Object Tracking Using Drone Swarms

27.10.2021

E25 Final Report

Sponsors: Graeme Woodward, the Wireless Research Centre

Research supported by Steve Pawson

Team Members:

Oli Dale (82258564)

Rowan Sinclair (42864286)

Nicholas Ranum (53147503)

Connor O'Reilly (16987735)

Alex Scott (54208312)

Supervisors: Richard Clare and Mike Shurety

Executive Summary

There is a fundamental gap in the behaviour of invertebrate species within New Zealand. This has led to poor conservation management, biosecurity risks, and ineffective pest control. The invertebrate species within New Zealand are under threat with 34.5% classed as either 'threatened' or 'at risk' [1]. Utilising tracking methods has shown to be crucial in previous conservation successes of threatened species. However, due to the small nature of invertebrates, traditional methods are unable to be used and alternative methods are ineffective. The lack of information surrounding invertebrate species has also led to ineffective pest control methods that are expensive and toxic. Invertebrate pests are estimated to cause $2.3 billion dollars of damage to the agricultural industry and $880 million dollars of biodiversity loss annually [2] [3]. There is a demand for non-toxic pesticide alternatives which can only be obtained with more information on invertebrate behaviour.

The Wireless Research Centre (WRC) is currently developing a new form of radar technology that is small and light enough to be placed on targets such as insects. The system works by transmitting a signal which is reflected by the target carrying a harmonic radar tag. The tag reflects the signal at a harmonic of the original frequency which is captured by a receiver. This signal can be used to measure the distance to the target. Due to the limited distance range of the technology, a drone swarm equipped with radar system has been proposed for tracking.

The harmonic radar technology is currently being developed in parallel with the sponsored project and will not be implemented within the final system. This project aims to take the output of the harmonic radar system to locate and track a moving target using a swarm of five drones. The swarm will consist of a transmitter drone which will emit a signal which is reflected by the tag and captured by four receiver drones. To simulate the radar readings, a GPS target will be used which broadcasts its position wirelessly to each drone. Using a robust inter-drone communication network, the transmitter drone will receive the current position and radar reading of each drone to determine the location of the target. Each drones desired position will then be updated using swarming logic. A flight control module will interface with the flight controller to move the drone to the next position. To ensure the swarm is safe and robust, suitable failsafes will be implemented.

This system builds upon work completed in a previous final year project. A robust system has been developed that uses improved localisation and tracking methods to locate the target more accurately. The inter-drone communication, failsafes and data logging have also been overhauled. The full system was tested in a practical flight test where one receiver drone was flown, and four drones were simulated. The system was able to successfully track a walking target while staying in the desired formation.

Table of Contents

1 Introduction

1.1 Purpose Statement

This project aims to use a swarm of drones to track small moving objects in 3D space. The team will build upon work completed in 2020 by adding multiple flying drones, fail-safes, and improving tracking and communication methods. This will culminate in a demonstration where a swarm of drones will track a moving target.

1.2 Project Sponsors

Last year's student group worked with the Wireless Research Centre (WRC) and Scion, a company specialising in research and technology within forestry. Scion's primary purpose was centered around tracking insects specifically. This year, the WRC is continuing the project with Graeme Woodward as the head, but Scion will not be partnered. A previous Scion entomologist Steve Pawson has begun working at the University of Canterbury Forestry department. He is in support of this project and its benefit to the field of biologic study, specifically entomology.

1.3 Project Definition

34.5% of the New Zealand invertebrate species are classified by the Department of Conservation as threatened or at risk [1]. Management of these species is limited by a lack of knowledge about their behaviour, e.g., migration and movement habits. This gap has led to poor conservation outcomes with operational management not reflecting the biology of individual species.

Our group continued a long-term project by the Wireless Research Centre (WRC) at the University of Canterbury which seeks to develop a method of tracking insects using a swarm of drones and harmonic radar [4] [5].

The drones form a swarm consisting of one transmitter drone and four receiver drones, shown in Figure 1.1. The transmitter emits a signal which reflects off a harmonic tag mounted on the target. It is then received at different times by each receiver drone depending on the drone's distance from the target. Based on the radar signal's time of arrival, the target's position can be determined. The harmonic tag, mounted on the target, is an import part of this system. It receives the radar signal and re-radiates it at a harmonic of its original frequency. In this way the signal from the target can be distinguished from signals bouncing off other objects. This gives harmonic radar an advantage over traditional radar systems.

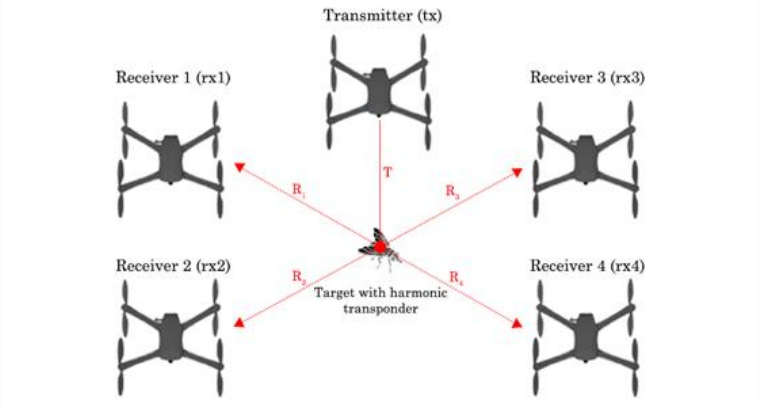


Figure 1.1: The drone swarm layout and multistatic radar arrangement.

Drone tracking using harmonic radar can theoretically track any object that can carry a transponder tag. The harmonic radar tags intended for this project are passive, i.e. battery free, and thus are significantly

lighter (~10mg) than traditional tracking methods such as GPS and VHF radio modules (~250mg) [6]. In addition, because there is no dependence upon a battery, the tags do not have a finite field life. In contrast, the smallest available active tags have a field life of about 48hrs before the batteries are depleted. Therefore, this technology may be applied to other contexts and can be used by parties outside the insect tracking field.

The development of the drone-based harmonic radar transponder equipment is currently independent from this project and is not yet ready for integration. For this reason, the harmonic transponder component which will be mounted on the target has been replaced by a transponder which broadcasts its GPS position over Wi-Fi. The transmitter drone uses these GPS readings to simulate the time reading which the harmonic radar would produce. The project's goal is to create an operational autonomous UAV swarm, where the harmonic radar system could be substituted in place of the simulated range readings to produce a working prototype.

1.4 Summary of Progress

As stated earlier, this project is a continuation from a student group last year. A git repository was provided, allowing the code to be built and improved upon. They were able to achieve one drone physically flying with the others represented in a simulated environment.

The systems capability is tested by conducting practical and simulated flight tests. Simulated flight tests are conducted using a physics-based simulation, Gazebo. This allows for a replication of the practical flight test environment.

The Gazebo simulation environment can emulate the swarm tracking a set of GPS coordinates representing the moving target. The simulation is fully capable of simulating the real time interactions and events that occur in the real practical test flights. This allows for quicker development and testing of the code without organising practical test flights. It also removes the risk of collision during the early stages of development. In its current state, the full drone swarm can successfully track the target in the simulated environment.

From building upon the previous code, we achieved a flight test with two real drones and three simulated drones successfully tracking a GPS target. Two receiver drones were flown into the air and then connected to the Gazebo simulation environment which simulated the remaining two receiver drones and the transmitter drone. The flight tests successfully tracked the moving target (a laptop emitting the GPS signal), which moved around an open field at jogging pace.

We were not able to perform a test flight with all the drones flying. It was determined later in the project, that the base code provided would not be reliable enough to fly four drones physically. This was caused by GPS error and a rudimentary approach to tracking. Because of this, the code was reworked to implement 3D multilateration and a Kalman filter. The inter-drone communication, failsafes and data logging were also overhauled to provide a more robust system. With this, the position readings for the drones and the tracking would theoretically be more consistent. The new system had two weeks of testing before the project conclude. With it we were able to achieve one drone physically flying with the others simulated.

1.5 COVID Consideration

The recent coronavirus lockdown had a significant impact on this project. The period of level 4 lockdown directly intervened with our ability to conduct flight tests. While we were able to create a plan and work around it, the inability to physically meet made any practical tests on the project impossible.

The revised plan was to spend the lockdown improving the code and adding in the needed improvements. However, this only left two remaining weeks to practically test before the project had to be concluded. Given no lockdown and having the time to gradually test the code as additions were made would have greatly benefitted the project. Regardless, zoom meetings were held with our supervisors which allowed us to continue having regular meetings.

1.6 Topic Order

The project was a collaborative effort and many of the tasks required multiple people to complete. However, each member of the team was given an aspect of the project to focus on:

1. **Oli Dale** – Target Localization (Multilateration): Using multilateration calculations to accurately determine the target's position in 3-dimensional space.
2. **Rowan Sinclair** – Target Tracking and State Machine: Implementing a Kalman Filter to track the target through signal dropouts and noise. Designing a state machine to control the drones in the air.
3. **Nicholas Ranum** – Communications: Ensuring the communications between the drones are stable.
4. **Connor O'Reilly** – Data Logging and Visualization: Logging data from practical flights and displaying them in a readable manner.
5. **Alex Scott** – Drone Failsafes: Ensuring the drones and the surrounding environment remain safe during flight.

2 Target Localization (Multilateration) - Oli Dale

2.1 System Overview

To locate the target, each receiver drone calculated a simulated bistatic radar range from the target. A bistatic radar reading consists of two components. The distance from the transmitter drone to the target, summed with the distance from the target to itself. Using these four simulated radar readings, one for each drone, the target could be located in 3D space (Figure 2.1).

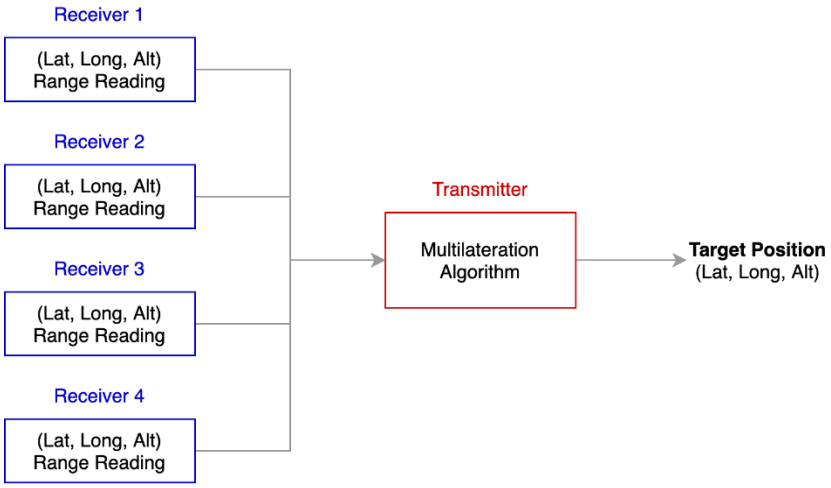
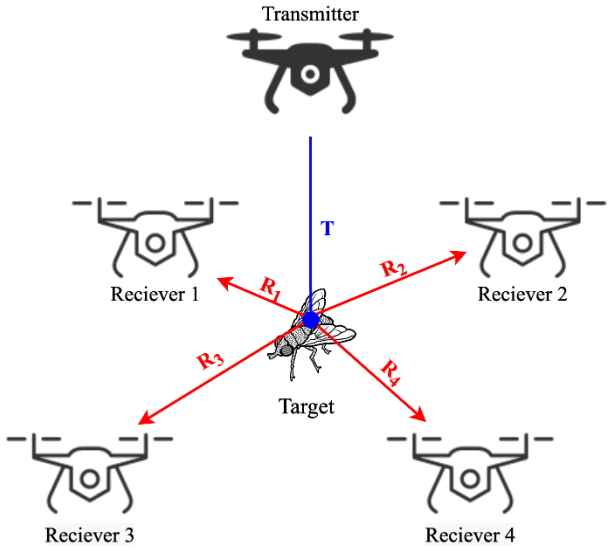


Figure 2.1: Multilateration system diagram

As the radar technology is still being developed by the Wireless Research Centre, a laptop with a GPS module was used as the target. Each receiver drone calculated a simulated bistatic radar reading by considering the distance between the drones and target. This process is shown in Figure 2.2.



$$\textbf{Range} = T + R_i$$

Figure 2.2: Drone swarm arrangement

The localisation was designed to meet a range of design specifications. These described the characteristics of the target being tracked and the accuracy required.

R1.1 The system must operate in 3 dimensions.
R1.2 The system must be able to track a kinematically constrained target.
R1.3 Must operate with an update frequency of 1 second.
R1.4 Have a localisation accuracy less than 1 metre from the true location.

2.2 Previous Implementation

The system developed in the 2020 Final Year Project involved a multilateration module which located the target using a 2D grid-based search method. This assumed that the target was in the same plane as the drones. If the target was above or below the drones, this assumption introduced projection errors into the solution. This is shown in Figure 2.3.

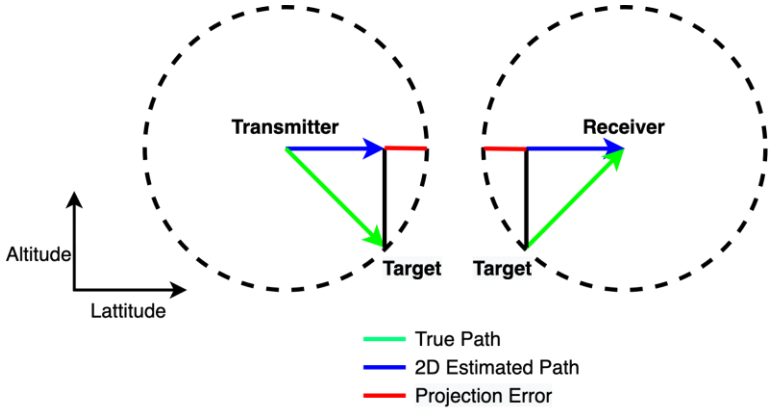


Figure 2.3: Projection errors from the 2D grid search method

The projection errors introduced can be as large as the altitude difference between the transmitter and target, and get smaller as the target moves away from the centre of the swarm. While this method provided a proof of concept for target localisation, a robust method which considered altitude needed to be developed for the future implications of this project. Within the environment, it will be vital for the swarm formation to be flexible, allowing for obstacle avoidance and swarm maintenance. It is also likely that the drones will swarm above the target to optimise radar measurement accuracy and to minimise the sound disturbances to the target. All of these require the ability to accurately locate the target within 3-dimensional space. Hence, a new method was researched and tested.

2.3 Research

Due to the unique arrangement of the transmitter and receivers, conventional localisation methods cannot be used. The harmonic radar produces radar readings that consist of two components, forming a bistatic radar signal. In contrast, conventional multilateration methods consider radar readings which are transmitted directly from the target. Common methods for locating the target include Bancrofts Algorithm and Fangs method. With the current multistatic arrangement (multiple transmitters or receivers) formed by the drone swarm, the most popular methods for locating the target include Time Difference of Arrival (TDOA) and Time Sum of Arrival (TSOA). These algorithms both form a set of non-linear equations which involve intersecting hyperbolas for TDOA and ellipses for TSOA. Both methods were researched and tested for implementation.

2.3.1 Time Difference of Arrival (Hyperbolic)

The TDOA method considers the difference between each distance radar signal. Each measurement is subtracted from another measurement within the swarm. The common distance from the transmitter to the target is removed, leaving the difference in distance from the target to each receiver drone. In 2 dimensions, readings from 3 receiver drones produce 2 hyperbolas. The intersection of these curves is the

estimated location of the target. When this is shifted to 3 dimensions, the 2D lines turn into 3D hyperbolic surfaces. The following set of non-linear equations are formed with four receiver drones.

$$R_1 - R_2 = \sqrt{(x_1 - x)^2 + (y_1 - y)^2 + (z_1 - z)^2} - \sqrt{(x_2 - x)^2 + (y_2 - y)^2 + (z_2 - z)^2} \quad (2.1)$$

$$R_1 - R_3 = \sqrt{(x_1 - x)^2 + (y_1 - y)^2 + (z_1 - z)^2} - \sqrt{(x_3 - x)^2 + (y_3 - y)^2 + (z_3 - z)^2} \quad (2.2)$$

$$R_1 - R_4 = \sqrt{(x_1 - x)^2 + (y_1 - y)^2 + (z_1 - z)^2} - \sqrt{(x_4 - x)^2 + (y_4 - y)^2 + (z_4 - z)^2} \quad (2.3)$$

Where *R* represents the corresponding range reading for each receiver drone and *(x, y, z)* represents the current position of the receiver drones and target.

2.3.2 Time Sum of Arrival (Elliptic)

The TSOA method considers the summation of each radar reading. An equation is formed for each receiver drone which forms an ellipse around the drones, outlining the possible locations of the target. Where the ellipses intersect is deemed to the position of the target. The following equations can be formed to locate a target in 3 dimensions. As the system is overdetermined, a solver such as least squares must be used.

$$T + R_i = \sqrt{(x_T - x)^2 + (y_T - y)^2 + (z_T - z)^2} + \sqrt{(x_i - x)^2 + (y_i - y)^2 + (z_i - z)^2} \quad (2.4)$$

Where *i* ranges from 1 to 4, representing each receiver drone.

2.4 Testing

Following the design specifications, a successful localisation method had to operate in 3 dimensions, be robust to additive noise and consider the swarm/system constraints. A testing structure was formed to ensure the algorithm could pass all requirements. These were as follows:

1. Test the algorithm in 2 dimensions with both zero and additive noise.
2. Test the algorithm in 3 dimensions with additive noise following a path.
3. Test the algorithm within the entire system using a physics simulator to simulate the swarm.

2.4.1 Two-Dimensional Testing

Both methods were first implemented in 2 dimensions using three receiver readings instead of the total four. The equations were solved in python using a non-linear least squares solver from the SciPy python library. By only working in 2 dimensions, the hyperbolas (TDOA) and ellipses (TSOA) could be plotted to gain a graphical understanding of the solver. With zero additive noise on the radar measurements, both algorithms were able to successfully locate the target with zero error. Additive noise was added to the radar measurements to simulate error from the harmonic radar tag. The results are shown in Figure 2.4. The black dot represents the estimated location, and the red dot represents the real target location.

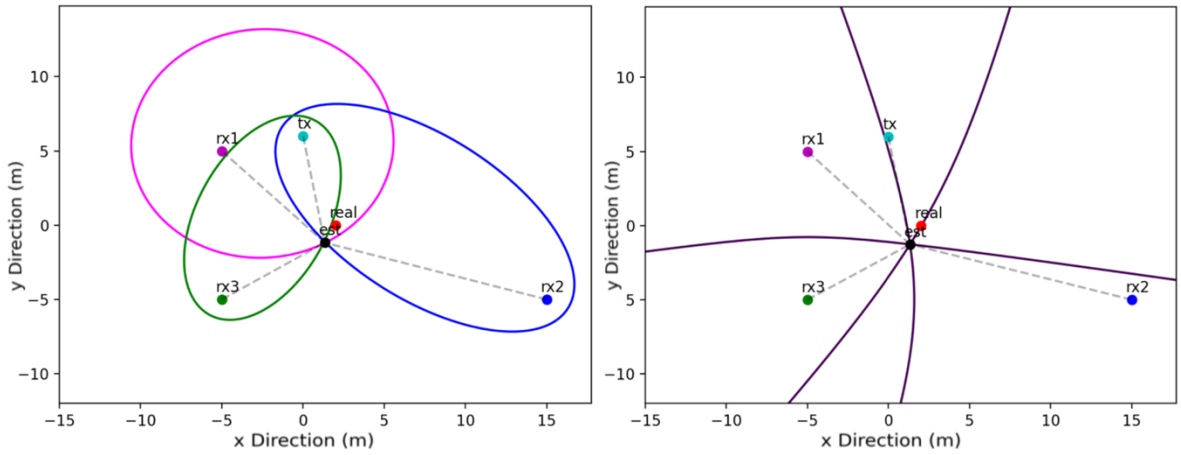


Figure 2.4: 2D testing of the TDOA (left) and TSOA (right) algorithms

As the radar technology is still being developed, an accurate model for the noise was unable to be obtained. Thus, a gaussian distribution with a variance of 1 metre was selected as it models white noise. Both algorithms were able to locate the target with an average accuracy of 1.5 m. The hyperbolic method was computationally more efficient, taking 0.5 ms to compute – an order of magnitude faster than the elliptic method. The computation time was comparable to the grid search implemented last year which took 2.5 ms.

2.4.2 Three-Dimensional Testing

With both methods successfully locating the target in 2 dimensions, the altitude of the target and drones was considered – expanding the scope of the multilateration from the previous year. The same process was conducted. The algorithms were first tested with no noise, and both were successful at locating the target with zero error. Considering noise, the elliptic method was able to robustly locate the target. However, once noise was considered the hyperbolic method went unstable and occasionally did not converge to a solution. The cause of this was investigated but was not determined. As the representation of the possible locations of the target shifted from 2-dimensional lines to 3-dimensional surfaces, it was difficult to analyse the cause graphically. The cause was suspected to be due to the hyperbolic surfaces not intersecting. As the elliptic method was able to produce reliable results it was selected for further testing.

I developed a program to generate a random target path to test the stability of the algorithm. At each time step the previous estimate was used for the initial guess for the linear solver. By testing the algorithm on a series of points, the stability of the localisation was evaluated. The target path was generated by kinematically constraining the movement. The target behaviour was discussed with an entomologist to gain a better understanding of the insect's movement. However, due to insufficient information on the behavior of insects, the maximum values were unable to be estimated. As the full localisation system applies a Kalman Filter (developed by Rowan Sinclair) to track the target, the movement constraints were closely integrated with the Kalman Filter to provide a high level of localisation accuracy. The current target represents a person in a light jog.

Table 2.1: Kinematic constraints on the target

Specification	Limits
Velocity	3 m/s
Acceleration	1 m/s^2
Jerk	1 m/s^3
Horizontal Rate of Change	30 deg/s
Vertical Rate of Change	30 deg/s

These values can be tweaked to change the behaviour of the target to further test the robustness of the multilateration. Where higher values simulate a faster moving, erratic target, and lower values provide an easier target to track. To test the multilateration on a continuous path, the following assumptions were made:

1. The drones positions and radar readings were updated every second.
2. Each radar reading contained noise with a gaussian distribution and standard deviation of 1 m.
3. The drones could instantly move to the swarm formation between timesteps.

The target path lasted for 2 minutes and can be seen in Figure 2.5.

