documented (U.S. Air Force 2017), the precision of the sensor on this drone (CN-06 v2) may differ. To test, it was simply left static on the ground for five minutes, logging every five seconds, providing 60 data points for each of the three tests. In all of these tests, the GPS readings were based on seven or more satellites on average.

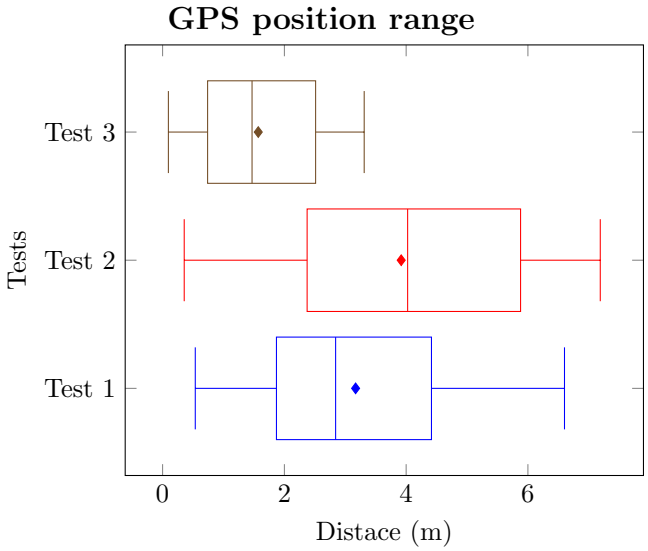


Figure 5.13: GPS reported distance from the initial point when drone remains static on the ground

Figure 5.13 shows the range of data from the initial recorded point - this is not the true location, however the drone does not rely on knowledge of the exact geographic location, only a relative measure from its takeoff position, hence this graph is showing the precision of the data. The precision would likely increase with greater visibility to the sky when the drone was airborne, however, with the

recorded mean averages, this would direct the size of the trilateration towards 10m edges. Mapping the points onto the globe, the average of the geographical location of the points for tests 1 & 3 was less than a metre away from the initial point, conversely for test 2 it was 3.78m (2 d.p.) - given this and the large inter-quartile ranges, it is necessary to average a small number of points.

5.2.3 Altimeter Tests

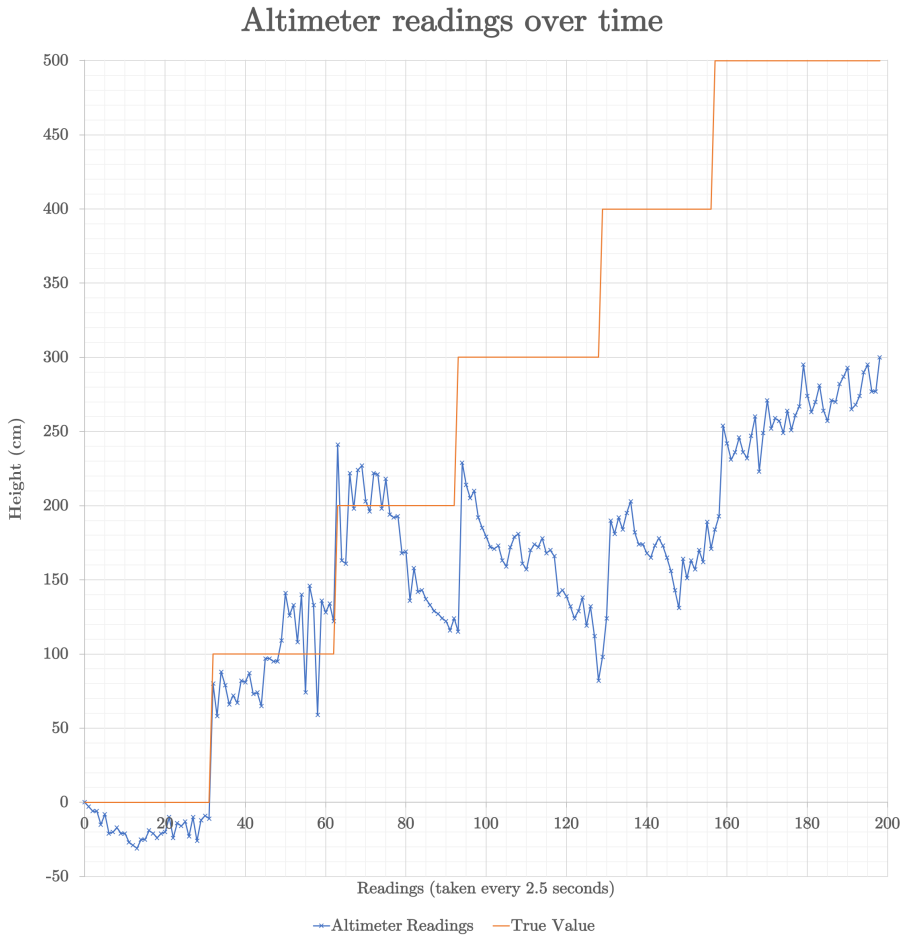


Figure 5.14: Altimeter readings taken from the barometric sensor on the flight controller over time whilst varying altitude

These measurements were taken by hoisting the drone in one metre increments from a ladder *(similar to the method used for measuring RSS, figure 5.1, only vertical)*. Initially the test was performed on a smaller scale (only up to 2m), however the data was thought to be corrupted due to the foggy weather,

so the test was repeated on a clear day with a larger range of height, shown in figure 5.14.

The altimeter data is imprecise and unreliable for the range required, with limited accuracy: traits that are undesirable for the stability of the drone. Efforts were made to verify that the readings were being taken from the Flight Controller's sensors correctly, but to no avail: delaying the project development.

Whilst there is some accuracy to the data, it is not enough for reliable vertical stability of the drone, mandated by requirement M.4 - as a solution, manual control of the throttle was given to the operator, *chapter 6*, which also increased safety. There are also altitude values from GPS, however GPS was primarily designed for horizontal accuracy (U.S. Air Force 2017) and vertical accuracy is typically within 10-20m (Gladstone 2006).

5.2.4 Compass Tests

The values from compass internal to the Flight Controller (FC) were found to be very unpredictable: they would sporadically range between approximately 0 and 410 and change each time the FC was initialised. This in of itself is an inconvenience, however the values were also imprecise and couldn't be resolved to individual directions, e.g. in one test the values for the drone facing north ranged between 392 and 398, and when facing west between 385 and 395. As there was such variation in the data, it was impractical to plot graphically.

With this in mind and not wanting to delay the project further, the proof of concept conceived in this project will not rotate, however algorithms and prototypes can be designed, leaving future work to address this this issue.

5.3 Algorithm

5.3.1 General Principle

Once the drone is flying, the program can essentially be reduced down to a maximisation problem. For example, figure 5.15 shows how the RSS is perceived by the drone from two devices at two separate locations, where one device has a more powerful transmitter than the other, *e.g. a laptop and a phone.* Given this plot, it is trivial to locate the optimum position for the drone to provide the best signal to the two devices.

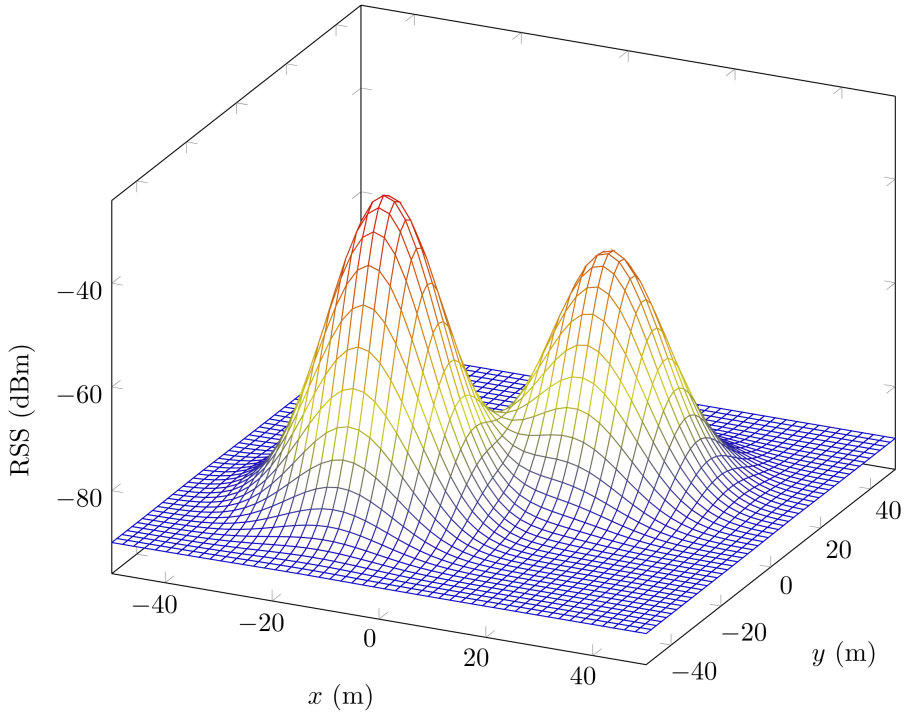


Figure 5.15: Representation of the RSS from two devices when the drone moves in the horizontal plane

However, as seen from the previous tests, there is a lot of noise in the data and the search landscape is more likely going to look like figure 5.16. This is the entire search space at only a single point in time, however the drone can only sample at single (x, y) coordinate. Furthermore, the devices can move, the drone's WiFi antenna is slightly directional and the drone's position is difficult to exactly locate in all six degrees of freedom. With these points in mind, the following algorithm has been designed.

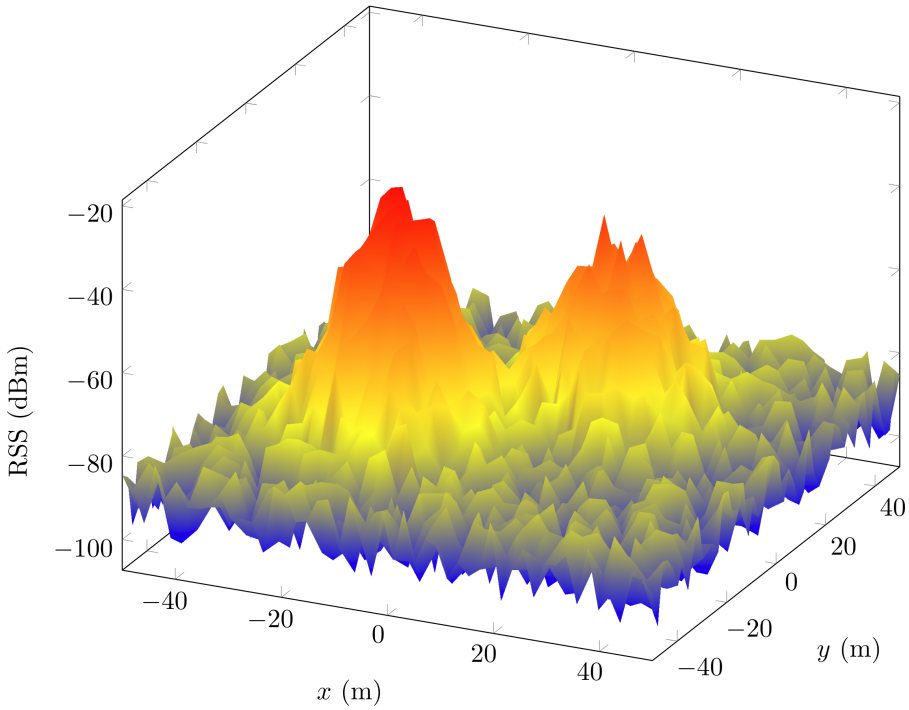


Figure 5.16: Representation of the RSS from two devices, with noise, when the drone moves in the horizontal plane

The drone takes off and looks for the best direction to start searching, then performs a square, starting in that direction. Once the square is completed, the drone trilaterates the users and moves to the position which it decides will give the best signal to the users. This is detailed in algorithm 2, which assumes perfect flight. In addition, there would be further fail-safes and cases where the drone may land, however, these depend more on the exact implementation. In this algorithm, the drone lands after 10 minutes, specified on lines 5 & 6 of algorithm 2 - this would need to be tuned for the real world application.

5.3.2 States

The states defined on line 1 of algorithm 2 are:

- **TAKEOFF**
  The drone's initial state. Gets the drone up to its minimum height of 6m - this is based from the data shown in figures 5.2 & 5.3: the best WiFi distance fidelity is within the 10m range, however for safety, a height of 6m is a balance.

Algorithm 2: General algorithm for the drone

```
1  TAKEOFF ← 0, LANDING ← 1, STARTSEARCH ← 2,
     SEARCH ← 3, OPTIMAL ← 4;
2  state ← TAKEOFF, RSSLog ← [], rotation ← 0, minHeight ← 6;
3  while true do
4  |  RSSLog ← RSSLog + getRSSLog();
5  |  if flightTime > 10 minutes then
6  |  |  state ← LANDING;
7  |  end
8  |  switch state do
9  |  |  case TAKEOFF do
10 |  |  |  if height is 0 then
11 |  |  |  |  takeOffPosition ← currentPosition();
12 |  |  |  end
13 |  |  |  if height < minHeight then
14 |  |  |  |  goUp();
15 |  |  |  else
16 |  |  |  |  state ← STARTSEARCH;
17 |  |  |  end
18 |  |  end
19 |  |  case LANDING do
20 |  |  |  if currentPosition() ≠ takeOffPosition then
21 |  |  |  |  goTo(takeOffPosition);
22 |  |  |  else if height > 0 then
23 |  |  |  |  goDown();
24 |  |  |  else
25 |  |  |  |  quit();
26 |  |  |  end
27 |  |  end
28 |  |  case STARTSEARCH do
29 |  |  |  rotateAndWait();
30 |  |  |  if finished rotating then
31 |  |  |  |  turnTo(bestDirection(RSSLog));
32 |  |  |  |  state ← SEARCH;
33 |  |  |  end
34 |  |  end
35 |  |  case SEARCH do
36 |  |  |  square();
37 |  |  |  if no-one connected during square then
38 |  |  |  |  state ← LANDING;
39 |  |  |  end
40 |  |  |  optimalPosition ← calculateOptimalPosition(RSSLog);
41 |  |  end
42 |  |  case OPTIMAL do
43 |  |  |  if currentPosition() ≠ optimalPosition then
44 |  |  |  |  goTo(optimalPosition);
45 |  |  |  else if no-one connected or weak RSS then
46 |  |  |  |  state ← SEARCH;
47 |  |  end
48 |  end
49 |  wait(1);
50 end
```

- **LANDING**
  Lands the drone in its takeoff position and then powers it down.

- **STARTSEARCH**
  Rotates the drone and waits in each orientation for three seconds in order to log three RSS readings - this is in an effort to assess the direction where the most users are, utilising the slight directionality of the antenna on the Pi, shown in figures 5.5 - 5.10. $rotateAndWait()$ would remain in each of the four compass directions for three seconds in order to log three RSS reading to average. Then $bestDirection()$ would return the direction with the lowest sum of averages from the recent RSS readings in the $RSSLog$.

- **SEARCH**
  The search consists of performing a 10m square. This was chosen because there is approximately only 10m of fidelity in RSS values, shown in figures 5.11 & 5.12. From the data collected whilst performing the square, the drone will trilaterate the position of each user using the lateration formula, equation 2.1. It will then be a matter of predicting where the users will be, based on the recent locations of the user, and then averaging these positions in order to provide the best signal to the most people. An assumption of this algorithm is that there will be at least one device connected: the drone will need to be monitored by an operator, if this operator can not be located, the drone has potentially flown too far, or the operators device has failed - for safety, the drone lands *(defined on lines 37 & 38)*.

