Cover page

Project Title: Target Tracking using Drone Swarms

Industrial Sponsor: Wireless Research Centre

Supervisor: Le Yang

Student Names & IDs

1: Benjamin Ireland
93524067

2: Finlay Cross
46667692

3: Se Hyun Kim
17430351

4: Ronniel Padua
17413145

Executive Summary

New Zealand has many endangered and invasive insect species that are extremely difficult to track. Information about these species is incredibly important for conservation efforts of our endangered insects and native flora. This project, with the help of the Wireless Research Centre, aims to further the development of a target tracking drone swarm capable of collecting data on these insects' interactions with their environment. This technology is being developed for use by the Department of Forestry while additionally providing a research platform for flying ad hoc networks.

This project builds upon the implementation of a previous drone swarm. The newest implementation aimed to remove limitations of the previous drone swarm, relating to the swarm's performance and scalability, while adapting the swarm to be able to track an arbitrary target. Additionally, the project aimed to continue development of the drone's network protocol, the drone coordination protocol (DCP), and streamline the configuration and operation of the drone swarm.

The project undertaking was partitioned into four subprojects, listed as follows: Path planning and trajectory following, target implementation, network protocol development, and the drone management and visualisation application. For each of these subproject's research was conducted to investigate the applications of drone swarms and current implementations of similar technologies. A new design for the drone swarm was developed and simulated before being tested in the field to measure the performance of the new implementation.

From the field tests conducted a two-drone swarm was observed tracking a target in a field test, and the formation forming of a triangular drone swarm was achieved. The drone swarms new design demonstrated better performance and the capability to be easily scaled. The integration of DCP and the management application were never fully realised by the conclusion of the project. However, these have been fully developed and merely need to be integrated.

The simulated drone swarm closely matched what was observed in field tests. Hardware limitations and external software configurations prevented further testing of the drone swarm and hindered further progress. It is recommended in the continued development of the drone swarm that the integration of DCP and the management application be completed, along with further enhancements to the current drone swarm design, will see successfully operating drone swarms at a larger scale.

Table of Contents

Introduction

Purpose Statement

The purpose of this project is to further develop the Wireless Research Centre's (WRC) target tracking drone swarm technology. Potential applications of this technology are extensive and will impact key industries. The specific application the swarms are being developed for is tracking endangered NZ insects using harmonic radar technology developed by WRC. Other applications include infrastructure monitoring, providing support in natural disasters, and search and rescue. The stakeholders for this final year project (FYP) are the WRC, the Department of Computer Science and Software Engineering, and the Department of Forestry. The 2024 FYP team aimed to be able to coordinate multiple drones in formation tracking a slow-moving target in the field.

Project Overview

The Wireless Research Centre is one branch of the University of Canterbury's research sector that focuses on wireless communication and signal applications. They are responsible for sponsoring the drone swarm FYP and are the acting clients for this project. This FYP project is a continuation which builds upon previous final year project developments with a goal to create an autonomous drone swarm capable of finding, tracking, and following a target.

The primary application the WRC intends to use this technology for is to reliably track endangered insects. Insects are incredibly difficult to track through conventional means. Because of this little is known about how these insects interact with the environment. This information is important for the conservation of our native insects. In development at WRC is harmonic radar technology that can be used to locate tagged insects. The drone swarm will be fitted with this technology in future once it is fully developed.

There is currently no known commercially operative drone swarm deployments in NZ as the development of this technology is relatively new. The most relatable technology which uses drone swarm communications are drone light shows. These differ from the drone swarm being developed. In these applications each drone has their flight path pre-orchestrated, whereas the drone swarm is dynamic. The WRC requires a reliable and scalable drone swarm that accurately tracks a target and removes the rigidity of the previous implementation. This means that the drone swarm must be able to be deployed repeatedly without failure and additionally, increasing the number of drones should not require any significant changes to system software. Listed in Table 1 are the overall project goal and requirements as prescribed by WRC.

***Table 1** - Overarching Project Goals*

Project Goals
Scale up to larger drone swarms.
Improve drone swarm performance.

Develop an exemplary tracking application.
Make swarm formations more easily configurable.
Develop and integrate DCP.
Implementation of formation algorithms.
Streamline the process of configuring, running, and diagnosing field tests.
Develop an application for updating drone software and configuring swarms.

Project Breakdown

The overall project was partitioned into four subprojects outlined as follows:

- **Path Planning and Trajectory Following** – Redesigning the current flight control system to improve the efficiency and scalability of system software while decoupling the swarm formation from target tracking.
- **Drone Management and Visualisation Application** – Development of an application that oversees the management of drone software, pre-flight diagnostics, and flight data.
- **Target Integration** – Development of a pseudo-target to be used in flight testing of the drone swarm.
- **Network Protocol and Data Dissemination** – The implementation of a network protocol for ad-hoc drone communication and data dissemination throughout the drone swarm.

These subprojects were divvied amongst the FYP team, resulting in the project delegation outlined in Table 2. The first two projects were further split into different areas due to the volume of work required to complete them.

Table 2 - *Subproject Delegation*

Subprojects	Subproject Areas	Team Member Responsible
Network Protocol	VarDis Protocol Implementation	Raith Fullam[1]
	Drone Coordination Protocol API	Benjamin Ireland
Path Planning & Trajectory Following	Drone Lifecycle Manager	Finlay Cross
	Drone Control System	Benjamin Ireland
Drone Management and Visualisation Application	-	Ronniel Padua
Target Implementation	-	Se Hyun Kim

Project Background

The UAVs being used for the drone swarm are HolyBro X500 V2's equipped with a Pixhawk V5 flight controller (FCU) [1] using PX4 Autopilot software. The software developed as a part of this project is executed on an onboard companion computer, for this project a Raspberry Pi 4 was used.

[1] Raith Fullam was the projects Software Engineer. His work is not covered in this report but assessed separately.

The companion computer communicates through an ethernet link between the FCU and the companion computer. A diagram of this setup is depicted in Figure 1 adapted from [2]. A lightweight communication protocol known as MAVLink is used for communication between the onboard flight computer and neighbouring drones, in association with Robot Operating System broadcasts. More details about this protocol and ROS 2 can be found in [3] and [4].

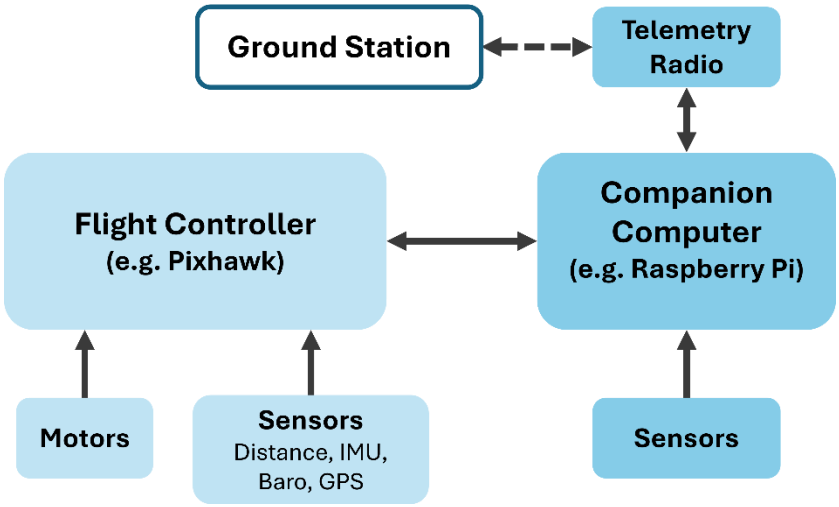

Figure 1 *- PX4 System Architecture with a Companion Computer. Adapted from [2], Fig. 2.*

The network protocol being implemented was predefined by project sponsors before the project began. The Drone Coordination Protocol (DCP), outlined in [5], explains how the drone swarm will eventually communicate. DCP consists of three sub-protocols:

- The Beaconing Protocol (BP).
- The State Reporting Protocol (SRP).
- The Variable Dissemination Protocol (VarDis).

BP is the overarching transmission protocol; it is used to encapsulate data from SRP and VarDis for reliable transmission. SRP manages the drone's state. Sharing information like, velocity, position, trajectory, etc. Whereas VarDis disseminates any arbitrary variable throughout drone swarm. This could be information such as drone IDs and software versions.

Currently the drone swarm does not communicate using DCP but using a combination of MAVLink and ROS 2 broadcasts. This is done so that the drone swarm can be developed and tested independently of the DCP protocol.

Benjamin Ireland – Drone Control System & Network Protocol Integration

1. Introduction

An important aspect of the drone swarm is the individual control system each drone employs to maintain their position both in and out of the swarm. As stated in the project introduction, my contribution to the project lies with the redesign and implementation of the drone control system. Additionally, contributions were made to the integration of the drone coordination protocol (DCP), the network protocol used for the communication of critical flight information and mission data. The overarching goals of this subproject were focused around improving the drone swarm's performance and scalability. In this section of the report, the research conducted, work undertaken, and results achieved throughout this subproject are discussed in length.

2. Background

2.1 Swarm Design & Subproject Goals

To gain some necessary context for this subproject, the previously developed drone swarm needed to be analysed. The wireless research centre (WRC) performed various field tests for the final year project team to demonstrate the previous drone swarm's operation. Initial inspection of the previous implementation, alongside discussions with the WRC, highlighted the issues prevalent with the previous drone control system. Table 3 provides an outline of the predefined project goals that related to this subproject and their associated subproject objectives.

Table 3 *- Control System & Swarm Design Objectives.*

Project Goals	**Subproject Objectives**
Scale up to larger drone swarms.	Remove inter-drone dependencies.
	Decouple individual drone trajectory from swarm formation activities.
Improve drone swarm performance.	Remove redundancy within the software.
	Reduce individual drone workload.
	Redesign control algorithm for improved performance.
Develop an exemplary tracking application.	Provide interchangeable targets for a variety of applications.
	Allow for fully automated tracking as well as manual swarm control.
Make swarm formations more easily configurable.	Simplify swarm configuration.
	Unify configuration format.
Develop and integrate DCP.	Provide an API for DCP.
	Develop software for DCP integration.
Implementation of formation algorithms	Develop an algorithm to prevent collisions when forming a swarm.

From these objectives it was clear that the most important aspects that needed to be addressed was the swarm's scalability and performance. Additionally, the new design had to provide reliable target tracking. Developing a metric for the performance of the previous implementation proved difficult. In all field tests the drones were unable to demonstrate swarming behaviour due to their inability to converge on their individual positions. Figure 2 depicts the 'leader-follower' structure that the previous drone control system enforced.