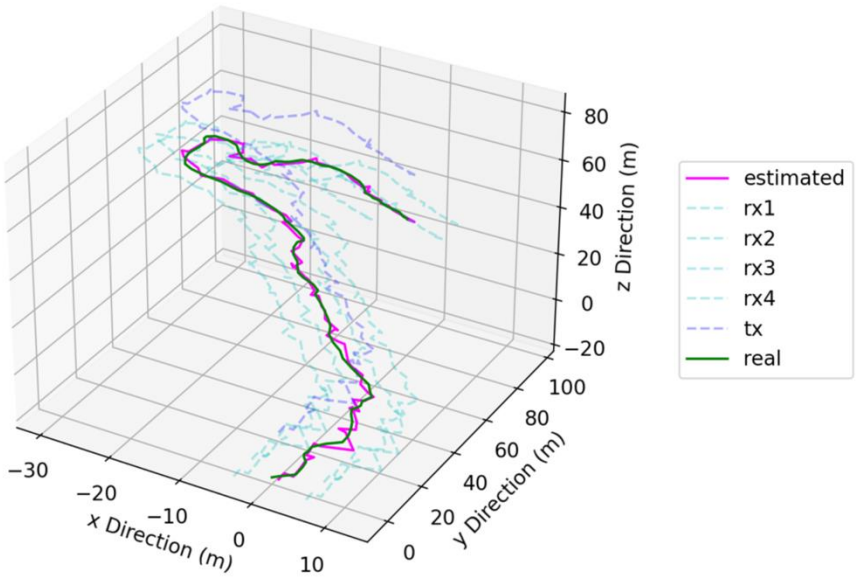


Figure 2.5: Multilateration applied to a target path for 2 minutes

The elliptic method was able to locate the target over 10,000 iterations with an average accuracy of 0.7 m and computation time of 4.5 ms. In Figure 2.5, the drones locations are plotted around the target path, forming a 10 x 10 m square pyramid. As previously stated, the assumption was made that each drone could instantly move to the desired position between time steps. This is not an accurate representation of the swarm dynamics, as the drones will move to each position using PID control on the flight controller. To simulate this movement, the algorithm was tested within a physics-based simulation.

2.4.1 Swarm Integration

The final step was to test the algorithm within the drone swarm system. This was achieved using a Gazebo simulation. Gazebo provides a simulated environment for the drone swarm to be tested. Each drone has a simulated flight controller, which is identical to the real drones. The simulation considers the dynamics of the drones flying, and inaccuracies such as GPS error. The goal was to replace the multilateration algorithm from last year with the improved 3D method. To do this, the system required alterations to shift it from operating in 2-dimensions to 3. The alterations included:

- Updating the swarm to record the altitude of each drone
- Converting the radar-readings on the receiver drones to 3-dimensions, considering altitude.
- Changing the communication packet structure to handle an extra dimension.

To test the multilateration algorithm, a text file of coordinates was used which recorded a person walking around the Wireless Research Centre. Within a simulation, the computational time increased to 30 ms and the average multilateration error was 0.037 m. Comparing this to the previous 2D method, the computational time was 500 ms and had an average error of 0.46 m. Thus, the new method developed was 12 times more accurate and 16 times more efficient. The computational times of both localisation methods significantly increased within the simulation. The cause of this was investigated and was likely to be due to the simulation running slower than a real time system. Despite the extra time, there were no signs of computational lag, and the test flight was successful.

2.5 Practical Flight Tests

In parallel with the multilateration development, the team conducted weekly flight tests to test the system within a practical environment. A practical test flight usually consisted of one or two real drones, with the rest of the system simulated in Gazebo. Each real drone was connected to remotely, to launch the required scripts. With more drones being added to the swarm, more terminals were required for executing and monitoring the launch commands. A typical screen during this process is shown in Figure 2.6.

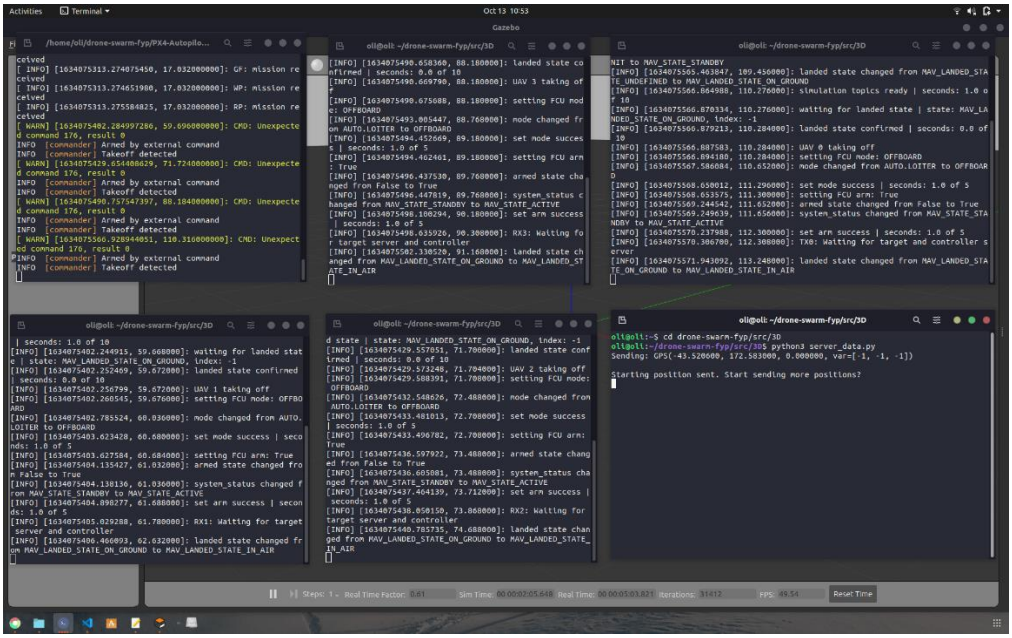


Figure 2.6: Screenshot during practical flight test

As it was difficult to keep track of each drone and its current state, the process was streamlined by developing a user interface. This removed the need for opening multiple terminals to interact with the real and simulated drones. My role was to automate the script launching and monitoring process on the real drones and in the simulation. The developed GUI is shown in Figure 2.7.

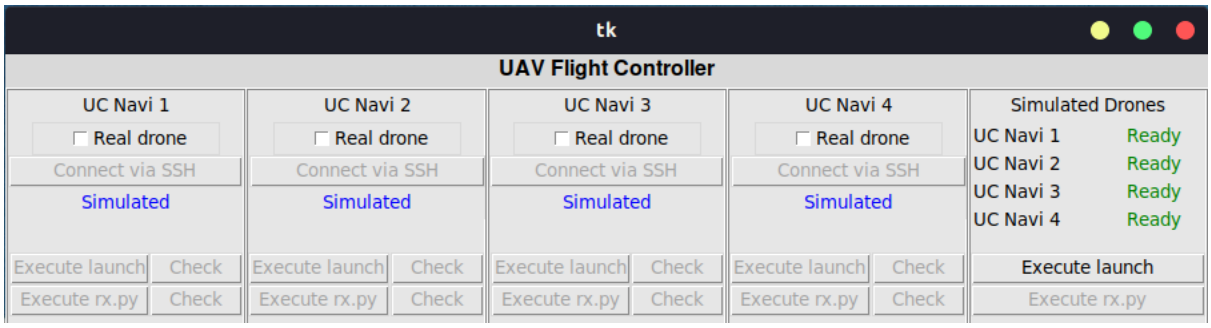


Figure 2.7: User interface to launch the swarm

The GUI was separated into two sections to control the real and simulated drones. Currently, the controller supports four practical receiver drones with the transmitter drone simulated in Gazebo. The GUI allows each receiver drone to be flown in a field test or simulated. The front end of the GUI was developed in Tkinter for its simplicity. The backend was developed using Paramiko and subprocess. Paramiko provides API for a python implementation of the SSHv2 protocol, while subprocess is a python module to invoke external commands to the OS.

Real Drone Launch Procedure: By selecting a drone to be real, an SSH connection could be formed with the click of a button. The unique IP address, username and password of each drone is stored in a json file that allows each drone to be connected to. The connection status is updated at 2Hz to ensure a reliable connection. Once a connection has been formed, each script can be run. The output of the commands is buffered to separate windows that can be opened for monitoring. Each command is executed in a new thread to ensure the GUI does not freeze while executing and buffering the output of the commands.

Simulated Drone Launch Procedure: The drones that are selected to be simulated are launched on the right-hand side of the GUI in one Gazebo simulation. The first button launches a Gazebo simulation which spawns in the simulated receiver drones and transmitter drones. This process could not be executed by simply running a child subprocess within the python script, as it requires shell environment variables to be set. To set these, a bash script was launched instead that ran the required commands and then launched the simulation. Once the Gazebo simulation was launched, the swarming code could be executed. This commands the drones to take off to 10 metres, and the prompts the user to begin tracking.

2.6 Discussion

2.6.1 Localisation Considerations

The goal of the multilateration was to locate the target in 3 dimensions using the four distance radar measurements from the swarm. The work undertaken was suitable for the practical flight tests, but further research was necessary to understand its full capability. The final method (used to track targets such as insects) will need to be robust to handle the changing dynamics of the swarm and target, as well handling inaccuracies in the harmonic radar signals. These effects were investigated.

2.6.1.1 Swarm and Target Dynamics

The effect of the targets position on the multilateration error was investigated. While tracking a target such as an insect, it is important that the localisation is still accurate, even when the target is located away from the centre of the swarm. To investigate this, the swarm was kept in a square pyramid formation,

while the target moved around in a 50 x 50 m grid surrounding the swarm. The multilateration error was calculated 100 times at each position and then averaged. A resolution of 1 metre was used to iterate through the grid.

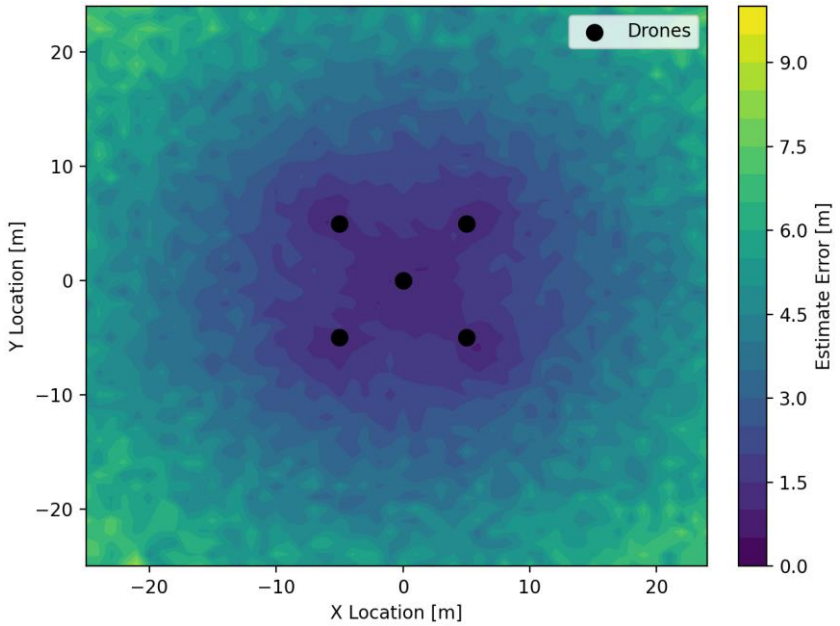


Figure 2.8: Effect of target position on multilateration error

As the target moved away from the centre of the swarm, the multilateration error increased. The error ranged from 0.5 metres - directly under the swarm, to 9.0 metres at 35 metres away from the swarm. To meet the required design specification, the target needs to be located within the swarm. This will require a very responsive and accurate swarm controller. As this is likely to be unachievable, further research will need to be conducted to identify an optimum formation to minimise the localisation error.

During tracking, the swarm is likely to change shape to handle obstacle avoidance and swarm maintenance. Currently, all testing has been conducted in a square pyramid formation with the transmitter drone 10 metres above the receiver drones. The effect of the transmitter height relative to the receiver drones was investigated. The error of the multilateration localisation was measured using the random target path for 1.5 minutes. The offset of the transmitter drone was varied above the receiver drones from 0 metres to 30 metres. The results are shown in Figure 2.9.

It was discovered that there was a minimum height of 5 metres to ensure the multilateration was stable. At heights greater than 5 metres, the error remained relatively constant, gradually increasing. However, at heights lower than this, the swarm diverged from the true target path due to errors at each time step accumulating. This signifies the importance of wrapping the multilateration algorithm in a Kalman filter. Over the 1.5 minutes, it is likely that a bad radar reading caused highly inaccurate estimate. As this is used as the initial guess for the next time step, without any filtering the localisation can quickly go unstable.

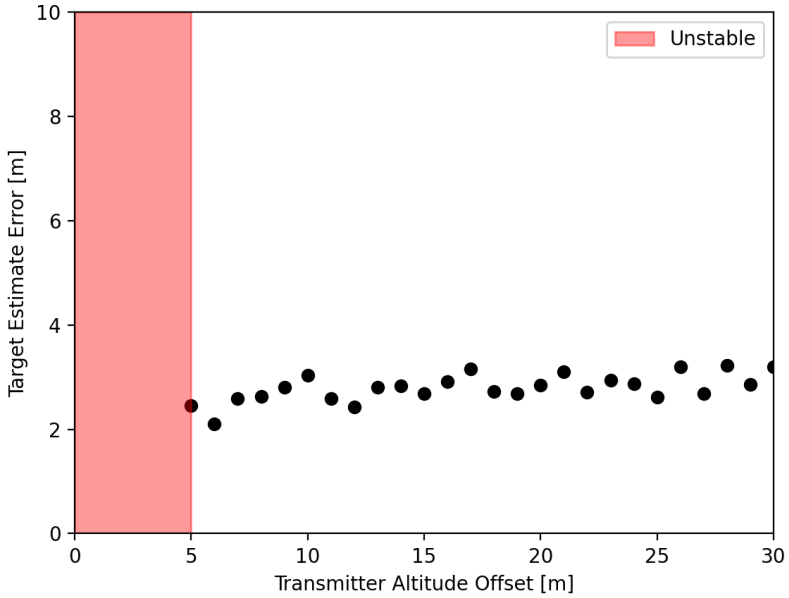


Figure 2.9: Investigating transmitter height above swarm.

2.6.1.2 Harmonic Radar

Two assumptions have been made about the harmonics radar technology for the proof-of-concept swarm system. The measurements were assumed to be normally distributed random variables with a constant variance and mean, as well as having no maximum range measurement. This crude model allowed the algorithm to be tested without the technology implemented and was suitable for a GPS target. However, considering the practical implications of the project, the effect of these assumptions was investigated.

The current harmonic radar technology can support a range up to 28 metres. This range considers the distance from the transmitter drone to the target, summed with distance from the target to the receiver. With an optimised swarm formation, the target could be located within a circle of radius 2.9 m, as shown in Figure 2.10.

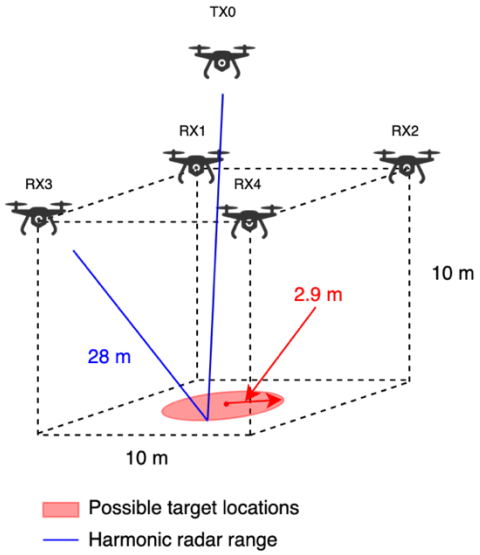


Figure 2.10: Harmonic radar range considerations

This is example considers a swarm with the transmitter drone at 5 metres above the receiver drones. This was deemed to be the minimum height to produce stable tracking without any filtering. The drones were estimated to swarm 10 metres above the target. While the current technology is technically able to locate the target with the current swarm formation, the performance is very limited. At this height, the target can only be located within a circle of 2.9 metres, directly below the transmitter drone. To increase the reliability of the system, the swarm could be brought closer together, or flown closer to the target. This will reduce the distance that the radar signal must travel, allowing the target to be located further away from the centre of the swarm. As the radar technology is still being developed, it is likely that the maximum range of the tag will also increase, supporting a target further away. The harmonic radar technology is known to have a variance that varies with distance. In future, the effect of this should be investigated. It is likely this will further highlight the importance of keeping the target as close as possible to the drone swarm to reduce localisation errors.

2.6.2 GUI Considerations

The current user interface for launching the drones has been tested within the Gazebo simulation environment. No terminals are required to be opened, making the process quick and straight forward. This has proved valuable in speeding up testing procedures. In parallel with this controller, Nicholas has been responsible for a user interface to control the drones during flight. Future development of the controller involves integrating these two modules together to produce a fully functional central controller. As they have both been developed in the same GUI environment using an object orientated approach, the integration should be straight forward. Future development also includes robustly testing the GUI in preparation for use in a practical flight test. Currently, practical flights are still conducted using multiple terminals, as only one or two drones are used. With more real drones being added to the swarm, the demand for the GUI controller increases. In preparation, the GUI should be robustly tested to ensure situations such as communication dropout are accounted for.

2.7 Summary

The goal of the multilateration module was to locate the target in 3 dimensions using the four distance radar measurements from the swarm. The method implemented last year only operated in 2 dimensions which introduced projection errors into the solution. I researched two alternative methods of locating the target that operated in 3 dimensions without requiring a grid search. It was important that the methods met the specified design requirements outlining the latency, accuracy, and search area of the design. The proposed elliptic method meets the specifications and was thoroughly tested and eventually integrated into the swarm system. The algorithm was extensively tested within a Gazebo environment and produced reliable results. The developed method was evaluated to be 12 times more accurate and 16 times more efficient. The target was able to be located with an average error 0.037 m within 30 ms. Further testing was conducted to evaluate the performance of the algorithm by considering the practical application of the swarm tracking an insect.

3 Target Tracking and State Machine - Rowan Sinclair

3.1 Introduction

3.1.1 Target Tracking

Target tracking is the process of combining sensor measurements from all the drones to estimate the target's position. This was my focus for the first half of the year. All five drones are equipped with a GPS sensor, and a radar system which is simulated, for this project. To combine the output of all these sensors, a tracking algorithm was developed which used a Kalman Filter and a constant velocity model. Four sources of error were considered when designing the filter: measurement noise, temporal noise, processing time, and dropout. The tracking algorithm was extensively tested to determine its accuracy and limits.

I also redesigned how all the algorithms within the swarm interact to work with a new central controller. The previous system was built around a target server, which was used to synchronize the drones. The harmonic radar will eventually replace the target server and a new source of synchronization will be needed. The new system will work with a controller, a target server, or both (for testing). The failsafe algorithm was also distributed throughout the drones to improve its reliability.

3.1.2 State Machine

A state machine is a robust behavior model consisting of several states. A state machine can move between states based on inputs and its current state. It behaves differently, and has different outputs, depending on which state it is in. A state machine is run on each drone. It defines the setup process, how they behave while tracking, and how they handle errors. This was my main focus for the second half of the year. It is important that the code which runs on the drones is reliable and behaves predictably. Therefore the state machine was designed to safely handle errors and make setting the drones up for a flight easy. The state machine was also designed to integrate with a controller running on a separate laptop. The user could change the state of the drones and see tracking information through the GUI of the controller. I also defined the interactions between the controller and the drones.

3.2 Work Undertaken

3.2.1 System Description

For the swarm to successfully follow a target several processes, which are spread between the drones, must work in unison. The processes form a chain which starts at the target server, then moves to the receiver drones, then the transmitter drones, and back to the receivers. It ends when all the drones have moved to a new, estimated target position. This process is shown in Figure 3.1.

The process starts at the target server. As well as providing a GPS coordinate for the swarm to follow, the target server acts a source of synchronization between the drones. Each time the server sends a coordinate, the receiver drones increment a counter. All the counter values are sent to the transmitter, along with simulated range data, and the receiver drones' positions. The counter values are used to group position and range readings. This is necessary because there will be a time difference when receiving data from each receiver drone. Once a full group of readings are ready, the tracking algorithm is used to estimate the target's position. This position is logged and sent back to all the receiver drones. All the drones check if the position they are required to move to is safe, before moving to that position.

Once the harmonic radar is ready to be deployed the target server will be removed and the controller will become the new source of synchronization. Similarly, radar ranges will no longer need to be simulated therefore this component can be removed too. The controller will also be used to control the drones' states and how the behave in the air. The components that will be removed once the harmonic radar is ready are outlined in dashed lines in Figure 3.1. This system is redesigned from the one used last year. Several connections which were only during the setup were removed and replaced with a state machine. The dependency on the target server was removed, to facilitate the addition of the harmonic radar.

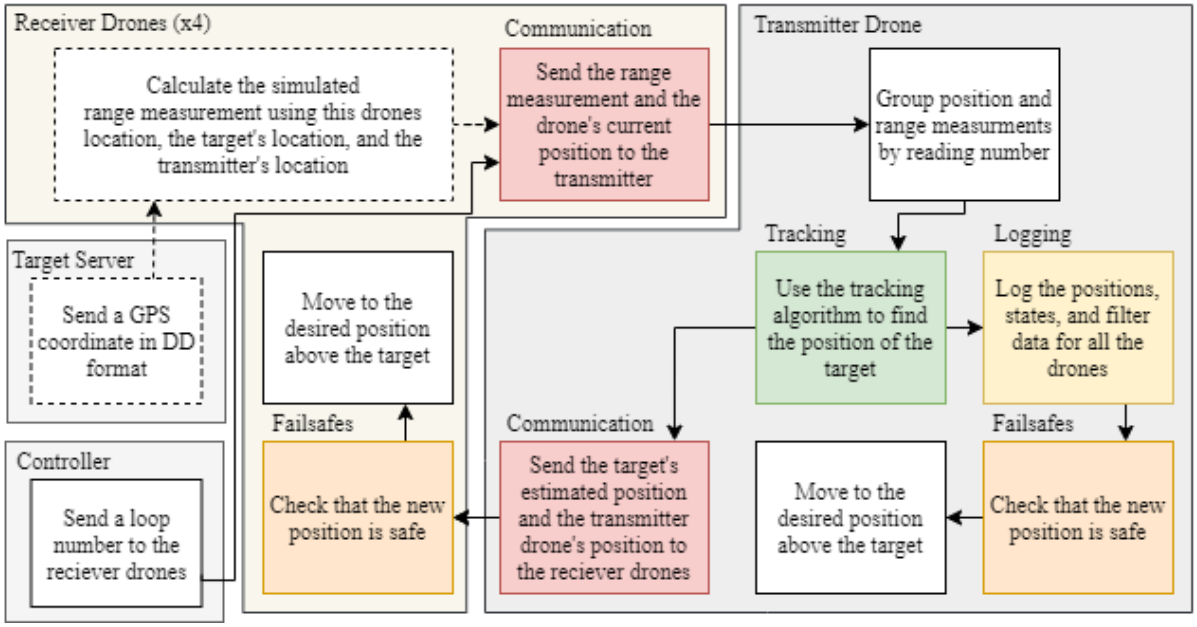


Figure 3.1: System block diagram for the overall drone swarm. The colored blocks correspond to colored sections in Figure 3.2.