- **OPTIMAL**
  The drone moves to where it believes is the best position to provide the best coverage to the devices which have recently connected.

5.3.3 Finding the Optimal Location

As there is not a constant correlation between RSS and distance, shown by the difference in magnitudes between figures 5.2 & 5.3, it is necessary to approximate the values for A, the RSS from the device at one metre, and n, the path loss exponent, in equation 2.4 so that the only unknown is r *(where S is the RSS from the device)* - the distance from drone, restated:

$$r = 10^{\frac{S-A}{-10n}} \qquad \text{(2.4 restated)}$$

Rearranging equation 2.4 to make A the subject gives equation 5.1:

$$A = S + 10n \log_{10} r \tag{5.1}$$

It will be assumed that the lowest absolute RSS values received during the duration of the flight are logged when the drone is above the user and hence both S and r will be known: the drone hovers at 6m and the user's device is assumed to be at approximately 1m from the ground, so $r = 6 - 1 = 5$. To reduce the effect of noise, shown by the box plots in figures 5.5 - 5.10, three values will be averaged.

The value for n can be approximated from the test data shown in figures 5.2 & 5.3, shown in table 5.1:

Line of Best Fit	**Equation**
Phone Test 1	$y = -10.28 \ln x - 52.918$
Phone Test 2	$y = -9.325 \ln x - 56.28$
Phone Test 3	$y = -8.998 \ln x - 59.532$
Laptop Test 1	$y = -9.138 \ln x - 44.648$
Laptop Test 2	$y = -9.123 \ln x - 42.413$
Laptop Test 3	$y = -9.368 \ln x - 42.664$

Table 5.1: Lines of best fit from figures 5.2 & 5.3

As an aside, these show observational values for A.

Converting these from expressions in the form $y = m \ln x + c$ to those with base 10 *(to match equation 5.1)* using the method $y = \frac{m}{\log_{10} e} log_{10} x + c$ produces table 5.2:

Line of Best Fit	**Equation**	*n* ***(3 d.p.)***
Phone Test 1	$y = -23.670 \log_{10} x - 52.918$	2.367
Phone Test 2	$y = -21.472 \log_{10} x - 56.28$	2.147
Phone Test 3	$y = -20.719 \log_{10} x - 59.532$	2.072
Laptop Test 1	$y = -21.006 \log_{10} x - 44.648$	2.101
Laptop Test 2	$y = -21.571 \log_{10} x - 42.413$	2.157
Laptop Test 3	$y = -21.041 \log_{10} x - 42.664$	2.104
	Average	**2.157**

Table 5.2: Lines of best fit from figures 5.2 & 5.3 in log_{10}

From this, the value for n is approximated to 2.2, giving allowances for anything blocking line of sight and other extraneous factors. Filling in the previous unknowns in equation 5.1 gives equation 5.2, where S is the average of the three best RSS recordings:

$$A = S + 10 \cdot 2.2 \cdot \log_{10} 5 \tag{5.2}$$

This is calculated on line 7 of algorithm 3 for all devices.

It is then a matter of applying equations 2.1 and 2.3 to all devices, restated:

$$\begin{bmatrix} r_1^2 - r_2^2 + x_2^2 + y_2^2 - x_1^2 - y_1^2 \\ r_1^2 - r_3^2 + x_3^2 + y_3^2 - x_1^2 - y_1^2 \\ \cdots \\ r_1^2 - r_N^2 + x_N^2 + y_N^2 - x_1^2 - y_1^2 \end{bmatrix} = \begin{bmatrix} 2 \cdot (x_2 - x_1) & 2 \cdot (y_2 - y_1) \\ 2 \cdot (x_3 - x_1) & 2 \cdot (y_3 - y_1) \\ \cdots & \cdots \\ 2 \cdot (x_N - x_1) & 2 \cdot (y_N - y_1) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

(2.1 restated)

$$D = C \begin{bmatrix} x \\ y \end{bmatrix} \Leftrightarrow \begin{bmatrix} x \\ y \end{bmatrix} = (C^T \cdot C)^{-1} C^T D \qquad \text{(2.3 restated)}$$

Arrays C & D in algorithm 3 represent associative arrays of matrices, where the key is the device number and C & D correspond to the matrices in equation 2.3. The result of the matrix multiplication is stored in a similar array, *estimated*, where *estimated*[3][0] refers to the first row of the fourth device's estimated location *(the x coordinate, as per equation 2.3)*.

For simplicity, this algorithm assumes that all devices are connected throughout and the optimal location is simply an average of the estimated locations - *in reality the devices may become disconnected and in the implementation there would be some form of mobility prediction.*

5.4 Summary

There is a clear relationship shown between RSS and distance in figures 5.2 & 5.3, which gives a founded backing to compute the values in table 5.2, meaning the user can be trilaterated. Notably, this proves the viability of the project and satisfies objective 6; however, there is notable noise in the data, *figures 5.5 - 5.12*. The drone's GPS sensor is acceptable for use on this project, however the altimeter and compass are not accurate enough to meet objective 4, so cannot be used - this will affect the implementation on the drone, chapter 6.

Algorithm 3: calculateOptimalPosition()

Input : **RSSLog**: array of RSS values and the location at which they were taken, each stored as an object where $RSSLog[i].x$, $RSSLog[i].y$ & $RSSLog[i].RSSs$ are the X & Y coordinates of the drone and RSS values at the time of the reading i respectively. $.RSSs$ is structured as an associative zero-indexed array, where $.RSSs[0]$ refers to the RSS value from the first device.
devicesCount: number of connected devices

Output: The optimal location as (X, Y) coordinates

$lowestRSSs \leftarrow []$;
for *dev in length(devicesCount)* **do**
  $lowestRSSs[dev] \leftarrow threeLowestValues(RSSLog, dev)$;
end
$n \leftarrow 2.2, A \leftarrow [], C \leftarrow [], D \leftarrow [], estimated \leftarrow []$;
for *dev in length(devicesCount)* **do**
  $A[dev] \leftarrow lowestRSSs[dev] + 10 \cdot n \cdot log10(minHeight - 1)$;
  $C[dev] \leftarrow [], D[dev] \leftarrow []$;
end
for *i in length(RSSLog)* **do**
  $RSSLog[i].distances \leftarrow []$;
  for *dev in length(devicesCount)* **do**
    $RSSLog[i].distances[dev] \leftarrow 10^{\frac{RSSLog[i].RSSs[dev] - A[dev]}{-10 \cdot n}}$;
  end
end
for *i in 1 to length(RSSLog)* **do**
  $CLine \leftarrow$
  $[2 \cdot (RSSLog[i].x - RSSLog[0].x), 2 \cdot (RSSLog[i].y - RSSLog[0].y)]$;
  for *dev in length(devicesCount)* **do**
    addMatrixLine(C[dev], CLine);
    $DLine \leftarrow [$
    $(RSSLog[0].RSSs[dev])^2 - (RSSLog[i].RSSs[dev])^2 +$
    $(RSSLog[i].x)^2 + (RSSLog[i].y)^2 -$
    $(RSSLog[0].x)^2 - (RSSLog[0].y)^2]$;
    addMatrixLine(D[dev], DLine);
  end
end
for *dev in length(devicesCount)* **do**
  $estimated[dev] \leftarrow (C[dev]^T \cdot C[dev])^{-1} \cdot C[dev]^T \cdot D[dev]$
end
$optimalx = 0, optimaly = 0$;
for *dev in length(devicesCount)* **do**
  $optimalx = optimalx + (estimated[dev][0]/n)$;
  $optimaly = optimaly + (estimated[dev][1]/n)$;
end
return (optimalx, optimaly);

Chapter 6

Implementation

6.1 Prototype

To avoid potential disasters, a virtual prototype was built: reducing the possibility of algorithmic problems whilst in the air, like non-termination of the program or damage to the drone.

6.1.1 Technology

Figure 6.1: Prototype in the **starting search** state *(rendered in Google Chrome)*

The prototype was written in JavaScript (JS) with the Three.js library, which renders to a WebGL HTML canvas, much alike figure 6.1. This was chosen as it was fast to develop, gave visual feedback and didn't require specialist knowledge of the physics. This is as opposed to either writing a program specific simulation from scratch, or using a maths/physics based package, like MATLAB, Maple or Simulink.

6.1.2 Overview

The elements of the prototype are the:

- **pink cuboid**, representing the location of the user. This can either remain static or is controllable via the arrow keys.
- **dark blue cuboid**, representing the estimated location of the user. This is not shown until the an estimate has been made, e.g. it is not visible in figure 6.1 when the drone is initially starting to search, however is visible in figures 6.3 & 6.2, when the drone has already searched once.
- **purple rounded triangular prism**, representing the drone. Triangular so that it is clear which way it is facing.
- **green plane**, representing a 100m^2 field. However the drone has been limited to only operate within 50m^2, centred on its take off location. Similarly, the user is limited to the central 50m^2 of the field.
- **debug output**, showing key information about the the drone's state and estimates.

It is possible to adjust the location of the camera using mouse gestures, shown in 6.2.

The JS prototype is largely an implementation of algorithm 2. To make the movements and decisions of the drone clearest, there is only one user in the prototype, however, wherever possible it was written to accommodate additional users.

The drone will take off and enter the starting search state, shown in figure 6.1, where it rotates to each compass orientation and waits at each for three seconds to log three RSS values for the user. It then moves into the searching state and executes a square, starting in the best direction; having completed this, the drone will estimate the location of the user and move to where it considers

the optimal position, using a heuristic of the user's previous estimated locations. The drone transitions into the optimal state, picture in figure 6.2. This then repeats when the user's signal is considered to be poor or 30 seconds have passed in the optimal state and the drone will start searching again, shown in figure 6.3.

Figure 6.2: Prototype in the **optimal** state *(rendered in Google Chrome)*

Figure 6.3: Prototype in the **searching** state again after finding an estimate for the user *(rendered in Google Chrome)*

6.1.3 Simulating RSS

Listing 6.1 shows the method used to simulate the RSS values from the user, where the inputs are the drone, `drone`, and any user, `obj`. The function is based on equation 2.4, shown in line 10, where `n` is the path loss exponent and `a` is the RSS from the device at one metre - these are based on the observed values from the phone, figure 5.2.

```
1  function perceivedRSS(drone,obj) {
2    // estimate what the RSS is for the obj from the drone
3
4    distance = distanceBetweenObjects(drone,obj);
5
6    // rough relation between distance and RSS (from recorded data)
7    var n = 2.2;
8    var a = -55;
9
10   var RSS = -10 * n * Math.log10(distance) + a;
11
12   angleToUser = getHeading(drone) - angleToObject(drone,obj);
13
14   // the signal is better when the user is to the rear of the drone
15   // this is an approximation using a bell curve
16   backwardFactor = normalPDF(angleToUser,180,45) * 2;
17
18   // when the user is closer the effect seems to be greater
19   // this is approximation of how that works (from recorded data)
20   // y = -2.453ln(x) + 13.747
21   RSS += backwardFactor * (-2.453 * Math.log(distance) + 13.747);
22
23   // need to add noise
24   var noiseVariance = 2 // rough variance of the noise
25
26   // it seems the noise is approximately 1.5x larger at the back
27   noiseVariance *= 1 + 0.5 * backwardFactor;
28
29   RSS += (randomNormal() - 0.5) * (noiseVariance);
30
31   return Math.round(RSS);
32 }
```

Listing 6.1: `perceivedRSS` function from prototype

To account for the increased signal when the user is to the back of the drone, the `backwardFactor` is greatest when the user is directly behind the drone *(when `angleToUser` is 180)*. As the exact relation between the RSS from a device and the angle to drone is unclear, it was approximated to a normal distribution,

displayed in figure 6.4 and assigned on line 16.

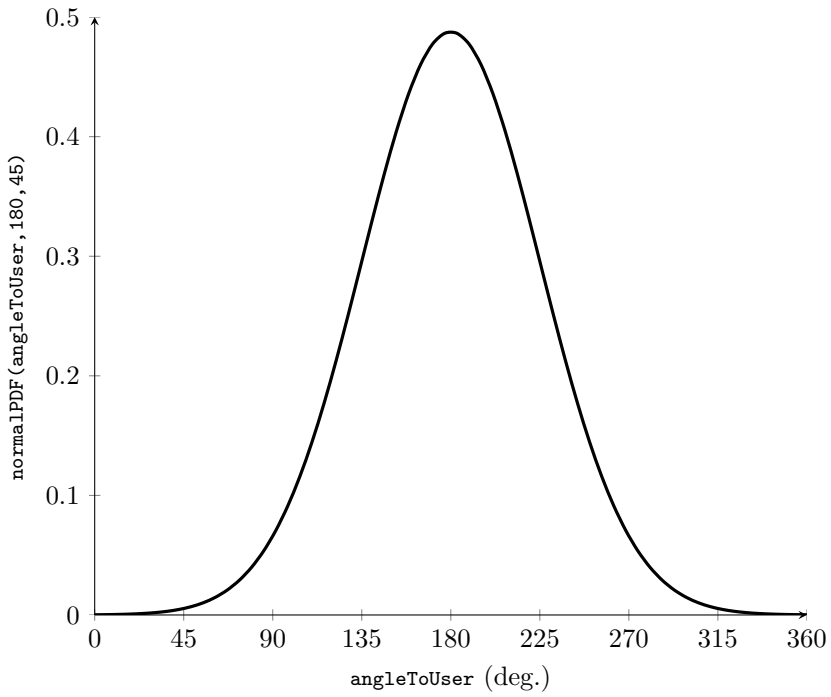


Figure 6.4: Output of `normalPDF(angleToUser,180,45)` used in listing 6.1

From figures 5.2 - 5.12, it is clear that there is a lot of noise in the data, so noise is added to the data, as explained by the comments in listing 6.1, where `randomNormal()` returns a random number, from a normal distribution with $\mu = 0$ & $\sigma^2 = 1$.

6.1.4 Data Structures

Through developing this prototype, considerations were made for how the data should be stored whilst the drone was operating. For example, to log the RSS values, the `RSSLogItem` & `RSS` objects were created, shown in listing 6.2. The `RSSLog` consists of an array of `RSSLogItem` elements, with each of these elements containing the time, position and heading of the RSS readings; the readings themselves are stored in `RSSLogItem.RSSs`, which is an array of `RSS` objects. These `RSS` elements are for each device connected at the time of the reading, storing the MAC address, IP address and Received Signal Strength from the device.

```
1  class RSS {
2    // an RSS measurement (stored in RSSLogItem.RSSs)
3    constructor(signal, MAC, IP) {
```

```
    this.MAC = MAC;
    this.IP = IP;
    this.signal = signal;
  }
}

class RSSLogItem {
  // a single log entry
  constructor(position, RSSs, time, heading) {
    this.position = new point3D(
      position.x,
      position.y,
      position.z,
    );
    this.RSSs = RSSs; // array of RSS elements
    this.time = time;
    this.heading = heading;
  }
}
```

Listing 6.2: RSS Classes (`RSS` & `RSSLogItem`) from prototype

6.1.5 Findings

The prototype was useful for establishing:

- How to implement algorithm 2 in the sense of a embedded system where the program loops multiple times a second - hence there is little context in the current loop and it must be stored in global variables.
- The application of the maths with realistic data and seeing how the estimates can at times be quite far from the true value - hence it was necessary to add some limitations and assumptions to make sure the drone did not fly too far from its initial location.
- Which data to store and how it should be stored, including the construction of objects and the use of global variables - like storing the starting time of when a state had been entered.
- When to start searching for the user again once the optimal location had been decided upon: in the prototype the drone remains at the optimal position for 30 seconds, unless the RSS from the connected devices drops below -80dBm (as the service provided is poor below this level (Metageek 2017)), or the current average signal from the connected devices is 40% worse than the best values received.