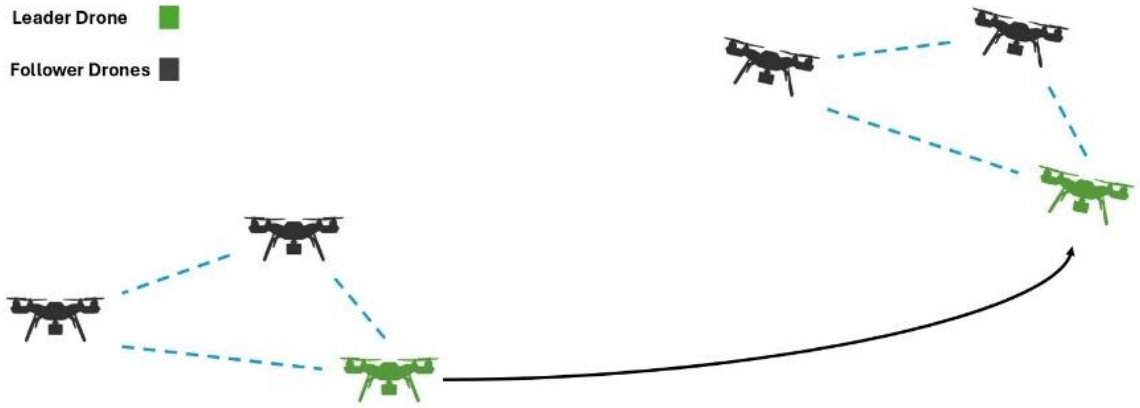

***Figure 2** - Previous Drone Swarm Flight Structure.*

This structure constricted the drone swarm's scalability and performance. Listed are the issues identified with the previous implementation:

- To maintain swarm formation, strict constraints were enforced between every drone. Each drone was required to define their constraints with respect to every other drone in the swarm, making configuration overly complex.
- Every additional drone decreased the speed of the control system. Each drone would instantiate a separately running PID controller for each additional constraints added.
- The PID controllers would work against each other when the drones were attempting to find their position in the swarm, preventing the drones from converging on their goal positions.
- The leader drone controls the whole swarm, this was identified as a single point of failure.

2.2 Control System Research

With the deficits of the drone swarm identified, research into commercially available and patented drone swarms was conducted. From this investigation it was found that commercially available drone swarm technology is limited. Most available swarms are designed for static predefined flights such as the light shows demonstrated by *Crostars* [6]. This is not particularly useful for target tracking. Alternatively, active patents exist for the use of drone swarms in wildfire management detailed in [7]. Although, as with commercially available options, these swarms also follow predefined paths.

Researching the technology that was used by the drone swarm provided insight into some of the limitations and applicable technologies for the drone swarm. Alternative control methods were investigated to address potential performance benefits. As outlined in [8], a type-2 fuzzy neural network could be used for trajectory control. This was found to be more effective than traditional PID control. However, after referring to the PX4 User Guide, it was outlined in [9] that the flight controller already employs a complex control architecture that provides stable drone control. The software developed on the companion computer would simply send position or velocity commands to the FCU to enact. As the companion computer only had to adjust the drone's velocity to maintain swarm formation, not keep the drone in a stable flight, it was determined using PID controllers would be suitable.
From this research the technology already employed by the WRC could complete multi-drone flights with suitable accuracy for drone swarming, and no additional hardware was required. Regardless, the flexibility of the Pixhawk FCU meant it could adapt to any additional requirements from the drone control system.

2.3 Network Protocol Research

VANETs provide inter-vehicle communications using WLAN and cellular technologies. Some examples of VANETS using drones include [10], [11]. In [11] a review of wireless communications amongst multi-UAV systems was conducted, concluding that drone-based communication systems are becoming increasingly popular. My contribution to the network protocol resides in creating an interface for drone coordination protocol. This API must be able to obtain information from both the state reporting protocol (SRP) and the variable dissemination protocol (VarDis).

An investigation into what is good practice in API design was conducted, so that a robust API could be developed for DCP. In [12], maxims for good practice in API development are defined. The most important maxims were itemised to be:

- APIs should be easy to use and hard to misuse.
- APIs should be self-documenting.
- API design is not a solitary activity.
- Documentation matters.

These claims are backed up in [13]. Both [12] and [13], discuss the importance of documentation and developing robust software if it is to be used by many different applications. These maxims were considered when API development took place.

3. Work Undertaken

To develop a solution that met the overarching project goals and subproject objectives, the following development steps were completed:

1) Swarm Redesign
2) Software Implementation

3) Simulation
4) Hardware Integration
5) Field testing & Validation
6) Network Protocol Integration

The work completed throughout these stages is documented below under their respective subheadings.

3.1 Swarm Redesign

To address the identified issues, it was decided that the drone swarm structure would be revisited. Through correspondence with the WRC, a new swarm structure was designed. To remove the coupling between each drone in the swarm, the leader-follower structure was replaced with the virtual reference point structure depicted in Figure 3.

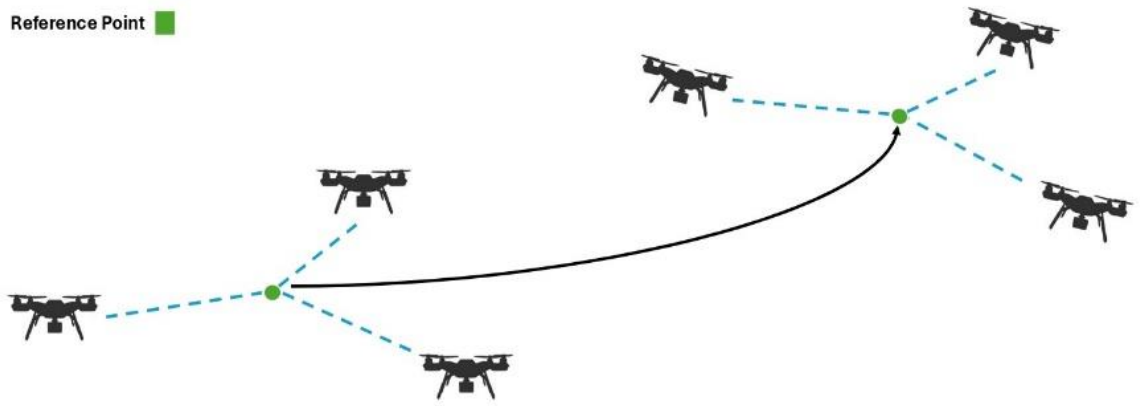

Figure 3 *- Virtual Reference Point Swarm Structure.*

This structure was designed to provide scalability and improved performance by allowing drones to have their own independence within the swarm. Instead of drones maintaining strict constraints with every swarmed drone, they now only had to maintain their position with respect to the reference point (RP). The benefits of this design are expressed as follows:

- Simpler Configuration – By removing inter-drone dependencies configuration only related to the drone's position, altitude, and heading w.r.t the reference point.
- Less Computation – With this design the number of PID control instances per drone would not scale with the number of drones. Improving not only the scalability but also the performance.
- Target Agnostic – By using a virtual reference point when the swarm needs to track a specified target, the reference point is simply superimposed over the target[2].

[2] This would be used to track any target if its position information can be interpreted. In practice, this may be a computational node that transmits its GPS location to the drones using DCP or, in following with the proposed application of the drone swarm, a harmonic signal from a tagged insect.

- Decoupling Trajectory & Formation – With drones independent from the swarm, they could now leave formation and act independently as required.

This new design did come with its own potential issues. As the drones no longer constrain themselves with respect to the position of other drones, they are blind to their neighbours' positions in the swarm. Collisions in high density swarms with non-ideal conditions would be likely. However, many potential solutions to this problem could be employed to eliminate this risk. Once DCP is integrated with the swarm, safety checks could be made against the flight safety data to stop the drones before a potential collision. As an intermediate precaution, PX4 autopilot already provides a traffic avoidance failsafe to be configured as outlined in [14].

Subsequently, a standard structure for drone configurations needed to be defined. This was broken down into three main flight constraints: position, heading, and altitude. Figure 4 represents an example configuration file alongside the physical mapping of these constraints.

```
"id": 1,
"takeoff_height": 3,
"alt_constraint": {
  "Min_altitude": 3,
  "height_displacement": 2,
  "gains": [2.2, 0, 0.2]
},
"pos_constraint": {
  "distance": 6,
  "angle": 45.0,
  "gains": [0.9, 0, 0.01]
},
"heading_constraint": {
  "angle": 0,
  "gains": [1, 0, 0]
}
```

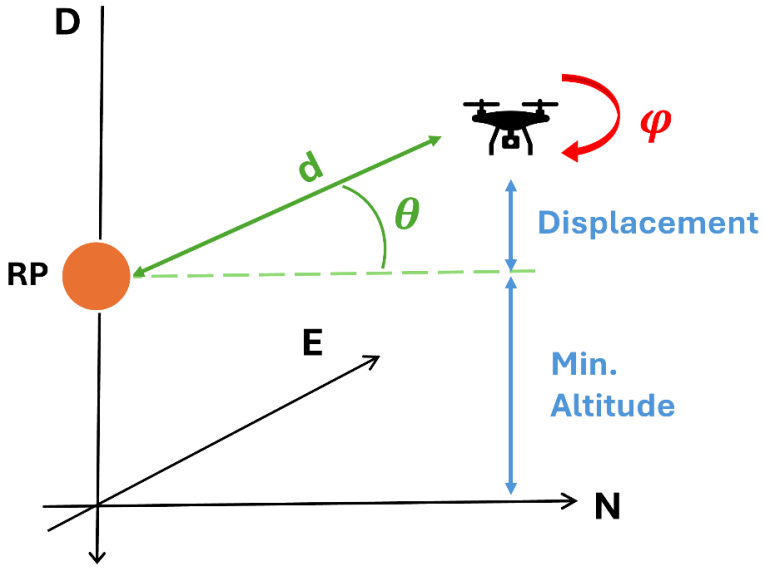

Figure 4 *- Drone Configuration*

The coordinate system adhered to for drone configuration is a north, east, down (NED) frame. This matches what is typical for aircraft. The altitude constraints define a drones' minimum altitude in addition to their height displacement above this. This is defined so that when target tracking, if the target/reference point was airborne the drone would maintain a height displacement above this but will never fly below its minimum altitude. The position constraint is defined as a distance and an angle w.r.t to the north and east axis, and the heading constraint is an angular displacement about the down axis w.r.t the heading of the reference point. For each of the constraints are associated proportional, integral, and derivative control gains.

3.2 Software Architecture

Now that a suitable swarm structure meeting the project requirements had been developed, a software implementation was required. Initially, the previous swarm software was reviewed to understand the software architecture in place. To improve performance, the new swarm software would be written in C++ and use the Robot Operating System (ROS 2) to communicate with the Pixhawk FCU and other drones. The drone swarm software was broken down into four processes:

- **Drone Controller Node –** Controls individual drone flight and position.
- **Reference Point Node –** Manages and propagates reference point position throughout the swarm.
- **Lifecycle Node –** Manages the drone swarm's missions, and state information. *Developed by Finlay Cross.*
- **Target Node –** A separate target dependant process used to provide the swarm with the targets global position information. *Developed by Se Hyun Kim.*

The system software relating directly to the drones control system were the drone control and reference point nodes, along with a target interface for the target node developed by Se Hyun Kim. The interconnections between these processes and the drones FCU are depicted in Figure 5.