3.2.2 A Filter For the Tracking Algorithm

A filter was required within the tracking algorithm to solve the multilateration equations and estimate the target's position. A Kalman Filter was chosen as the best option to track the target. A Kalman Filter is a type of Bayesian filter that compares sensor measurements with model predictions to produce a more accurate estimate than either option alone. This type of filter was chosen as it could model the five drones in the swarm and the target simultaneously using a constant velocity model. Furthermore, the filter is computationally light and produces an estimate along with a standard deviation as an output. A specific variety of Kalman Filter called the Unscented Kalman Filter was used [7]. This type of filter was chosen as it efficiently deals with the non-linear nature of the multilateration equations. This is done by sampling several weighted points around the mean of the nonlinear function. Linear filtering is performed on each point then the points are re-combined based on their weights to produce an approximation of the nonlinear function.

Two other alternatives to the Unscented Kalman Filter were also considered. The first alternative, the Extended Kalman Filter, performs a similar linear approximation to the unscented variety however this filter uses a Taylor Approximation which is more computationally expensive [8]. The second alternative was a Particle filter. This filter uses many Gaussian samples (particles) to produce an accurate approximation of the system [9]. However these filters are computationally intense and difficult to tune.

3.2.3 The Tracking Algorithm Loop

The tracking algorithm consists of a measurement grouping algorithm, the Kalman Filter loop, the multilateration equations, and a conversion algorithm between local and global coordinates. These components form a closed loop, shown in Figure 3.2.

The algorithm starts with an initial estimate of the target's location. This project assumes that the initial location of the target is known. This assumption is reasonable as the target needs to be captured to place the harmonic tag on it anyway. The target's location is supplied in decimal degrees (DD) format and needs to be converted to cartesian coordinates to be processed by the Kalman Filter. The origin of the cartesian system is defined to be the initial starting position of the target. A DD coordinate is converted to a cartesian coordinate by calculating its distance in metres from the origin using an equirectangular approximation. When a coordinate enters the Kalman filter, shown in blue, in Figure 3.2, it is converted to cartesian coordinates and when one leaves it is converted back to DD.

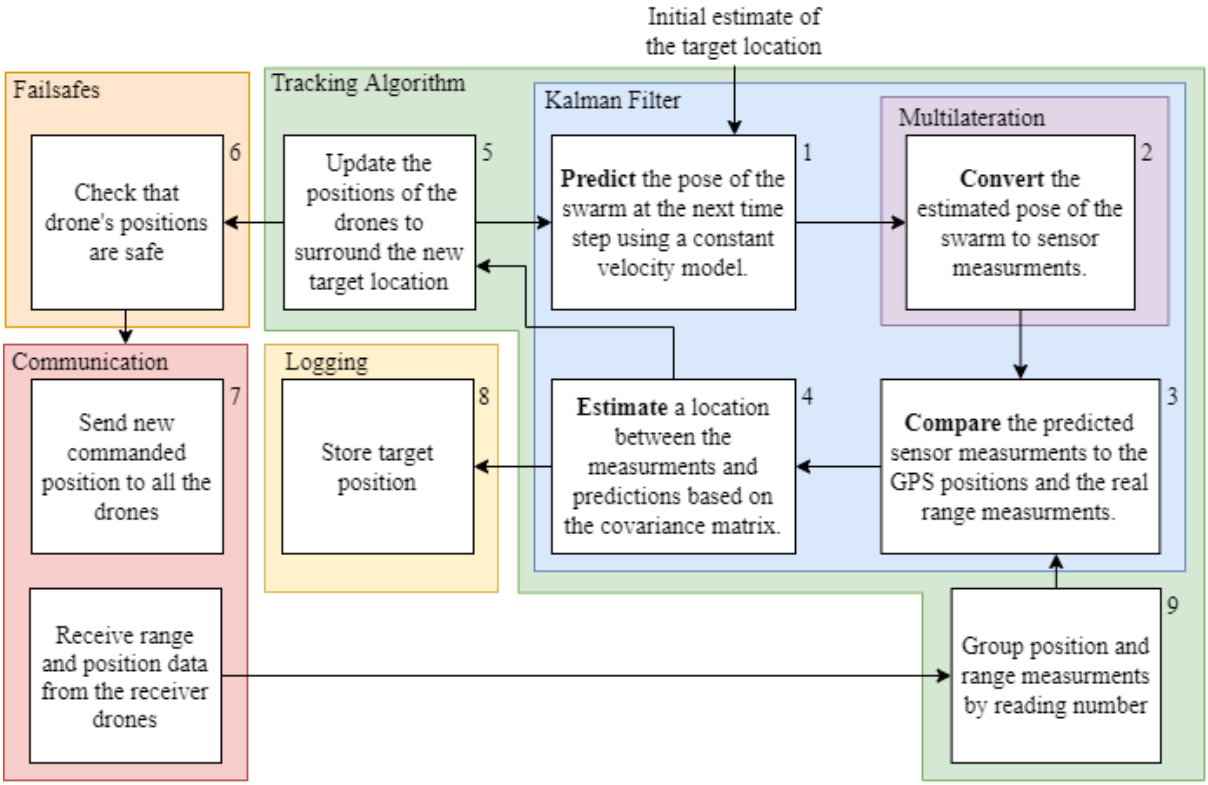


Figure 3.2: System block diagram for the Kalman Filter loop and supporting algorithms. Step numbers are shown in the top-right of each block. The colored sections correspond to colored blocks in Figure X.

The Kalman Filter tracks the positions of all the drones and the target in 3D space. Rotational values are not tracked as they are not required in the multilateration equations, however the positional values are. The state vector for this Kalman Filer has 36 values representing the positions and velocities of all five drones, and the target in 3D space. The state vector is shown in Equation 3.1. For convenience the all the parts of the swarm were given ID numbers. The transmitter drone has an ID of 0, each of the receiver drones are numbered 1 through 4 and the target was given an ID of 5. The IDs of each drone are shown in the subscripts in Equation 3.1.

$$\boldsymbol{x}(t) = (\mathbf{s}_0 \quad \mathbf{s}_1 \quad \mathbf{s}_2 \quad \mathbf{s}_3 \quad \mathbf{s}_4 \quad \mathbf{s}_5 \quad \dot{\mathbf{s}}_0 \quad \dot{\mathbf{s}}_1 \quad \dot{\mathbf{s}}_2 \quad \dot{\mathbf{s}}_3 \quad \dot{\mathbf{s}}_4 \quad \dot{\mathbf{s}}_5)^{\mathrm{T}} \tag{3.1}$$

$$where\ \mathbf{s}_{\mathrm{n}} = (\mathrm{s}_{\mathrm{x,n}} \quad \mathrm{s}_{\mathrm{y,n}} \quad \mathrm{s}_{\mathrm{z,n}})\ and\ n = 1,2,3,4,5,6$$

The Kalman filter uses the initial target location to calculate the initial (prior) drone locations by adding offsets for each drone. For instance the transmitter drone is placed 15 metres above the target. The initial velocity of the target and the drones is assumed to be zero. Kalman Filters normally incorporate some sort of odometry into their motion model. Odometry uses supplied information regarding movement or a commanded position to improve the accuracy of the prediction. Odometry could not be used for this project as the target does not disclose its movements. Instead, a constant velocity model was used to predict the positions of the target and the drones at the next timestep. This model is depicted in step 1 in Figure 3.2. Constant velocity models cannot track, large accelerations and may need to be updated as the movement behavior of the target becomes known. The position of swarm part n at time t is given by a vector $\mathbf{s}_{\mathrm{n}}(t)$. The part's velocity is $\dot{\mathbf{s}}_{\mathrm{n}}(t)$. Equations 3.2 and 3.3 were used for the constant velocity model.

$$\mathbf{s}_{\mathrm{n}}(t + \Delta t) = \dot{\mathbf{s}}_{\mathrm{n}}(t) + \mathbf{s}_{\mathrm{n}}(t)\Delta t, \qquad \dot{\mathbf{s}}_{\mathrm{n}}(t + \Delta t) = \dot{\mathbf{s}}_{\mathrm{n}}(t) \tag{3.2,3.3}$$

Once a full group of sensor measurements are received, the Kalman Filter compares the model prediction to the sensor outputs. A sensor model, $\boldsymbol{h}(\boldsymbol{x})$, is required to convert the state vector, $\boldsymbol{x}$, to a measurement vector. The sensor model calculates the outputs that the sensors would produce if the swarm were in the position which the motion model predicted. The sensor measurements consist of a 3D position coordinate for each drone, and four range measurements. The position coordinates can be matched directly to their corresponding state vector $\mathbf{s}_{\mathrm{n}}$. However the range measurements must be calculated using the multilateration equations. Equations 3.4 and 3.5 were used to find the measurement vector, $\boldsymbol{h}(\boldsymbol{x})$, from the state vector, $\boldsymbol{x}$. The subscripts of $\boldsymbol{x}$ and $\boldsymbol{h}(\boldsymbol{x})$ denote indexes in the vectors.

$$\boldsymbol{h}(\boldsymbol{x})_{[0:11]} = \boldsymbol{x}_{[0:11]} \tag{3.4}$$

$$\boldsymbol{h}(\boldsymbol{x})_{[12+n]} = \|\boldsymbol{s}_0 - \boldsymbol{s}_5\|_2 + \|\boldsymbol{s}_n - \boldsymbol{s}_5\|_2 \tag{3.5}$$

Once $\boldsymbol{h}(\boldsymbol{x})$ is found, it can be compared to the real measurement vector, $\boldsymbol{z}$, to find the residual. The Kalman Filter then estimates a new posterior state vector, $\boldsymbol{x}(t + \Delta t)$ based on the residual, and three covariance matrices: P, Q, and R. The target's position is extracted from the posterior state vector and transmitted to every drone so they can move to follow the target. The process repeats again using the posterior state vector as the prior estimate in the next iteration. This is a closed loop process consisting of prediction, conversion, comparison, then estimation.

3.2.4 Covariance Matrices

P is the covariance matrix for the prior estimate. It is a measure of the accuracy of the Kalman filters posterior estimate and it changes at each iteration. P is used in the failsafe algorithm to probabilistically determine whether the drones are too close to each other.

Q is the process noise matrix for the motion model. It is defined before the Kalman Filter starts and contains information on the accuracy of the motion model. It is notoriously difficult to find accurate values for this matrix particularly when the movement behaviour of the target is unknown. Therefore the constant velocity model was assumed to produce outputs with added discrete white noise [10] and a variance equal to that of the drone position measurements.

R is the covariance matrix for the sensor model. For this project it is computed before step 4. The GPS sensors on each drone output a covariance matrix containing information regarding the accuracy of their position measurements. These matrices are substituted into R to increase the accuracy of the filter. The covariance matrix for the radar ranges is unknown. However while testing the filter's robustness with added gaussian noise, the variance was known. Therefore the sensor noise values were substituted into the R matrix for these tests.

3.2.5 Error Sources

Sensor noise, temporal noise, processing delay, and dropout were considered as sources of error while designing the filter. It is impossible for any sensor to exactly measure a real-world phenomenon. All sensors including those on the drones have measurement noise, which the target tracking algorithm must deal with. For this project it is assumed that all the sensor's noise is normally distributed with a mean of zero. As the multilateration radar is not complete its noise levels are unknown therefore the tracking algorithm was designed to account for as much noise as possible. Its maximum noise limits were quantitively tested.

Temporal noise is introduced through the differences in clock timings between the drones. To calculate the position of the target, the time between the transmitter emitting a radar signal and the receivers detecting the signal must be found. Four time values will be measured for each radar pulse. These values will be converted to distances and used to calculate the position of the target through multilateration. To measure distances accurately the clocks on all five drones should be not differ by more than 10 nanoseconds. This value is exceptionally small therefore the problem of clock synchronisation is beyond the scope of this project. However it was considered when designing the filter as this type of temporal error manifests as error in the range measurements. By designing the filter to handle a large amount of sensor noise it can cope with more temporal noise.

Processing delay is introduced while the drones are wirelessly sending data, and processing information. Each receiver drone receives a signal from the transmitter. The drone then processes its data, then sends a range measurement back to the transmitter. The range measurement is processed at the transmitter to produce an estimate of the target's position. This position is sent to all the drones so they can move to surround the target. By the time the swarm has moved to the estimated target position, the target has already moved to a new position. This delay becomes greater the more computationally intense the tracking algorithm is. An unscented Kalman Filter is efficient compared to other non-linear filters.

The drones used in this project must communicate wirelessly and have a limited flight time and range. It is enviable that a dropout in communications between one or more of the drones will occur while they are flying. A temporary dropout could occur because of an obstruction between the drones, a pause in the software, or a drone running out of power. The tracking algorithm was designed to cope with temporary losses in signal from one or more of the drones. When a dropout occurs the tracking algorithm uses the Kalman Filter's constant velocity prediction. It combines the sensor measurement it does have with this prediction to yield a less accurate, but still valid estimate of the target's position. The maximum dropout time was found for several sensor noise levels.

3.3 State Machine

The drone swarm must behave in a predictable manner to avoid damage to property or others. We found that the drones often behaved unexpectedly during the initial flight tests. I spent the second half of the year reintegrating the communications, tracking, failsafe, and logging algorithms so that errors could be handled safely. The swarm must be robust and should continue tracking, as long as it can do so safely. If a critical error occurs, which it cannot recover from, the swarm must fail gracefully and in a predictable manner. A state machine was developed around the other components to control the setup and flight of the drones in an intuitive and predictable way. This is shown in Figure 3.3. One state machine runs on each drone in the swarm and each drone's current state is sent to all the other drones.

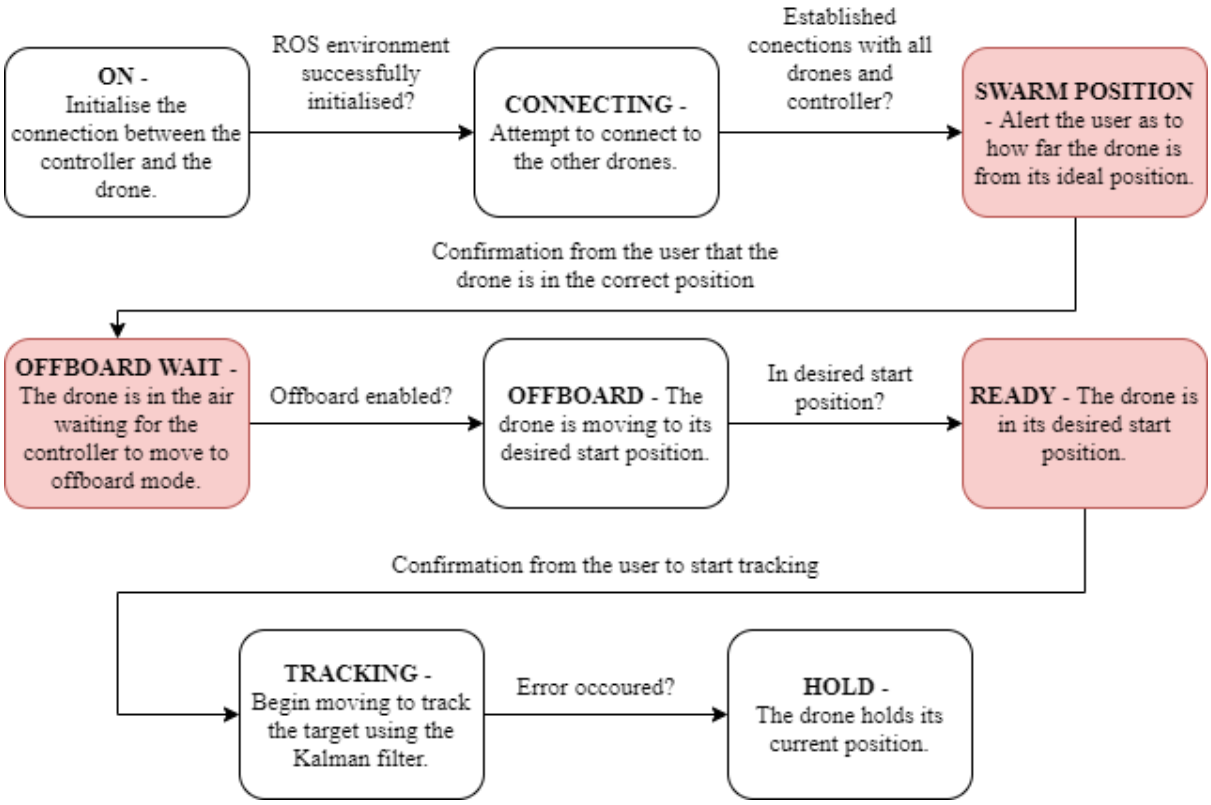


Figure 3.3: The state machine which is run on the drones. Sections where the user must interact with the drones are marked in red.

The state machine requires a user conformation at three points during the setup process. These points are shown in red in Figure 3.3. The first confirmation is in the swarm position state. In this state the drones are placed on the ground directly below where they intend to start in the air. Each drone alerts the user as to how far it is out of position so it can be moved to its correct position. Once the drones are in the correct position the user can confirm this and the drones will move to the next state.

The next user input occurs once a drone has been manually piloted to the correct altitude. While a drone is being flown to the correct altitude, it is in the offboard wait state. The drone moves to the offboard state once the pilot flicks the offboard switch on the controller. In the offboard state the tracking algorithm takes control of the drone and moves it to its intended start position. If the drone was placed accurately at its correct starting position on the ground it should not need to move to reach its start position.

The final user input occurs once all the drones are in their correct starting positions. The transmitter drone checks that all the drones are in the ready state. If this is true, the transmitter drone presents the user with the option to start tracking. If the user accepts this, by pressing enter in the terminal, the tracking loop will start, and the drones will begin to follow the target.

The three aforementioned inputs were added either before the drones move or to minimise the amount the drones move. It is inevitable that the drones will behave unexpectedly during testing. However if they are going to behave unexpectedly it is important that everyone around the drones know when it is going to occur. Having multiple user inputs before important events ensures that we know when the drones are going to move and can catch unexpected events.

3.4 Results

The system was tested under a variety of circumstances to determine its maximum operating parameters. Figures A, B and C show only a few of the hundreds of simulations which were run. Unfortunately the final tracking algorithm was only run once in a real flight test. This test was unsuccessful as the update frequency of the system was too high. Figure 3.4 shows the systems response to one of the full swarm simulation tests with a duration of 50 seconds. The standard deviation in the radar readings during this test was 0.5m. Only the path of the transmitter drone is shown. All four receiver drones were also simulated however they are not plotted for clarity.

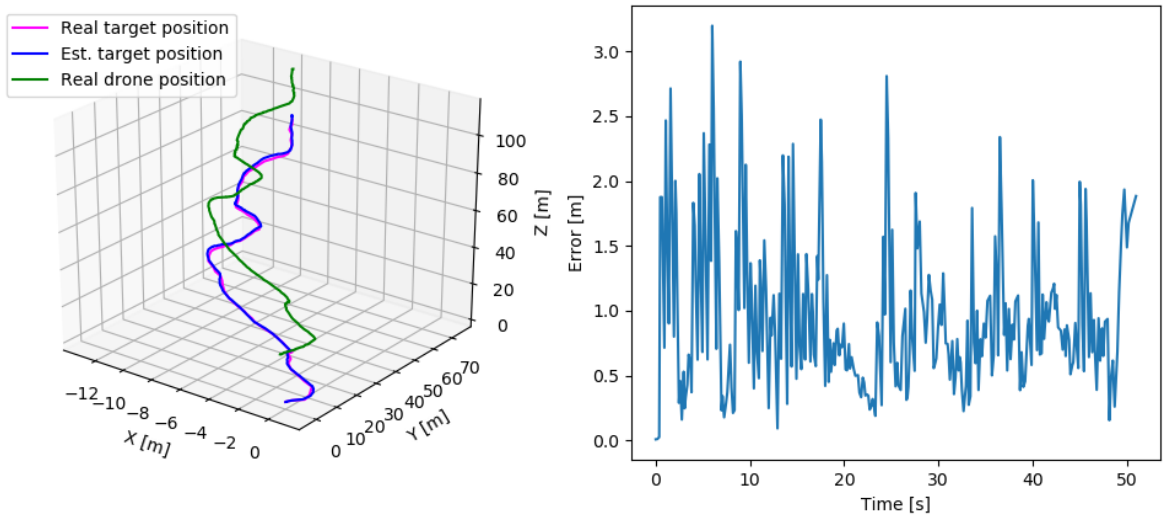


Figure 3.4: Left: A full swarm simulation using the tracking algorithm and a generated target path. Right: Error between the tracking algorithm output and the generated target path for the same full swarm simulation.

The tracking algorithm's response to a dropout was also tested. Figure 3.5 shows the 3D path of the swarm following a target. A five second dropout in range and GPS data was simulated from 10 to 15 seconds. Normally distributed noise was also added to the sensor inputs with a mean amplitude of 1m. maximum speed of the simulated target was 4m/s. The error in the system is shown in Figure 3.6. The constant parameters used for all the tests are shown in Table 3.1.

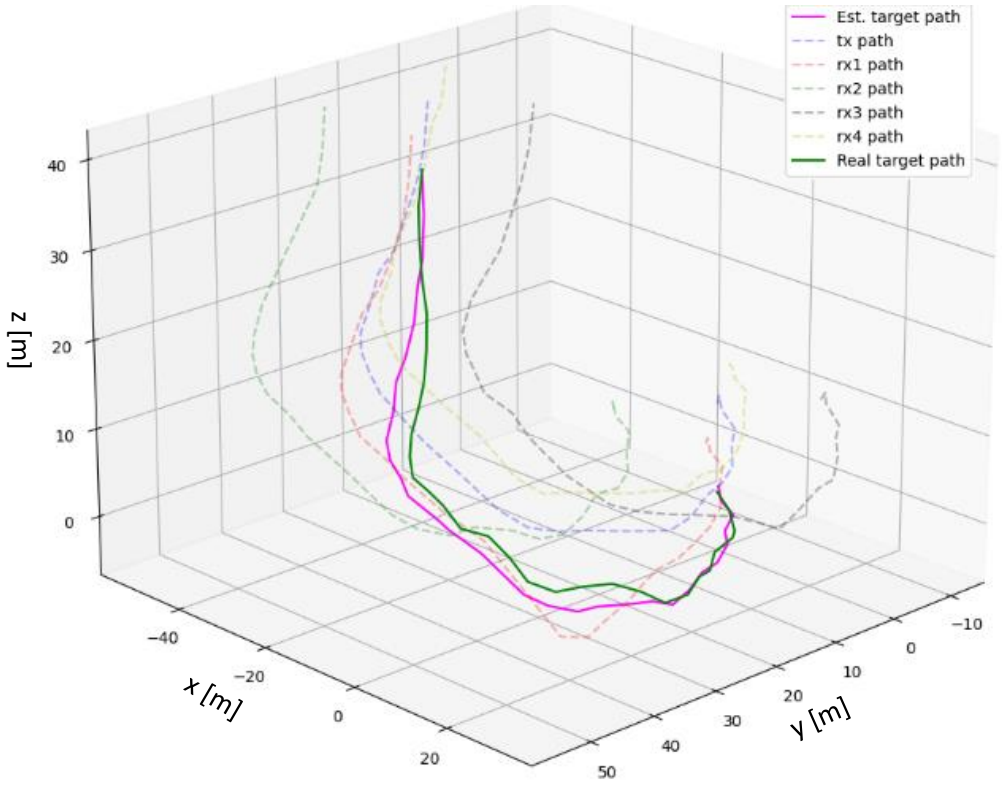


Figure 3.5: Comparison between the real and estimated path of the target with a 5s dropout in all receiver drones at 10s.