6.2 Drone

This section covers the construction and configuration of the components on the drone, figure 6.5.

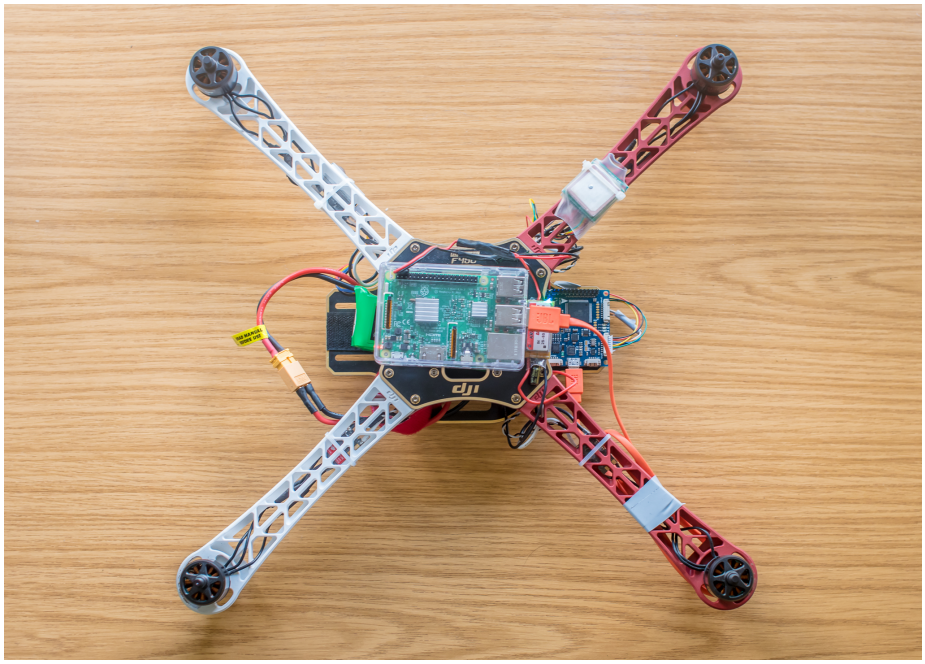


Figure 6.5: Completed drone from above

6.2.1 Components

The components used in this project were:

- Raspberry Pi 3 Model B in clear case *(indicated as 1 on figure 6.6)*
- Crius CN-06 v2.0 GPS *(indicated as 2 on figure 6.6)*
- CRIUS All in One Pro Flight Controller v2.0 *(indicated as 4 on figure 6.6)*
- DJI Flame Wheel F450 ARF Kit, including the frame *(indicated as 5 on figure 6.6)*, DJI 2312E motors *(indicated as 3 on figure 6.6)*, ESCs *(indicated as 7 on figure 6.7)* and DJI self-tightening 9443 propellers
- Multistar 3S 3000mAh batteries *(indicated as 6 on figure 6.6)*
- HobbyKing Lipoly Low Voltage Alarm (2s-4s) *(indicated as 8 on figure 6.7)*

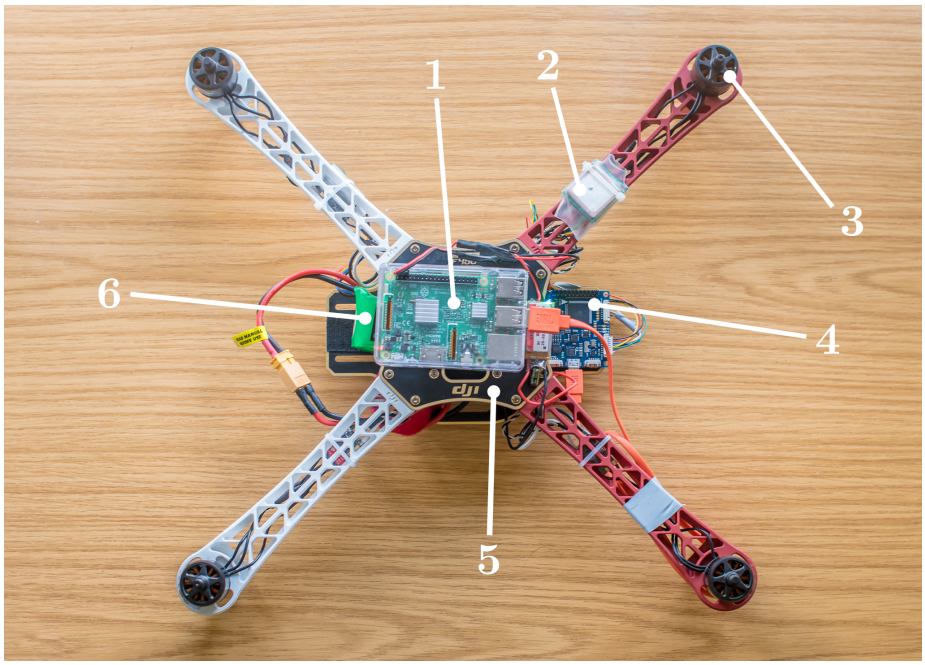


Figure 6.6: Completed drone from above, labelled

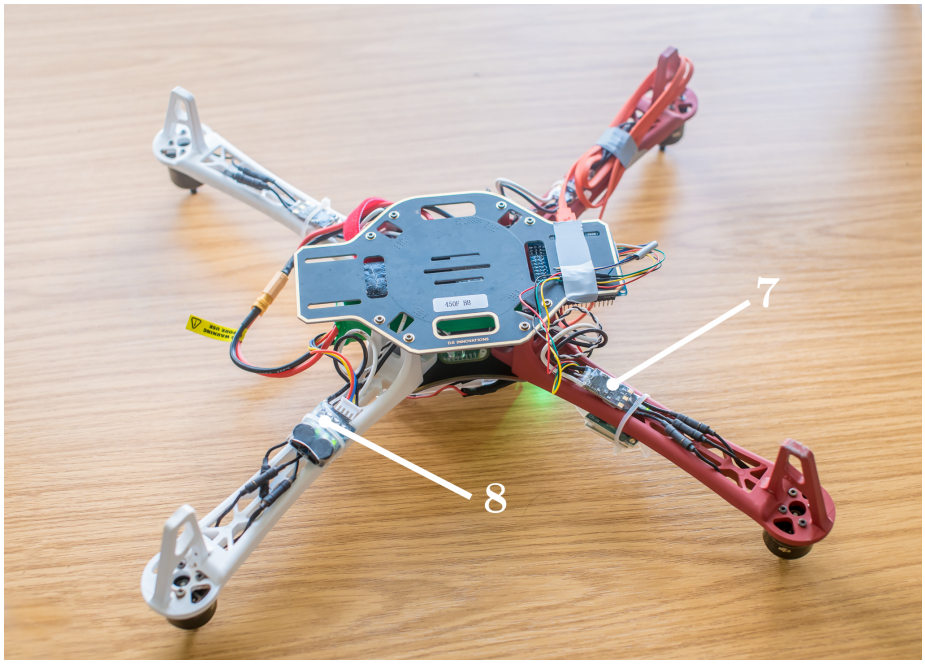


Figure 6.7: Completed drone from below, labelled

6.2.2 Build

First the frame was constructed, figure 6.8. There are multiple guides available to aid with the construction of the drone, including the official video by DJI (2012). This involved soldering a XT60 connector to the bottom board of the frame (the yellow connector seen in figure 6.5), which acts as power rails to the ESCs, which were also soldered on. The arms were screwed into top and bottom boards with Allen keys. Similarly, the motors were screwed into the frame and connected to the ESCs using the three phase cables, shown clearly in the lower right-hand side of figure 6.7. Figure 6.8 shows the completion of these steps.

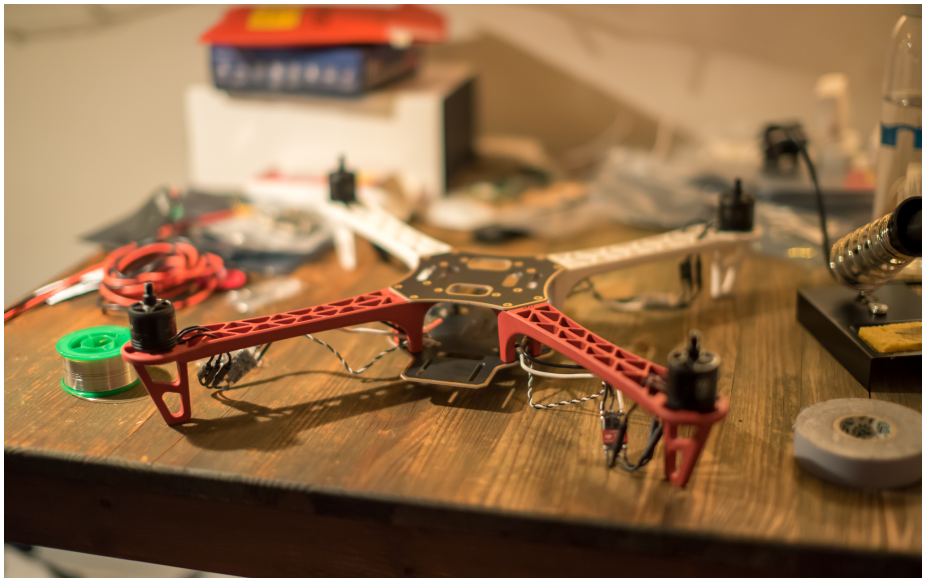


Figure 6.8: Construction of the drone's frame

The Flight Controller (FC) was affixed to the front of the bottom board of the frame using double sides foam tape; the front being the two red arms of the drone, shown in figure 6.5.

Figure 6.9 shows the configuration of the motors and their corresponding motor connection pin number on the FC, as detailed in the various guides for the FC (Gaza07 2012, Jumpy07 & Gaza07 2013). The motor connections on the FC can be seen in figure 6.10 - the four connectors plugged into the series of pins closest to the centre of the drone.

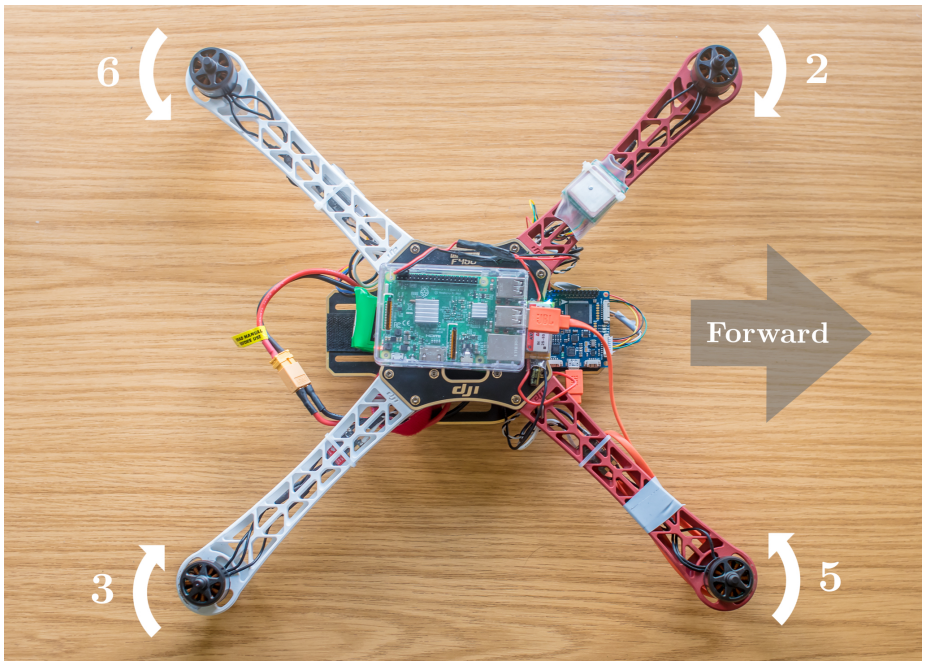


Figure 6.9: Direction of the motors and the pins they connect to

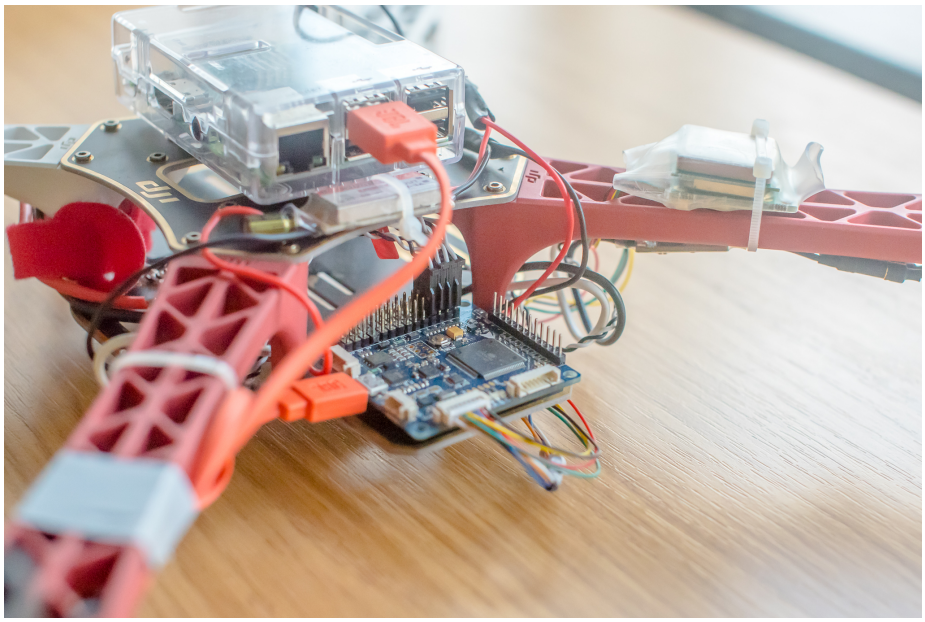


Figure 6.10: Connections to the flight controller on the drone

The GPS is connected to the serial pins of the FC, shown in the foreground of figure 6.10, with the receiver pin on the FC connected to the transmitter pin of the GPS module (Gaza07 2012, p. 8). The GPS is then attached to one of

the arms of the drone using a cable tie, with the antenna facing upward, shown in figures 6.5 & 6.10. The FC is connected to the Pi via the orange USB cable, which is used to send the control signals.

Both the Pi and the FC are powered by a 5V 3A Universal Battery Elimination Circuit (UBEC) *(shown under the USB connections on the Pi in figure 6.10)*, which regulates the battery's voltage and converts it to a consistent 5V. As the FC is being powered externally, the jumper pin was removed (Gaza07 2012, p. 6). The power is connected to the Pi via the General Purpose Input Output (GPIO) pins, the red & black cables in figure 6.5.

Likewise, the case for the Pi is attached to the top board using double sided foam tape, as well as a cable tie underneath. The Pi itself is then screwed into the case, however, there is no mechanism for securing the top of the case down - electrical tape can be used, to allow future access to the Pi.

To power the drone, the battery is plugged into the XT60 connector, the yellow connection in figure 6.5. The battery also plugs into the low voltage alarm *(indicated as 8 on figure 6.7)*, which emits a loud tone when the battery could become damaged if continued to be used. It is fastened to the underside of the drone so that it is also visible during flight. The battery's voltage level could have also been connected to the FC, however it was thought best not to have a single point of failure.

6.2.3 Configuring Components

In order to confirm the GPS module was working correctly and to store settings to it, a custom cable was made to connect the GPS to a USB. A config file was sent to the module and stored to the EEPROM (memory). Figure 6.11 shows ubloc's u-center software confirming the GPS module works at 38400 baud.