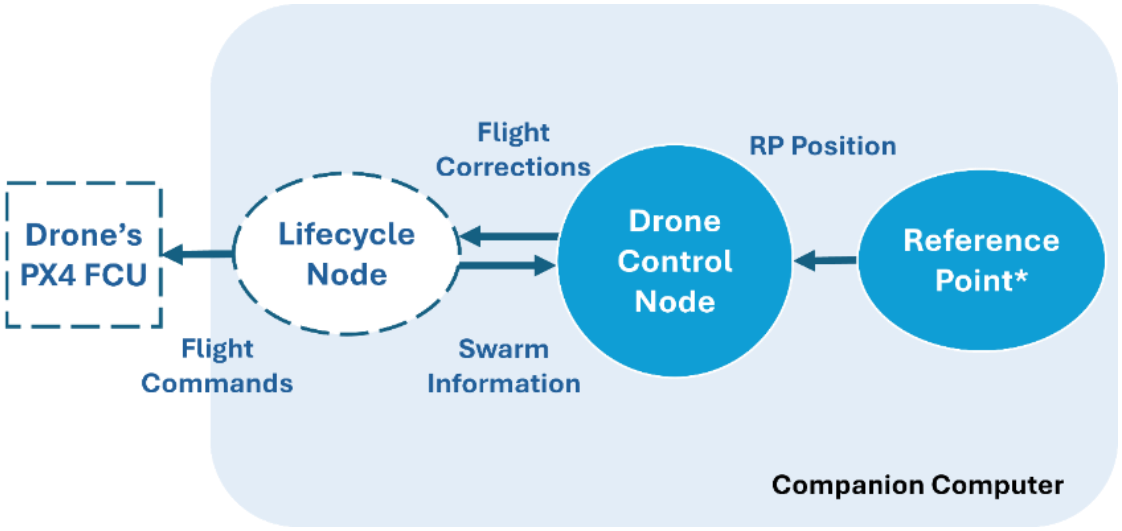

Figure 5- *System Software Relationships*

The drone control node has been designed so that it takes information about the swarm such as, swarm state (landed, take-off, track, etc.) and drone health updates, along with the position of the reference point to calculate individual flight corrections. The flight corrections computed are in terms of velocities with respect to a NED coordinate frame. These are then passed through the lifecycle node and passed onto the Pixhawk FCU to interpret and enact. Each drone is running an instance of both the control node and the lifecycle node, only one of the drones run the reference point node, propagating information about the RP throughout the swarm. Each of these processes

come together to manage an individual drone's flight as a part of the swarm. The specifics of their operation are outlined as follows.

3.2.1 Drone Control Node Implementation

The Control Node is comprised of a controller class and three primary supporting classes. This structure is shown in Figure 6, a conceptual UML class diagram. The three supporting classes are outlined as follows:

- **Pose –** All position and heading information of the drone is monitored using this class. Pose has two derived classes, Global and Local Pose. These were more specific classes developed for a GPS pose and a Local NED pose.
- **Constraints –** This class was used to parse and manage individual drone configurations for the drone.
- **PID Manager –** Used by the controller to find the drones goal position w.r.t the reference point computing flight corrections to match the goal position, from the constraints outlined in the drone's configuration. A separate PID controller is instantiated for each constraint: altitude, position, and heading.

The controller listens for prompts from the lifecycle node to determine when automated control is needed and the type of control that it needs to implement. For example, take-off and landing are managed by the FCU, but control needs to be applied for swarm movement and formation forming.

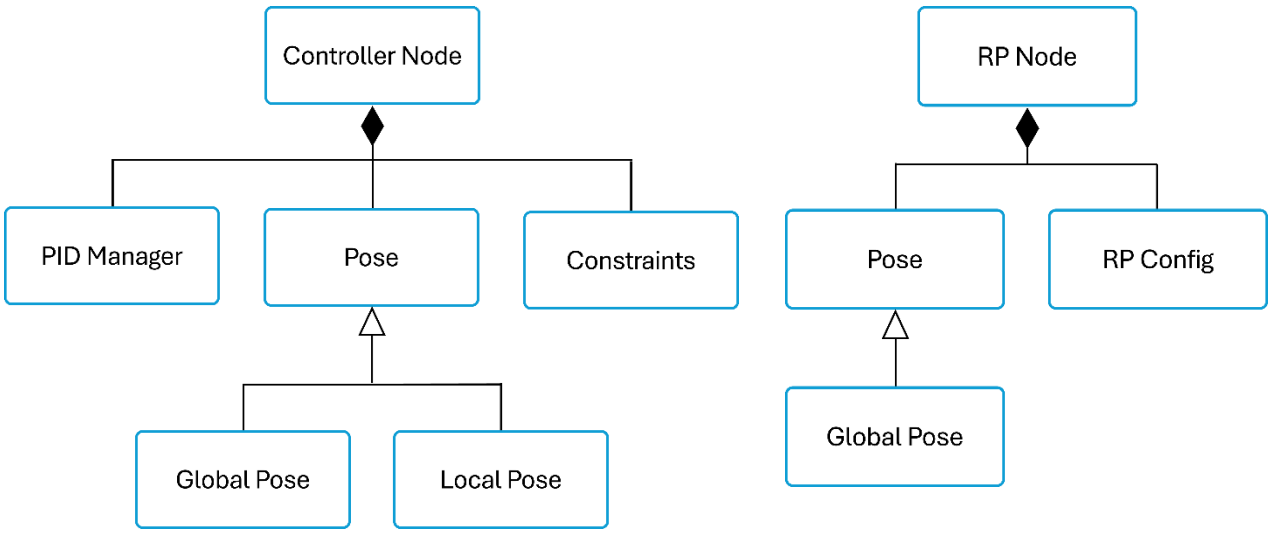

***Figure 6** - Conceptual UML Diagrams for the RP and Controller Nodes.*

3.2.2 Reference Point Implementation

The reference point node uses the drones GPS information along with the configuration it provides to define the location of the virtual reference point. This information is broadcasted throughout the drone swarm until all drones are agreed upon its location. Upon start up the reference point is defined as the origin of the local coordinate drone. The RP will remain stationary until the drone

swarm enters a state where it is moved either through manual manipulation or because it is tracking the target.

As the RP node is only operating on one of the drones, in the event this drone becomes inactive, the swarm will remain stationary in formation about the last known position of the reference point. The drone swarm will become aware of the loss of the reference point, and at this point it is left up to the user to determine whether the drones return home, land or elect a new drone to operate the reference point.

3.2.3 Target Interface

The target interface implements a simple abstract class that has one method, publish position. Whichever target node is being used must enforce this interface publishing the target position to the reference point for it to mimic.

3.3 Simulation Testing

Throughout software implementation, the drone swarm was tested in Gazebo [15]. Gazebo is the simulation software that PX4 recommends for simulating automated drone flight. PX4 provides a model for the Holybro X500 drones used in real world testing. This allowed for each software implementation to be tested in Gazebo to identify any prevalent issues with the swarm. Initially, this tested the start-up sequence of the swarm defined as follows: configuration, take-off, and rendezvous (formation forming). When formation was found the drones would hover in place awaiting further commands. Once swarm start-up was fully tested in the simulation, a virtual target node was developed to test the target tracking of the drone swarm to model a physical target. Allowing for dynamic testing of the drone swarms control system.

3.4 Hardware Integration

With a fully functional simulation of the drone swarm, development moved to integration with the physical hardware. The following steps were taken to complete the hardware integration:

1) Pixhawk FCU Firmware Upgrades v1.10 -> v1.14.
2) Drone Flight & Safety Configuration.
3) Drone Sensor Calibration.
4) Companion Computer Set-up.
5) FCU -> Companion Computer Communication.

Majority of these steps related to upgrading the system software to their most recent stable releases. These upgrades required many additional components to be replaced with their newly supported counterparts. Additionally, it was important to make sure that safety procedures were upheld in flight tests. Multiple failsafes were configured on the drones in the event of communication lost, battery depletion, or other possible issues that could arise during flight. As a part of the Federal Aviation Associations flight regulations, explained in [16], every drone being used in the field must have a pilot capable of taking manual control of the drone using an RC controller. To minimise the skill requirements needed, the controllers were configured such that lifting control stick would put the drone into manual control where the drone would hover in place. After which

at the press of a button the drone would land. This form of guided flight allows the drones to be piloted with ease, where little to no training was required. An outline of the RC controller's configuration is outlined in Appendix A.

3.5 Field Testing & Validation

Field testing begun once the behaviour exhibited in Gazebo was stable. Field tests were conducted by following flight test plans. Flight plans consisted of the safety procedures that needed to be undertaken before flight could ensue, the functionality that was to be tested, and the observations that were made.

The initial field tests tested the functionality of a single drone. This started with simple formation forming to see if the drone was able to find its position in the swarm w.r.t the virtual reference point. With this validated the physical target was introduced. This sequence was repeated for two drones in a parallel formation, and finally a three-drone swarm in a triangle formation as shown in Figure 7. Logs of every flight test were recorded by each drone. These were used to validate flight behaviour.

***Figure 7** - Flight Test of Drones in a Triangular Formation.*

3.6 Network Protocol Integration

Unfortunately, much of the network protocol integration was not completed. This is due to the protocol not being in a state where testing had confirmed the operation of DCP with reliable confidence. A roadmap has been provided in Appendix B that outlines how DCP could be integrated and the issues that could possibly arise with the current implementation.

4. Discussion

This subproject aimed to develop a drone swarm that performed better than the previous iteration, while removing inter-drone coupling that caused scalability of the swarm to be limited. The objectives that were outlined in Table 3, summarise what had aimed to be achieved by the conclusion of the project.

4.1 Results

Using the Gazebo simulation the formation forming, and target tracking of the drone swarm was successfully demonstrated. The simulation model provided was an accurate representation of the

behaviour observed in field tests. From the flight tests conducted a two-drone swarm was successfully demonstrated tracking of a target, and the formation forming of a three-drone swarm. The logs retrieved for the flight tests on the drone's contained information on the goal position that the drone was attempting to maintain, unfortunately in successful flight tests logging of the control error was disabled. Ideally, it was hoped that the visualisation and management application would have been implemented and integrated such that this information could be represented in a meaningful way. Additionally, it was impossible to compare against the previous design from flight recordings as the previous swarm was never seen flying in formation by the current FYP team.

Tracking was never achieved with a three-drone swarm. The issues that arose throughout flight testing were related to hardware and configuration issues. The PX4 mission planning software used to communicate with the drones would often crash or lose connection with drones amid flight testing, causing the drones to land as a precaution. These issues could not be known in advance as there are hundreds of preconfigured flight parameters that are set on the drones, most of which are not set with multi-drone in mind. If the PX4 configuration issues were remedied the control system would be able to operate unhindered and replicate what is seen in simulation.