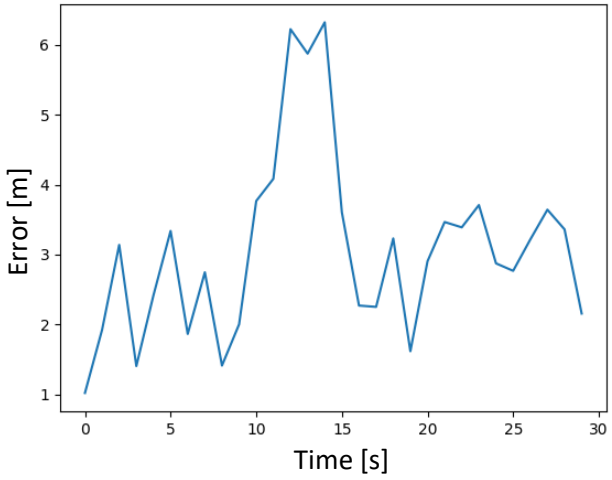


Figure 3.6: Absolute error between the real and estimated position of the target for the simulation run in Figure 3.5.

Table 3.1: Parameters used to find the limits of the tracking algorithm.

GPS Standard deviation	Drone distance	Receiver altitude	Transmitter altitude	Update period	Maximum target velocity	Maximum target acceleration
0.3m	20m	10m	15m	0.5s	5m/s	$5m/s^2$

3.5 Discussion and Recommendations

3.5.1 Tracking Algorithm

The maximum radar noise levels that the tracking algorithm can handle are determined by four factors: the speed of the target, the distance the drones are from each other, the GPS position accuracy, and the update rate of the system. These are shown in Table 3.1. The standard deviation in the GPS sensors was found through testing, as the modules return a covariance matrix with each position update. The drone distance, receiver altitude, and transmitter altitude were selected such that the drones were a safe distance from each other. They could not be too far from each other as the harmonic radar has a limited power and range. The maximum target velocity was chosen arbitrarily based on a jogging pace.

Figure 3.4 shows that the swarm can successfully track a target, in simulation, with a standard deviation in the radar noise of 0.5m. The average error in the target's estimated position during this test was 1.5m. While the system could occasionally handle errors of up to 2m, 1m was the maximum error which it could handle consistently. If the target was moving more slowly or the drones were further apart the maximum standard deviation would increase.

In Figure 3.5 the system's response to a dropout was tested. A five second dropout in range and GPS data from all four receiver drones was simulated from 10 to 15 seconds. Figure X shows an increase in error at this time however the swarm re-acquires the target, and the error reduces again once the dropout finishes. The standard deviation of the range measurements was 0.5m during this test. There are many dropout configurations which could be tested including dropouts from one, two or more drones. The swarm handled a 5 second dropout in the test shown in Figure X, however during repeated testing it could not always handle this scenario. Furthermore the dropout tests were performed on the Kalman Filter in isolation and the rest of the swarm was not simulated. Therefore further testing is required to determine the system's capacity to handle dropout.

All these tests were performed on a single PC running the full swarm simulation. In practice the performance of the system is reduced while it is running over multiple PCs. This may be due to the communication latency between them. This delay decreases the maximum noise levels which the system can handle significantly. This factor contributed to the unsuccessful test of the tracking system in a practical flight test. It is recommended that the tracking, failsafe, and logging algorithms be moved to separate threads to speed up the entire system and improve its accuracy.

The transmitter drone, which runs the tracking algorithm was not flown in a real flight test. Before flying all five drones the transmitter drone should be flown physically in a real flight test. As this drone runs the tracking algorithm it is the most important drone in the swarm. If it fails, the tracking fails. Flying this drone outside a simulation will add more latency which the system may not be capable of handling.

3.5.2 State Machine

The state machine was tested in a real flight test. It successfully ran on all five drones. We could easily place the drones in their correct positions on the ground using feedback from the swarm position state. Once the drones were switched to offboard mode they moved very little to reach their starting positions. During the test an error occurred in the tracking algorithm and the drones, simulated and real, moves to the hold state as intended. The state machine is not complete yet. Each time a user input is required the state machine pauses and waits for a response. This is because the state machine uses Python's input statement. Therefore none of the failsafe, logging, or communication algorithms are active while waiting for a user response. It is recommended that the controller is implemented to send user inputs to the drones, rather than using the input statement. This would eliminate all pauses from the system and allow it to run continuously.

4 Communications and Simulation – Nicholas Ranum

4.1 4.1 Introduction/Background

4.1.1 Introduction to My Contribution

My responsibility in this project was initially being involved with the development of the computer simulations used in this project as well as designing and improving both new and the existing communication frameworks used in the project. The use of the Gazebo [11] based simulation is a necessary tool for the testing and development of work completed by both individuals and the team as a whole. Producing a working simulation was my main focus in the beginning of the project as it would be required by multiple team members for verification of the multilateration, tracking, fail-safes and data logging aspects of the project. Working with the simulation also allowed me to gain an understanding of the project's communication frameworks and how improvements could be applied.

Throughout the project my involvement in test flights demonstrated developed new requirements for the project that where initially unseen by the group. This included upgrading the physical Wi-Fi connections used for communications between the operators and the drones themselves. Difficulties in our launch procedures also highlighted an unmet need for a more efficient and easier to operate method for controlling the program flow during test flights. This led to my involvement in the development of a flight controller with a graphical user interface.

4.1.2 Background to Simulation

The previous final year project was successful in producing a Gazebo based simulation that enabled for simultaneous control over five drones in an empty environment. The simulation environment was produced using the Gazebo simulator which is often used as the de facto choice for simulating environments that require the use of robot operating systems (ROS). The simulation shown in Figure 4.1 allows for the user to specify a number of GPS setpoint locations which will be broadcasted between the drones using the multilateration and swarm logic functions such that the drone swarm will follow the GPS setpoints as though it is tracking and insect like target.

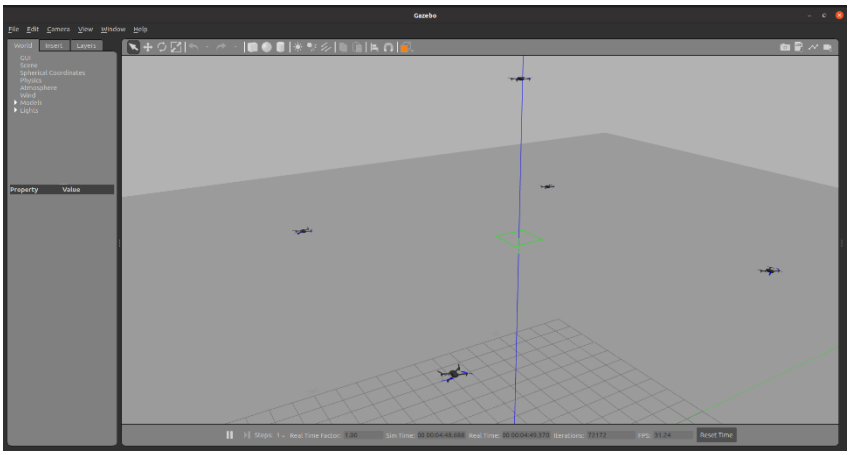


Figure 4.1. Gazebo Simulation Environment with Five Drones Flying

The Gazebo simulation software allows for integration with the Pihawk flight controller software PX4 [12] using the MAVLink communication protocol [13], which is the same communication protocol used to control the drone behaviour during practical test flights. This means that development of Python scripts

used to control drone behaviour during the practical test flights can be used in the simulation also. Performing test flights is a time-consuming process and always has the potential risk of damaging the drones and so simulation development is highly preferable. This ability to test our scripts in the simulation greatly reduces the development time for the project and allows us to test and verify the performance of our scripts in the simulation before using them in a real test flight.

The Python scripts that control the drone behaviour use the MAVROS package [14] to communicate to the Pixhawk flight controller. Within the Python scripts, calls to the MAVROS package are made using a TCP based publish/subscribe to communicate commands such as reaching a GPS location or switching the state of the drone form offboard to manual control. These commands are translated by the MAVROS API to the MAVLink protocol which are sent to the flight controller which are responsible for adjusting the physical drone such as increasing and decreasing speeds for each of the drone's copter motors. Figure 4.2 illustrates the flow of information between these components for both the simulation (dashed lines) and practical test flights (solid lines).

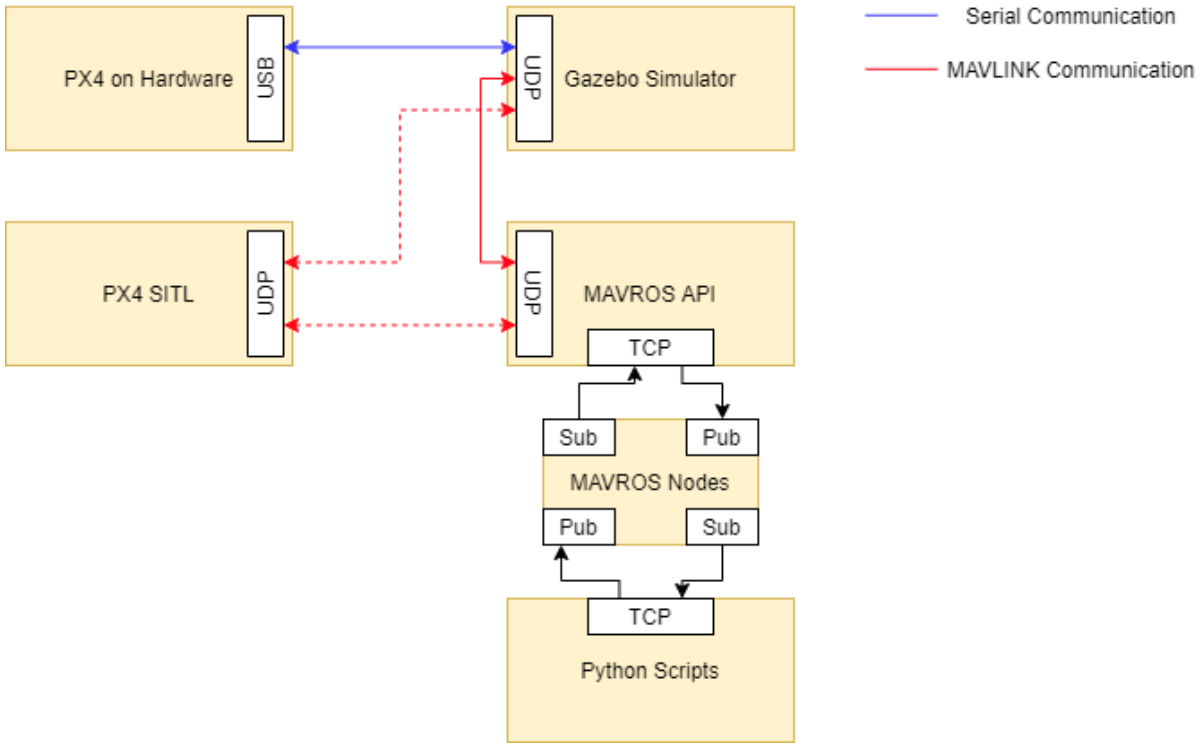


Figure 4.2. Simulation and Practical Flight Model

4.1.3 Background to Communications

To implement the communication of messages between each instance of the drones the ZeroMQ [15] library architecture was used. This library uses TCP socket connections to establish communication between two parties without the need for a central broker. The main communication pipes used in this communication was between the Tx and Rx drones which require communication to send target readings, Rx GPS locations, and new GPS setpoints for which the Rx drones should move to.

This ZeroMQ communication was implemented using a basic client/server model where the client sends a request and the server replies to the request (Figure 4.3). Using this architecture, the Tx drone will periodically request range readings from each of the Rx drones. Once the Rx drones reply to this message

they will perform a request to the Tx drones for a message containing their new GPS position they have to move to. When the Tx drone has received replies from each of the Rx drones the range readings are used in the multilateration and swarming logic to produce new GPS positions for each of the Rx drones. The Tx drone then replies individually to each of the Rx drones with their new positions completing the cycle of communication.

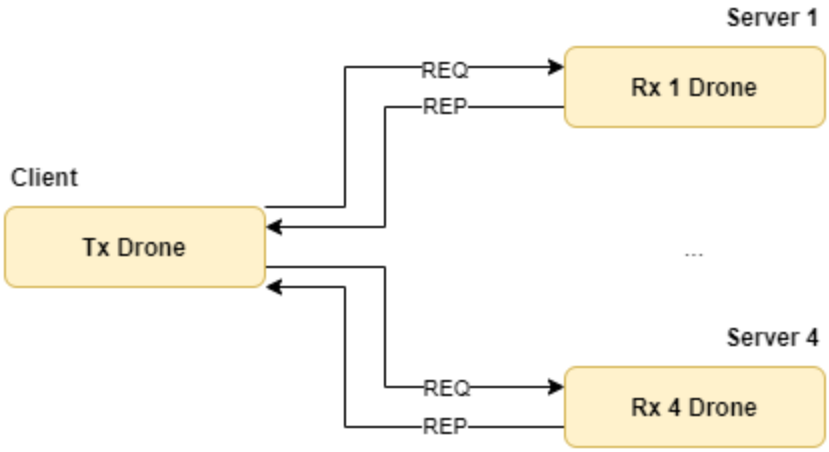


Figure 4.3. ZeroMQ Client/Server Model

The physical communication between the drones was done over a Wi-Fi hotspot, hosted by one of the drones and connected to by the remaining drones as well as the ground station. The Intel NUC's [16] on which the hotspot is created uses the onboard Wi-Fi which was emitted by 2.4 GHz Micro-Coaxial RF connector (Figure 4.4). Figure 4.5 illustrates how the Wi-Fi communications was set up for a practical flight test involving two flying drones and 3 simulated drones.

Figure 4.4 Micro-Coaxial RF Connector Wi-Fi Module

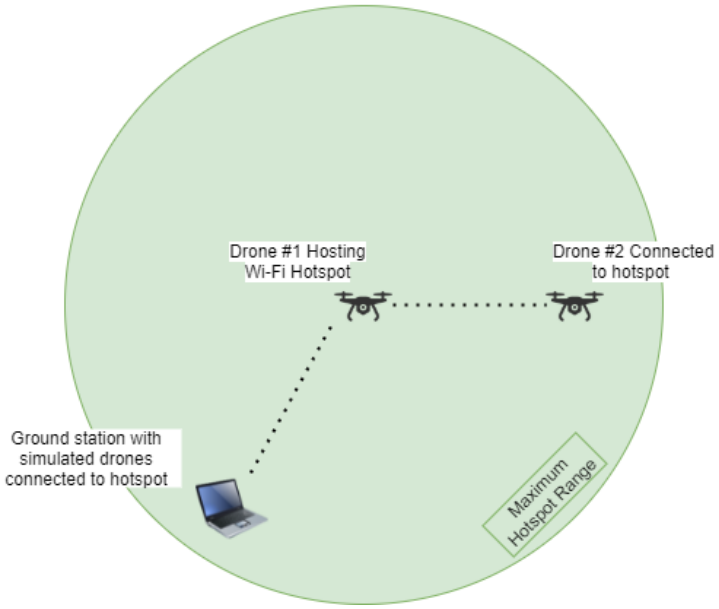


Figure 4.5 Wi-Fi Communication Setup for Practical Flights

4.2 Work Undertaken and Discussions

4.2.1 Simulation

The bulk of my initial work in this project involved setting up the simulation to replicate the results produced by the previous year's FYP. The first steps for doing so involved installing the Ubuntu 16.04 operating system and configuring the environment required to properly run the simulation. To do so, took longer than expected and much of the information described in Section 4.1.2 was gained directly by experimenting and attempting to run the simulation. Ultimately, we succeeded in running the simulation however both poor documentation and difficult to navigate program structures slowed us down.

One such issue that slowed down this development was having parameters redefined in multiple parts of the program. This resulted in us running the program with the correct parameter in one place, but not realising that it needed to be changed in another file buried deep in the repository. This led to the development of a centralised configuration file which contains all the constants which need to be correctly set when running the simulation on a new machine. The individual source files within the program reference this file rather then redefining the variables themselves.

Throughout this project and for future development of this project the simulation will continue to be a vital instrument for testing and verifying the validity of modules. The ability to run the same code used in practical test flights within the simulation allows for a reduced number of practical test flights needed, which will save a large amount of time. The simulation currently only uses a fraction of Gazebos capabilities. Future work on this project could include creating worlds with difficult to navigate terrain as well as developing obstacle avoidance within Gazebo [17] in conjunction with the computer vision functions Alex has developed. Going through the difficult process of setting up the simulation has motivated our group to maintain an organised repository with multiple readme files for each section of the repository. In addition, bash scripts written by Rowan will be a great aid in helping future teams quickly configure their machines with the necessary environment required to run the simulation.

4.2.2 Communications

Using the ZeroMQ library a number of improvements were made to the communication structure between the Rx and Tx drones.

A publisher/subscriber model was used to facilitate the Tx to Rx communication. This mode of communication allows the Tx drone to send publish a single message for which all the Rx drones can receive. The advantage of this model is that the Tx drone is unaware of the existence of the Rx drones and will publish messages with or without the Rx drones being on the other end to receive them. This is an important feature when considering the potential for drone dropouts. In a scenario where a Rx drone becomes temporarily disconnected from the network, the Tx drone will continue to publish messages. This way, the drones who have not dropped out are unaffected and can continue to receive messages from the Tx and track the target. If the drone who drops out reconnects in time to be in range of the transmitter drone, it will be able to re-subscribe to the topic and continue receiving updates. A push/pull model was used to facilitate the Rx to Tx communication. In this model, each Rx drone has a PUSH socket for sending updates and the Tx drone has a single socket for receiving updates from all the Rx drones. The ZeroMQ architecture when used in this mode is entirely asynchronous [18], which allows messages to be pushed to the Tx drone without the Tx drone being ready to receive the messages. This is an important feature because the Tx drone has a heavy computational load for performing the multilateration, tracking, and swarming logic checks. This asynchronous structure means that the Tx drone can receive the Rx messages when it is ready to, rather than waiting for the Rx drones to be ready. Figure 4.6 illustrates the two-way communication architecture developed between the Tx and Rx drones.

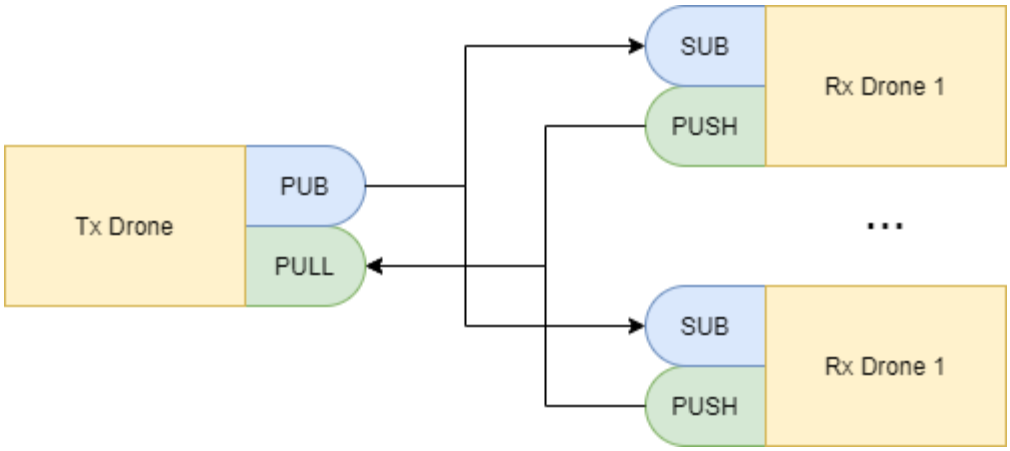


Figure 4.6 Tx-Rx Communication with ZeroMQ

During the practical test flights, we encountered issues with the strength of the signal emitted by the drone hosting the Wi-Fi hotspot. This primarily effected the communication link between the ground station hosting the simulated drones and the real drones. The poor signal caused drop outs in this link and resulted in the missed updates between the drones being simulated at the ground station and the real drones in the air. The issue causing these drop outs was the limited range of the NUCs onboard Wi-Fi antennas (Figure 4.4).

To improve the signal strength and robustness of the communication network the NUC's onboard Wi-Fi based hotspot was replaced with a Ubiquiti Rocket M2 [19] module equipped with two terminal mount dipole antennas [20] (Figure 4.7). The Rocket module has an maximum output power of 6.5W, over three times greater than the output power of the NUC's onboard Wi-Fi when using the Micro-Coaxial RF connector antennas. During testing, it was observed that the rocket module equipped with the dipole antennas was capable of producing a signal strength above -65 dBm for a distance up to 100m. This

increased range is sufficient for the test flights, and was noticeable with the team with us experiencing no further communication drop outs.

Figure 4.7. Ubiquiti Rocket M2 (left) and Terminal mount Dipole Antenna (Right)

The rocket module is mounted to the underside of each drone using a custom made mounting plate. The rocket modules are suited for this use in providing a flying ad hoc network as they take power over ethernet which can be supplied by the drones lithium-ion pack. To supply the power, wires 4, 5, 7, and 8 of a typical network cable [21] are connected to the power terminals provided from the battery pack.

To make use of the rocket's extended range, each drone connects to the rocket module using a standard RJ45 connector using the remaining pins on the network cable. This provides a LAN connection between the don and its equipped rocket module so that they can communicate over Ethernet. One drone is then selected to host the Wi-Fi hotspot and operate in router mode. The difference in this set up that the remaining drones will be connected over ethernet to their rocket modules operating in bridge mode. In bridge mode, the rocket will act as a device which connects to the hotspot and forwards all incoming traffic to its drone through the wired network interface. Figure 4.8 illustrates the new communication structure with the rocket modules.