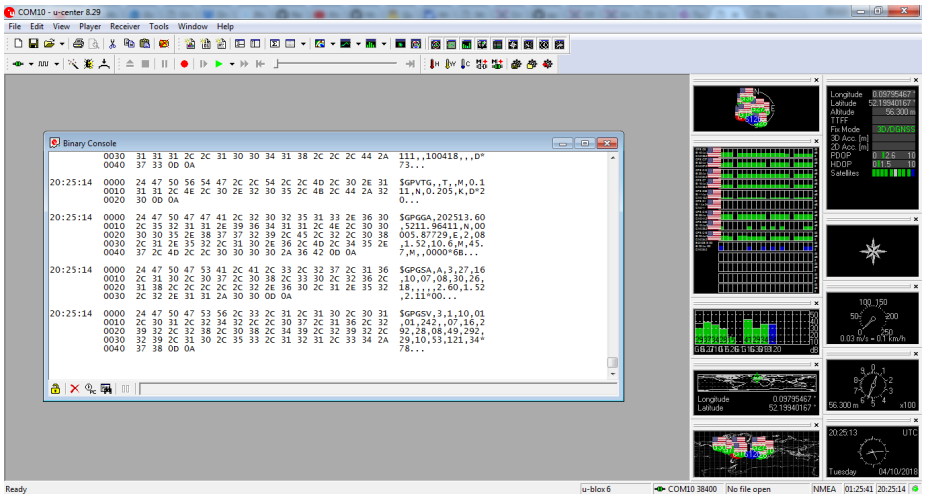


Figure 6.11: ublox u-center GUI

The settings of the FC were configured in `config.h` of the MultiWii source code (MultiWii 2015*b*), *with other minimal changes made to the code.* The key settings are:

- `#define QUADX` - the layout of the drone's motors
- `#define CRIUS_AIO_PRO` - the type of FC
- GPS settings:
  - `#define GPS_SERIAL 2` - the serial port that the GPS is connected to
  - `#define GPS_BAUD 38400` - the baud rate for communication with the GPS
  - `#define UBLOX` - the GPS protocol used by the Crius CN-06 v2.0

These settings were confirmed by the use of the MultiWiiConf software, which also shows the status of all of the peripherals, figure 6.12. Also seen on this screen are PID values, which affect the stability and handling of the drone (Liang 2013), which were tuned to benefit object 3.

Figure 6.12: MultiWiiConf GUI

6.2.4 Safety

It would be remiss of me not to include the common, key safety points associated with this equipment:

- **Lithium Polymer (LiPo) Batteries**
  There are many comprehensive guides on the safe handling of LiPo batteries (HobbyKing 2018) and the consequences of malpractice with Lithium-based batteries are severe (Samsung Group 2017). The key points can be summarised under the topics of:
  - **Storage**
    The batteries should be stored in a LiPo safe (fire proof) container at 3.8V per cell, *i.e. for the three cell batteries used in thise project, they should be stored at 11.4V*. Check the voltage of the battery if they have not been used for a long period of time ($>$ 6 months).
  - **Charging**
    Charge the batteries using a balanced charger designed for LiPo batteries, keeping them in the fire proof container. Ensure that they are not over charged and never charge a battery with a puncture or any sign of swelling.
- **Propeller Safe Practice**
  Only attach the propellers when flying the drone: they are fragile and will

break easily; also, they rotate at a rate fast enough to break fingers - avoid contact with them whilst they are in motion. During all programming, the propellers remained off the drone, as seen in the various figures.

There is also the risk to the wider public - fly responsibly and within the law (Ministry of Defence & Military Aviation Authority 2017). For all tests in this project using experimental code, the drone had a tether attached to limit its range and prevent loss.

- **Internet Safety**
  Strong security should be used when connecting the drone to the internet - there is potential for control to fall into the wrong hands. All of the testing was undertaken offline.

Throughout the project, all of these points were adhered to and no damages occurred.

Please note, improper handling of this equipment can result in serious injury and legal consequences: the author holds no legal standing or liability for injury or damage caused by the further use of the drone built in this project, nor attempts to replicate a similar aircraft.

6.2.5 Pi Configuration

The Raspberry Pi 3 Model B, with a distribution of Debian installed on a 16GB microSD card, was configured as an AP using Hostapd and ISC DHCP was used to assign IP addresses.

The secured network was configured with the `hostapd.conf` file shown in listing 6.3. Using this configuration, the drone will appear in the WiFi devices list of other devices as "pidrone", and connected to with the password "garethnunns"; however, with only Hostapd alone, the Pi will not assign the connecting device an IP address and the connection will fail.

```
1  interface=wlan0
2  driver=nl80211
3  ssid=pidrone
4  hw_mode=g
5  channel=7
6  wmm_enabled=0
7  macaddr_acl=0
8  auth_algs=1
9  ignore_broadcast_ssid=0
10 wpa=2
```

```
wpa_passphrase=garethnunns
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
```

Listing 6.3: `/etc/hostapd/hostapd.conf` file on the Pi

Listing 6.4 shows how IP addresses are assigned to those who connect to the Pi. The drone operates in the 192.168.10.0/24 IP space, with the Pi acting as a router on 192.168.10.1 and addresses are assigned to those who connect in the range 192.168.10.10 - 192.168.10.254. This allows for plenty of devices to connect to the drone and leaves space for other additional drones to operate in the 192.168.0.0/16 IP space; this drone was placed in this range to avoid IP address conflicts with home routers, which are typically in the 192.168.0.0/22 range.

```
default-lease-time 600;
max-lease-time 7200;

ddns-update-style none;

subnet 192.168.10.0 netmask 255.255.255.0 {
  authoritative;
  range 192.168.10.100 192.168.10.254;
  option domain-name "local-network";
  option domain-name-servers 8.8.8.8, 8.8.4.4;
  option routers 192.168.10.1;
  option broadcast-address 192.168.10.255;
  default-lease-time 3600;
  max-lease-time 7200;
}
```

Listing 6.4: Key settings from `/etc/dhcp/dhcpd.conf` file on the Pi

Listing 6.5 covers the configuration of the network interfaces, where the IP address of the ethernet port is assigned by the router it is connected to, whereas the WiFi antenna is set static at 192.168.10.1. The packets were also set to forward between the ethernet and the WiFi, meaning that when the Pi is connected to the internet via the ethernet port, when connected to the Pi's AP you can connected to the internet.

```
auto lo

iface lo inet loopback
allow-hotplug eth0
iface eth0 inet dhcp

```

```
allow-hotplug wlan0
iface wlan0 inet static
  address 192.168.10.1
  netmask 255.255.255.0

# restore iptables
up iptables-restore < /etc/iptables.ipv4.nat
```

Listing 6.5: `/etc/network/interfaces` file on the Pi

Hostapd and ISC DHCP were both run at start-up by calling `start.sh`, listing 6.6, with a `cron` job.

```
sudo hostapd /etc/hostapd/hostapd.conf -B
sudo service isc-dhcp-server restart
```

Listing 6.6: `drone/start.sh` file on the Pi

The Pi was also configured so that it could be connected to via Secure Shell (SSH), which meant code could be transferred and executed remotely, including when it is offline and in flight. To map the Pi's IP address to a domain name, the `/etc/hosts` file on Unix-based systems must be edited to match the example given in listing 6.7.

```
# This computer connected directly to the Pi
192.168.10.1  pidrone
# This computer and the Pi are connected to a network and the Pi
    has been assigned an IP address of 192.168.2.35 on that network
192.168.2.35  pidrone
```

Listing 6.7: `/etc/hosts` file on the development computer

6.3 Program

This section covers the main implementation of the project on the Pi.

6.3.1 Development Environment

Developing for embedded systems inherently means transferring the code to the device and the need for versioning. The whole project was maintained using a Git repository, to aid coding between various devices and backing up the project to separate storage locations. The Pi microSD was also routinely backed up, in case of corruption or loss.

Microsoft Visual Studio Code was chosen as the source code editor, as it has a large feature set, including Git control, syntax highlighting and an integrated

terminal, figure 6.13. The terminal was used to control the Pi over SSH. To make the state and movement of files clearer, Transmit was used as an SFTP client 6.13.

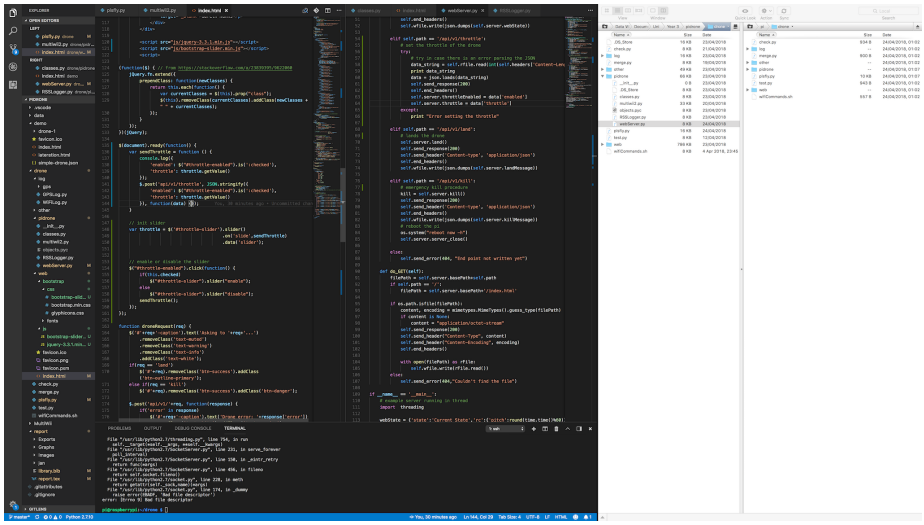


Figure 6.13: Development Environment: *(left)* Microsoft Visual Studio Code, *(right)* Transmit

An interesting element was powering the drone with the LiPo batteries, which limited the connection time to the drone, plus how often one could connect to it which lengthened the development period. When the motors and FC were not required, the Pi could be powered via USB.

6.3.2 Interacting with Flight Controller

There were some difficulties interacting with the Flight Controller (FC) over MultiWii Serial Protocol (MSP), such as:

- Unable to set the virtual Remote Control (RC) stick values
- Sensor values outside of possible bounds, *e.g. a heading of -400, or a longitude of 500*
- Not recognising the GPS unit

This of course delayed the development, as all of these are essential for the program to run correctly. Whilst there is some documentation for MultiWii Serial Protocol (MSP) (MultiWii 2014*b*), it is often incomplete (MultiWii 2014*a*), so there was a lot of trial and error involved.

There are various libraries available for interacting with a MultiWii Controller (Vargas 2017, Dean 2015), alas these too are not fully functional and some do not fully implement all of the features of the protocol, *e.g. accessing GPS coordinates*. The library provided by Dean (2015) was used for this project with some small modifications.

6.3.3 Web Server

In order to view the current status of the drone more clearly, a web dashboard interface was built, figure 6.14; this is as opposed to looking at the script's output in an SSH terminal window (as a direct connection is impractical during flight), which would be difficult to access on mobile devices, hard to read and would take time to enter key commands - like landing the drone. This is similiar to the debug window in the prototype which conveniently outputs key attributes, figure 6.1.

The web server is run in a thread of the main drone program, all served by a Python `HTTPServer`. Having not written a web server before there was a slight learning curve, however a functioning server was produced: an example of how to initiate it is given in listing 6.8 (assumes the corresponding modules have been imported).

The server services all GET requests by looking for the requested file in the folder specified at its initialisation, stored in `WebServer.basePath` - in line 1 of listing 6.8, `webDir` is assigned the path of the `/web` directory in the folder above and passed as an argument to the constructor of `WebServer` on line 2. Also passed to the constructor on line 2 is the request handler class, `WebHttpServer`, and the port it will operate on: `8000`; to connect to the drone's dashboard, the operator goes to `http://192.168.10.1:8000`.

```
1  webDir = os.path.dirname(os.path.abspath(__file__)) + '/../web'
2  server = WebServer(webDir, ('', 8000), WebHttpServer)
3
4  server.webState.state = 'Current State'
5  server.webState.rc['pitch'] = 1234
6  server.webState.devices['88:53:95:57:b2:64'] = webDeviceLogItem(
7    -55,
8    '88:53:95:57:b2:64',
9    '192.168.10.100'
10 )
11 server.webState.devices['a3:4d:21:f5:22:de'] = webDeviceLogItem(
12   # device that won't get updates
13   -77,
14   'a3:4d:21:f5:22:de',
```

```
15    '192.168.10.102'
16  )
17
18  serverThread = threading.Thread(target = server.serve_forever)
19  serverThread.daemon = True
20  serverThread.start()
21
22  try:
23    while True:
24      # simulate values from drone in the air
25
26      time.sleep(1)
27
28      server.webState.state = 'Testing Server'
29      server.webState.rc['throttle'] = 1500 + 5*round(time.time()%60)
30      server.webState.heading = 200
31
32      server.webState.devices['88:53:95:57:b2:64'].signal = -50 -
            random.randint(0,10)
33      server.webState.devices['88:53:95:57:b2:64'].time = time.time()
34
35      if random.randint(0,10) == 10:
36        server.webState.devices['88:53:95:57:b2:64'].position =
              pointTime(
37          random.uniform(52,53),
38          random.uniform(0,1)
39        )
40
41  except KeyboardInterrupt:
42    pass
43  server.server_close()
```

Listing 6.8: Example of running the web server from `drone/webServer.py`

Figure 6.14 shows the dashboard that was created with Bootstrap styling - an open source toolkit for quickly building responsive web apps (Otto & Thornton 2017). It is very usable on mobile devices, figure 6.15, which is essential for the nature of the applications of this project, where there will not always be access to a large laptop or desktop computer.