4.2 Review

The reference point structure that the drone swarm adheres to has improved the scalability of the drone swarm. By removing the inter-drone dependencies, having individual drones unaware of the configuration of the whole swarm has allowed additional drones to be added with ease. The RP structure meant that the drones no longer needed to maintain their position relative to all other drones in the swarm, removing redundant PID controller instantiation and allowing drones to converge on their swarm positions. Multiple iterations of the drone swarm structure and the control system were developed. Listed as follows are the iterations developed:

1) Design that introduced the virtual RP in place for a leader drone.
2) Dependencies between drones were removed from swarm configurations.
3) The control system was developed using a global reference frame using GPS coordinates that had to be converted to a local position in meters.
4) Synchronising drones within the swarm to have the same cartesian reference frame allowed for local coordinates to be used minimising conversion error.

The final iteration of the drone swarms control system and swarm structure achieved majority of the original subproject objectives. The current implementation can provide suitable target tracking and formation forming. Most importantly the software developed is well documented able to be enhanced by future project teams. Even though a three-drone swarm tracking an arbitrary target was not achieved, the new design solved many of the issues of the previous design. A two-drone swarm was demonstrated in the field tracking an arbitrary target, and with the PX4 FCU configuration revised, there is no reason that the swarm would not work with more drones due to the separation of drone trajectory and swarm formation.

Finlay Cross – Lifecycle Manger & Automation

1. Introduction

The lifecycle control of a drone is a very important feature of a drone swarm, being the main control for each mission and drone health. My contribution was largely based around developing a comprehensive lifecycle manager in software for the drone swarms. While developing this, I contributed to many more tasks; particularly to the development and porting of the drone software, the hardware integration, and the automation of drone setup. This subproject was mainly focused on improving the software in all facets, including the scalability, time performance, and structure. I will outline the research conducted, the previous implementation of the software, the process of improvements and other tasks completed during development, and then finally the end results of this sub-project.

2. Background

2.1 Sub-Project Goals

My primary contribution focused on developing a comprehensive lifecycle manager for the drone swarm software. This sub-project aimed to improve the software in all facets, including scalability, performance, and structure. The specific objectives were aligned with the overarching project goals, as summarized below:

Table 4 *- Lifecycle Project Goals*

Project Goals	**Sub-Project Objectives**
Extend previous drone swarm implementation to achieve successful multi-drone flight tests	Improve software scalability, performance, and structure
Improve scalability and performance of the swarm	Remove inter-drone dependencies to enhance scalability
Make swarm formations more easily configurable	Simplify swarm configuration and unify configuration formats
Implement formation algorithms to prevent collisions	Develop algorithms to prevent collisions during swarm formation
Streamline hardware integration and setup processes	Automate drone setup and configuration to reduce manual intervention and potential errors
Provide flexibility for mission expansion and scalability	Design an event-based lifecycle system allowing easy addition of new states and missions without major code changes

2.2 Previous Work

The project was analysed to gain an understanding of the structure of the current software, the implementation of the lifecycle controller, and any improvements that could be made. This software was all implemented in Python, utilising ROS1 and a Mavros for the communication protocol between the PX4 and the Raspberry Pi.

2.2.1 ROS

ROS1, Robot Operating System 1, is the software framework that is commonly used to develop software for robotic systems. Its main functionality is based around a publisher-subscriber messaging system. While researching ROS, we found that ROS 1 was planned to be deprecated in 2025 [17] [18] and replaced by ROS 2, originally released in 2017[19]. ROS 2 improves by enhancing the publisher-subscriber messaging system established in ROS 1 while introducing a more modular and flexible architecture with significant performance increase.

2.2.2 MAVLINK

As stated in the introduction, the FCU used was a Pixhawk PX4 FCU. This FCU communicated with the companion computer using the MAVLink protocol through the MAVROS [20] wrapper. Investigating the official documentation for PX4 FCU, PX4 Autopilot [21], we found that MAVROS was no longer the main officially supported method for communication. The new method uses a new middleware protocol, µXRCE-DDS, that translates the PX4 uORB MAVLINK messages into ROS2 DDS subscribe topics [22]. This involves setting up a, µXRCE-DDS Agent on the companion computer. The main reason for this change is ROS2 uses communication method based on Data Distribution Service (DDS) [23]. DDS allows for efficient and scalable communication, and improves real-time performance over the older modified TCP socket used by ROS1 [23]. As a consequence, because MAVROS was designed for ROS 1 communication paradigms [24], it does not seamless integrate with ROS 2's new DDS communication system.

2.2.3 Lifecycle Structure

The lifecycle control was a polling system implemented in a Finite State Machine. Depending on what state the drone was in, different variables and information would be polled by each drone. This included the battery health, drone synchronisation state, and other drone's dropout checking. This did not utilise the event-based possibilities of the publisher subscriber system. The states were hard coded in Python, which is not inherently a bad thing.

3. Work Undertaken

3.1 Software Porting

As mentioned in Benjamins section, he mentions how the performance concerns of the controller mean the software was ported to C++. There were originally plans to only port the controller loop to C++, as this is the time critical section because of the PID error calculations. However, after researching the implementation of the previous work and finding ROS1 was to be deprecated and MAVROS was no longer the primary support of the Pixhawk company, we decided to investigate porting all of the software. This would be done while simultaneously updating to the newer ROS2 and the µXRCE-DDS communication method. This proposal can be seen in Figure 8.

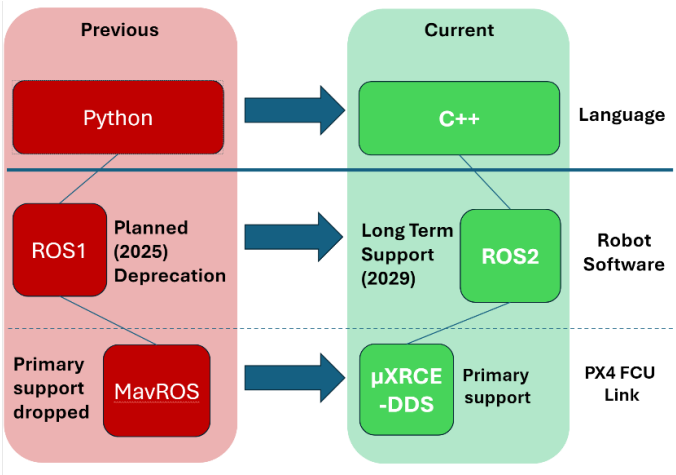

Figure 8 *- Proposed Software Update*

One of the major considerations was the acknowledgement that this project is likely to be continued into the future. Updating to the longer term and officially supported frameworks would be beneficial for whoever continued this project and would make further development easier. Another consideration was consistency. Having two different software languages in the same source code would make development more complicated. It was concluded that porting and updating all of the software would be the most effective option. Instead of having to learn two different implementations of ROS2 and µXRCE-DDS middleware, one in Python and one in C++, we were able to focus on just implementing the latter. This also improved the collaboration opportunity for me working on the Lifecycle and on Benjamin on the Controller.

3.2 Lifecycle Redesign

The main contribution to this project was the development of the Lifecycle controller. As we concluded that we would be porting the software to C++ from Python, while changing the frameworks used within the software, there was an opportunity to address some of the concerns I had with the previous lifecycle controller. The previous implementation utilised a polling system, which was functional, but did not utilise the full capabilities of the ROS2 framework. The goals

of the project was to make sure that the drone swarms are easily scalable. Part of this is to have a highly configurable lifecycle controller and mission manager to allow for rapid changes and ease of mission expansion. To achieve this, the software structure from Figure 9 was implemented, where the main logic of the system would be event based and have no internal loop. There is one base class, Lifecycle, and then two types of sub-classes. There are the internal classes which are dependencies of Lifecycle and hold specific functions that Lifecycle calls. Then there are the friend classes, which are independent of Lifecycle and can be changed and developed without having to change any internal logic in Lifecycle class. The benefit of the friend classes is that they can be plugged in and out with different implementations without changes to the internal software.

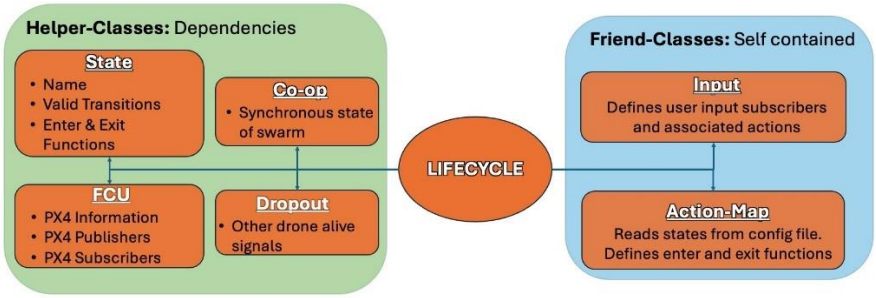

***Figure 9** - Lifecycle Class Diagram*

3.2.1 State

This class is used to hold information about each state. This information is: The name of the state, whether it has off or onboard control, on enter and exit functions, and states that are valid to transition to. Valid Transitions was an important feature to add, as this protects the system from unintended state changes that could cause unknown swarm behaviour. A state will enter using the *transition* function in Lifecycle. The function attempts to change the current state to the next state if the current state exists and the transition is either valid or forced; if successful, it calls the exit functions of the current state, changes current state, and then calls the enter functions of the new state.

By holding pointers to the on enter and on exit functions, the need for any state checking logic blocks or loops was removed. This means that new states can easily be added without having to change much of the internal code.

3.2.2 FCU

This class is the communication point with the PX4 FCU. In this class is all of the subscriber and publisher definitions that use the µXRCE-DDS ROS bridge. Having this central communication point means that all specifics of message types and the µXRCE-DDS definitions are contained in a single location, making the code more portable to other FCUs types, or further firmware updates within the µXRCE-DDS ROS bridge. There were 5 messages that were important to our project. These are as follows:

- vehicle_status subscriber: this message holds all of the info about the status of the drone: the current flight mode, armed status, battery status, and the general health status of the drone.
- vehicle_command publisher: This is a major message type that sends message for vehicle commands, these are: arming the drone, changing state (takeoff, flight, hold, land), and setting the origin point of the drones' local frame.
- vehicle_command_ack subscriber: This is the message sent back to acknowledge a vehicle command that is sent. It returns success or error with a message of what the error was.
- offboard_control_mode publisher: This message is required to be received by the FCU at a rate of at least 2 Hz. This lets the FCU know that an offboard computer is in charge of the drones current control.
- trajectory_setpoint publisher: This is the message that tells the FCU what the trajectory of the drone is, and is what the controller utilises to send its PID calculated trajectory. This message is also required to be sent at regular intervals.

3.2.3 Cooperative

This class holds the logic for checking if the drones are in a cooperative state, for example all within range of their individual point. This is to ensure the swarm is all in the same state before moving to any major move methods. Currently, however, this is not used, and this logic is being sent by Benjamin's Controller. This class still exists as it will be developed more once the new network communication protocol is integrated. The cooperative class is one of two classes that requires knowledge of the other drones, which is currently done with ROS2 subscribers but will be replaced with the new communication method.