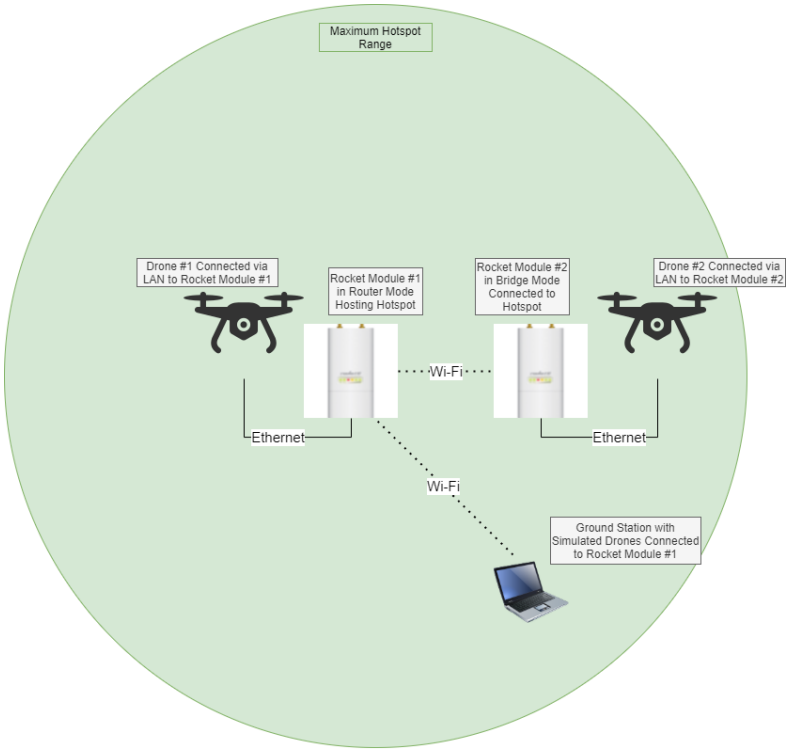


Figure 4.8 Practical Flight Communication Network with Rocket Modules

The current communication framework has proven to be successful through several test flights which involved no issues with sending messages between the Tx and Rx drones. The ZeroMQ architecture has the ability to expand for a greater number of drones through the flexibility of push/pull and pub/sub patterns. In addition, the ZeroMQ library is available in all commonly used programming languages which would enable communication between any future components to the project written in a different language then Python. Using the Ubiquiti Rocket modules involves a greater level set up and configuration but is necessary for completing successful practical test flights. Splitting the wiring of the network cable to provide both communication between the drone and the rocket modules may be a difficult procedure and using the project's partial technicians may be the best course of action for doing so. One drawback of using the rocket module during the development and testing stages is that they rely on the battery pack for power. The best suggestion for this it use the drone's onboard Wi-Fi hotspot for development and make use of the project's configuration files for switching easily between the two communication systems.

4.2.3 Controller

To perform the estimations of the target positioning using the multilateration and tracking algorithms the Tx drone requires a radar reading from each of the Rx drones. The multilateration algorithm then uses these readings to determine the most likely position of the target. To do so, the radar readings are required to be grouped, such that each of the readings where made by the Rx drones at the same time. This requires synchronisation between each of the Rx drones we implemented by attaching a sequence number to each of the update messages sent by the Rx drones to the Tx drones. The Tx drone would receive the updates, and group them by using the sequence numbers.

During the development of the system each of the drones where created and initialised from a single launch file which ran on one machine. This meant that all four Rx drones ran an almost identical copy of

the scripts determining their behaviour, and where running relative synchronised. During the test flights however, each of the real drones would run a separate instance of the launch file that initialised the drone. It was required that we would SSH into each drone and begin the launch file sequentially. This resulted in the programs for each of the real drones and the simulated drones being initialised at separate times. The result of this is that one drone would begin periodically producing target estimates and sending updates to the Tx drone, with each update containing an increasing sequence number. When every Rx drone had been initialised and begun sending periodic updates to the Tx drone, although the radar readings may have been taken at similar times could have differing sequence numbers. The Tx drone would then be unable to successfully group the radar readings and could not perform the multilateration or tracking, meaning the whole system would be unable to track the target.

To overcome this effect of running the system over multiple machines, we decided to incorporate synchronisation through the use of a centralised controller. The controller would establish communication with each of the Rx drones, and would send periodic updates to each Rx drone with its own sequence number appended. When the Rx drone receives this controller update it is triggered to append their radar readings and pass the update upstream to the Tx drone. The result of using the controller to trigger the updates is that the sequence numbers for each of the Rx updates are entirely dependent on a single source – the controller. If one Rx drone was initialised later than the other drones, it would not cause a mismatch in sequence numbers as it would simply use the sequence number provided by the controller. This resulted in the Tx drone always being able to group the radar readings successfully.

This flow of information was implemented using the ZeroMQ Push/Pull architecture which allows for the best implementation of a producer-consumer-collector model (Figure 4.9). In this model, the controller (producer) can distribute an identical controller update to each of the Rx drones (consumers) over a single connection. The producer sends the update messages in a round-robin fashion using a separate output queue for each of the connected consumers. Each of the Rx drones (consumers) then send their modified controller update message to the Tx drone (collector), with all the consumers sharing a single connection on which the Tx drone receives messages. This architecture has the advantage of only requiring a single connection to be created for each stage of the process and has the ability to easily scale up and down the number of consumers, which is useful for testing the application with a single consumer for simplification and may be necessary if the drone swarm is to expand to a larger number of drones.

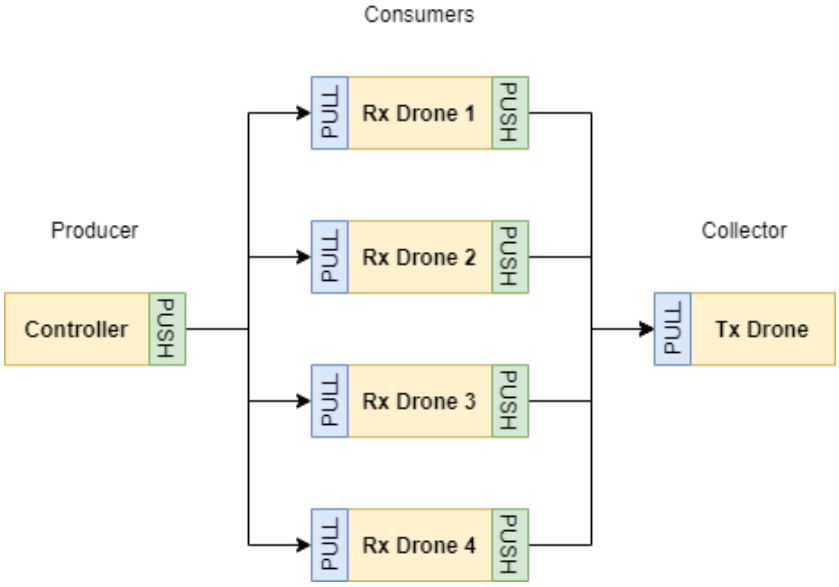


Figure 4.9 Controller-Rx-Tx ZeroMQ Push/Pull Architecture

The Practical test flights also revealed issues with our launching procedures being too complicated and difficult to use. For each real drone that was being used in the test flight, we were required to SSH into the drone and launch two files over the SSH terminal. For an example, if we were performing a test flight with two real drones and four simulated drones, two terminals were required for each real drone as well as two terminals for the simulated drones. This resulted in the laptop we used as the ground station being overwhelmed with open terminals that not only required user input to navigate through the launch procedures but also required careful monitoring. Figure 2.6 shows the ground station screen during this type of test flight. This method was prone to failure from incorrect input caused by confusion as well as missing output messages indicating warnings and failures in the drone states.

To ease the process of performing practical test flights, I developed a graphical user interface (GUI) that assists in the launch procedures (Figure 4.10). The GUI interfaces with the controller module by including state information in the controller update messages which are sent in the controller-Rx-Tx communication pipeline. These state information packets navigate each of the drones through the state machine model developed by Rowan. The state of each drone is updated by pressing buttons in the GUI which are included with prompts indicating actions required by the user. This removes the need for the user to provide inputs to the terminals. The GUI also uses a ZeroMQ pub/sub model to receive information from both the Rx and Tx drones indicating information vital to the user such as GPS positions and possible error messages. This removes the need for the user to monitor multiple terminals at once as the relevant information is all collected in the GUI.

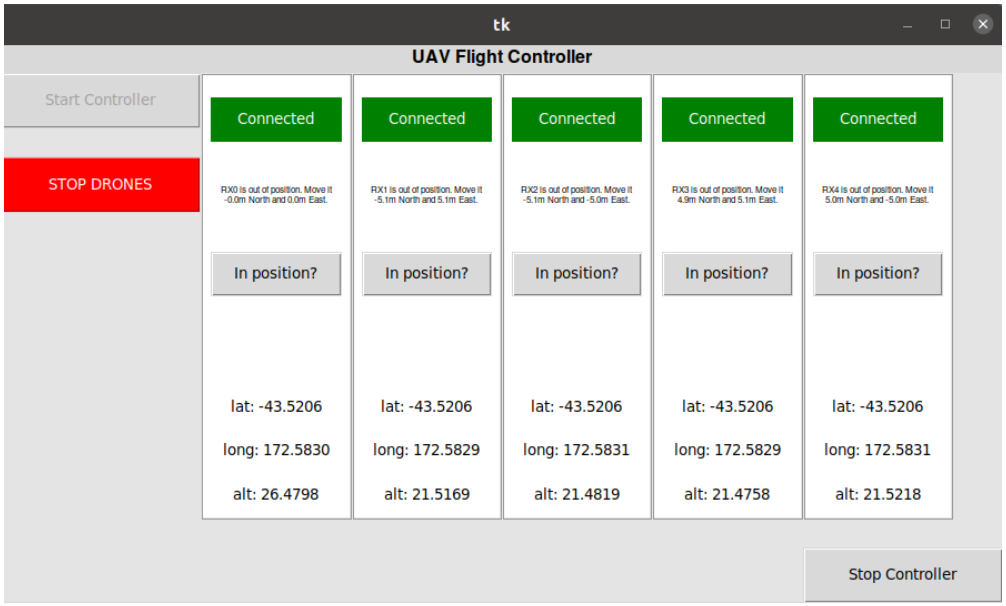


Figure 4.10 Controller Based GUI

The development of the GUI was done in TKinter [22], which provided all the necessary implementation while being suitably simple for development. The GUI is initialised by the controller class and runs in a separate thread so that the GUI can be interactive and respond to the user without blocking vital controller functions such as sending updates and receiving drone positions.

The use of the controller module as a centralised location for initiating the flow of information between the Rx and Tx drones is a viable option for fixing the issue with mismatching sequence numbers being received at the Tx drone. However, the controller is currently being run and operated at the ground station which in a final product will not be a part of the system. With the idea of the drone swarm tracking a

moving insect, the concept of using a centralised platform may not work in this system. Future investigation into options for synchronisation should be explored such as including timestamps taken from GPS readings or using the Networking Time Protocol [23].

As part of the GUI development, Oli was responsible for creating a separate GUI that automated the process of performing the SSH commands and script launching commands that begin the first stages of the practical flight launch procedure. These GUIs are currently independent, and for future development of this project it would be preferable to combine the two. By using Tkinter for both of the GUI designs the process of combining the two should be straightforward by using Tkinter 'frames' to create containers for each part of the GUI to go in. In addition, the GUI has not been tested in a successful practical flight scenario but rather tested by replicating the process of a practical flight without having the drones in the air. Although the process of this test was identical to that of a real test flight, unexpected issues may occur that will require further refining of this part of the project.

4.3 Summary

Throughout this project the simulation environment was used thoroughly for the testing and development of individual components of the project. The repository maintained by our group has been designed to be intuitive to use and easy to replicate by restructuring the way that the program runs and creating clear and informative documentation. It is our goal to make the process of replicating the work done by this group as well as performing modifications and improvements as easy as possible. The current communication architecture for this project works well using the ZeroMQ architecture and is designed to have the ability to add or remove drones with minimal alteration. Using the Ubiquiti Rocket modules as a replacement for the NUC's onboard Wi-Fi hotspot was a necessary change and enabled us to perform practical flights without facing issues caused by poor signal performance. The controller module is a viable solution for creating a centralised platform to synchronise the communication flow and grouping of radar readings at the Tx drone. With the graphical user interface being used in conjunction with the controller, the process of performing test flights will be streamlined in the future.

5 Data logging and visualization – Connor O'Reilly

5.1 Introduction and Background

5.1.1 Introduction – Responsibility

A task I was originally assigned was implementation and optimization of advanced control and maintenance of the drone swarm arrangement algorithms. After analysis of code and careful considerations of team progress, I realised this would slow the progress of each other team member. I then began development on a data logger and visualisation tool to improve flight analysis.

5.1.2 Background – reasoning, research, and problem formulation

A crucial requirement of object tracking is collation of data. In particular, flight performance measurements to aid in proof of concept. All data relevant to flight performance metrics are dynamic memory variables part of the robotic operating system software. The drone machine should log as at most as frequent as the software operation; and at least as frequent as the expected rate the data changes. This is because it indicates the object location, and how the swarm operates in response to object movement. A logger of flight data relevant to flight path and progress of each module designed and investigated is required upon request of academic supervisor, as discussed in later sections.

A problem with the logged data, established in previous years work, is there are difficulties with interpretation of the data. Natural operation of a real-time multiple robot operating system includes interrupts and flight controller protocols. These vary in length of time, and occur at unknown rates for each machine as these depend on the surrounding environment as of .As each drone operates in different microcosms, the interrupts and flight controller protocols occur are different in amount of time for each machine. Therefore, the drones become desynchronised and hence the difficulty in interpretation of logged data. A visualisation tool and elimination of desynchronisation is required to improve flight analysis and investigation.

There are many solutions to the data logging problem, data visualisation tool, and elimination of desynchronisation. The solutions are compared as one solution may have greater benefit over another solution. For example, it would be dangerous to implement synchronisation software within an autonomously operating drone swarm whilst tracking a fast-moving insect. This was done to ensure the best logger and best data visualisation tools were implemented in the most sustainable way possible.

The UAV is an asset to the WRC. The centralised mothership is very large in comparison to all four slave drones. The drones are indispensable and have many modules. All hardware components are buyable online, and are critical to drone swarm formation, data logging, and data visualisation. The UAV hardware components which are essential for these processes are:

- A flight-controller, PX4 firmware installed onto Pixhawk-series flight-controller boards that enable autonomous flight control as part of
- An Intel NUC Kit with 8th Generation Intel Core Processors. The Intel NUC can set up WiFi hotpots for all UAVs to communicate, also executing essential software and therefore UAV instructions aside from that of the flight controller as with .
- A high-performance GPS module with proficient accuracy as specified in . Having an accuracy of 1 micro-second, and very fast acquisition rates.
- ZeroMQ (ZMQ), enables fast communications. Forming an integral component the communications system. ZMQ provides functional operation within the multiple UAV swarming system sending packets from mothership to slave in one burst as with .

5.2 Overview

5.2.1 Overview – Data logging

Data logging is crucial for drone swarm performance analysis. A data logger collates a set of memory variables stored on the machine, where all memory variables logged are important for flight operation. As such, the data logger is required to operate within the real time robotic operating system and capture all relevant data of flight test performance. The data is required for analysis of system response to inputs; collating data on flight performance, and having data on the path an object took enables a qualitative insight. Without data logging, it is very difficult to determine whether or not software performs as intended. Performance metrics are measured at the same time, and include the analysis of:

1. GPS coordinates of both current location, heading location, and height.
2. Failsafe activation and causes.
3. Multilateration accuracy and performance.
4. Communication system efficacy.
5. Kalman filter effects and desirable performance.

There are many other metrics that can be recorded, however the other metrics were not measured because the measurements were overcomplicated and out-of-scope. For instance, the battery voltage can be logged and may be indicative of a failsafe activation and thus deactivation of the drone swarm; however, these were measured with a handheld device regularly during the occasional test flight.

5.2.2 Overview – Data visualization

There are significant issues with interpreting logged data as there are countless lines of logs for each drone. Analysis of drone swarm maintenance requires interpretation of more than one drone log file at once. Furthermore, each drone logs asynchronously/independently from other drones, therefore, a set of data on a particular line of the log does not match a set of data on the same line for another drone. Data visualisation tools are required as it allows for easy interpretation of the data collected from a flight, and enables further data processing which ultimately provides a great set of pros and cons especially for eliminating desynchronisation. Reconstruction of data onto a 3-dimensional figure allows an easy comparison of heading GPS coordinates with current GPS coordinates.

5.3 Planning

5.3.1 Problem Formulation and Solution – Data Logging

5.3.1.1 Planning – Original Solution – ROSLOGS

Robot Operating System (ROS) alongside the PX4 Autopilot flight control software are a set of software frameworks. These have a set of flexible tools related to data logging ; however, the logging provided by ROS is inadequate. ROS logging occurs infrequently (inconsistently) and only contains coordinates. The inconsistent logs and lack of performance metrics limits analysis of flight performance, swarm operation, and modelling of the flight. The lack of data restricted analysis of each flight to hypotheticals, so a better logging strategy was required.

5.3.1.2 Planning – New Solutions

A set of three solutions are delivered below. These solutions address the data logging problem defined in section 5.2.1 and 5.3.1.1. All solutions have a set of pros and cons and the ultimate solution is chosen for

the beneficial set of advantages and disadvantages it has in contrast to other solutions. The three alternative solutions to the ROS logging are:

- A headquarters (HQ) computer that pulls data off the ZMQ communications framework.
- The transmitter drone pulling all data off the ZMQ communications framework and logging everything.
- Having the respective drone log its own data.

5.3.1.2.1 New Solution – HQ logs data from ZMQ

The data pulled from the ZMQ communications framework would be logged on the HQ computer. This solution is easiest and operates outside the swarm. The entire CPU load of logging data is not part of swarm operation which improves the drone software performance. All the data of a drone can be logged if and only if the data is pushed onto to the ZMQ communications framework. The problem is there is an increase in transmission overhead, as more data must get pushed onto the ZMQ communications framework for it to be pulled and logged. Furthermore, as the drones operate over a short-range WiFi hotspot, the drones have to operate close to HQ in order for data to be pulled and logged. This does not resemble a drone swarm tracking an insect traversing through a forest.

5.3.1.2.2 New Solution – Transmitter logs everything

The transmitter drone pulls the data off the ZMQ communications framework and logs all information of each drone. The transmitter drone can log all relevant information to itself, but only the information the receiver drones push onto the ZMQ communications framework. As such, all information of drone performance are transmitted from each receiver drone, increasing the transmission overhead. The problem is that the transmitter drone has an additional CPU load of 5 loggers which is a very bad disadvantage because the transmitter drone already has highest CPU loading.

5.3.1.2.3 New Solution – Respective drone logs everything

After the drone software uses the flight context stored as memory variables in multiple data classes, the data is logged into the log file stored locally on the machine. This solution is most difficult to implement as it operates on all drones. Similar to the transmitter logging everything, all drones can operate far from HQ. The CPU load required for the logging operation is distributed throughout all the drones, and the logging is not limited to what is transmitted.

5.3.2 Problem Formulation and Solution – Data Visualisation

5.3.2.1 Asynchronous Logs / Desynchronisation

As discussed in later sections, profiling was performed and estimates the software operates at over 300 Hz and depends on the number of interrupts that occurs in each cycle. An unknown amount of system interrupts occurs as a result of normal operation of pix-hawk flight-controller and Intel NUC. As such, each drone logs at different rates resulting in asynchronous logging – termed desynchronisation. Desynchronisation occurs and becomes a problem because data does not match; for example, the time stamp on the 150th line of a drone's log file does not match the time stamp on the 150th line of another drone's log file. As such, further software is required to remove the extent of desynchronisation to improve the interpretation of logs and thus interpretation of flight performance. The process of synchronisation was the first problem encountered in the data logging and data visualisation task, and is found as highly relevant for interpretation of swarm data logs.

5.3.2.2 Requirements

The requirements that enable improved analysis of drone swarm flight logs and comparison between heading and current coordinates of all drones are:

1. Maximal reduction of desynchronisation of logs between each machine.
2. Capture concerning major deviations of measurements, yet do not trigger a premature ending of flight.

After implementation of data visualisation, further requirements were established such that:

- Drones are unaffected by synchronisation processes matching data.
- Align completely with the data logging principles established in the introduction.

5.3.2.3 *Solutions*

5.3.2.3.1 Solution – Synchronisation of drones and therefore data

A solution of maximally reducing desynchronisation is incorporating synchronisation software into drone operation. The data would log at the beginning of each cycle, and each drone would pause operation until a confirmatory signal is received. The software of all drones (while moving to a set of coordinates) would pause and wait until each drone receives the next set of coordinates. Upon receival of new coordinates (the confirmatory signal), the drones would log the coordinates and then proceed with operation. This solution reduces the smoothness of the swarm operation as any delay of one machine impacts the operation of other machines. This is a bad problem for an autonomous drone swarm, and thus not a viable solution. This solution achieves synchronisation, however severely disables drone operation.

5.3.2.3.2 Solution – Synchronisation of data on the drone

Another solution is similar to the solution discussed in section 5.3.2.3.1. This would incorporate synchronisation software into the drone software, but without incorporating a confirmatory signal disabling the operation of a functional drone swarm. Instead, the logger performs the synchronisation process and the drone software would operate the same as it would without the synchronisation process; all data would be logged at the end of a software cycle. The difference is the data is logged at the beginning of each 10^{th} of a second; thereby satisfying the synchronisation of data and capturing major deviations of concern – fulfilling the requirements stated in section 5.3.1.2. However, there are large losses of data in between each 10^{th} of a second – failing to meet the requirements of data logging. Therefore, proving as a non-viable solution to synchronisation and data logging.

5.3.2.3.3 Solution – Synchronisation of data in the data *visualisation software*

The third solution operates independently from the drone swarm. Application of data synchronisation in the data visualisation allows abundant collection of data, a requirement of the data logging software; and maximal reduction of desynchronisation, without affecting the drone swarm operation. These fulfill the requirements listed in 5.3.2.2; outlined with reasoning in prior sections. Any major deviations of measurements are captured by setting the time difference of timestamps between consecutive logs. If more detail is required, the difference in timestamps can be reduced; regardless of difference in consecutive time stamps, desynchronisation is virtually eliminated because the likes of a difference equal to 0.025s (between plot logs) is indiscernible lag. As with previous solutions, every 10^{th} or 40^{th} of a second is capable of capturing any major deviations of coordinates that do not trigger a premature ending of a flight – typically a result of corrupt GPS data received by the GPS module from a satellite.