The dashboard shows all of the key live settings from the drone, including the current state, true virtual stick values from the FC, heading, altitude and location. Furthermore, all of the devices connected over WiFi are listed at the bottom, along with their key statistics and their estimated position *(if the drone has searched and trilaterated the RSS values for that device).*

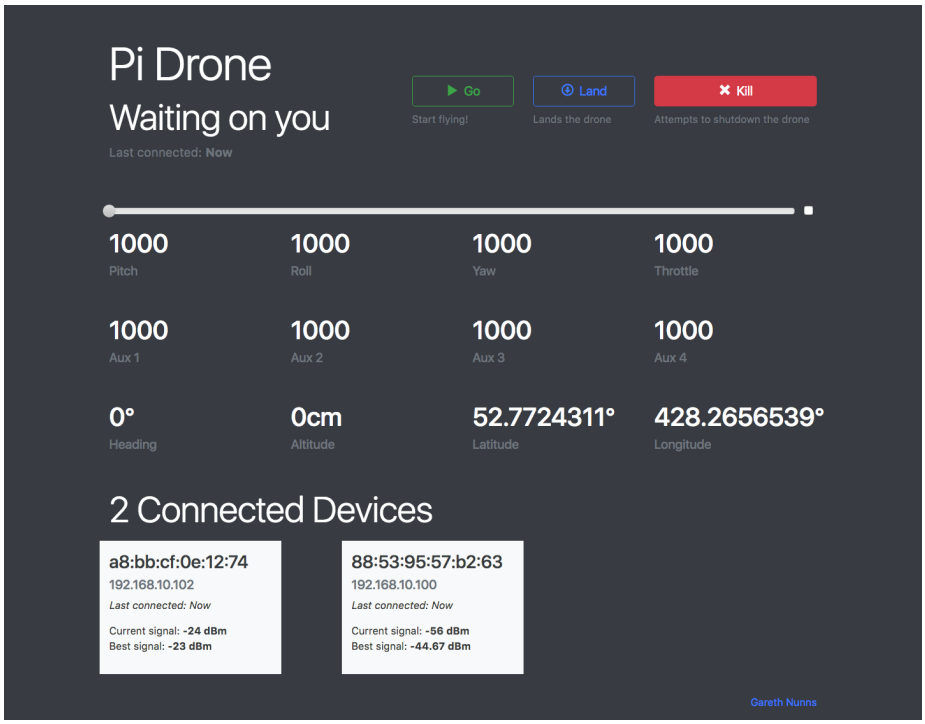
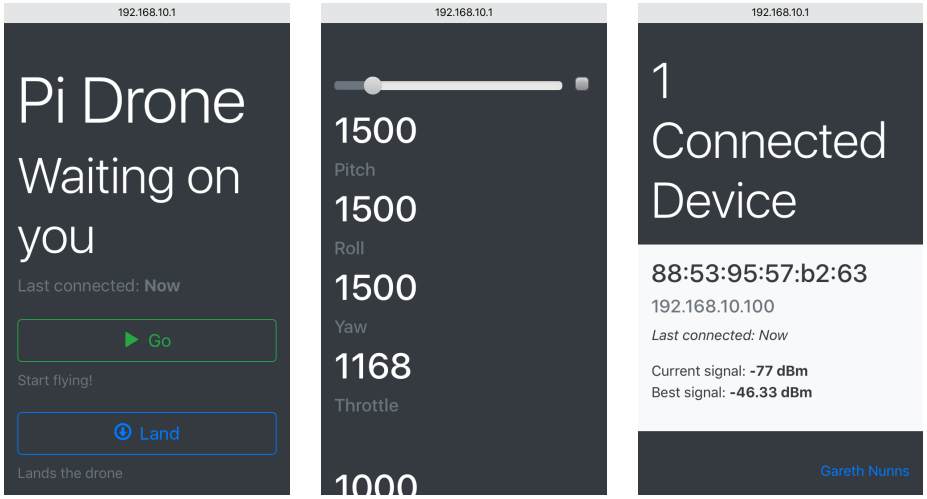


Figure 6.14: Drone dashboard



(a) (b) (c)

Figure 6.15: Drone dashboard viewed on a mobile

This works by the dashboard making POST requests to:

`http://192.168.10.1:8000/api/v1/drone`

This is handled by the `WebHttpServer`, which returns `webStatus.dict_copy()` as a JSON object. `webStatus`, listing 6.9, provides a standardised way of sending the data to the user, containing all of the fields seen on the dashboard, figure 6.14.

The Pi's internal clock may be set to a different timezone *(e.g. UTC vs. daylight saving time)*, or the Pi has been disconnected from power & internet, so its internal clock has not updated - this would cause an issue for calculating how long ago a time value sent by the Pi was; to avoid this problem, the drone's current time, in seconds since the Unix Epoch *(1^{st} January 1970)*, is included in each response, updated on line 26 of listing 6.9.

The purpose of using the `webStatus.dict_copy()` function, line 24, over the built-in `webStatus.__dict__` function is not only to update `self.time`, but also to avoid accidental editing of the object and to convert the object references to static data. For example, if one were to execute:

```
state = webStatus()
```

And then request it as a `dict` *(useful for converting to JSON)* and change a value:

```
stateDict = state.__dict__
stateDict['heading'] = 30
```

Then this will also in turn update `state.heading`; hence, a new instantiation of the object is created in line 29. As `webStatus.gps` is an object itself, `stateDict['gps']` would simply be a reference to the object and not contain the data itself. Lines 30 - 34 convert the objects in `webStatus` to static data, using similar `dict_copy()` functions in those classes.

```
1  class webStatus(object):
2    # returned by the web server
3    def __init__(self):
4      self.state = 'Initialising'
5      self.rc = {
6        'pitch': 0,
7        'roll': 0,
8        'yaw': 0,
9        'throttle': 0,
10       'aux1': 0,
11       'aux2': 0,
12       'aux3': 0,
13       'aux4': 0,
14     }
15     self.heading = 0
16     self.altitude = 0
17     self.gps = point(0,0)
18     self.started = False
```

```
19      # dict of form: {MAC:webDeviceLogItem()}
20      self.devices = {}
21      # if the clock is set incorrectly, relative times can be
            calculated from this
22      self.time = time.time()
23
24    def dict_copy(self):
25      # update the time
26      self.time = time.time()
27
28      # returns a dict as a new instantiation
29      dic = dict(self.__dict__)
30      dic['gps'] = self.gps.dict_copy()
31      # convert dict of objects to dict of dicts
32      dic['devices'] = {}
33      for MAC in self.devices:
34        dic['devices'][MAC] = self.devices[MAC].dict_copy()
35      return dic
```

Listing 6.9: `webStatus` from `drone/classes.py`

The `webDeviceLogItem` class, listing 6.10 includes all of the attributes which are sent to the dashboard about the devices connected to the Pi's WiFi: the device's MAC & IP address, its current RSS from the drone, the average of the three highest RSS values received by the Pi from the device, its estimated position and last time the device connected.

The estimated location defaults to 0 latitude, 0 longitude, estimated at a time of 0 (the start of the Unix Epoch) on line 9. Hence, a simple test to see if the position has been estimated by the drone is to check if `webDeviceLogItem.time > 0`. This keeps the data types consistent and removes additional fields, e.g. `webDeviceLogItem.positionEstimate = True`

Instantiations of `webDeviceLogItem` are maintained by the `RSSLogger` class, covered in the next section.

```
1  class webDeviceLogItem(object):
2    # the information sent by the web server about the devices
         connected
3    def __init__(self,signal,MAC,IP):
4      self.MAC = MAC
5      self.IP = IP
6      self.signal = signal
7      self.best = signal # average of the best signals
8      # most recent estimate of their location as pointTime()
9      self.position = pointTime(0,0,0)
10     # most recent connection time
11     self.time = time.time()
```

```
12
13  def dict_copy(self):
14    # returns a dict as a new instantiation
15    dic = dict(self.__dict__)
16    # convert position object to dict
17    dic['position'] = self.position.dict_copy()
18    return dic
```

Listing 6.10: `webDeviceLogItem` from `drone/classes.py`

Lines 23 - 39 of listing 6.8 update `server.webState`, which is an instantiation of the `webStatus` class, emulating the drone's changing virtual stick and RSS values during flight, having initialised the devices first in the lines above them.

On the dashboard, figure 6.14, there are also options to send requests to these endpoints:

- `http://192.168.10.1:8000/api/v1/go`

  Go button - the drone takes off: this ensures there is an operator in control of the drone before it flies. *Return nothing.*

- `http://192.168.10.1:8000/api/v1/land`

  Land button - changes the drone into the landing state, which should then land the drone. *Returns a message or an error in JSON format.*

- `http://192.168.10.1:8000/api/v1/kill`

  Kill button - stops the main drone program running, disconnecting from the FC and also restarts the Pi. *Returns a message or an error in JSON format.*

- `http://192.168.10.1:8000/api/v1/throttle`

  Throttle slider - this is the large slider in the centre of the dashboard with the checkbox next to it, figures 6.14 & 6.15b. This allows the operator to manually override the throttle; this is especially useful to ensure the drone hovers in a stable fashion. *Accepts JSON requests in the form of listing 6.11 and returns nothing.*

```
1 {
2     'enabled': true,
3     'throttle': 1600
4 }
```

Listing 6.11: Example throttle request to `/api/v1/throttle`

This is currently an unsecured page open to all those connected to the drone, which is ideal for this proof of concept. However, for production it would need to be secured, either via a password, or there could be a set of whitelisted MAC addresses which have access.

6.3.4 Logging RSS

In order to trilaterate the connected devices, the RSS values of the devices must be logged. As with the web server, the RSS logs are stored in objects and the constructors for these objects bear very similar likenesses to those used in the prototype; comparing listing 6.2 to 6.12, the same data is stored in the same structure. A slight difference is that `RSSLogItem.time` defaults to the current time if no argument is passed, line 13.

```
1  class RSS(object):
2    # an RSS measurement (stored in RSSLogItem.RSSs)
3    def __init__(self,signal,MAC,IP):
4      self.MAC = MAC
5      self.IP = IP
6      self.signal = signal
7
8  class RSSLogItem(object):
9    # a single RSS log entry
10   def __init__(self,RSSs,position,heading,second=None):
11     self.position = point3D(position.lat,position.long,position.alt
           )
12     self.RSSs = RSSs
13     self.time = second if second != None else time.time()
14     self.heading = heading
```

Listing 6.12: Constructors for `RSS` & `RSSLogItem` from `drone/pidrone/classes.py`

The logging performed by the `RSSLogger` class is executed in a separate thread from the main program thread: this means that neither slow each other down and can run independently, whilst sharing objects. Within `RSSLogger.start()`, there is a loop that runs once a second, by default, and each time at the start of the loop a new thread is started which executes `RSSLogger.log()`. This means the log times are more consistent, as opposed to each `RSSLogger.log()` call running asynchronously, which may not be a constant time due to varying numbers of devices.

The crux of the `RSSLogger.log()` is shown in listing 6.13, which first runs:

```
$ iw dev wlan0 station dump
```

This provides a list of all of the devices connected to the interface `wlan0`, from which it picks out the MAC addresses of those using `grep` and stores them in a `list`. With these, it then gets the corresponding, assigned IP address using `ip neigh`, stored similarly. This is required to `ping` the device, which updates the RSS value, procured with:

```
$ iw dev wlan0 station get aa:bb:cc:dd:ee:ff
```

`RSSLogger.log()` iterates over the `MACs`, running the line above for each and storing them in a `list`, lines 27 - 33.

```
1  # get mac addresses of connected devices
2  procMAC = subprocess.Popen("iw dev wlan0 station dump | grep '
       Station' | awk '{print $2}'",
3    shell=True, stdout=subprocess.PIPE, stderr=subprocess.STDOUT)
4
5  MACs = []
6
7  for line in procMAC.stdout.readlines():
8    MACs.append(line.strip())
9
10 sorted(MACs)
11
12 # get IPs from MACs
13 ipCommand = "&&".join(["ip neigh | grep '"+MAC+"' | awk '{print $1}'
       " for MAC in MACs])
14
15 procIP = subprocess.Popen(ipCommand,shell=True, stdout=subprocess.
       PIPE, stderr=subprocess.STDOUT)
16
17 IPs = []
18
19 for line in procIP.stdout.readlines():
20   IPs.append(line.strip())
21
22 # ping connected devices (updates RSS as no inactive time)
23 pingCommand = "&&".join(["ping -c 1 "+IP for IP in IPs])
24 subprocess.Popen(pingCommand,shell=True, stdout=subprocess.PIPE,
       stderr=subprocess.STDOUT)
25
26 # get signal levels
27 signals = []
28 sigCommand = "&&".join(["iw dev wlan0 station get '"+MAC+"' | grep '
       signal' | awk '{print $2}'" for MAC in MACs])
29
30 procSig = subprocess.Popen(sigCommand, shell=True, stdout=
       subprocess.PIPE, stderr=subprocess.STDOUT)
31
```

```
32 for line in procSig.stdout.readlines():
33   signals.append(line.strip())
```

Listing 6.13: Excerpt from `RSSLogger.log()` function in `drone/pidrone/RSSLogger.py`

Using the data gathered above, the `RSSLogger.RSSLog` is updated, adding another `RSSLogItem` object *(listing 6.12)* to the `list`. The log of all devices that have connected to the drone is then stored in `RSSLogger.devices` as a `dict` of `deviceLogItem` objects with the device's MAC address as the key, as well as a simplified version to send over the web, similarly stored in `RSSLogger.webDevices` as `webDeviceLogItem` objects *(listing 6.10)*.

6.3.5 Main Program

The main program is largely an implementation of algorithm 2 and a conversion from the JS prototype into Python. A key section of this is the `squareSearch.trilateration()` function, detailed in listing 6.14 and similar to algorithm 3, designed in chapter 5.

The function accepts a set of `points`, typically computed using `squareSearch.calculateDistances()` which converts the RSS values into distances. These `points` pertain to multiple devices which are trilaterated with the function; however, at least 3 points are required for trilateration, line 13.

The location for the device is then calculated using `numpy` (`np`) matrix functions, line 43, to produce the `location`. This is based on equations 2.1 and 2.3, restated:

$$\begin{bmatrix} r_1^2 - r_2^2 + x_2^2 + y_2^2 - x_1^2 - y_1^2 \\ r_1^2 - r_3^2 + x_3^2 + y_3^2 - x_1^2 - y_1^2 \\ \dots \\ r_1^2 - r_N^2 + x_N^2 + y_N^2 - x_1^2 - y_1^2 \end{bmatrix} = \begin{bmatrix} 2 \cdot (x_2 - x_1) & 2 \cdot (y_2 - y_1) \\ 2 \cdot (x_3 - x_1) & 2 \cdot (y_3 - y_1) \\ \dots & \dots \\ 2 \cdot (x_N - x_1) & 2 \cdot (y_N - y_1) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{2.1 restated}$$

$$D = C \begin{bmatrix} x \\ y \end{bmatrix} \Leftrightarrow \begin{bmatrix} x \\ y \end{bmatrix} = (C^T \cdot C)^{-1} C^T D \tag{2.3 restated}$$

```
def trilateration(self,points):
  """Trilaterates the devices connected during the sqaure
  Returns their positions in the form {MAC: pointTime()}
  points is a dict in the form {MAC: [signalPoint()]} containing the distance"""

  # initialise stores
  C = {}
  D = {}
  locations = {}

  for dev in points:
    # loop through all the devices that connected to the drone during the square
    if len(points[dev]) < 4:
      # don't consider a device if there are not enough points to trilaterate
      break

    # initialise the matrices for this device
    C[dev] = []
    D[dev] = []

    i = 1
    while i < len(points[dev]):
      # computing the rows of the matrices
      C[dev].append([
        2 * ( points[dev][i].position.lat - points[dev][0].position.lat ),
        2 * ( points[dev][i].position.long - points[dev][0].position.long )
      ])

      D[dev].append([
        np.square(points[dev][0].signal) -
        np.square(points[dev][i].signal) +
        np.square(points[dev][i].position.lat) +
        np.square(points[dev][i].position.long) -
        np.square(points[dev][0].position.lat) -
        np.square(points[dev][0].position.long)
      ])
      i += 1

    # trilaterate the points for this device using this matrix calculation
    # (C^T * C)^-1 * C^T * D
    CM = np.matrix(C[dev])
    DM = np.matrix(D[dev])
```

```
43      location = (CM.T * CM).I * CM.T * DM
44
45      locations[dev] = pointTime( # add to the list of locations to
            return
46        location[0,0],
47        location[1,0]
48      )
49
50    return locations
```

Listing 6.14: `squareSearch.trilateration()` function in `drone/pidrone/classes.py`

Listing 6.15 gives an example use of `squareSearch.trilateration()`, with the a set of points and distances for device `'a'`, represented in figure 6.16 (where the bottom left hand corner is (0,0), each minor line represents a unit of 1 and the blue circles represent the estimated distance away from the drone at each point).