3.2.4 Dropout

The dropout class checks on the life of other drones in the swarm. This is implemented using the service – client feature available on ROS2. Each drone sends out a client request to a service set up on each drone. It will then wait for a response for a set amount of time. If no response is received back, the drone will flag this drone as having dropped out. To handle a dropout, the swarm is paused as it is assumed that there has been a fatal error.

3.2.5 Input

Input is a friend class that defines all of the access points to the drone swarm during runtime. Currently defined inside are subscribers that listen for messages about state changes, PID value changes, and custom command strings for debugging manual controls. This class is a friend class of Lifecycle and therefore can interact with the private and protected variables. The debugging manual control and runtime PID tuning was particularly useful during testing, as we could force the drone into states and change its control behaviour during runtime without having to reset the swarm. As stated, input is a friend class of Lifecycle, not a dependency, meaning it can be swapped in and out with different implementations very easily.

3.2.6 Action-Map

Action map is another friend class, and it implements the state and action creation. In this context action refers to the enter-exit functions, and hence action-map refers to the mapping of function pointers to each state. This class currently implements this by reading states from a configuration YAML file. These YAML files define the states; their names, offboard mode, enter and exit functions, and the valid transitions. This implementation utilises YAMLS as it streamlines running different missions – where the software does not need to be rebuilt from source. Instead, the launch file just needs to define a different path to another mission YAML. The enter and exit functions have to be explicitly mapped to a string in this class, which is sim

3.2.7 Lifecycle

Lifecycle is the base class. It creates all of the subclasses and implements their logic, such as what defines a drone drop out for Dropout, or what cooperative state is being checked by each instance of the cooperative class. Internally it also defines Input and Action-Map as friend classes, allowing these classes to have access to the private members of lifecycle.

3.3 Designed Mission

The mission state machine that is currently used can be seen in Figure 10. Configure is the starting state. Once a user command *start* is received, the state switches to start. In the start state, there will be continuous attempts to arm the drone. Once a acknowledgement is received, the state will switch to Takeoff. Once at take-off height, the drones will move to rendezvous, where they will now use Offboard control with our PID control to move to their starting position in the swarm. Each drone's state changes to move when in full formation, which is the main target tracking state. At any time during *Takeoff, Rendezvous,* and *Move,* the drones can enter PX4 Hold or Pause. PX4 Hold utilises the on board hold control to keep the drone in place. Pause pauses target tracking but keeps control on the Raspberry Pi's. Anytime other than *Configure* and *Start* drones can enter the *Land* or *Return* home states. Only after returned home or landing can the drones enter the final finishing state Finish. For simplicity, the diagram joins the arrows to PX4 Hold and Pause, and Land and Return, but these are still separate connections and states.

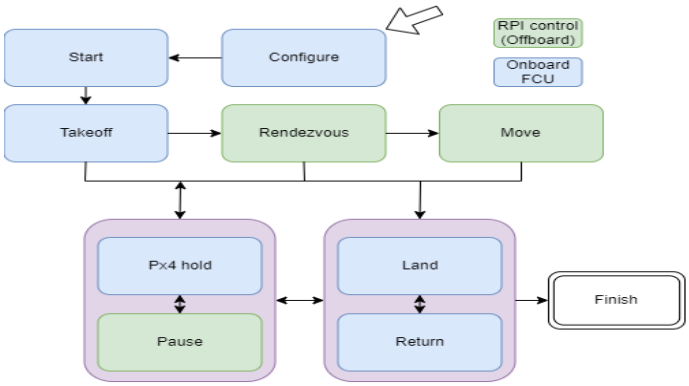

Figure 10 *- Drone Mission*

3.4 Swarm Configuration App Proof of Concept

To test and enforce the scalability of our control system, a proof-of-concept drone configuration CustomTkinter application was created, seen in Figure 11. The new configuration is developed to highlight the scalability the new configuration outlined in Benjamins Section. Drone nodes can be added using the button *Add Node* and have their [x,y,z] coordinates adjusted with the sliders. A polygon of drones could also be created by defining the number of points and a radius, which would create a number of evenly spaced drones corresponding to that polygon. The frame of this configuration is with respect to the target, meaning [0,0,0] is the position of the target. The created configuration can then be saved with a name, which would create a directory and save a drone configuration file for each drone in the shape.

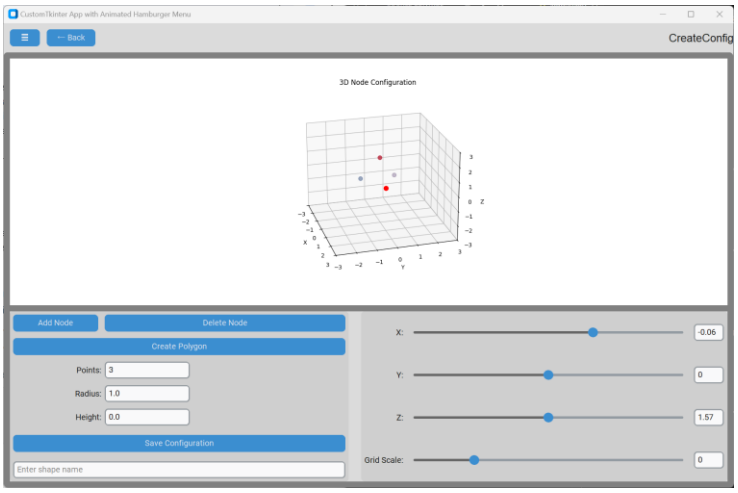

Figure 11 *- Drone Configuration App*

3.5 Manual Launching

To launch the drones, the µXRCE-DDS agent needs to be started on the companion computer as well as launching all of the ROS nodes for controller, lifecycle, and sometimes the reference node. As one of the overarching project goals was to ensure a scalable swarm, this process is automated per drone. A launch bash script was created that sets up separate subprocesses, and then launches the agent and the ROS package. It is in this script that the path to the configuration file the drone uses is passed. Unfortunately, even though this manual drone setup and launch script is stable and functions seamlessly in the field, it requires to be manually called on each drone through separate ssh terminals which is tedious.

A separate but similar functioning launch script was created for the simulation on a single computer as well, which can launch all from one terminal. This launches all of the simulation drones required by the selected configuration folder.

3.6 Automated Drone Setup

Setting up a new drone was a tedious process. Each new companion computer requires the following to be installed on first launch:curl, nlohmann-json3-dev, ros-humble, python3-rosdep, libreadline-dev, unzip, network-manager, python3-pip, micro-xrce-dds-agent, and many more.

Additionally, the drones need a new network configuration setup to allow ethernet communication between itself and the PX4. For this reason, an *install_on_new_drone.sh* script was written. Each companion computer is flashed with Ubuntu 22.04 and then has the install script downloaded either over SSH or USB stick. The script requires an internet connection, but once connected will automatically install and setup all requirements mentioned above.

Part of the setup of each new drone was configuring the PX4 FCU by installing new firmware. Once the firmware is installed, parameters to set up µXRCE-DDS and other configuration parameters had to be set using their software *QgroundControl.* This process was tedious and so it has now been automated. This was achieved by cloning official the open source Github repository into our own UC Gitlab. Once in our own repository, all custom changes and configurations were made to the firmware and saved. This means that now other than installing the new firmware, there is no manual setup for the new PX4 FCU configuration. A flow diagram of how this is done is shown in Figure 12.

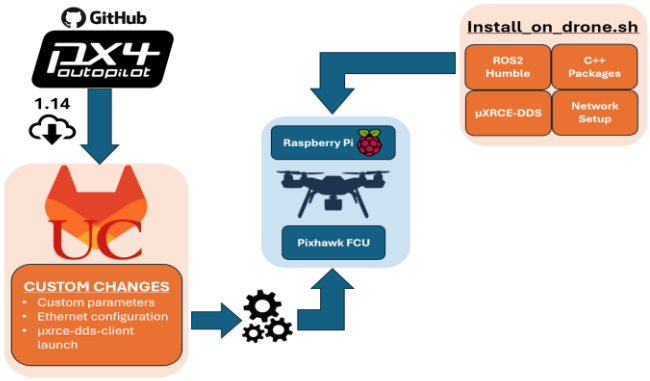

***Figure 12** - Automatic Install Flow Chart*

3.7 Command Line Interface

A Command Line Interface (CLI) has been developed to interact with the drones during run time. This is another ROS2 node called *User Input* and is run on a host computer that is connected to the same network as the drones. This CLI interacts with the Input class described above and must be developed to match to ensure the CLI is interacting correctly with the input. A screenshot of the interface can be seen in Figure 13.

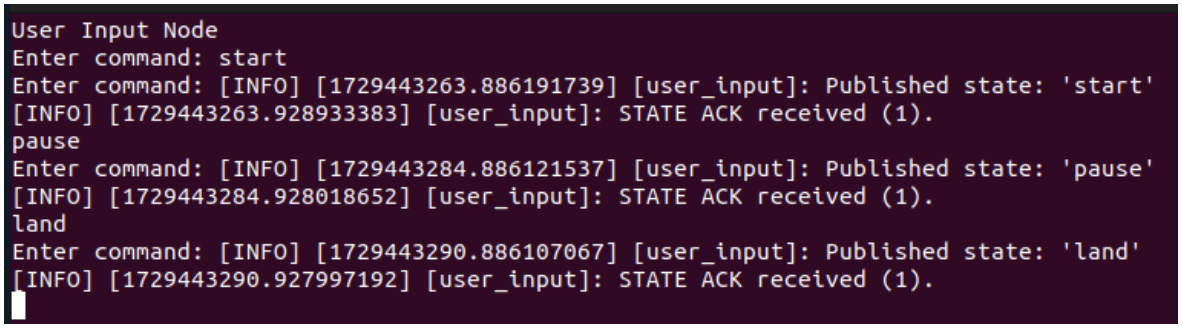

***Figure 13** - Command Line Interface*

To ensure comfortable user experience, the CLI was designed to match what is expected with normal terminal interaction, such as history scrolling and insert movement. This is achieved with the *readline.h* package. The logic of the CLI is essentially a string parser that looks for patterns. Some single word commands are also available for the common commands, such as start and land. "{Address}! --{command} {Arguments}" is the generic pattern for a typical message, with Appendix D showing a subset of the possible inputs. The messages are addressable to either individual or all drones

Multithreading is also implemented so the front end CLI does not block the backend message sending. This is because some messages continue to send until an acknowledgement is received from the drones. Multithreading lets this happen while still being able to type in the CLI.

This command line interface has been the main form of interacting with the drones and is how we launch and land the drones. The command option also helps significantly with debugging and run time changes for testing.

4. Discussion

I was able to meet most of the design requirements laid out at the start of this section. Unfortunately, the formation algorithms were unable to be completed. Below is a review of the major components of this sub-project.