5.4 Operation

5.4.1 Operation – Data Logging

Overall, the solution discussed in section 5.3.1.2.3 where data is logged on the respective drone is highly advantageous. Despite this solution being the hardest to implement, the CPU load is distributed across all drones and the swarm can operate far from base; proving highly beneficial and reflective of the proof of concept. However, as discussed in section 5.3.1, the same problem exists in each solution: the log files have a large number of lines which have time stamps that do not match the time stamps of other drone log files.

5.4.1.1 *Operation – Data Logging – Software*

The data logger is implemented within the real time operating system (RTOS) of the drone swarm. The data is accurately captured, and written to a log file. The data captured includes:

- A time stamp of the time when the data enters the drone.
- The state of each drone.
- The mode of each drone.
- Heading and current coordinates.
- Flight performance metrics established in 5.2.1.

The operation that includes only the data logging process follows the block diagram below as of figure 5.1:

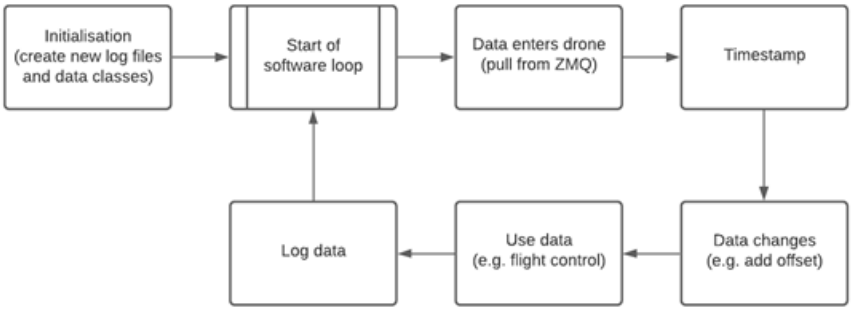


Fig 5.1 - Data Logging Block Diagram Overview

Upon running the drone swarm software, a new log file is created. The drones put the data on the ZMQ communications framework and all other drones grab the data, then afterwards immediately notes the time. Any changes of swarm context is applied, such as setting an offset. The drones then use the data, such as passing into the flight controller. The logger will then log data upon change of relevant data. After the logger completes the final instruction, the process repeats. The block diagram represents the operation the data logger performs, this process allows the drones to use the data before logging it. The drone software generates a highly accurate time stamp (based on Unix epoch time) once the drone receives new data. The logger saves a record of all data relevant to flight operation. The time stamp is made with a function call to the timer library; and the time stamp is highly accurate because the timer module uses the GPS to accurately determine the time.

5.4.2 Operation – Data Visualisation

The operation of the data logger is to store data locally on the machine. After a flight is performed, all the data is taken off each of the machines and loaded onto a computer for analysis. The logged data is

processed to remove desynchronisation. Any data which corresponds with the beginning of each 10th (or 40th) of a second is extracted and appended to a new array. The heading and current GPS coordinates extracted are plot. The extracted data is processed again, this is done to place markers at each second and 10th (or 40th) of a second; illustrating a set of heading GPS coordinates plot in close proximity to the current GPS coordinates. The markers allow easy comparison of plot data for each drone and thus enhancing analysis of flight context. The time between each consecutive data log appended to an array is adjustable to increase the quantity of data that is plot as there are between 30-40 logs each millisecond; for example, selecting data corresponding to the beginning of each 50th of a second increases the quantity of data that is plot. This virtually eliminates desynchronisation while extracting all data (including changes) as discussed later in section 5.6.1.

5.4.2.1 Operation – Data Visualisation – Software

The data visualisation process is illustrated as a block diagram in figure 5.2 below. All log files are loaded into the software, incremental variables are initialized, and all data classes are instantiated. The first drone log file is loaded, and the first line of data is stored, stripped, ultimately performing data splitting into separate memory variables. The time stamp is loaded and passed through a conditional: if the time stamp is the first time stamp encountered in the millisecond, further processing occurs. The data is extracted and appended to an array of the new data class; otherwise, the data is not appended. Assuming the end of the file is not reached, indicated by a string containing 'STOP', the process of extracting data will continue. Once the string containing 'STOP' is read, the file is at the end and the data of the next drone is loaded. The process proceeds until there are no more drone log files. Once all drone log files have the data completely extracted, the nested data class for each drone is plot which represents a 3-dimensional reconstruction of the data collected from a flight.

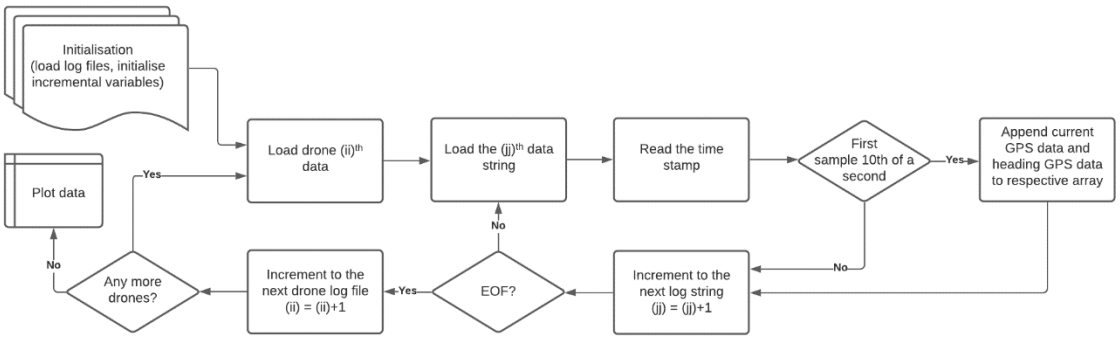


Figure 5.2 - Data Visualisation and Desynchronisation Block Diagram Overview

5.5 Results

5.5.1 Results – Data Logging and Profiling

A sample of the logs taken is below. All drones have loggers that collate the data contents as mentioned in 5.4.1.1.

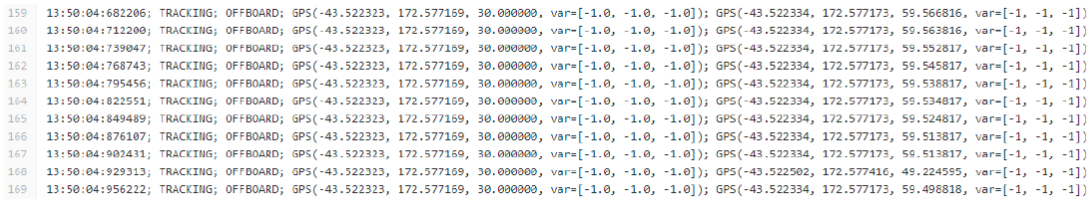


Fig 5.3 - Drone Data Log Snippet

The columns from left to right represent the data: time stamp, state, mode, heading GPS coordinate, and current GPS coordinates. The transmitter drone captures additional context, including measurements of the Kalman Filter efficacy, multilateration accuracy, and a states list of all drones measuring the activation of fail safes or not. These ultimately represent the operation of the swarm.

In the example provided, figure 5.3 above, the data logger is specified to log data upon receival of new GPS current coordinates rather than every cycle of software. This effectively shows the GPS data acquisition rates; how fast new GPS current coordinates are received as a result of change of location. Comparatively, the Intel NUC operates much faster than the GPS coordinate acquisition rates. This ultimately proves the software operating at 300 Hz, but has new data arrive at 30-40 Hz, can log sufficient amounts of data. Figure 5.4.1 and figure 5.4.2 below present data collected by two different drones of the swarm at 1PM on the same day. 5 lines of code are selected and the figures only show time, as this illustrates the nature of desynchronisation within a drone swarm.

Figure 5.4.1 - Drone 1 Stamps

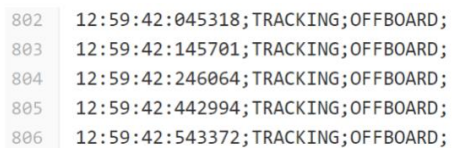


Figure 5.4.2 - Drone 2 Stamps

5 lines of logs are selected, and the logs are taken at roughly the same time. There are two problems these figures present. The first problem is the line number does not match, despite the two data sets operating at the same time. The second problem is the time stamp do not match, even though the logs span 5 lines. These prove the need for requirements stated in 5.3.2.2, as these aim to improve the analysis of data. These illustrate the need for data visualisation software, selecting and allocating data based on the time stamp. Logging data at rates faster than observed in figure 5.4.1 and figure 5.4.2 will allow for desynchronisation as the GPS current coordinates stored as memory variables do not change while waiting for data acquisition.

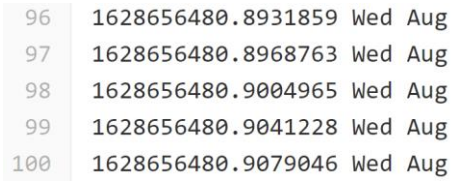


Figure 5.4.3 - Drone Profile

Profiling data proves the system to operate at 300 Hz as observable in figure 5.4.3 above. There is 0.0035 seconds between each log corresponding to 300 Hz. A comparatively much faster rate than the change of coordinates (suggesting rate of transmission success).

5.5.2 Results – Data Visualisation

The data logged from a successful test flight is reconstructed as a 3-dimensional figure below. The plot contents are the data of two drones without any markers. The current and heading GPS coordinates are plot from an early version of the data visualisation software. Figure 5.5 below presents data on a drone swarm which maintains swarm formation.

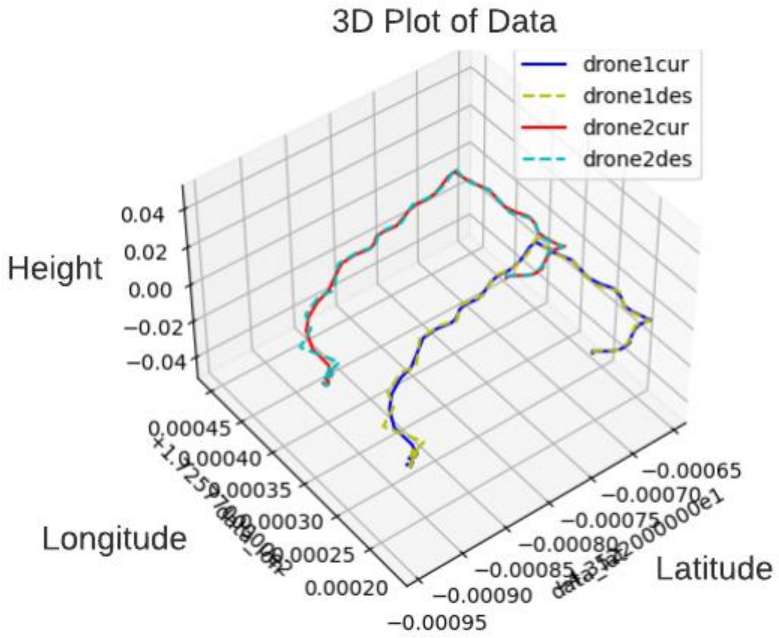


Fig 5.5 - Reconstructed Successful Test Flight With Two Drones

The second plot, figure 5.6 below, contains a close up of one plot of a successful test flight. The data logged of current (blue line) and heading (yellow line) GPS coordinates are plot. Markers represent the data selected and allocated to attain a reference. A cross marker represents the data at the beginning of each 10th (or 40th) of a second for the current (blue cross) and heading (green cross) GPS coordinates. A circular marker represents the 10th (or 40th) piece of data extracted from the logs for the current (orange) and heading (red) GPS coordinates; this represents the beginning of a new second. An error in heading coordinates is identified by a large deviation of coordinates, illustrating proof of capturing major deviation of concern that does not trigger a premature ending of flight – most likely the result of GPS hardware malfunction, or communications error between GPS and satellites.

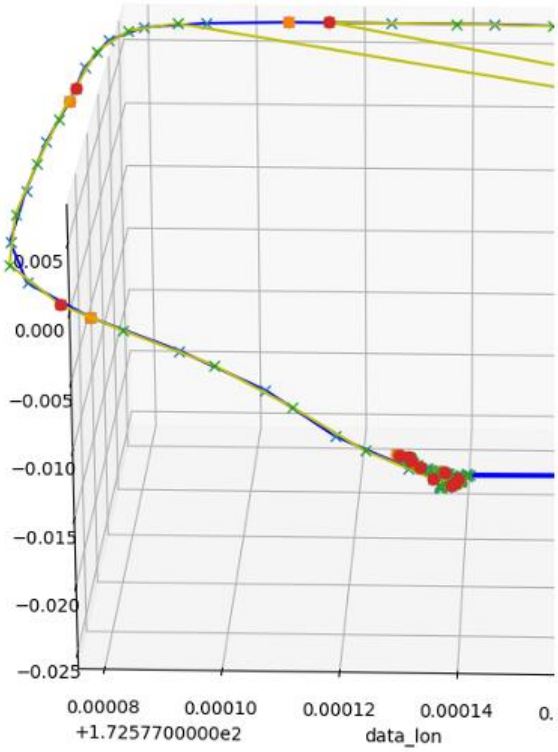


Figure 5.6 - Insight into Operation of Removal of Desynchronisation

5.6 Outcome / Summary

5.6.1 Outcome – Data Logging

Data logging has proven an effective tool in the aid of analysis of flight performance. The logger fulfils the intended operation as discussed in section 5.4.1. Profiling had proven the software to run beyond 300 Hz, and data acquisition rates are proven to arrive at 30Hz; overall, logging substantial data as part of the requirements, and capturing adequate amounts of data required for analysis of flight. The vast amounts of data has proven the need for data extraction and allocation to a time for elimination of desynchronisation between machines; enhancing analysis of data gathered from a flight as stated as the requirements in section 5.2.1. The logger accurately collates data relevant to flight performance such as drone coordinates the swarm relies on, and flight performance metrics including others progress. The logger is accurate because all data stored as memory variables are passed into the logger; therefore, any inaccuracy is a result of hardware malfunction. Without the data logging and data visualisation software, investigating the possible causes (and time of cause) of premature end of flight is very difficult.

5.6.2 Outcome – Data Visualisation

The programmed data visualisation software has proven an effective method of reconstructing the flight on a 3-dimensional plot. Data extracted from a successful test flight accurately represents the path taken from one location to another. The data visualisation software accurately depicts the observation made on the Ilam field test flight. Furthermore, the data visualisation software has proven effective is eliminating the effects of desynchronisation; however, there are improvements to improve the analytical assessment of the 3-dimensional reconstruction of collated data. Overall, data the visualisation tool currently selects data corresponding to the beginning of every 10^{th} of a second; a change of this time interval can be adjusted to operate selecting data corresponding to above 30 Hz which ensures all changes of data are captured. The data visualisation results section does not show virtually synchronised data; however, both

plots in conjunction illude to achievement of this because a single plot with multiple drones and markers does not illustrate the concepts in enough detail.

5.7 Improvements

The data logger and data visualisation software designed have the greatest set of advantages and disadvantages in comparison to other solutions explained in section 5.3. The analytical approach taken to first learn about the data classes and hardware, design a solution, then integrate into a real time operating system enabled a successful design in conjunction with others progress.

Improvements to the data logger depend on future installations and operation of drones. For instance, to investigate the cause of a failsafe breaking constellatory maintenance during tracking of an insect in the wild; logs of battery voltage should be taken. However, as a proof of concept sending the drones up and bringing them back down during a test flight, logging of battery voltage is not required.

Improvements to the data visualisation software is broken into two: first is the extraction of data, and second is the plotting of data. Firstly, the data extracted from a drone's flight log file should be assigned to a common reference between each drone; for example, if data is extracted is at the beginning of every 50^{th} of a second, the data could be assigned to the closest 0.025 seconds. This will allow for an easier identification of desynchronisation. Secondly, the data marks that are plotted to represent every 1 second could have a number or time on the plot. This would allow greater insight into the reconstructed swarm operation. This would reduce the time to identify an error in the flight pathing, and then investigation of the log files for more detailed interpretation. An out-of-scope improvement would be to incorporate interpolation of data between data sets through using measurements taken by the accelerometer and barometer sensors.

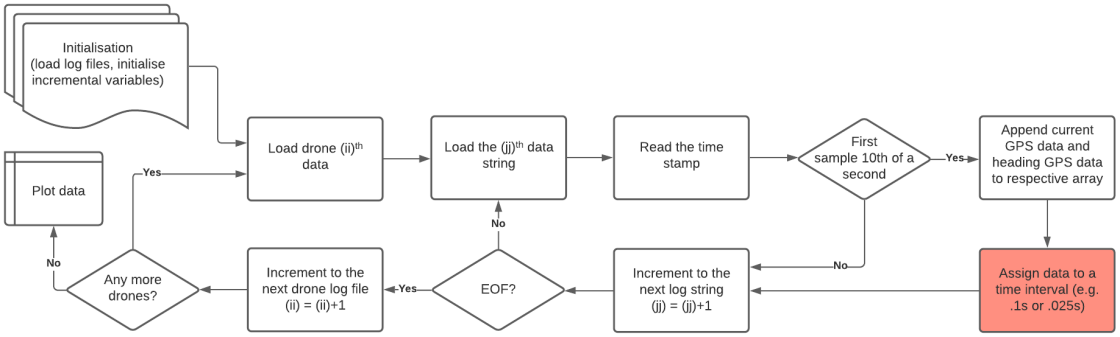


Figure 5.7 - Improvement with Assignment to a Time Interval

6 Drone fail-safes – Alex Scott

6.1 Introduction/Background

In the project, my work focused around implementing fail-safes to keep the drone and its surroundings environment safe while they were automatically flying. This was split into two main categories:

- Individual fail-safes, which looked at the safety measures for the individual drone. For example, monitoring battery life and connection strength.
- Swarm fail-safes, which looked at the safety measures for the drones interacting with one another. For example, making sure the drones do not collide or get too far out of formation.

6.1.1 Research – Previous Year's Work

This project continued from another student groups work in 2020. As detailed earlier, the desired accomplishment of this project was to have a working demonstration of all 4 slave drones flying in formation while tracking a transmitting target. The previous group was able to fly and track a target using a singular drone with the others represented in a simulated environment. This meant that most of the research for this project was focused on the reports and files of the previous group rather than external sources.

The reports of that group covered flight controls, communications, multilateration calculations, and swarm formation. Flight control ensured the individual drones experienced a steady flight and set up the simulation on Gazebo, the multilateration report focused on tracking the target's position, and communications focused on ensuring reliable communications between the drones. Swarm formation is the most applicable to fail-safes because it focused on keeping the drones in formation, which was critical to achieving the swarm flying portion of the project.

The swarm formation could monitor the position of each drone and check their position in relation to each other. If they were considerably out of position, the formation will stop tracking temporarily to correct itself. The code also implemented a failsafe function when the swarm is critically out of formation, causing the drone to stop and remain in position until it is manually landed.

The results were reported to be quite successful. The swarm formation worked using the simulated environment, and in the test with one physical drone and other drones being simulation. But, while not mentioned in the report itself, the two flight tests using two physical drones both failed to due false critical formation readings.

6.1.2 Research – Pixhawk Fail-safes

The Pixhawk flight controller is the system that controls the motors and, therefore, movements of the drone. Using QGroundControl, the flight of the drones can be monitored from a ground control position. Before the flight, QGroundControl can also be used to implement geofencing and drone fail-safes.

Geofencing ensures the drones remain in a desired area during operation. Because of the 730% increase in drone related airport incidents in recent years , it is critical that our operations stay in the desired area and do not cause problems for other aircraft. The geofencing will be set to have boundaries around Ilam Fields for the demonstration.

The QGroundControl fail-safes monitor all potential issues an individual drone can experience during flight. These include low battery, connection losses, and geofence breaches. The battery monitoring can

be set for both low and critical levels. Connection loss can monitor both the remote-control signals and data signals.

Once a failsafe has been triggered, the QGroundControl function can implement any of the following actions:

- Return to base, which involves flying vertically upward to a set altitude and safely returning to the home point.
- Hold mode, which holds the drone stationary in its current position in the air while adjusting against outside influences such as wind conditions.
- Land mode, which has the drone descend directly downward until it has landed.
- Warning, which sends a warning message back to ground control.

These four could find use under different conditions in the project. There are two other actions. Flight termination, which activates external failsafe measures like a parachute and lockdown, which disables the motors and lets the drone drop. These two did not find use in the project.

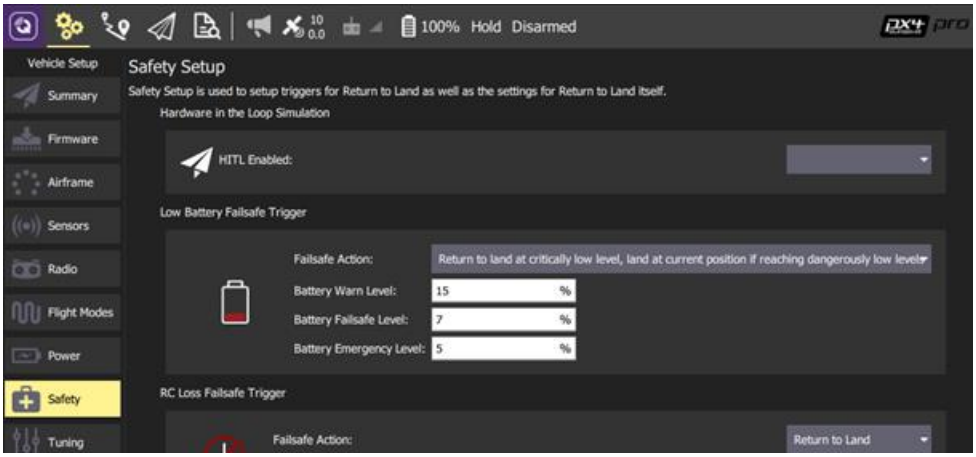


Figure 6.1: QGroundControl's safety settings panel

6.1.3 Problems Encountered

The previous groups datalogging and reporting on results presented a problem. The conclusions around the flight tests and the swarming logic appear to present a different outcome than the test results shown. As mentioned earlier, the case with two drones in flight did not work and had problems that the swarm logic report does not address. The research paper concludes "a simplified setup with up to two UAVs" was able to track the target successfully . However out of the two flight tests using two drones, only the first of them had the setup successfully track the target briefly before registering that it was critically out of formation. Because of the contradictions, a deeper look into the flight logs was deemed necessary to determine what error had occurred.

The git repository contains three sets of logs, the PX4 logs which tracked the movement of the drones during the flight tests, the ROS logs which tracked the commands going into the Pixhawk flight controller, and the flightlogs which was the data logging implemented by last year's group.

Flightlogs contained a read me that detailed the three test flights:

1. The one drone test which worked well
2. The first two-drone test which worked initially then falsely registered a critically unsafe formation and went stationary as per the safety requirement
3. The second two drone test did the same as the first test but failed earlier.