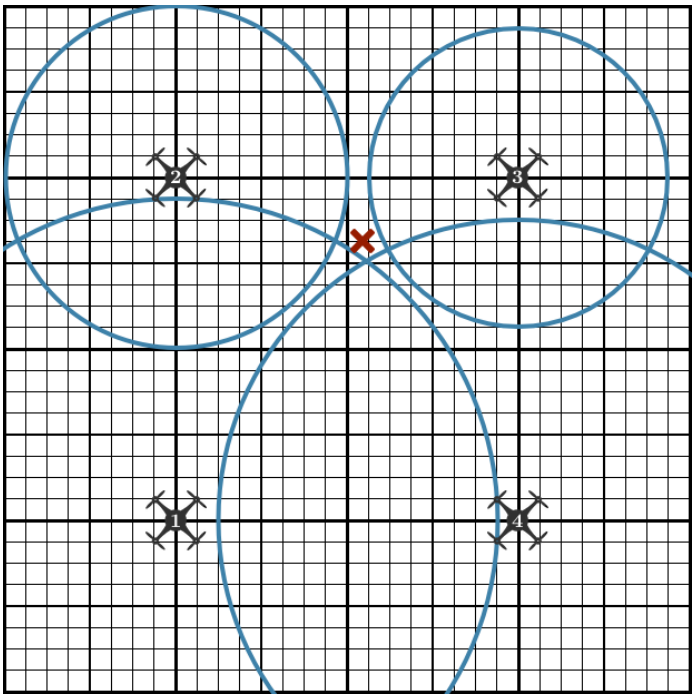


Figure 6.16: Trilateration example, location computed by listing 6.15

```
1  points = {'a': [
2    signalPoint(15, point3D(8, 8, 0), 0), # 1
3    signalPoint(8, point3D(16, 8, 0), 0), # 2
4    signalPoint(7, point3D(24, 24, 0), 0), # 3
5    signalPoint(14, point3D(8, 24, 0), 0), # 4
```

```
]}
s = squareSearch()
aLocation = s.trilateration(points)['a']
```

Listing 6.15: Example use of `squareSearch.trilateration()` function

The computed `aLocation` is shown on figure 6.16 as a red cross, with coordinates:

```
aLocation.lat = 21.08
aLocation.long = 16.66
```

6.4 Summary

The virtual prototype really cements algorithms 2 & 3, established in chapter 5; it also helps clearly explain the intentions of the drone and how it should react to various situations, making the project more accessible to others, *objective 2*.

The build of the drone gives a platform for others to work from and satisfies key requirement M.1. The construction and was less trivial than expected in chapter 2 and delayed the project slightly.

The main program on the drone itself has many components to it, including the RSS logging, *meeting objective 6*, hosting a web server to display key information and control the drone, *objective 2*, *requirements M.4 - M.6* and controlling flight in various states, *objective 1*.

It was not possible to implement all of algorithm 2 due to the invalid sensor data, *chapter 5*, however the drone does attempt to move in a square and go to an optimal location for all users that are connected, the average location of the trilaterated RSS values.

Chapter 7

Testing

7.1 Test-Driven Development

The use of Test-Driven Development (TDD) was especially useful for verifying the class constructors and methods in `drone/classes.py`, as these tests could easily be written before development, largely based on the prototype. They were not only testing simple data structures, but also mathematical checks on the more complex functions in the `squareSearch` class with known values. An example unit test is shown in listing 7.1, which checks the `squareSearch().trilateration` function calculates the correct value with a single device and four readings at different locations with corresponding distances to the device, *similar to listing 6.15.*

```
1  def test_squareSearch_trilateration_single_device(self):
2    s = squareSearch()
3
4    points = {'a': [
5      signalPoint(15,point3D(8,8,0),0),
6      signalPoint(8,point3D(16,8,0),0),
7      signalPoint(7,point3D(24,24,0),0),
8      signalPoint(14,point3D(8,24,0),0),
9    ]}
10
11   aLocation = s.trilateration(points)['a']
12
13   self.assertAlmostEqual(aLocation.lat,21.08,2)
14   self.assertAlmostEqual(aLocation.long,16.66,2)
```

Listing 7.1: Example unit test

The tests were written in a test suite native to Python, `unittest`, and were configured in Microsoft Visual Studio Code, figure 6.13, to run automatically upon saving the file. This provided a way to perform regression testing as code was written and instantly verify correctness of newly written code. Listing 7.2 gives an example output from when `drone/classes.py` was saved.

```
..............................................................
----------------------------------------------------------------------
Ran 66 tests in 0.004s

OK
```

Listing 7.2: Output from running the classes test suite

7.2 Flight Tests

Figure 7.1 shows the set up used for the flight tests of the drone. Following the initial flight test (Nunns 2017), figure 7.1a, which was performed in a slightly wooded area, subsequent tests were performed at locations such as on one of the university's many sports pitches, which were large and clear of people & obstacles 7.1c.

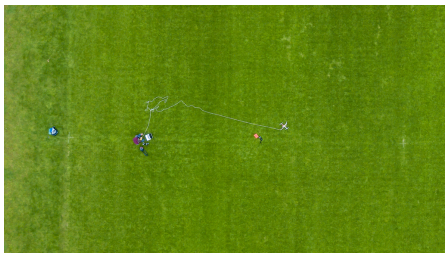
In order to prevent the drone flying too far from flight control, a large length of string was attached to the drone 7.1g. The drone was then controlled from a laptop, figure 7.1f, via the web dashboard, figure 7.1e.

During all testing cameras were recording, either filmed by kind individuals (thanks to Fraser Stockley, seen in figure 7.1c), left on a tripod, filmed from another drone, figure 7.1c, or shot using a chest-mounted wide angle camera, figures 7.1a & 7.1e. This provided a log of events and helped track progress.

Inevitably the effect of adding the string slightly affected flight, however it was a necessary precaution to ensure safety and prevented possible loss of the drone. The flight tests were useful for configuring parameters of flight and testing algorithms. The difficulty was finding a weather window long enough to test the drone.

The testing showed the drone was capable of flight, figures 7.1g & 7.1h, and was under control of the main program running on the Pi, with remote supervision from the web dashboard figure 7.1e.

(a)

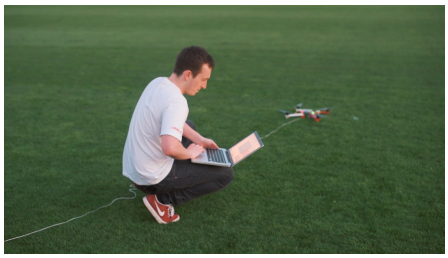


(b)

(c)

(d)

(e)



(f)

(g)

(h)

Figure 7.1: *Flight tests:*
Figure 7.1a: First flight test, Figure 7.1b: Test Setup *from above*,
Figure 7.1c: Test Setup, Figure 7.1d: Connecting the battery,
Figure 7.1e: Drone's web dashboard, Figure 7.1f: Configuring the drone,
Figure 7.1g: Drone taking off, Figure 7.1h: Drone in flight

From the aforementioned dashboard, the trilaterated positions of the users were displayed and the optimal state was entered. The drone performed an ap-

proximate square whilst logging the RSS values once per second, from which it determined an estimate for the position of the users and moved itself appropriately. Unfortunately, the effects of the strong winds prevalent during the testing phase, and common to Loughborough, meant the precision of the movement was minimal: an area for improvement in future work.

7.3 Summary

The TDD tests ensured the implementation behaved as expected throughout development, meeting objective 2. The flight tests were used to assess the success in meeting objectives and test that the code ran as expected from the implementation, chapter 6. Further refinements of the hardware and software, including the suggestions made in chapter 8, would make the drone's movement more refined and could provide better compensation for the wind.

Chapter 8

Recommendations

This project was the basis for a proof of concept and there are still plenty of possibilities left unexplored.

8.1 Search Method

The current algorithm used could be bettered by the use of least squares lateration, which could improve the accuracy of the lateration fourfold (Dag & Arsan 2017). For simplicity's sake, this was not implemented on this project.

With a large collected data set from known data, a machine learning method could be trained to produce more reliable results, drawing from the large number of data points collected by the drone. This may well greatly improve the accuracy of the trilateration and the speed of the search algorithm.

It may be faster to choose a random direction to start searching in, as opposed to the rotation method - in a similar vein to a random mutation in a genetic algorithm. This may improve the speed at which the user's location was determined, although would require testing to see if it were statistically beneficial. There are various hill climbing algorithms that could be employed to maximise the RSS from the connected devices.

8.2 Drone Fleet

With a fleet of drones, a door of opportunities is opened:

- Classic trilateration is possible with the different drones acting as the base

stations, figure 2.3.

- If a drone needs to land for whatever reason, another drone can dynamically fills its place and service will be maintained.
- The drones can act as a swarm and cover a greater area of users.
- Mesh WiFi algorithms can be used to ensure the best signal.
- A shared database of RSS values from the devices connected to the different drones and their predicted locations would improve trilateration and mobility prediction.

8.3 Drone Modifications

To add internet connectivity to the drone whilst in flight, a 4G connection could be routed through Pi's WiFi AP. Or, if it was operating as part of a fleet it could get its internet connection via another drone, either using WiFi or via a separate technology, like microwaves.

An interesting addition to the drone would be the use of a 360 camera on underside. From this, users could be identified by using image-processing algorithms to recognise people in the image: this would give an approximate distance and angle to each user. Similar objectives could be achieved by providing the best signal to the users which it can see, because for many applications line of sight can be assumed.

The current WiFi transceiver on the Pi is only designed for short-range indoor connections and this project stretches its capabilities. The range of the drone could be enlarged by fitting it with a more powerful WiFi antenna, which may also add more fidelity to the RSS values, leading to more accurate trilateration. Moreover, a transmitter which gave an approximate direction of the signal (often achieved by multiple directional antennas in a single housing) could be used for triangulation, which would reduce search times and most likely enhance user location estimates.

Putting lights on the drone would make it easier to see in the sky and could indicate the state of the drone, *e.g. flashing red: low battery.*

In general the sensors on the drone could be improved to give more precise values and a Ultrasonic/Infrared sensor could be added to avoid obstacles.

Any additions made to the drone would impact on the weight that the motors have to lift, which in turn would be to the detriment of flight time, if the same

capacity battery were used.

8.4 Web Dashboard

The web dashboard could be greatly expanded, including, but not limited to:

- **Operating over WebSockets**
  WebSockets allow full duplex communication between the server and client, where one can send messages to a server and receive event-driven responses, without the need for the traditional polling to the server (MDN 2018). This would mean the `webStatus`, listing 6.9, could be constantly streamed over the WebSocket; likewise, the throttle commands could be streamed as well.

  This would reduce the number of HTTP requests made to the drone and improve the stability of the web app.

- **Map of Devices**
  A map on the web app of the connected devices may well make it more clear what the drone's intentions are. Either an online mapping service, like Google Maps, could be used, or to reduce web traffic, a simple JS canvas method, like the virtual prototype, could be used to represent the drone and the predicted locations of the users.

- **Single-Page Application**
  The dashboard could be converted into a much more powerful and robust single-page web application, using a framework such as AngularJS, Vue.js or React. These provide strong footholds into building a complex, scalable web and mobile apps. This would make the code base simpler and more modular.

- **Device Names**
  A little research was made to look into retrieving the local hostnames of devices, not only to make the web dashboard clearer *(as opposed to simply showing the device's MAC address, figure 6.15c)*, but also could be used as a heuristic for calculating the transmitter power of the device. For example, if the connected device's hostname was "Bob'siPhone", then that would likely have a weaker power than "Bob'sMacBook". This would of course not be entirely accurate, but a simple decision tree may aid the estimation of the transmission power, in turn aiding the RSS trilateration.

- **API**
  A standards-based API (Application Programming Interface) for the drone would mean different apps could connect to the drone to read and write data. This could include native apps on phones, requests from other drones or operators. Initial moves were made towards this in this project, chapter 6.

- **Secuirty**
  If the drone were to be used in a public setting then there should be a level of protect added to the web dashboard, to avoid unintended access to the controls of the drone. When connected to the internet, there should also be strong firewalls in place to prevent others tampering with the drone.

  Additionally, the hotspot itself could have increased security, with a login screen before access to the internet was granted, including terms and conditions.

8.5 Flight Recorder

Before the drone went into production, it would be important to fit it with a flight recorder, also known as a black box. If there were to be an accident, then this would aid the investigation of what was the cause. Currently, there is a simple log kept by running the script with the following command on boot, which stores the output of the script to a dated file:

```
$ sudo python -u /home/pi/drone/drone.py &>> /home/pi/drone/log/
    start/start-"$(date +%Y%m%d-%H%M%S)".txt &
```

However, a more rigorous flight recorder could be written to run in another thread, with thorough logging of key variables to a file. Furthermore, there could be a separate device on the drone, independent of the Pi, dedicated to monitoring the flight, as the drone may have failed due to an error on the Pi.

8.6 Summary

This report lays the groundwork for many exciting directions in which the work could be taken, including various hardware and software improvements. These would largely help the localisation of users, benefiting objective 7. This field remains very open and is rife for developments which can add to the scope of this project.

Chapter 9

Discussion

This section considers the constraints on the project and evaluates the methods undertaken, including what has been learnt from the project.

9.1 Constraints Evaluation

One of the most difficult parts of this project was the physical, outdoor nature of it. An ever-uncontrollable factor was the weather, which was untypically cold, with snow multiple times during the project (Godsall 2018, Nunns 2017). Rough conditions often stunted progress on the project, furthermore, testing could only occur during daylight hours for safety's sake, very limiting during the winter period.

Finding suitable test sites was also important, not only to operate within the law (Ministry of Defence & Military Aviation Authority 2017), but to make sure there were no bystanders who could be injured as a result of the drone's experimental and erratic movements. This often involved travelling out to fields or university sports pitches on foot with all of the equipment *(development drone, camera drone, batteries, laptop, camera, etc.)*. There was only one slight mishap with a tree close to the launch site of the first test flight where a propeller was damaged (Nunns 2018), thus, all subsequent tests were performed with greater safety distances.

These constraints were not initially recognised to the full extent and greater planning could have been put in to account for these issues.

A lot of the components used in this project were specialist parts shipped from China, which delayed the build and pushed other factors back. These

could have been ordered sooner to allow more development time.

9.2 Technical Evaluation

I believe Python suited this project well, however it is difficult to say if the use of the library provided by Dean (2015) expedited the progress project due to difficulties interacting with the Flight Controller (FC). The ability to flexibly add threads and host a web server definitely made it a suitable language for the project and I would use it again.

The drone components used in this project varied in how well they were fit for purpose:

- **Frame**
  It was reasonably capable of housing the various components, but also wasn't overly bulky: I would most likely choose the same frame again.

- **Motors & ESCs**
  The motors are easily powerful enough to lift the drone and a little more - *the throttle is at approximately* $\frac{2}{3}$ *to hover*; these motors were a good choice, especially as they seem to be fairly efficient too. Similarly in relation to the model of Electronic Speed Controller (ESC), I had no reservations.

- **Flight Controller (FC)**
  The MultiWii Serial Protocol (MSP) is one of the more open-source methods for interacting with FCs, however other firmware options are available and possibly greater time should have been spent researching these. Reading the sensor values from the FC was more difficult than anticipated and potentially a better solution is to attach sensors directly to the Pi and control the ESCs directly from the Pi. This may be easier said than done, especially if the Pi can not produce a fast enough Pulse Width Modulated (PWM) signal to the ESCs, as well as implementing the many hidden sub-processes that the FC performs.

9.3 Personal Development

There was a slight learning curve to writing Python, as this was the first major project I had written in the language. As a result of this project, I have become

aware of its technicalities, foibles and best practices, in order to produce efficient and understandable code.

Threads were key to this project and I had not used them before, either academically or personally. The complexities of sharing information and calling functions outside of classes in separate threads produced some interesting challenges, but led to learning and creative solutions. Another new technology to me was building a web server to handle requests, which furthered my understanding of the client-server model and HTTP methods. Whilst configuring the Pi and accessing it from SSH, I increased confidence in Bash (Unix shell) scripting.

Whilst having a passion for drones already (Nunns 2017), I had never before built a drone, nor was I aware of a lot of the associated terminology, processes and techniques. Constructing the drone involved more soldering than anticipated, a skill which I have bettered greatly, with thanks to my father for his tuition. However, this could have potentially been avoided by exploring commercial-off-the-shelf solutions further, but it is difficult to say whether more challenges may have been encountered interacting with that hardware.