4.1 Software Update Review

To determine whether the redesign and porting was worth, our achievements, time for development, and whether there was a positive trade off. The rebuild using ROS2 and µXRCE-DDS extended the software development time significantly compared to our original Gantt chart (Appendix E), as not only did we need to understand how to port the software to C++, but to also port the messages from MAVROS to µXRCE-DDS. The message porting was less straight forward than the software porting and took more time than expected. The time delay was particularly because we found there were deprecated messages or slight differences in message content. The main time sync was learning where to find all information on the official documentation. However, after the

initial learning curve, having the official documentation and primary support meant that further development was sped up. To update to ROS2 was far more straightforward. ROS2 framework is generally an improvement on ROS1 not a redesign. As it's an improvement, most of the functionality and concepts were the same. Even given these hurdles, we were able to have a working simulation quickly.

The new network protocol was not ready to integrate with our software by the time we would have needed it to be. Because of this, our timeline was able to be updated. With one less project to complete, the lost time in software development did not affect our achievements.

The majority of the lost time came from setting up the new firmware of the drone. Admittedly, this was unforeseen. The process of figuring out which parameters needed to be changed took far longer than expected. This is because there was no clear single source for documentation on this aspect. This meant we would only discover issues with drone parameters during flight tests, which wasted a lot of valuable flight test afternoons.

Ultimately, the decision to not only port to C++, but update the framework seems like the correct decision. I believe that the benefits that future developers will have from the newer systems far outweigh the lost time that our group encountered. We were still able to achieve a successful multi-drone tracking flight which was the ultimate goal of this project, showing that we were still able to complete a significant amount of development.

4.2 Lifecycle Review

The new lifecycle implementation is a good one. The change to an event-based system makes the system easier to develop in the future. Instead of having to edit the code significantly and change the logic blocks and for loops, individual events can be introduced without interaction with existing code. New events can be defined anywhere and only need to publish a message that a subscriber in lifecycle listens to, and a callback function that relates to the lifecycle

The new state system makes the drone lifecycles easier to configure and design. Each state has its own enter and exit functions, and valid state transitions defined. This ensures that drones behave predictably and maintain operational safety. Additionally, this modular approach simplifies debugging and testing because the code itself should not have to be inspected too closely.

Overall, the Lifecycle redesign meets the sub-project objectives with a heavy focus on enhancing scalability, adaptability, and simplifying configuration. By introducing event-driven architecture we created a robust foundation for future development, making the system adaptable to a wide range of missions and applications. The implementation we ended up with ensures that the drone swarm software stays stable and usable, while also being capable of evolving to meet the requirements of future different projects.

Se Hyun Kim – Target Implementation

1. Introduction/Background

As of the beginning of the year, the drone swarm application had no real-time tracking targets but rather used a virtual set point as its target with a preplanned path, such as moving in a circle, sinusoidal wave, square, etc. This had limited the movements of the drone swarm to a linear motion and hence is not suitable to properly test the drone swarm integration. The target subproject aims to create a real-time tracking target which the drone swarm would track during flight tests to better examine and evaluate the movement, behaviour, and robustness of the drone swarm. It is also important to state, the final implementation of the drone swarm application already has a target in mind to use for the final implementation, however this target technology is currently under development by the UC Forestry Research Department [25] and does not have a completion date for the foreseeable future.

***Table 5** - Requirement List For Real-Time Tracking Target*

ID.	Requirement
R1	Target needs to be real-time tracking with the capability of dynamic motion simulation
R2	Target needs to be able to accurately determine its position
R3	Target needs to be able to communicate its position to the drone swarm within operating swarm range with ease
R4	Target needs to be able to update its position at a high frequency to the drone swarm
R5	Target needs to be suited to the testing environment
R6	Target needs to have a battery life equal or greater than the operation life span of a drone swarm
R7	Target needs to be within the budget of $350

To understand what was required for the target, a list of necessary conditions was listed out as seen in Table 5. This covers the necessary requirements which need to be met to create a robust real-time tracking target. The justification for each requirement can be seen below.

R1: Ensures the swarm's tracking algorithms are stress-tested and adaptable to different movement patterns, such as erratic or unpredictable motions, reflecting real-world scenarios.

R2: Accurate position data is important for ensuring the validity and proper testing of the drone swarm's tracking ability. Without accurate positioning from the target, the error will flow over to the behaviour of the drone swarm making it harder to analyse the proper behaviour of the swarm. A high precision within 3 metres is important to ensure a proper accurate testing.

R3: This ensures that the information the drone swarm is able to update its position in relation with the target. The target should be able to communicate at a range of up to 50 m.

R4: This ensures that the swarm is able to maintain smooth and responsive tracking of the target's movements. A high update frequency (e.g., 10 Hz or higher) is expected to achieve this.

R5: This ensures that the target is suitable for use in the environment conditions. The field tests will be done out in Ilam fields which is an open space with no obstacles in the air. The field tests will also only be conducted when there is no rain, or excessive wind as advised by our sponsor.

R6: This ensures the target remains operational throughout the entire swarm mission. Losing battery life during a flight test would cause malfunctioning / collateral damage to the drown swarm behaviour at worst and cause inconvenience at best.

R7: This is the budget given to the FYP project to use to purchase new items.

2. Work Undertaken

2.1 Real-Time Tracking Target Research

The approach to finding the suitable real-time target method was to list different target tracking methods and compare them against the requirement table listed in Table 5. The different methods researched were using Wi-Fi RTLS (Real-Time Location System), active RFID (Radio-Frequency Identification), and GPS (Global Positioning System) receiver.

To give a brief summary of how each target implementation method works,

Wi-Fi RTLS uses virtual tags and anchors to determine a position. This can be achieved using various methods such as checking the received signal strength and timing the signal travel distance. As the HolyBro X500 drones are installed with Wi-Fi capabilities, in theory, a drone should be able to act as an anchor to determine positions of a target.

Active RFID uses a battery powered tag which periodically sends signals, and the readers collects the data. The signal of the tags can have frequencies at ultra-high frequencies which allow the signal to travel over 100 meters.

GPS receiver uses satellite signals which broadcast their own position and time. By using trilateration, the GPS receiver is able to calculate its position by reading multiple satellite locations at once.

Table 6 - Determining Suitable Target Implementation Method

	GPS Receiver	Wi-Fi RTLS	Active RFID
R1	✓	✓	✓
R2	✓	✓	x
R3	✓	x	✓
R4	✓	✓	✓
R5	✓	✓	✓
R6	✓	✓	✓

R7	✓	✓	x

By analysing different target tracking methods against the requirements listed earlier as seen in Table 6, the usage of Wi-Fi RTLS and Active RFID is incompatible with what was necessary to create the target. Wi-Fi RTLS is not suitable for the usage of this drone swarm application due to the importance of distance between the tag and reader. The performance drop-off in determining position for longer ranges meant the Wi-Fi RTLS was not capable of determining an accurate position. Active RFID has an accuracy of around 5-10 meters which means that this implementation is also not suitable for the target application. Active RFID also has a linearly increasing cost (1 reader per drone), which quickly reaches the budget limit with an increasing number of drones. For example, the cheapest active RFID suitable for this application had a price of $260 NZD per reader [26]. The most suitable method for using as the target was a GPS receiver. Initially, the SiRF Star IV GPS receiver was used which was provided by the WRC to investigate as they had one lying around.

2.2 Sirf Star IV GPS receiver

The first thing done with the Sirf Star IV GPS receiver, was to look through its datasheet [27] and make sure that the specifications were up to par with the requirements. Although at this time I had thought that all the specifications met what the requirements needed, I had a severe oversight which would mean that my GPS would need to be upgraded. However, at this stage I had not realized and continued to work with the Sirf Star IV GPS.

With the SiRF STAR IV GPS being selected as the target tracking method, the GPS driver has to be installed which can be found on its manufacturer's website [27]. By also downloading a compatible GPS software, it is used to test the functionaltiy alongside the performance of the Sirf Star. The software used is called VisualGPSView [28] and has many features such as azimuth and elevation plot, scatter plot, signal quality, position plot, and most importantly NMEA data support which is the standard GPS data format and will be used to transmit for the drones to be used. The GPS receiver was tested inside and outside buildings, near structures and out in open fields. The received coordinates can be written into google maps to check the location. Setup, software, and coordinates given to Google maps can be seen in Figure 14.

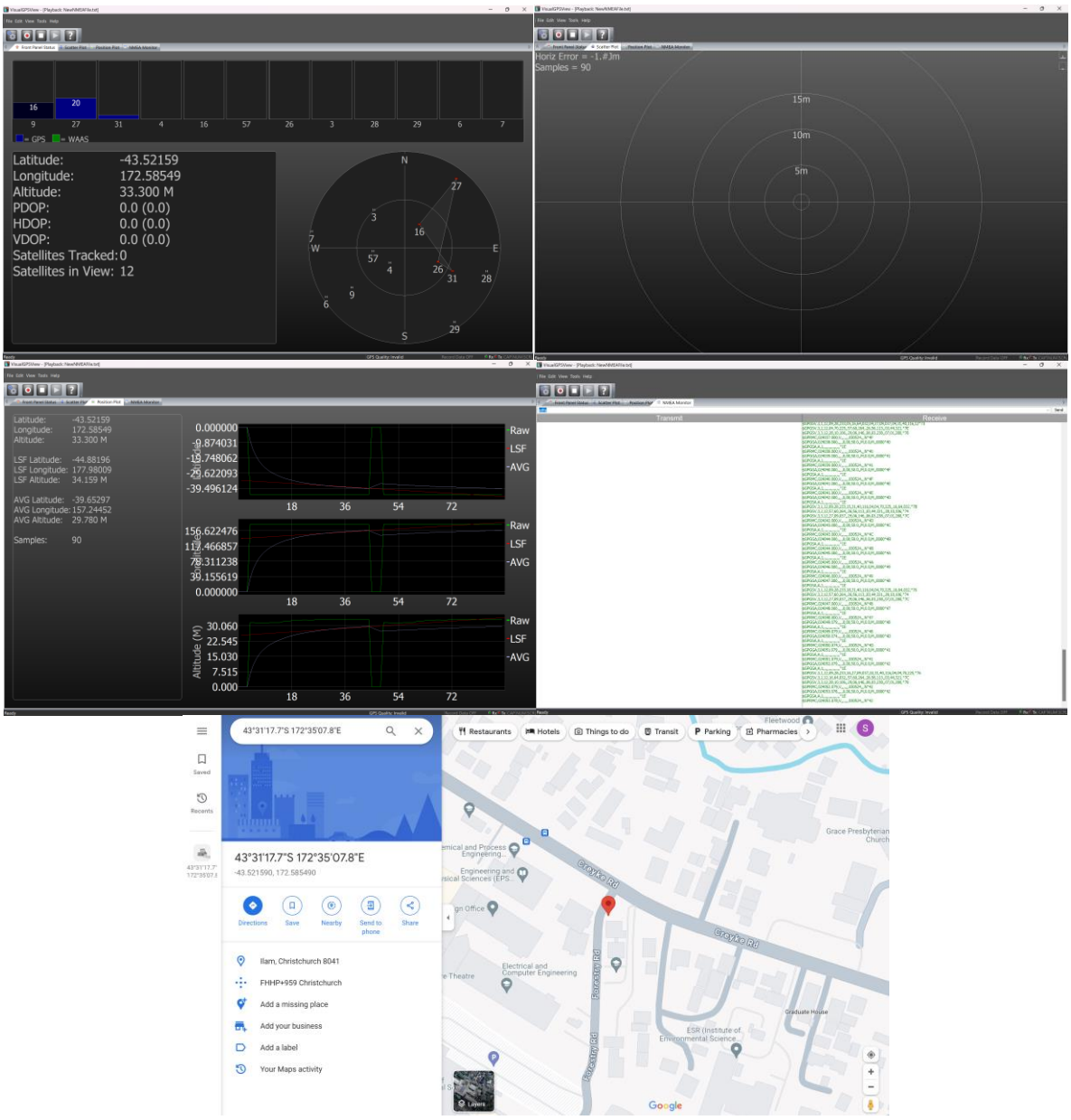

Figure 14 *- VisualGPSView Software User Interface Features And Google Maps Coordinate Location*