The other files in flight logs show, drone positions which shows no data of the drones moving. The target positions and estimated target positions which appear to work well but cannot be evaluated properly without a drone position reference.

I evaluated the PX4 and ROS logs to find more information about the two-drone flight tests:

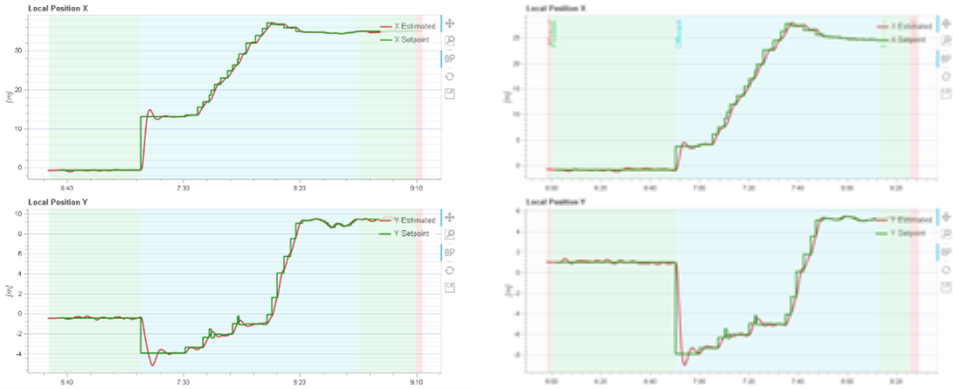


Figure 6.2: PX4 logs of both drones' X and Y positions for the first test

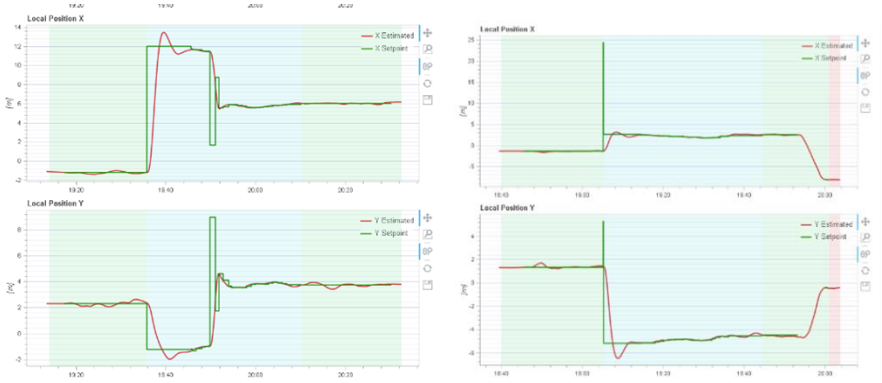


Figure 6.3: PX4 logs for the second test

To confirm the statement in flightlogs, the first test does appear to be stable and successfully tracking a target for a given time. However, the second test appears to go unstable almost immediately and the drones attempt to change their position very rapidly.

The ROS logs showed both the positions of each drone and the desired positions of the drones. The data showed no overlapping of the two in the X or Y coordinates and appeared to be moving steady and without increasing error before they went stationary.

The first test showed that the code can work with multiple physical drones and can track the target for a while. However, it still went stationary at the end as if it were critically out of position. The second test shows that the current system can go unstable at the start of automatic flight. The theory at the time was that an anomalous GPS reading must have caused the system to think the formation was unstable. Although, there is no presence of an outlier value in the ROS files so it must have been registered in the NUC only if that is the case. Another theory for the second flight is that the two drones started critically out of position before they switched to automatic flight which would explain the sudden jump in desired position.

When practical flight tests started, a checking the drones' start positions was implemented to ensure a correct starting formation. Additionally, the communications on the drone were improved by team member Nicholas. This gave the drones far less cutouts and overall, more consistent performance. With these solutions in place, the problem of false failsafe readings had stopped occurring. This meant that it was not a problem with the swarm formation code.

6.2 Work Done/Discussion

6.2.1 Landing function

The setup for drone fail-safes is as follows:

- Low battery issues a warning to the ground control so the operator can be aware of it.
- Critically low battery when operating in an open aerial space issues the drone to return to base.
- Critically low battery when operating in a crowded aerial space (tree canopy above or planes and other aircraft commonly above) the drone will land.
- All connection losses will cause the drones to enter hold mode until the connection returns. If it does not return after a certain time, the drone will follow the same protocol as critically low battery.
- A substantial breach in the geofencing will have the drones return to base.

The drone landing function is necessary in the system but could endanger the drone in its current state. It causes the drone to directly descend with no regard for the environment below. While useful in an unpopulated urban setting or open field where the ground below the drone is typically flat and safe, this does not apply to forests and other environments this project will need to track in. Descending in a forest without vision of the area below the drone could cause it serious damage.

To solve this problem, I looked at applying computer vision to survey the area below the drone and determine a safe area to land. Prior research on this topic showed few results. Most computer vision studies on automatic drone landing use fiducial markers, which indicate a predetermined safe area for the drone to land which is not useful. One research paper uses monocular simultaneous localization and mapping (SLAM) to detect a safe landing zone. This paper gave a starting point for the program, but it starts from too high and suffers from scale drift too much to be directly applicable .

Scale drift is the when the camera effectively loses track of how big objects are from the receiving frames. While it can be accounted for, generally this will cause inaccuracy problems when operating for a long enough time which could cause a problem. Scale drift does not affect the mapping if it also receives depth values to read the distance directly. Because of this, a D435 depth camera was chosen for testing.

The code designed for the landing protocol was made using OpenCV and Python. It has four steps for the procedure:

- Use pyrealsense to get frames from both the RGB and depth camera.
- Use the open3d's SLAM algorithm to create a 3d point map of the area from the frames provided.
- Use pyransac3d's ransac algorithm to fit a plane to the 3d map, finding the greatest area of flat ground.
- Create a box in the point map directly under the drone that is the proposed landing area. Determine whether that area is safe to land.

This program also reads the distance and angle between the camera and the plane. This uses a secondary reading for the drone's altitude from the ground and determines whether the plane is on an appropriate slope to land. It also requires multiple consecutive obstruction positive readings to register an obstruction. This is to combat noise which produces false readings in some rogue frames.

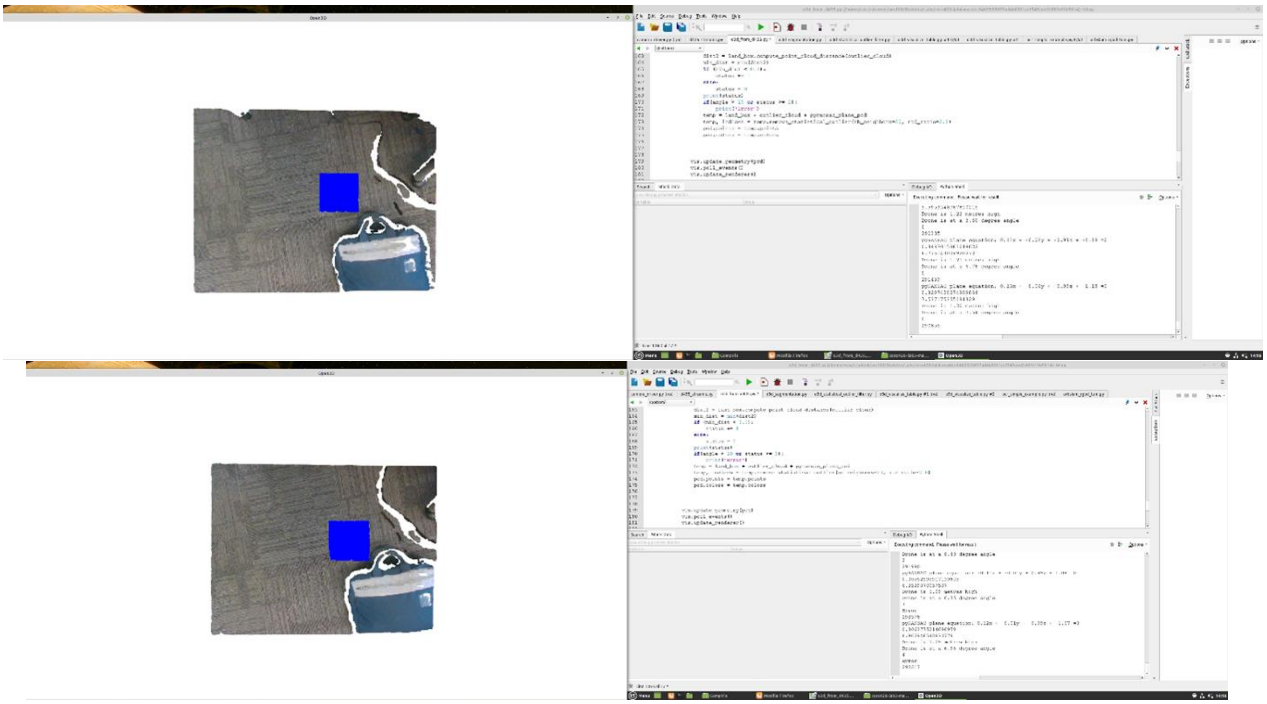


Fig 6.4: Results of landing protocol code. Safe landing case above, obstructed landing case below.

Shown in Fig 6.2, the blue box represents the area just below the drone. It can successfully detect when the bag is too close to the area to land safely, and it can detect if the angle of the ground is too steep to land.

The code still needs to be improved before it can be applied to the system. Primarily, the code fails to recognize obstacles that are considerably higher than the ground plane. This is because the function to detect distance between points works in three dimensions. So, the difference in height is also read as a safe distance. Additionally, the code will need to be optimized and the point cloud will likely need to be down sampled from its current form. This is because the code is slow on a regular computer and will take up too much processing power when run by the Intel NUC.

The landing function was put to the side for the remaining duration of the project. This is because it was not a high priority. Considering the testing for the year would be conducted on Ilam fields, the system would not have to fly below a canopy. Overall, the swarm formation fail-safes needed improvement before the system could fly. Whereas the landing function was not necessary for the working prototype to perform.

6.2.2 Improving swarm logic code

Once flight tests had started, we were able to practically test the code. While testing in the simulation had presented no errors in the failsafe code, practical tests allowed us to see how it would operate under circumstances that aren't ideal.

The previous year's swarm formation code used a Swarm class. This was where each receiver drone was given an offset location from the transmitter drone according to their designated position. The four receiver drones were ± 5 metres in longitude and latitude from the transmitter drone. This creates the 10 by 10 metre square pyramid formation. Then three failsafes were in place to maintain it:

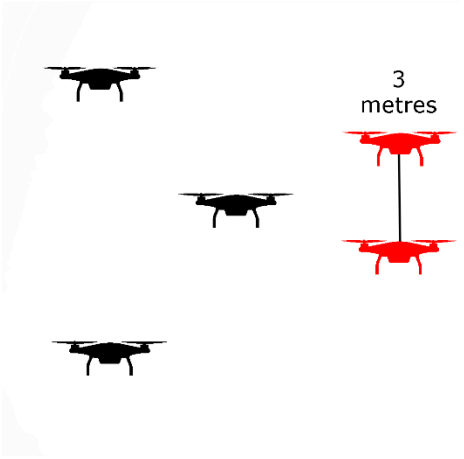


Figure 6.5: Drone distance failsafe

The drone distance failsafe looked at the distance between the GPS readings between each drone. If they got within 3 metres of each other, the failsafe would trigger.

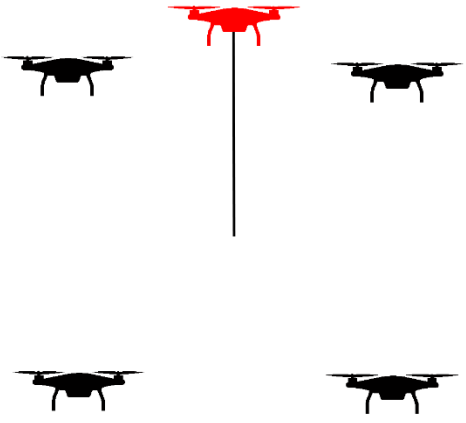


Figure 6.6: Drone overlap failsafe

Drone overlap failsafe would check to see if any drones overlap with one another in latitude or longitude. If any did, the failsafe would trigger.

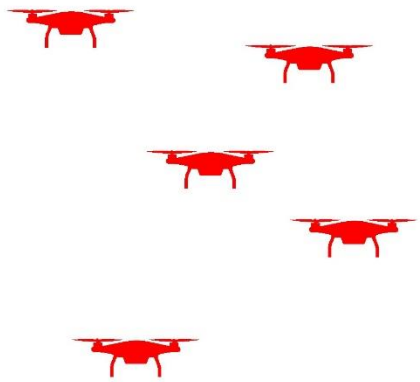


Figure 6.7: General formation failsafe

The general drone formation failsafe would remove the offset from each drone's position for calculation, meaning the only distance between the points is caused by GPS error. Then the net error between the points is calculated. If this exceeds 6 metres, the failsafe triggers.

This provided a good base to build upon because the code did work in the simulation. However, in matters of readability there were several improvements that I implemented.

I increased the minimum distance on the drone distance function. While the comments had noted the minimum distance was set was 3 metres, it was only set to 1. So, I started by increasing that to 3 metres. However, the uncertainty for the GPS modules we were using was between 2 and 2.5 metres. This meant that, while very unlikely, there was a possibility that two drones could collide without the failsafe being triggered. So, the distance had to be increased to 5 metres to be considered safe.

When a failsafe was triggered previously, it would only indicate that a critical formation had occurred. However, this gave no readable information on what failsafe had been triggered. This required looking through the data logs and each individual GPS reading to determine the problem. To solve this, I placed appropriate statements onto each failsafe that would be sent to the data logs.

Both the drone distance and general formation errors also had warning levels put in place to indicate when the system is approaching critical levels. For the general formation error, it would send a warning message at a 3-metre net error. For the drone distance error, it would send a warning message when the drones varied 2 metres from their expected distance. During practical testing, there were multiple times the swarm would go into the hold position with seemingly no error occurring. This made diagnosing those problems difficult. But with the added warning and error messages, the process was considerably faster.

One problem had increasingly occurred when testing with more drones. The general formation error would occur more often with more physical drones flying. This makes sense because a real drone will have more GPS error and uncertainty than a simulated target. So, the more drones present in the system, the more net error occurs. Each GPS module has between 2 and 2.5 metres of uncertainty, meaning the net error can reach at least 7.5 metres for three physical drones. This would make flying three drones unreliable and flying four drones impossible. So, the system would need more reliable position readings for the drones before it could achieve the project goal.

6.2.3 New Swarm Failsafe Code

Major reworks were done on the code over lockdown. This included multiple improvements, but the key additions for fail-safes were the Kalman filter and state machine:

- The state machine added a hold state for the system. So now the failsafe tasks just need to switch the state machine instead of repeatedly sending the drone's current position as it's desired one.

- The Kalman filter provided a more consistent estimate for the drone's position readings and provides a continuous reading on the estimate's uncertainty called the variance. Additionally, the Kalman filter responds to signal dropouts by continuing to estimate the drone's position.

With both improvements implemented, the failsafe code also needed considerable reworks. The new failsafe code I created does not create the Swarm class like before and does not set the hold state either. Both are now handled elsewhere by the state machine.

This new code receives the drone positions, checks for failsafe conditions, and then sends back if the failsafe has been triggered or not. The reworked code is also significantly smaller than the previous one. The reason for reducing this module is to follow the single responsibility principle. This principle states that each module in a computer program should only be responsible for a single aspect of it . This is to make it easier for a continuing student group to understand it and continue the project. Overall, this is significantly more readable as seen in Appendix A. Whereas the previous code had mixed both swarm formation maintenance and fail-safes together.

Continuing to the functions presented in the new failsafe code. Previously all three safety measures were placed inside the same 'critical formation' function. I split these into three separate function to continue following the single responsibility principle. The three main failsafe checks in the new code are:

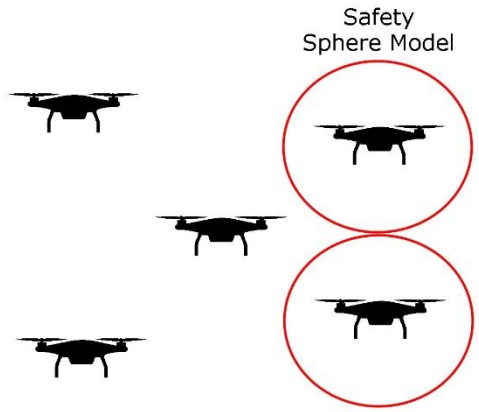


Figure 6.8: New drone distance failsafe

The new drone distance function has its safety distances informed by the Kalman filter. The filter provides the drone's latitude and longitude variances. This function takes the larger variance between the two and uses it to model a safety sphere. The safety sphere represents where a drone could realistically be according to the Kalman filter. So, if two of these spheres were to make contact, the failsafe would be

triggered and be put into the hold state. While Kalman filter's variance can be smaller than 1.5 metres, the drones will also go into the hold state if two drones come within 3 metres of one another.

The drone overlap function was put into its own function. Outside of that change, this failsafe remains the same.

Figure 6.9: Realistic readings failsafe

Realistic reading's function was made to account for unrealistic readings from a GPS module. On rare occasions, a GPS module will present a location far outside its uncertainty. For example, while flying on Ilam fields a drone might get a reading that places it in the Pacific Ocean. This is clearly incorrect and could cause the system to failsafe. However, considering it is a possible occurrence with any GPS module, the system should not be put into the hold position when this occurs.

The previous year's code would check to see if a reading was outside New Zealand. But what if the reading is in Auckland when the flight test is in Christchurch. This new function sets a distance threshold in metres. If the distance between the drone's current and previous positions exceeds the threshold, the failsafe is triggered.

Currently, the response to this failsafe has the drone's current position reading replaced by its previous position one. While this does allow the swarm to continue flying when the failsafe is triggered, it should instead treat the abnormal reading as a data dropout so the Kalman filter can properly estimate the drone's position.

6.2.4 Other work and administration

A portion of my work done was also administration and safety measures around the drone flight tests. It was my job to ensure that flight tests could occur on a weekly basis. This required learning and following the regulations around civilian drone flights.

The following had to be accounted for to conduct a flight test:

- A continuous line of sight was needed. Currently New Zealand regulation requires all drone flights to be within line of sight of the pilot. Considering our tests flights were held on Ilam fields, this was not a problem
- All flights must be below the tree line. Flying below the tree line has no risk of hitting commercial aircrafts, so it is open for civilians to fly drones there. If we were to fly above the tree line, it would require a qualification that would exceed the budget of this project to get.

- Experienced drone pilots were needed to fly the drones. This was to reliably launch and land them during flight tests. They were also required to take the drones back into manual control if anything were to go wrong
- UC security needed to be informed about every flight. This is because they do not allow unauthorized flights on Ilam fields.

During flights, an extra safety measure was put in place to monitor the battery voltage. The batteries provided for this project were 6-cell Lithium-ion batteries that were approximately 5 years old. Because of this, it was likely the capacity of the batteries were significantly lower than if they were freshly bought. So, an additional safety measure was deemed necessary.

The battery voltage monitor provided would check the voltage of each cell in the battery and compare it to a minimum threshold. If the cells voltage were to drop below the threshold, the monitor would make a loud beeping noise until removed. This indicated to the pilot to take manual control and land as soon as possible.

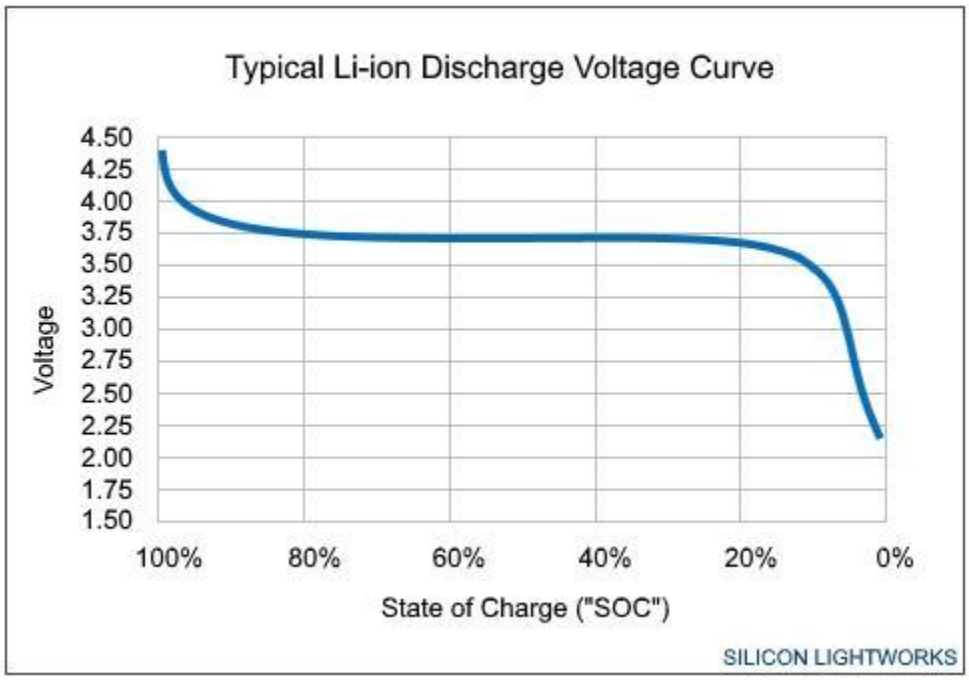


Fig 6.10: Discharge curve for the cell of a lithium-ion battery .

As seen on the discharge curve, the voltage of a lithium-ion battery drops rapidly when coming to the end of its capacity. With batteries where the capacity is lower than expected, this drop can be very quick. The voltage monitors had their threshold set to 3.6V to give ample time to land the drones before the drop occurs.

7 Sustainability

A triple bottom line analysis was performed to assess the sustainability of the drone swarm. Triple bottom line analysis assesses societal, environmental, and economic indicators which provide professional engineers insight into the project sustainability. The positive implications a project has on the environment, society, and economy are factors which ultimately determine the sustainability, namely with younger generations in mind.

7.1 Environmental

This project's end goal is to provide a method of tracking insects in the natural environment. Insects have become a point of attention in recent years and knowing more about them can help us protect endangered species or contain invasive ones. The 1993 Convention on Biological Diversity and 2020 New Zealand Biodiversity Strategy outline the importance of environmental conservation and protection to New Zealand. However, the current state of invertebrate conservation management is dire with 34.5% classed as either threatened or at risk and a further 1252 classified as 'data deficient' [1]. The paucity of ecological and behavioural knowledge about these species hinders conservation efforts. The current methods for gathering knowledge about invertebrate species are ineffective and provide unreliable data. Previous methods include the marker technique where insects are marked with dye and then released. They are trapped in various locations around the release point allowing their dispersal characteristics to be mapped . This method is very labour intensive and introduces bias into the system (as traps are only placed in certain areas for surveying). It also relies on a large population of insects to be captured and released due to the difficulty of trapping insects and the bias introduced into the system. Hence, accurate methods that can map how insects utilize their environment are required to facilitate their management.