With regard to the report, this is the largest document I've ever composed and I'm pleased with the result, which has a wide range of figures, listings and sources. Similarly, prior to starting this project I had not used LaTeX before, however am very glad I did, as it made producing graphs, adding code snippets and maintaining versions much easier. I have since gone on to use LaTeX for other documents due to the professionality and features it provides.

9.4 Summary

The project was ambitious and multifaceted, with constraints that were not initially clear. My naivety in the field of drones and reliance on the advice of others in the literature review, chapter 2, possibly led to some decisions that in retrospect could have been made differently; however, as there is limited documentation on the exact specifications of components, they were difficult decisions to make. Throughout the project I have learnt many skills and methods, which will aid my future academic and professional work.

Chapter 10

Conclusion

The ambitious main aim of this project was to develop an autonomous drone controlled by a Raspberry Pi, which creates its own access point and flies to provide the best coverage to the devices connected. The success of this aim is expanded upon in the following sections, which analyse how well the objectives and requirements were met.

10.1 Objectives Revisited

The objectives defined in the introduction, *chapter 1*, guided the project, prompting literary research & primary research, *chapters 2 & 5 respectively*, as well as forming the basis for decisions throughout the implementation and testing, *chapters 6 & 7 respectively*. This section considers whether the objectives were covered in this report.

10.1.1 Objective 1

"Simple flight controlled by the Pi - pitch, roll, yaw and altitude commands sent from the Pi which result in the correct movement of the drone."

The drone's flight is fully controlled by the Pi, hosting its web dashboard, *figure 6.14*, where the operator can remotely start and stop flight, plus control the drone's throttle; photos of flight tests are shown in chapter 7.

10.1.2 Objective 2

"Creating a project that is accessible for others to progress - clear, concise code and a user interface that explains what the drone is doing."

The virtual prototype developed in chapter 6 is a great starting point for those new to the project as it's an interactive demo of the project and shows the methods used by the drone to localise users. Similarly the web dashboard, *figure 6.14*, gives a good overview of the drone's current state.

The code is well commented and largely self documenting, with useful object classes for data and methods. Furthermore, this report acts as a good starting point for future work: notably the recommendations, *chapter 8*.

10.1.3 Objective 3

"Maintaining horizontal position - naturally a drone won't hover in a static position, due to the wind or differences in the motors: commonly this is achieved through the use of GPS and correcting for these problems."

The drone was configured for stability, *chapter 6*, maintaining its position from gyroscopes and GPS. However, there was still drift due to strong winds, as discussed in chapters 7 & 9 - solutions to this problem are covered in chapter 8.

10.1.4 Objective 4

"Maintaining vertical position - similar to the maintaining horizontal position issue, this can be solved by GPS (to a limited extent), radar or barometer."

This was more difficult than anticipated due to the erroneous data from the barometric altimeter, *chapter 5*, hence manual control was implemented on the web dashboard, *figure 6.14*, which also serendipitously increased safety.

10.1.5 Objective 5

"Forward obstacle avoidance - sense an object in front of the drone and take action to avoid a collision. This may be beyond the scope and time frame of this project."

This was not covered by this project as it was deemed to be out of scope in the requirements, *chapter 4*; nonetheless, this is one of the fields for further work on the project, *chapter 8*.

10.1.6 Objective 6

"A functioning wireless access point with the ability to measure connected devices signal strength - allowing the drone to make a decision of where to move to deliver the best signal."

This was fully implemented, with the commands used included in listing 6.13, the RSS of connected devices shown in figure 6.14, the method used for trilateration described in chapter 2, formulated into algorithm 3 and the implemented algorithm shown in listing 6.14.

10.1.7 Objective 7

"Movement based on the proximity to users connected to the access point, involving ascertaining the location of the connected users from trilateration and then providing the best signal."

This was the most difficult objective and was only partially covered in this project, this is due to the aforementioned issues covered in chapters 7 & 9. The position was trilaterated, but the precision of the localisation and movement was minimal and there is still further work to be pursued on this topic, *chapter 8*.

10.2 Requirements Revisited

Table 10.1 reexamines the requirements specified in chapter 4, considering whether they were achieved within this project.

#	**Priority**	**Requirement**	**Coverage**
M.1	Must have	A constructed drone that is capable of flight	This was fully achieved, researched in chapter 2 and its construction & configuration are described in chapter 6; photos of flight tests are found in chapter 7.

#	Priority	Requirement	Coverage
M.2	Must have	Drone is controlled by Pi	This was also fully achieved, with photos of flight tests found in chapter 7.
M.3	Must have	The drone maintains its position in the horizontal plane, with stable pitch, roll and yaw	There was only some success with this requirement - it did not fully compensate for strong winds, but in calm conditions it was relatively stable.
M.4	Must have	The drone maintains its position in the vertical axis, within +/- 1m	As the altimeter was found to be unreliable, *chapter 5*, the throttle was controlled manually which meant it was more precise. This was slightly difficult during testing as the increasing weight of the string the drone was lifting affected the throttle value required, figure 7.1g.
M.5	Must have	A means to force the drone to land	One can simply press the "Land" button on the web dashboard, figure 6.14, which achieves this requirement. This will work if the main program thread is still running, otherwise the "Kill" button could be used.
M.6	Must have	An emergency stop button which stops the rotors spinning	One can simply press the "Kill" button on the web dashboard, figure 6.14, which achieves this requirement. This attempts to stop the program and restart the Pi, disconnecting from the FC and hence stopping the motors.

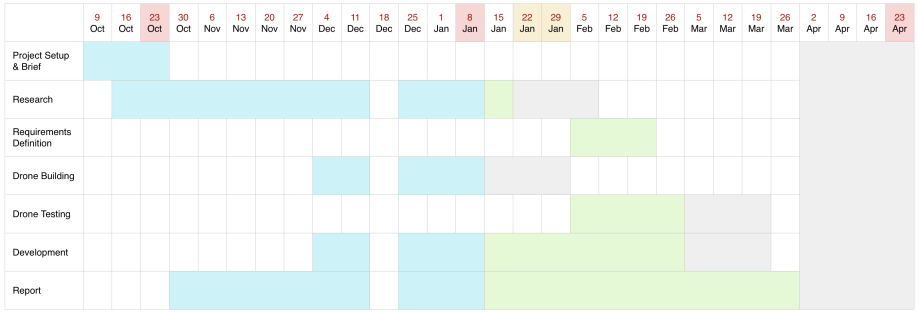
#	Priority	Requirement	Coverage
M.7	Must have	Measuring of the battery level	Fully achieved: the low voltage alarm, labelled as 8 on figure 6.7, flashes red lights when the battery is running low and emits a loud tone when it is depleted to a very low level.
M.8	Must have	Pi acting as an AP	This requirement was fully met, with the configuration described in chapter 6
M.9	Must have	Log the RSS of the devices connected to the Pi's AP	This requirement was also fully met: the method used is laid out in chapter 6 and the RSS of two devices can be seen on the web dashboard, figure 6.14.
M.10	Must have	Trilaterate the users' locations, based off the RSS	Achieved with scope for improvement, *chapter 8*: the mathematics to calculate this is covered in chapter 2, an algorithm is proposed in chapter 5 and an example use of its implementation is shown in chapter 6.
M.11	Must have	Self-documenting code, which is efficient and well-structured	To a large extent this has been covered, especially in the use of Object-Oriented programming, which makes the data types and methods clear; however there could be greater clarity in the main program.

#	Priority	Requirement	Coverage
S.1	Should have	The drone should make movements based on the trilaterated location of multiple connected users	Due to the difficulties described in chapters 7 & 9, its movements were slightly unclear, masked by the wind. However, in the virtual prototype, a clear implementation is presented.
C.1	Could have	Predict the future location of users, based on the prior predictions made	Achieved by the means of a simple weighting method which was used in the virtual prototype, however it was not particularly complex and was not implemented on the drone.
W.1	Won't have	Obstacle avoidance, using US, IR or a similar technology	This was not explored and a suitable sensor was not fitted to the drone. Further developments, including this addition, are described in recommendations, *chapter 8.*

Table 10.1: Key Requirements, revisited

10.3 Project Management

The initial plan shown in figure 3.1 was subsequently updated in January, *figure 10.1*, to reflect the progress of the project and forecast future work, taking better account of final year exams.



Key:

	Deadline Week
	Ongoing
	Completed
	Contingency
	Exams Week

Figure 10.1: Updated project plan *(10/1/18)*

The main delays to the projects were:

- As discussed in chapter 9, the drone components took quite some time to be shipped which slightly pushed back the start of the build, design and implementation, as all of these relied upon the drone.
- The parts themselves took time to configure, *chapter 6*, and additional work was done to attempt to rectify the poor sensor values from the Flight Controller (FC) covered in chapter 5.
- The weather during the development and testing period was very poor (Godsall 2018, Nunns 2017), which reduced available test weather windows.

These factors meant that the assigned contingency period at the end of the project (figure 10.1) was used more than hoped, however the project still met the strict 30$^{\text{th}}$ April 2018 deadline, set out in the constraints in chapter 1. Whilst some of these factors were partially foreseeable, they were largely out of my hands and should be considered for future projects.

10.4 Final Conclusion

I have presented a full report into the literary background of the problem from a wide range of sources and built a functional drone using this research.

From the preceding sections, the objectives and requirements were, for the most part, achieved, and where they weren't fully met leave strong footholds for future additions to the project.

To summarise, the project has largely met its aims of producing a flying, autonomous WiFi access point, which can be developed further in future.

Glossary

Git An open source distributed version control system (Git 2018). 60

MultiWii A general purpose software to control a multirotor RC model (MultiWii 2015*a*). 9, 56, 61, 62, 83, 94

WebGL Web Graphics Library native to compatible browsers. 1, 46

Acronyms

AP access point. 1–3, 9, 12–15, 20, 21, 58, 59, 79, 85, 87, 89, 92

ESC Electronic Speed Controller. 9, 51, 53, 83

FC Flight Controller. 5, 9, 36, 51, 53–56, 61, 63, 67, 83, 88, 91

GPS Global Positioning System. 3, 6, 13, 15, 33, 34, 36, 43, 51, 54–56, 61, 86

IR Infrared. 16, 22, 79, 90

JS JavaScript. 46, 70, 80

LiPo Lithium Polymer. 4, 57, 61

MSP MultiWii Serial Protocol. 61, 83

RC Remote Control. 61, 93

RSS Received Signal Strength. 1, 5, 10, 12–14, 21, 23–26, 30–33, 35, 36, 40–44, 46, 48–50, 63, 66–70, 73, 77–80, 87, 89

SSH Secure Shell. 24, 60, 62, 84

TDD Test-Driven Development. 5, 18, 74, 77

US Ultrasonic. 16, 22, 79, 90

Bibliography

Ahmed, A. (2017), 'Driving an ESC/Brushless-Motor Using Raspberry Pi : 5 Steps'. Date accessed: 2018-01-10.
URL: *http://www.instructables.com/id/Driving-an-ESCBrushless-Motor-Using-Raspberry-Pi/*

Aljadhai, A. & Znati, T. F. (2001), 'Predictive mobility support for QoS provisioning in mobile wireless environments', *IEEE Journal on Selected Areas in Communications* **19**(10), 1915–1930.

B. Benchoff (2016), 'Introducing the Raspberry Pi 3 — Hackaday'. Date accessed: 2018-02-27.
URL: *http://www.raspberry-pi-geek.com/Archive/2016/17/Raspberry-Pi-3-Model-B-in-detail https://hackaday.com/2016/02/28/introducing-the-raspberry-pi-3/*

Balaji, S. (2012), 'Waterfall vs V-Model vs Agile : A comparative study on SDLC', *WATEERFALL Vs V-MODEL Vs AGILE : A COMPARATIVE STUDY ON SDLC* **2**(1), 26–30.

Benkic, K., Malajner, M., Planinsic, P. & Cucej, Z. (2008), Using RSSI value for distance estimation in wireless sensor networks based on ZigBee, *in* 'Proceedings of the 15th International Conference on Systems, Signals and Image Processing', pp. 303–306.
URL: *http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4604427*

Bicheno, S. (2017), 'Exclusive the facts and figures behind EE's Glastonbury temporary network — Telecoms.com'. Date accessed: 2018-01-09.
URL: *http://telecoms.com/482875/exclusive-the-facts-and-figures-behind-ees-glastonbury-temporary-network/*

Brennan, K. (2009), *A Guide to the Business Analysis Body of Knowledge*, 2

edn, International Institute of Business Analysis.
URL: *http://www.amazon.com/Guide-Business-Analysis-Knowledge-BABOK/dp/1927584027*

Cellan-Jones, R. (2011), 'BBC - A 15 pound computer to inspire young programmers'. Date accessed: 2017-11-28.
URL: *http://www.bbc.co.uk/blogs/thereporters/rorycellanjones/2011/05/a_15_computer_to_inspire_young.html*

Cellan-Jones, R. (2016), 'Facebook's drones - made in Britain - BBC News'. Date accessed: 2018-02-25.
URL: *http://www.bbc.co.uk/news/technology-36855168*

Dag, T. & Arsan, T. (2017), 'Received signal strength based least squares lateration algorithm for indoor localization', *Computers and Electrical Engineering* **66**, 114–126.
URL: *https://doi.org/10.1016/j.compeleceng.2017.08.014*

Dean, J. (2015), 'Python MultiWii Serial Protocol communication library'. Date accessed: 2018-04-27.
URL: *https://github.com/ke4ukz/PyMSP*

DJI (2012), 'DJI F450 Setup Demo-Frame Assembly'. Date accessed: 2018-04-22.
URL: *https://www.youtube.com/watch?v=pUTHIL_Xfcc*

DJI (2017*a*), 'DJI MG-1S - Agricultural Wonder Drone - YouTube'. Date accessed: 2018-02-20.
URL: *https://www.youtube.com/watch?v=P2YPG8PO9JU*

DJI (2017*b*), 'How to use DJI's Return to Home (RTH) Safely - DJI Buying Guides'. Date accessed: 2018-01-10.
URL: *https://store.dji.com/guides/how-to-use-the-djis-return-to-home/*

Eames, A. (2016), 'Raspberry Pi 3 model B launches today'. Date accessed: 2018-04-10.
URL: *http://raspi.tv/2016/raspberry-pi-3-model-b-launches-today-64-bit-quad-a53-1-2-ghz-bcm2837*

Edwards, L. (2015), 'The internet space race is on: Google Loon vs Facebook drones vs SpaceX satellites'. Date accessed: 2018-02-24.
URL: *https://www.pocket-lint.com/gadgets/buyers-guides/131699-the-internet-space-race-is-on-google-loon-vs-facebook-drones-vs-spacex-satellites*

http://www.pocket-lint.com/news/131699-the-internet-space-race-is-on-google-loon-vs-facebook-drones-vs-spacex-satell

EE (2017), 'Glastonbury 2017: A record year of 4G sharing with more data used than ever before'. Date accessed: 2018-01-09.
URL: *http://newsroom.ee.co.uk/glastonbury-2017-a-record-year-of-4g-sharing-with-more-data-used-than-ever-before/*

Elnahrawy, E., Li, X. L. X. & Martin, R. P. (2004), 'The limits of localization using signal strength: a comparative study', *2004 First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks 2004 IEEE SECON 2004* **00**(c), 406–414.
URL: *http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1381942*

Floreano, D. & Wood, R. J. (2015), 'Science, technology and the future of small autonomous drones'.

Gageik, N., Benz, P. & Montenegro, S. (2015), 'Obstacle detection and collision avoidance for a UAV with complementary low-cost sensors', *IEEE Access* **3**, 599–609.