After ensuring that the GPS receiver was functioning and measuring its positioning performance, several scripts were developed in c to test different functionalities of the GPS receiver. The scripts developed were:

***Table 7** - Test Filenames and their Description*

FILENAME:	DESCRIPTION
readNMEA.c	Reads NMEA data from serial port and prints into terminal
format.c	Takes NMEA data and converts into to latitude, longitude, altitude in degrees
testing_gps_baud_rate.c	Tests various GPS baud rate to check which is compatible with hardware
DOP.c	Reads NMEA data and calculates HDOP*

*HDOP is dilution of precision which gives an approximate range of error in the horizontal position.

The scripts written was functional and was able to collect NMEA data from the GPS receiver and format the data into useful information. However, this implementation was done in Windows operating system and needed to be ported over to Ubuntu 22.04 as this is the required operating system to use ROS 2. ROS 2 will be used to transmit the data between two different systems using a useful subscriber and publisher node feature. This feature is used throughout the entire drone swarm to communicate between each other.

Through various testing and troubles, the porting of the code into Ubuntu 22.04 and the integration of using ROS 2 subscriber and publisher node was complete as seen in Figure 15. The target script is uploaded to a raspberry pi 4 board, with a power bank and the GPS receiver connected to the USB ports as seen in Figure 16. However, I had realised that the GPS Receiver did not support its hardware to change the update frequency and had it set to 1 Hz by default. This oversight in hardware incompatibility meant that this GPS receiver could not be suitable to be used out for a field test. Despite this large oversight on my end, the target was taken out in one of the earlier field tests conducted and showed messages were being transmitted.

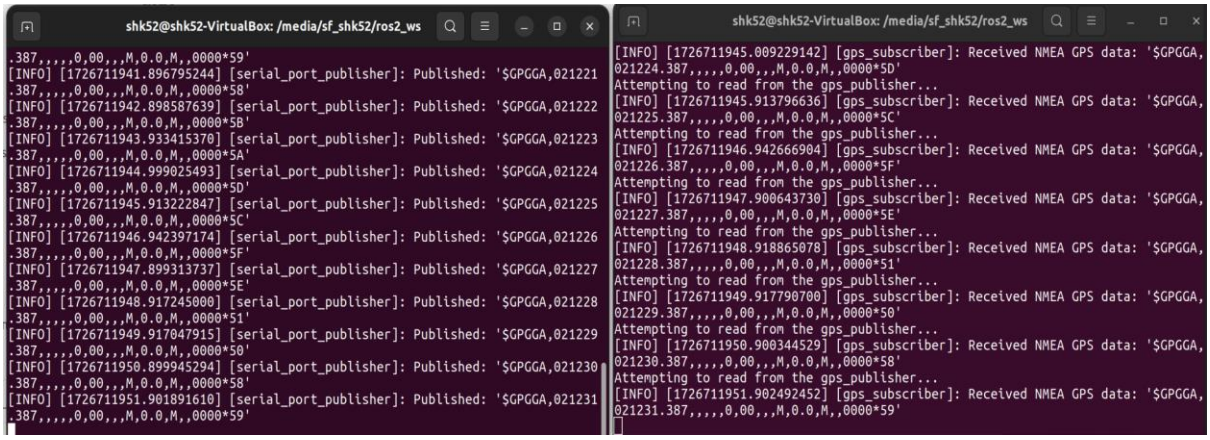

***Figure 15** - ROS 2 Subscriber and Publisher Node Implementation*

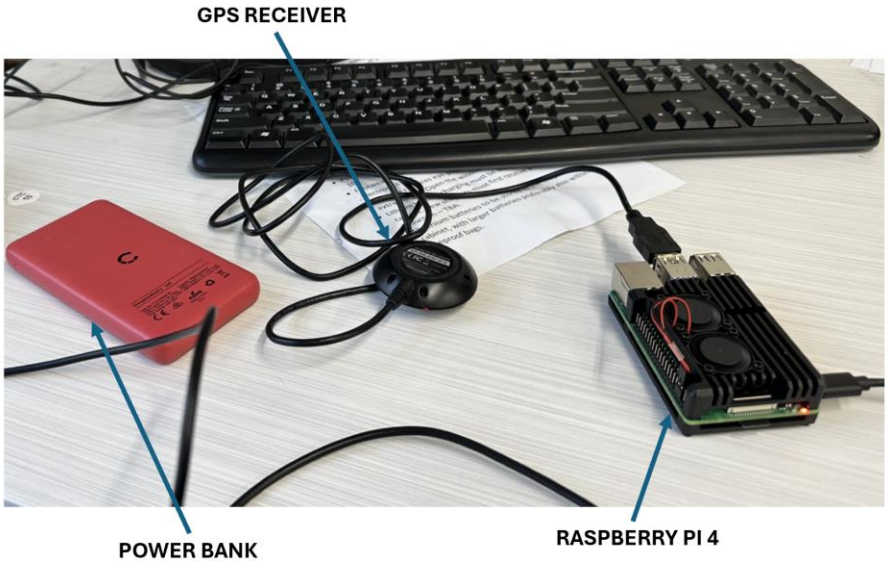

Figure 16 - *Target System Integration*

2.3 New GPS Receiver

Table 8 - *SiRF Star IV vs GU-504GGB-USB GPS Modules*

	SiRF Star IV GPS	GU-504GGB-USB GNSS
Accuracy	2.5m	1.5m
Update Freq.	1 Hz (Default)	10 Hz (Default)
Baud Rate (Max)	38400	921600
Satellite	GPS	GPS, GLONASS, GALILEO, BEIDO, SBAS
Cost (NZD)	Free (Provided by WRC)	$58.8

I had looked through various new GPS which would be suited for this target application and decided on the GU-504GGB-USB GNSS Receiver [29]. A comparison between the two GPS modules can be seen in Table 8. The new GPS should be more than sufficient to perform the required task for the implementation. This GPS receiver was then ordered online using the FYP project budget and took around 2-3 weeks to arrive. A new GPS receiver meant that the original code needed a couple adjustments such as setting up the serial port settings alongside the filtering messages. The implementation of the software alongside the new GPS receiver has not been able to be tested out in a field as it had arrived after we were finished conducting field tests. However, the functionality of the new GPS receiver has been tested to make sure it had worked locally on my computer. The final development of the target is able to collect data from the GPS receiver, format the data into position alongside DOP values. The position data is sent to both the drone swarm using ROS 2 and also stored in a logging file. The HDOP values are also stored in a separate logging file and graphed showing HDOP value over each point. The logged files for GPS coordinates can then be used into a GIS (Geographic Information System) such as google earth pro to map out the path of the target. A block diagram of the target can be seen in Figure 17.

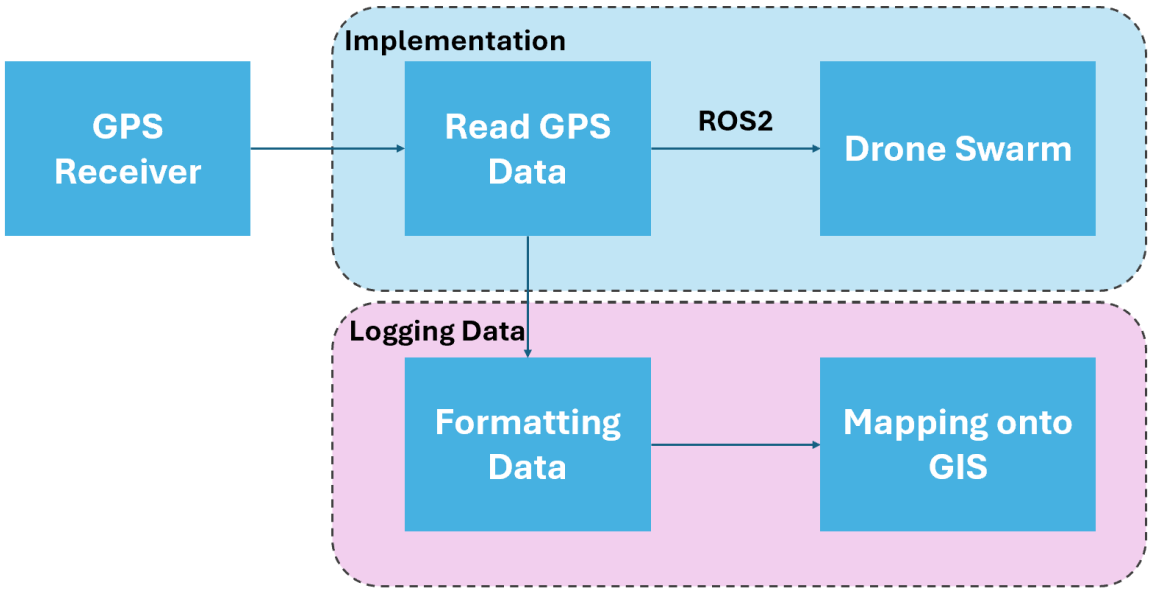

Figure 17 *- Block Diagram Of Target System*

3. Discussion

3.1 Performance

3.1.1 GPS Receiver

Currently, there is no actual field test data I was able to extract for the target system. However, there are several test data I had logged throughout the development of the target. The following data are what have been logged using SiRF Star GPS receiver and unfortunately have not been able to test the new GPS receiver to the same extent due to time constraints. The performance of the GU-504GGB-USB GNSS receiver is expected to be an improvement of what is seen as it is a direct upgrade from the SiRF Star GPS receiver in all fronts. The following Figure 18, Figure 19, Figure 20 shows an example of the designed target system in usage.