Invertebrates are the most diverse group of terrestrial species, providing vital components for local ecosystems across the country. They play crucial roles in pollination, organic decomposition, and the cycling of energy and nutrients . The need for monitoring insects has been an ever-growing concern. One well known case being the declining number of bees, which is alarming considering that they are increasingly needed for the global food supply . Another important case is the Vespa Mandarinia also known as the 'murder hornet' which has become a high priority pest when it entered the US as an invasive species. Harmonic radar presents a biosecurity tool to track these hornets back to their nests to facilitate eradication of the population.

This project aims to provide information about invertebrate behaviour, allowing tools to be developed that protect invertebrates, the native bush and natural ecosystem. The primary benefit of our project is that it allows us to track insect movement within the environment. Additionally, the tracking's effectiveness is not changed by environmental conditions like time of day. This allows broader applications in studying the tracked target. The vision of this project closely aligns with the Māori practice Kaitiakitanga. Kaitiakitanga outlines a sustainable way of managing the environment with underlying values in guardianship and protection.

7.2 Economic

Resulting from a lack of invertebrate knowledge, New Zealand's biosecurity is under threat from invasive species. Invertebrate pests are estimated to cause an annual loss of up to $2.3 billion dollars within the agricultural industry [2]. They are also estimated to cause $880 million dollars of damage to native biodiversity [3] . Pesticides are becoming less effective, with some species developing biological resistance [3]. Stricter regulations are being enforced to protect both the environment and human health. This leaves groups of invertebrate pests untreated as older pesticides are being phased out. One example of this is methyl bromide, a pesticide that was used to treat 22% of New Zealand's timber before export and will be completely banned in 2023 . Methyl bromide is an ozone-depleting chemical and has severe associated

health risks. There is a need for new pest reduction methods that are effective and do not damage the environment. This project aims to provide a better understanding of invertebrate behaviour to support this research.

As this technology is currently just a proof-of-concept, there will be costs associated to develop a product for commercial use. With the current swarm system, there is an initial upfront cost of $100,000 NZD to purchase five drones with the required hardware. With UAV technology rapidly advancing, it is likely that this cost will significantly reduce in the future. Extra hardware that is installed on these drones can also be minimised to reduce this cost. During operation, there will be costs associated with the maintenance of the swarm. As these drones will be facing harsh outdoor environments, there will likely be costs associated with replacement parts and units. Overall, these costs are likely to be offset by the economic gain of developing new pest eradication methods, considering they cause an annual loss of $2.3 Billion NZD.

Currently, the application of this project is limited by New Zealand Civil Aviation Authority (CAA) flight regulations that require drone activity to be within line of sight of the pilot. There is also an extended line of sight operation which allows the system to go further provided there is someone viewing it with constant communication with the pilot. This limits the current use of the project, but we are developing it in line with guidance about the future requirements for beyond visual line of sight drone safety and reliability. Thus, we are confident our swarm technique will be applicable for future applications.

7.3 Social

Analysis of societal sustainability requires knowledge of the stakeholders. Assessment of many societal aspects are considered. Object tracking using drone swarms looks to provide a method for gaining valuable information on animal movement patterns. These include migration habits to assist in developing a new pest reduction strategy. The WRC, Scion, and department of forestry share interests with society as there is primary interest in biological risk assessment, biosecurity, surveillance, diagnostics, biodiversity, and ecosystem function.

It is vital to use non-toxic pesticide alternatives or reduce the amount of pesticide applications. Pesticides such as methyl bromide are detrimental to all animal health and cause lung damage and long-term neurological damage after inhalation [29]. A drone swarm can locate the likes of a nest and therefore allows a minimalistic approach to pesticide application, reducing the harm to human health.

For societal sustainability, the manufacturers need; generally recognized cohesion and services, beliefs shared with the public, and an understanding with sympathy toward the public and environment. As for conservationists to agree, this stakeholder would agree to allow insect tracking, however, if and only if the underlying morals and principles of the stakeholders above (WRC, Scion, Department of Forestry, and manufacturers) coexist with the conservationist; further proving societally sustainable. As with environmental sustainability, there are almost no negative effects on landowners besides privacy concern and minimal noise pollution [30]. Another common problem is drones delaying commercial air lines by flying too close to airfields [19] – both of which can cast drone flights in a negative light on the social stage. As the swarm will operate in unpopulated areas such as forests, it is unlikely to interfere with airways and have implications on the public. The use of object tracking using drone swarms has positive social consequences, resulting in vast societal sustainability. Furthermore, as there are healthier biophysical environments for workers, disability-adjusted life years are reduced. The health and safety improvements also align with societal sustainability

8 Conclusions and Recommendations

Drones have an integral role in the future of remote sensing and tracking. The aim of this project was to develop a drone swarm which can track targets, such as insects, remotely and accurately. Investigation and a proof-of-concept UAV swarm tracking of insects / emulated targets is a great step toward better biosecurity and ecosystem stability. As insect numbers and diversity are declining, newer and faster approaches to insect tracking are required to enable greater research into insect behaviour and hive identification. Current techniques are inefficient and cannot operate in environments which are not accessible to humans.

Triple bottom line analysis has proven that the drone swarm is a sustainable project, with the potential to assist in conservation management and develop new pest reduction strategies. Most of the negative outcomes are not directly caused by the drone swarm itself. As mentioned in the life cycle analysis, the sustainability of using UAVs for tracking is influenced by the drone manufacturers. Regardless of the potential damages the manufacturers cause, the vast impact on environmental, societal, and economical is proven sustainable. This was concluded by considering the Resource Management Act and Environmental Impact Assessment (such as the brief product life cycle assessment), potential stakeholders, and economic gain from reducing destructive insect infestation.

The drone swarm described in this report consists of five drones: one transmitter drone and four receiver drones. The transmitter emits a signal which reflects off a harmonic tag on the target. It is then received at different times by each receiver drone depending on the drone's distance from the target. A multilateration equation was developed by Oli Dale to determine the target's position from the radar signal. A tracking algorithm was created by Rowan Sinclair to combine the sensors from all the drones, allowing the target to be tracked through the effects of noise and dropout. A communications system was developed by Nicholas Ranum to reliably send data between the drones and synchronise their movements. Connor O'Reilly developed a data logging system to assist in debugging and plot the target's path. Alex Scott developed failsafes to prevent the drones from colliding and prevent damage to people and property.

The drone swarm is an incredibly complex system, and its components have many subtle interactions and behavior's. A set of recommendations were compiled to assist in the continued development of the project:

1. The simulation is an incredibly powerful tool for testing the swarm and learning how it works. We recommend learning and running the simulation as soon as possible to understand the logic behind the code. The README within the git has instructions on how to run the simulation under multiple drone configurations.

2. Get other academic staff involved, including the course coordinator of the final year project. This is a new, immature project that has great potential in the coming years. All staff are interested and are willing to provide insightful information into improving the operation of the drone swarm.

3. Have many people train how to fly the drones. Prior to learning how to fly a drone, understand the operation of the code first; hence, the importance of the first recommendation and getting onto the project early. A major bottleneck in the development of the swarm was finding pilots to fly the drones. The more pilots you have the more real flight tests you can do.

4. Finally, the project in its current state is not complete. The code contains many readmes for components that still need to be completed. The main tasks that still need to be completed are implementing the controller, reducing communication latency, and testing the transmitter drone in a real flight.

9 References

[1] Department of Conveservation, New Zealand, "New Zealand Threat Classification System," 30 May 2021. [Online]. Available: https://nztcs.org.nz/.

[2] C. M. Ferguson and et al, "uantifying the economic cost of invertebrate pests to New Zealand's pastoral industry," *New Zealand Journal of Agricultural Research,* vol. 62, no. 3, pp. 255-315, 2019.

[3] The Royal Society of New Zealand, "Challenges for pest management in New Zealand," 2014. [Online]. Available: https://www.royalsociety.org.nz/assets/documents/Challenges-for-pest-management-in-NZ.pdf. [Accessed 26 May 2021].

[4] D. Kissling, D. E. Pattemore and M. Hagen, "Challenges and prospects in the telemetry of insects," *Biological Reviews,* vol. 89.3, pp. 511-530, 2014.

[5] A. Lavrenko, B. Litchfield, G. Woodward and S. Pawson, "Design and Evaluation of a Compact Harmonic Transponder for Insect Tracking," *IEEE Microwave and Wireless Components Letters,* vol. 30, no. 4, pp. 445-448, 2020.

[6] "Sirtrack: Wildlife Tracking Solutions. Coded VHF: NanoTag Series," 10 Oct 2019. [Online]. Available: https://www.sirtrack.co.nz/index.php/freshwater/coded-vhf/all-attachments.

[7] E. Wan and R. V. D. Merwe, "The unscented Kalman filter for nonlinear estimation," in *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No.00EX373)*, 2001.

[8] R. E. Kalman and R. S. Bucy, "New Results in Linear Filtering and Prediction Theory," *Journal of Basic Engineering,* vol. 83, no. 1, 1 3 1961.

[9] P. D. Moral, "Non Linear Filtering: Interacting Particle Solution," *Markov Processes and Related Fields,* vol. 2, pp. 555-580, 1996.

[10] Y. Bar-Shalom, X. R. Li and T. Kirubarajan, Estimation with applications to tracking navigation, 2001, p. 274.

[11] Open Source Robotics Foundation, "Gazebo," [Online]. Available: http://gazebosim.org/. [Accessed 27 October 2021].

[12] Dronecode Foundation, [Online]. Available: https://px4.io/. [Accessed 27 October 2021].

[13] Dronecode Project, "Mavlink Developer Guide," [Online]. Available: https://mavlink.io/en/. [Accessed 27 October 2021].

[14] Creative Commons Attribution 3.0, "ROS Wiki," [Online]. Available: http://wiki.ros.org/mavros. [Accessed 27 October 2020].

[15] The ZeroMQ authors, "ZeroMQ Documentation," [Online]. Available: https://zeromq.org/get-started/. [Accessed 27 October 2021].

[16] Intel, "Intel NUC Kit NUC8i5BEK," [Online]. Available: https://www.intel.com/content/www/us/en/products/sku/126147/intel-nuc-kitnuc8i5bek/specifications.html. [Accessed 7 April 2021].

[17] J. M. J. García, "Simulation in real conditions of navigation and obstacle avoidance with PX4/Gazebo platform.," *Pers Ubiquit Comput,* 2020.

[18] ZeroMQ,, "Get Started," [Online]. Available: ZeroMQ | Get started.. [Accessed 2021].

[19] Ubiquiti Inc, "rocket M datasheet," [Online]. Available: https://dl.ubnt.com/datasheets/rocketm/RocketM_DS.pdf. [Accessed 27 October 20211].

[20] Laird Connectivity, "2.4 GHz Dipole Antenna," [Online]. Available: https://www.lairdconnect.com/documentation/24-ghz-dipole-antenna-datasheet. [Accessed 27 October 2021].

[21] Farnell, "Cat5 Cable," [Online]. Available: http://www.farnell.com/datasheets/1311845.pdf. [Accessed 27 October 2021].

[22] Python Software Foundation, "Python interface to Tcl/Tk," [Online]. Available: https://docs.python.org/3/library/tkinter.html. [Accessed 27 October 2021].

[23] D. L. Mills, "Internet time synchronization: the network time protocol," *IEEE Transactions on Communications,* vol. vol. 39, pp. pp. 1482-1493, 1991.

[24] "SparkFun Pixhawk 4 Flight Controller,," Mouser.com., [Online]. Available: https://nz.mouser.com/new/sparkfun/sparkfun-pixhawk-4-flight-controller/. [Accessed 26 Oct 2021].

[25] "STM32F100CBT6B," Element14.com, [Online]. Available: https://nz.element14.com/stmicroelectronics/stm32f100cbt6b/mcu-32bit-cortex-m3-24mhz-lqfp/dp/1838512. [Accessed 26 Oct 2021].

[26] "System Time and Clock — NuttX latest documentation," Apache.org, [Online]. Available: http://nuttx.incubator.apache.org/docs/latest/reference/os/time_clock.html. [Accessed 26 Oct 2021].

[27] "PX4 User Guide," 14 March 2021. [Online]. Available: https://docs.px4.io/master/en/ .

[28] PBTech, "GlobalSat BU-353-S4 Weather-proof USB GPS receiver BY G.Sat SiRf Star IV GPS chipset compatible with Windows iRF Star IV," [Online]. Available: https://www.pbtech.co.nz/product/HHAGST0354/GlobalSat-BU-353-S4-Weather-proofUSB-GPS-receiver. [Accessed 10 April 2021].

[29] "roscpp/Overview/Logging - ROS Wiki," Ros.org, [Online]. Available: http://wiki.ros.org/roscpp/Overview/Logging. [Accessed 26 Oct 2021].

[30] K. Larbey, "Drone disruption at airports: A risk mitigation insurance response," June 2019. [Online]. Available: https://www.willistowerswatson.com/en-US/Insights/2019/07/drone-disruption-at-airports-a-risk-mitigation-and-insurance-response.

[31] A. Lavrenko, Z. Barry, R. Norman, C. Fraser, Y. Ma, G. Woodward and S. Pawson, "Autonomous Swarm of UAVs for Tracking of Flying Insects with Harmonic Radar," *2021 IEEE 93rd Vehicular Technology Conference,* 2021.

[32] T. Yang, P. Li, H. Zhang, J. Li and Z. Li, "Monocular Vision SLAM-Based UAV Autonomous Landing in Emergencies and Unknown Evironments," *Electronics 2018,* p. https://doi.org/10.3390/electronics7050073, 2018.

[33] T. Janssen, "SOLID Design Principles Explained: The Single Responsibility Principle," 1 April 2020. [Online]. Available: https://stackify.com/solid-design-principles/.

[34] Silicon Lightworks, "Li-ion Voltage Analysis," Silicon Lightworks, [Online]. Available: https://siliconlightworks.com/li-ion-voltage. [Accessed 25 Oct 2021].

[35] S. L. Goldson and et al, "New Zealand pest management: current and future challenges," *Journal of the Royal Society of New Zealand,* vol. 45, no. 1, pp. 31-58, 2015.

[36] G. Li, R. Gao, T. Michael and L. Kong, "Study on dispersal of anoplophora glabripennis population," *Forest Research,* vol. 23, no. 5, pp. 678-684, 10 2010.

[37] B. Patrick, "The importance of invertebrate biodiversity: an Otago Conservancy review," New Zealand Department of Conservation, 1994. [Online]. Available: https://www.doc.govt.nz/Documents/science-and-technical/casn053.pdf. [Accessed 27 May 2021].

[38] M. J. F. Brown and R. J. Paxton, "The conservation of bees: a global perspective," *Apidologie,* vol. 40.3, pp. 410-416, 2009.

[39] Ministry for Primary Industries, Biosecurity New Zealand, "Methyl bromide update," June 2021. [Online]. Available: https://www.mpi.govt.nz/dmsdocument/39575-Methyl-bromide-fact-sheet-.

[40] Ministry for Primary Industries, "Methyl Bromide Information," 2020. [Online]. Available: https://www.mpi.govt.nz/dmsdocument/14869/direct. [Accessed 27 May 2021].

[41] R. Maggiora, M. Saccani, D. Milanesio and et al, "An Innovative Harmonic Radar to Track Flying Insects: the Case of Vespa velutina," *Nature, Sci Rep 9, 11964,* 2019.

[42] A. G. Gopi, B. Rao and R. Maione, "The societal impact of commercial drones," *Technology in Society,* vol. 45, pp. 83-90, 2016.

10 Appendix

10.1 Appendix A: Swarm Failsafe code

```
"""
filename: swarming_logic.py
author: Alex Scott
date: 16th September 2021
description: Functions to check positions and variance of tx and rx positions.
It then returns if there is an error or warning for the drone formation's
safety.
"""

import math
import numpy as np
from gps import GPSCoord
from common import *
import rospy


class Failsafes:

    def __init__(self, is_local) -> None:
        # Get the constants from the json
        self.fp = get_parametres(is_local)
        self.size = self.fp['COMMON']['DRONE_DISTANCE']*2
        self.formation_error = self.fp['COMMON']['FORMATION_ERROR']
        self.var_def = self.fp['COMMON']['DEFAULT_VARIANCE']
        self.max_drone_movement = self.fp['COMMON']['MAX_DRONE_MOVEMENT']

        # TODO: At the moment the altitude of rx1 is assumed to be the altitude of all the rx
drones
        # Redo base distances matrix so that the rx drones can all have differnt altitudes
        self.rx_alt   =   abs(self.fp['1']["DEF_ALTITUDE"]-self.fp['0']["DEF_ALTITUDE"])   #   The
difference between the rx and tx alt
        self.tx_rx = np.sqrt((self.size*np.sqrt(2)/2)**2+self.rx_alt**2)
        self.rx_rx_diag = self.size*np.sqrt(2)

        self.prev_pos = None # The last position of the drone

    def positions(self, tx_pos, rx_pos):
        """
        This function takes the realtime GPS positions of the drones and returns True if the
formation
        is good. It checks whether the variance spheres overlap and give a warning if the drones
are
        getting too far from their ideal swarm positions
        """
        drones = [tx_pos] + rx_pos

        # The distances between the drones
        base_distances = np.array([
            [0,           self.tx_rx,       self.tx_rx,       self.tx_rx,       self.tx_rx       ],
            [self.tx_rx,  0,                  self.size,         self.size,         self.rx_rx_diag
],
            [self.tx_rx,  self.size,        0,                self.rx_rx_diag,  self.size        ],
            [self.tx_rx,  self.size,        self.rx_rx_diag,  0,                self.size        ],
            [self.tx_rx,  self.rx_rx_diag,  self.size,        self.size,        0              ]])

        return_val = True

        for i, d1 in enumerate(drones):
            for j, d2 in enumerate(drones):
                # We don't need to compare the distance between each drone twice or the distance
between the same drone
                if j < i:
                    # Sum the maximum variances to find the 1sd distance sphere
                    # TODO: Model the actuall ellipse for differing lat, long variances instead of
taking max.
```

```
                    # If any of the GPS variances are undefined, use the default variance
                    if any([var==-1 for var in d1.var+d2.var]):
                        limit = self.var_def*2
                    else:
                        limit = max(d1.var[0:2]) + max(d2.var[0:2])

                    # If the drone's variance spheres overlap return false
                    if d1.distance3d(d2) < limit:
                        rospy.logfatal("ERROR:    DRONES    {}    and    {}    ARE    TOO    CLOSE
({:.2f})".format(i,j,d1.distance3d(d2)))
                        return_val = False

                    # If the drones are within a specified range limit of eachother return false
                    if      abs(d1.distance3d(d2)-base_distances[i,j])/base_distances[i,j]      >
self.formation_error:
                        rospy.logwarn("WARNING: DRONES {} and {} ARE OUT OF FORMATION ({:.2f}m,
{:.2f}m)".format(i,j, d1.distance3d(d2),base_distances[i,j]))
                        print(base_distances[i,j])
                        return_val = True

        return return_val



    def overlaps(self, tx_pos, rx_pos):
        """
        This function takes the tx and rx drone positions and checks whether any of their latitudes
or longitudes overlap
        If they do the function returns False and a warning is printed
        """
        drones = [tx_pos] + rx_pos
        # Drone 1 left (lower long) of 0, 2, 4, and above (higher lat) 0, 3, 4
        # Indicates which drone has overlapped
        if  not(drones[1].long  <  drones[0].long  and  drones[1].long  <  drones[2].long  and
drones[1].long < drones[4].long):
                rospy.logfatal("Drone 1 longitude overlap")
                return False
        if not(drones[1].lat > drones[0].lat and drones[1].lat > drones[3].lat and drones[1].lat >
drones[4].lat):
                rospy.logfatal("Drone 1 latitude overlap")
                return False
        # Drone 2 right of (higher long) 0, 3, and above (higher lat) 0, 3, 4
        if not(drones[2].long > drones[0].long and drones[2].long > drones[3].long):
                rospy.logfatal("Drone 2 longitude overlap")
                return False
        if not(drones[2].lat > drones[0].lat and drones[2].lat > drones[3].lat and drones[2].lat >
drones[4].lat):
                rospy.logfatal("Drone 2 latitude overlap")
                return False
        # Drone 3 left of (lower long) 0, 4, and below (lower lat) 0
        if not(drones[3].long < drones[0].long and drones[3].long < drones[4].long):
                rospy.logfatal("Drone 3 longitude overlap")
                return False
        if not(drones[3].lat < drones[0].lat):
                rospy.logfatal("Drone 3 latitude overlap")
                return False
        # Drone 4 right of (higher long) 0 and below (lower lat) 0
        if not(drones[4].long > drones[0].long):
                rospy.logfatal("Drone 4 longitude overlap")
                return False
        if not(drones[4].lat < drones[0].lat):
                rospy.logfatal("Drone 4 latitude overlap")
                return False

        return True



    def move(self, pos):
        """
```

```
        This function takes too GPScoords and checks whether the distance between them is too
        large. This is to ensure the drones aren't commanded to fly away. Returns True if the
        drones are moving at a good speed else returns False and prints an error.
        """
        return_val = True
        # If the previous position is None the failsafes object has just been initialised, return
true
        if self.prev_pos is not None:
            if pos.distance(self.prev_pos) > self.max_drone_movement:
                rospy.logfatal('ERROR:          Drones          moving          too          fast
({:.2f})'.format(pos.distance(self.prev_pos)))
                return_val = False
            elif pos.distance(self.prev_pos) > self.max_drone_movement*0.8:
                rospy.logwarn('WARNING:     Drones     are     almost     moving     too     fast
({:.2f})'.format(pos.distance(self.prev_pos)))
                return_val = True
            else:
                return_val = True

        self.prev_pos = pos
        return return_val

def main():
    """
    Short program to test failsafes
    """
    failsafes = Failsafes(is_local=True)

    # Define drone locations
    target = GPSCoord(-43.5206,172.58302,0,[1,2,-1])
    rxs = [None]*4
    tx = update_loc(target, 0, True)
    tx = tx.add_x_offset(0)
    for i in range(1,5):
        rxs[i-1] = update_loc(target, i, True)
    # Add small error in one drones position
    rxs[0] = rxs[0].add_y_offset(3)
    # Test the function
    print('Positions output: {}'.format(failsafes.positions(tx, rxs)))

    tx_next = tx.add_y_offset(8)
    print('Move output: {}'.format(failsafes.move(tx)))
    print('Move2 output: {}'.format(failsafes.move(tx_next)))

if __name__ == "__main__":
    main()
```