Gageik, N., Müller, T. & Montenegro, S. (2012), 'OBSTACLE DETECTION AND COLLISION AVOIDANCE USING ULTRASONIC DISTANCE SENSORS FOR AN AUTONOMOUS QUADROCOPTER'.

Gaza07 (2012), 'The Crius All In One Pro Flight Controller (AIOP)'. Date accessed: 2018-03-22.
URL: *http://rctimer.com/download/Crius_AIOP_Manual_MWC.pdf*

Git (2018), 'Git'. Date accessed: 2018-04-28.
URL: *https://git-scm.com/*

Gladstone, P. (2006), 'Discussion about vertical GPS accuracy'. Date accessed: 2018-04-29.
URL: *http://weather.gladstonefamily.net/gps_elevation.html*

Glaser, A. (2017), 'DJI is running away with the drone market'. Date accessed: 2018-01-10.
URL: *https://www.recode.net/2017/4/14/14690576/drone-market-share-growth-charts-dji-forecast*

Godsall, D. (2018), 'Five days of snow in Loughborough forecast - Loughborough Echo'. Date accessed: 2018-04-28.

URL: *https://www.loughboroughecho.net/news/local-news/five-days-snow-loughborough-forecast-14341836*

Gonzalez, D. (2017), 'How to Build a Drone: A Beginner's Guide — Skilled Flyer'. Date accessed: 2018-01-10.
URL: *https://skilledflyer.com/how-to-build-a-drone/*

Heathman, A. (2018), 'SpaceX satellite launch: first step in Starlink internet project'. Date accessed: 2018-04-08.
URL: *https://www.verdict.co.uk/spacex-satellite-launch-starlink/*

Hern, A. (2016), 'Raspberry Pi 3: the credit card-sized 1.2GHz PC that costs $35'. Date accessed: 2017-11-28.
URL: *https://www.theguardian.com/technology/2016/feb/29/raspberry-pi-3-launch-computer-uk-bestselling*

Highsmith, J. & Cockburn, A. (2001), 'Agile software development: The business of innovation', *Computer* **34**(9), 120–122.
URL: *http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=947100*

HobbyKing (2018), 'Battery Safety Guidelines'. Date accessed: 2018-04-23.
URL: *https://hobbyking.com/en_us/battery-safety-guidelines*

Howell, J. (2015), 'The rise of drones in movie-making'. Date accessed: 2018-02-20.
URL: *http://www.macleans.ca/culture/movies/the-rise-of-drones-in-movie-making/*

Internet.org (2016), 'Connectivity Lab'. Date accessed: 2018-02-25.
URL: *https://info.internet.org/en/story/connectivity-lab/*

Janzen, D. & Saiedian, H. (2005), 'Test-driven development concepts, taxonomy, and future direction', *Computer* **38**(9), 43–50.

Joshi, D. (2017), 'Drone Technology and Usage: Current Uses and Future Drone Technology - Business Insider'. Date accessed: 2018-02-20.
URL: *http://uk.businessinsider.com/drone-technology-uses-2017-7*

Jumpy07 & Gaza07 (2013), 'Crius AIOP V2.0 Guide for MegaPirateNG'. Date accessed: 2018-03-20.
URL: *http://www.xxl-modellbau.de/abbildungen/quadcopter/flugsteuerung/crius_aiop_v2/Crius_AIOP_V2_0_Guide_for_MegaPirateNG.pdf*

Kukreja, N., Boehm, B., Payyavula, S. S. & Padmanabhuni, S. (2012), Selecting an appropriate framework for value-based requirements prioritization, *in* '2012 20th IEEE International Requirements Engineering Conference, RE 2012 - Proceedings', pp. 303–308.

Kulkarni, N. (2017), 'Filming using Drones'. Date accessed: 2018-02-20.
URL: *https://dronetrends.org/filming-using-drones/*

le Bon, A. (2015), 'The Drone Pi: 7 Steps (with Pictures)'. Date accessed: 2018-01-10.
URL: *http://www.instructables.com/id/The-Drone-Pi/*

Liang, O. (2013), 'Quadcopter PID Explained - Oscar Liang'. Date accessed: 2018-04-29.
URL: *https://oscarliang.com/quadcopter-pid-explained-tuning/*

Lim, C. H., Wan, Y., Ng, B. P. & See, C. M. S. (2007), 'A real-time indoor WiFi localization system utilizing smart antennas', *IEEE Transactions on Consumer Electronics* **53**(2), 618–622.

Liu, T., Bahl, P. & Chlamtac, I. (1998), 'Mobility modeling, location tracking, and trajectory prediction in wireless ATM networks', *IEEE Journal on Selected Areas in Communications* **16**(6), 922–935.

Liu, Z., Li, Z., Liu, B., Fu, X., Ioannis, R. & Ren, K. (2015), Rise of Mini-Drones, *in* 'Proceedings of the 2015 Workshop on Privacy-Aware Mobile Computing - PAMCO '15', pp. 7–12.
URL: *http://dl.acm.org/citation.cfm?doid=2757302.2757303*

Lu, H., Li, Y., Mu, S., Wang, D., Kim, H. & Serikawa, S. (2017), 'Motor Anomaly Detection for Unmanned Aerial Vehicles Using Reinforcement Learning', *IEEE Internet of Things Journal* p. 3.
URL: *http://ieeexplore.ieee.org/document/8004441/*

Lu, S. & Bharghavan, V. (1996), 'Adaptive resource management algorithms for indoor mobile computing environments', *ACM SIGCOMM Computer Communication Review* **26**(4), 231–242.
URL: *http://portal.acm.org/citation.cfm?doid=248157.248177*

Mathuranathan, V. (2013), 'Log Distance Path Loss or Log Normal Shadowing Model'. Date accessed: 2018-04-17.
URL: *http://www.gaussianwaves.com/2013/09/log-distance-path-loss-or-log-normal-shadowing-model/*

MDN (2018), 'WebSockets - Web APIs — MDN'. Date accessed: 2018-04-27.
URL: *https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API*

Metageek (2017), 'Understanding RSSI'. Date accessed: 2018-04-17.
URL: *https://www.metageek.com/training/resources/understanding-rssi.html*

Meyer, D. (2018), 'What You Need to Know About SpaceX's Satellite Broadband Plans'. Date accessed: 2018-02-25.
URL: *http://fortune.com/2018/02/22/spacex-starlink-satellite-broadband/*

Miller, P. (2017), 'Raspberry Pi sold over 12.5 million boards in five years'. Date accessed: 2017-11-28.
URL: *https://www.theverge.com/circuitbreaker/2017/3/17/14962170/raspberry-pi-sales-12-5-million-five-years-beats-commodore-64*

Ministry of Defence & Military Aviation Authority (2017), 'Drones - are you flying yours safely? (and legally?)'. Date accessed: 2018-04-23.
URL: *https://www.gov.uk/government/news/drones-are-you-flying-yours-safely-and-legally*

Miranda, J., Abrishambaf, R., Gomes, T., Goncalves, P., Cabral, J., Tavares, A. & Monteiro, J. (2013), Path loss exponent analysis in Wireless Sensor Networks: Experimental evaluation, *in* '2013 11th IEEE International Conference on Industrial Informatics (INDIN)', number July, IEEE, pp. 54–58.
URL: *http://ieeexplore.ieee.org/articleDetails.jsp?arnumber= 6622857 http://ieeexplore.ieee.org/document/6622857/*

MultiWii (2014*a*), 'ESCs - MultiWii'. Date accessed: 2018-04-27.
URL: *http://www.multiwii.com/wiki/index.php?title=ESCs*

MultiWii (2014*b*), 'MultiWii Wiki'. Date accessed: 2018-04-27.
URL: *http://www.multiwii.com/wiki/*

MultiWii (2015*a*), 'MultiWii'. Date accessed: 2018-04-26.
URL: *http://www.multiwii.com/*

MultiWii (2015*b*), 'multiwii - Google Code Archive'. Date accessed: 2018-04-23.
URL: *https://code.google.com/archive/p/multiwii/*

Nigel (2017), 'What is Pitch, Roll and Yaw?'. Date accessed: 2018-04-30.
URL: *https://emissarydrones.com/what-is-roll-pitch-and-yaw*

Nunns, G. (2017), 'Aerial views of snow in Loughborough'. Date accessed: 2018-04-28.
URL: *http://garethnunns.com/blog/aerial-views-of-snow-in-loughborough/*

Nunns, G. (2018), 'Maiden flight of the Pi WiFi drone'. Date accessed: 2018-04-28.
URL: *http://garethnunns.com/blog/maiden-flight-pi-wifi-drone/*

Ogden, M. (2012), 'droneduino'. Date accessed: 2017-11-28.
URL: *https://gist.github.com/maxogden/4152815*

Otto, M. & Thornton, J. (2017), 'Bootstrap · The most popular HTML, CSS, and JS library in the world.'. Date accessed: 2018-04-26.
URL: *https://getbootstrap.com/*

Parrot (2017), 'Quadcopter AR Drone 2.0 GPS Edition'. Date accessed: 2017-11-28.
URL: *https://www.parrot.com/uk/drones/parrot-ardrone-20-elite-edition#parrot-ardrone-20-elite-edition*

Pasztor, A. (2018), 'SpaceX Indicates Satellite-Based Internet System Will Take Longer Than Anticipated - WSJ'. Date accessed: 2018-02-25.
URL: *https://www.wsj.com/articles/spacex-indicates-satellite-based-internet-system-will-take-longer-than-anticipated-1519227620*

PiCopter (2017), 'PiCopter'. Date accessed: 2017-11-28.
URL: *https://www.picopter.org/*

Project Loon (2014), 'Welcome all to the inaugural Golden Balloon Awards 2014!'. Date accessed: 2018-04-08.
URL: *https://plus.google.com/+ProjectLoon/posts/aJVCL6dFbz9*

Project Loon (2017), 'Technology - Project Loon - Project Loon'. Date accessed: 2018-02-25.
URL: *https://x.company/loon/technology/*

Rajasekar, S., Philominathan, P. & Chinnathambi, V. (2013), 'RESEARCH METHODOLOGY', p. 8.
URL: *https://arxiv.org/pdf/physics/0601009.pdf*

Rao, B., Gopi, A. G. & Maione, R. (2016), 'The societal impact of commercial drones', *Technology in Society* **45**, 84.

URL: *https://ac.els-cdn.com/S0160791X15300828/1-s2.0-S0160791X15300828-main.pdf?_tid=839e7dba-15dd-11e8-808b-00000aacb362&acdnat=1519090366_c5b9bf9fe28558d40c2936d0ac55311e*

Rouse, M., Earls, A., Shea, S. & Wigmore, I. (2016), 'What is drone (unmanned aerial vehicle, UAV)?'. Date accessed: 2017-11-27.
URL: *http://internetofthingsagenda.techtarget.com/definition/drone*

Rozyyev, A., Hasbullah, H. & Subhan, F. (2011), 'Indoor Child Tracking in Wireless Sensor Network using Fuzzy Logic Technique'.

Rufus, D. (2016), 'How to Build a Drone - A Definitive Guide For Newbies'. Date accessed: 2018-01-10.
URL: *http://beginnerflyer.com/build-a-drone/*

Samsung Group (2017), 'Galaxy Note7: What We Discovered'. Date accessed: 2018-04-26.
URL: *https://news.samsung.com/global/infographic-galaxy-note7-what-we-discovered*

Soh, W.-S. & Kim, H. S. (2003), 'QoS provisioning in cellular networks based on mobility prediction techniques', *IEEE Communications Magazine* **41**(1), 1.
URL: *http://ieeexplore.ieee.org/document/1166661/*

Soh, W. S. & Kim, H. S. (2004), Dynamic bandwidth reservation in cellular networks using road topology based mobility predictions, *in* 'Proceedings - IEEE INFOCOM', Vol. 4, IEEE, pp. 2766–2777.
URL: *http://ieeexplore.ieee.org/document/1354694/*

Song, L., Deshpande, U., Kozat, U. C., Kotz, D. & Jain, R. (2006), Predictability of WLAN mobility and its effects on bandwidth provisioning, *in* 'Proceedings - IEEE INFOCOM', p. 2.
URL: *http://softlab-pro-web.technion.ac.il/projects/TrackMe/doc/22_02.PDF*

Tahat, A., Kaddoum, G., Yousefi, S., Valaee, S. & Gagnon, F. (2016), 'A Look at the Recent Wireless Positioning Techniques with a Focus on Algorithms for Moving Receivers', *IEEE Access* **4**, 6652–6680.

Thomas, S. (2015), 'Drone On', *Rail Profressional Magazine* **94**(May), 81–82.
URL: *http://heinonline.org/HOL/Page?handle=hein.journals/fora94&id=646&div=&collection= http://issuu.com/railpro/docs/may_issue_rail_pro/80*

Tiwari, A. & Dixit, A. (2015), 'Unmanned Aerial Vehicle and Geospatial Technology Pushing the Limits of Development', *American Journal of Engineering Research (AJER)* **4**(01), 16–21.
URL: *www.ajer.org*

Torres-González, A., Capitán, J., Cunha, R., Ollero, A. & Mademlis, I. (2017), ROBOT 2017 : Third Iberian Robotics Conference. Volume 1, Springer, p. 337.
URL: *https://books.google.co.uk/books?hl=en&lr=&id=Cyg-DwAAQBAJ*

U.S. Air Force (2017), 'GPS Accuracy'. Date accessed: 2018-04-12.
URL: *https://www.gps.gov/systems/gps/performance/accuracy/*

Valente, J., Sanz, D., Barrientos, A., del Cerro, J., Ribeiro, Á. & Rossi, C. (2011), 'An air-ground wireless sensor network for crop monitoring', *Sensors* **11**(6), 6088–6108.
URL: *http://www.mdpi.com/1424-8220/11/6/6088/*

Vargas, A. (2017), 'MultiWii Serial Protocol (MSP) API'. Date accessed: 2018-04-27.
URL: *https://github.com/alduxvm/pyMultiWii*

Vasisht, D., Kumar, S. & Katabi, D. (2016), 'Decimeter-Level Localization with a Single WiFi Access Point', *Nsdi 2016* (ii), 165–178.
URL: *https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/vasisht*

Wagster, J. & Rose, M. (2012), 'Obstacle Avoidance System for a Quadrotor UAV', *Infotech@ Aerospace 2012* p. 10.
URL: *http://arc.aiaa.org/doi/pdf/10.2514/6.2012-2548*

Waters, R. (2017), 'Google parent shifts stance on internet balloons'. Date accessed: 2018-02-25.
URL: *https://www.ft.com/content/b8066f88-f485-11e6-8758-6876151821a6*

Webster, J. & Watson, R. (2002), 'ANALYZING THE PAST TO PREPARE FOR THE FUTURE: WRITING A LITERATURE REVIEW Analyzing the past to prepare for the future', *Management Information Systems Quarterly* **26**(2).
URL: *http://www.jstor.org/stable/4132319 http://about.jstor.org/terms http://aisel.aisnet.org/misq/vol26/iss2/3/*

Wu, Y., Chou, P. a., Zhang, Q., Jain, K., Zhu, W. & Kung, S.-Y. (2005), 'Network Planning in Wireless Ad-hoc Networks : A Cross-Layer Approach', *IEEE Journal on Selected Areas in Communications* **23**(1), 136–150.

Xu, J., Liu, W., Lang, F., Zhang, Y. & Wang, C. (2010), 'Distance Measurement Model Based on RSSI in WSN', *Wireless Sensor Network* **02**(08), 606–611. **URL:** *http://www.scirp.org/journal/doi.aspx?DOI=10.4236/wsn.2010.28072*

Zhao, Y. (2000), 'Mobile Phone Location Determination and Its Impact on Intelligent Transportation Systems', *IEEE Transactions on Intelligent Transportation Systems* **1**(1), 55–64.