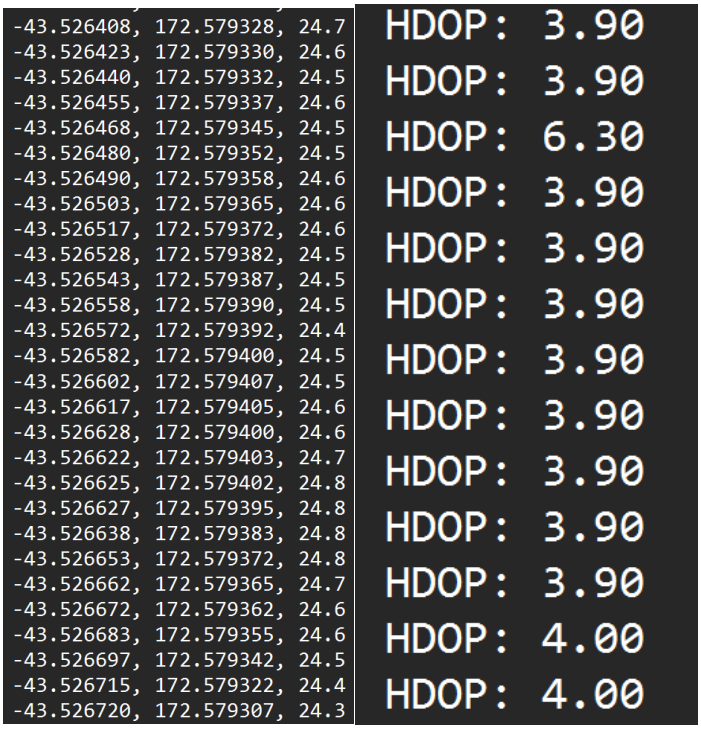

Figure 18 - *Formatted NMEA Data To Coordinates (Left) and HDOP (Right)*

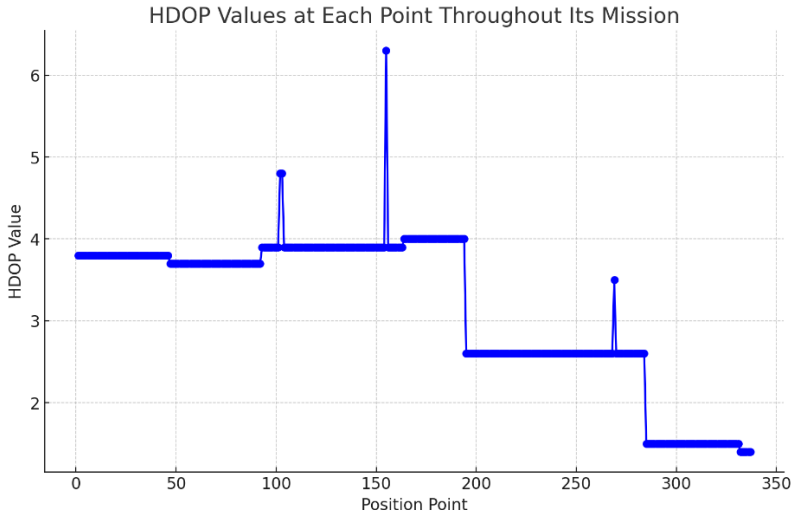

Figure 19 - *HDOP Values For Each Position*

Figure 20 *- Formatted Coordinates Mapped onto Google Earth Pro*

Table 9 *- Testing SiRF Star IV GPS Accuracy*

Location	Duration (minutes)	Weather	HDOP
Inside Building (Electrical Wing)	5	Sunny/No Clouds	-
Walking Around Ilam fields	10	Sunny/No Clouds	1.9
Walking Across Campus	15	Sunny/No Clouds	2.4
Walking Around at home	5	Sunny/ No Clouds	3.0
Walking from Electrical Wing to Parked Car	15	Spitting/Cloudy	3.2

Table 9 shows the variety of tests conducted to measure how well the SiRF Star IV GPS was able to get its position. Although other tests suggest a high value of HDOP, in a clear sky environment with no structures nearby, the GPS receiver was reading an average HDOP value of 1.9 which is sufficient to satisfy requirements R2. With the addition of the newly upgraded GPS receiver, the GU-504GGB-USB GNSS receiver will get better HDOP values as it is capable of collecting more satellite data which increases its precision.

3.1.2 Power Supply

One of the requirements (R7) was to ensure the operational time of the target were to be equal or to exceed the mission life span of a drone swarm. As the target system is run on a raspberry pi 4 board, the board was powered using a power bank.

As the power bank is 10000 mAh with a voltage supply of 5V,

$$10Ah \times 5V = 50Wh \quad (1)$$

The raspberry pi is connected to a relatively low power consumption GPS receiver which assuming a current draw of around 0.7 A,

$$0.7A \times 5V = 3.5W \quad (2)$$

$$\frac{50Wh}{3.5W} = 14.3\ h \quad (3)$$

This means a fully charged 10000 mAh power supply is able to power the target system for around 14.3 hours as seen in Equation 3. As the average battery lasts around 20 minutes for a drone, this is more than enough power to last dozens of drone swarm flight tests.

3.1.3 Summary

All requirements were met to create a robust target system which in theory should be capable of reliably sending its position to the drone swarm. The final designed implementation needs to have further testing carried out with the new GPS receiver; however, the behaviour or functionality should remain the same as it is only the hardware. Once the working implementation has been confirmed to work, this subproject can now be considered complete.

Ronniel Padua – Drone Management and Visualization Application

1. Background Research

All drones used in this project use a raspberry pi 4 running ubuntu 22.04. To communicate wirelessly with the Raspberry Pi's, Secure Shell (SSH) [30] is used to access the Raspberry pi remotely. From the SSH terminal, *shell commands* can be sent to the Raspberry Pi. Examples of shell commands are *ls, cd, chmod, python.* By sending the correct shell command, the Raspberry Pi can run Python or Bash scripts, activate ROS, or command the drone to fly.

There are various issues with using a terminal for controlling the drones. One issue is that using a terminal is not easily scalable. For each drone, a terminal must be brought up, an SSH connection made, and then each drone is given a command. The terminals are also not user friendly because all of the commands must be typed manually. Manual typing of commands often has errors, due to misspellings or incorrectly chosen commands. For this reason, a GUI should be made that decreases the amount of manual setup that has to be done by the user.

1.1 Previous GUI

Because E24 is a continuation project, a GUI was developed in 2023. However, it was not working when used by a summer student. Continued development was unfeasible because the code was using undocumented Typescript and had no explanation for the Python front-end.

1. 2 Purpose of the GUI

As described above, using a terminal to connect to the drones is not user friendly and does not scale to large drone sizes. For this reason, the GUI *aims to decrease the amount of user intervention required to get the drones to fly*.

1.3 Key Requirements

To decrease the amount of manual intervention required, the following key requirements were created. Their justifications are listed below each requirement.

1. Update software and configuration on the drones.
    a. Ensures that the drone swarm will fly in the correct formation.
    b. Ensures that up to date firmware is on the drones, which means all of the drones will be using the same firmware versions.
2. GUI should be checking to confirm launch readiness
    a. Ensures that key software on the drones is running.
    b. Ensuring that key software is up to date.
    c. Also ensures that the drone can physically fly, for example, by having enough battery.
3. Visualize runtime data from the formation

a. Live telemetry can be helpful for debugging each flight test.
b. Can be helpful for seeing the current state of the drone, for debugging or as confirmation that everything is working correctly.

1.4 Preference Requirements

These requirements are requirements that do not need to be met. However, following them provides an optional framework for the sub-project. Under each requirement is a short justification for why the requirement is important.

The overarching goal of these requirements is to *consider the future of the GUI*. Since E24 is a continuation project, future development will happen and so some thought must be given to the future students working on the GUI.

1. As much as possible, use open-sourced libraries and implementations.
    a. This decreases the necessity of keeping licenses updated in the future, simplifying development because libraries will not have to be replaced.
2. Use libraries with the longest support available.
    a. This decreases the need for the libraries to be updated immediately after this project finishes.
    b. Another reason is so the GUI does not suffer from 'tech-debt' which is when the GUI must continue to use outdated libraries to keep working.
    c. Decreases the chance that future versions will result in incompatibilities between libraries.

1.5 Specifications

The Key Requirements listed above lead to the specifications listed below. By meeting these specifications, the GUI will accomplish the goal of decreasing the amount of manual intervention required:

1. GUI should be able to send software to the drones from the computer used.
2. GUI should be able to run software on the drones.
    a. Key software is ROS and sending the launch commands.
    b. Other software is the webserver.
3. GUI should receive data from the drones.
    a. Receiving battery level, heading and current software versions.
    b. Other information such as MAC address, IP address.
4. GUI should receive telemetry data from the *data dissemination protocol* (developed by Raith Fullam, SENG, not assessed here)
    a. GUI should display received telemetry data on the screen.

2. Work Carried Out

2.1 Iteration One

The first iteration of the GUI was using Qt GUI, a GUI framework that used C++. Qt was originally chosen because of an assumption that C++ would speed up the operation of the GUI.

While developing the GUI in Qt, multiple issues were discovered. The first issue is that Qt is a large GUI framework with any features and ways to create a GUI. The issue of size became apparent by the lack of documentation while building a simple GUI. The official tutorial codes the GUI by hand, while other tutorials use the Qt Design studio to move the aesthetic features of the GUI into place. A lack of documentation while working with Qt wasted significant time trying to figure out how to program basic GUI features such as buttons.

The second issue encountered is building the GUI. Due to Qt being programmed in C++, a makefile was used to compile all of the code. Unfortunately, this meant moving the code from my git repository to an external SSD running ubuntu 22.04 broke the makefile, and Qt could not compile my GUI again. It was apparent that trying to reprogram everything would be an inefficient use of time. For this reason, I changed the GUI's programming language from C/C++ to Python, and used Tkinter to create the rest of the GUI.

2.2 Iteration Two (Current state)

Due to the time wasted trying to use Qt, I created new requirements while restarting GUI development, which are listed below. The primary goal of these requirements is to *increase development speed* so key features could be implemented.

1. The new GUI would have to be in a language I could easily develop in.
2. The language had to be portable, with as little setup to run the program as possible.
3. The GUI framework must have well-documented examples to get started with.

The language that best fit requirements 1 and 2 is Python. Since Python is taught across the ECE program, with beginner courses COSC131/122, I had knowledge of Python's syntax. Being widely taught also means future students working on this GUI would be able to develop the GUI without having to learn another programming language. Another benefit is that as an interpreted programming language, there is no need to compile and running programs could be done quickly once the correct dependencies were installed.

The GUI framework was PySimpleGUI and Tkinter. Both of these frameworks were measured against preference requirement 3. I chose the Tkinter libraries because the breadth of available resources allowed me to get started developing very quickly. Tkinter is also simple to develop in, because the GUI is created by programming all elements on the page. For these reasons, Tkinter meets requirements 3 over PySimpleGUI. Another benefit is that Tkinter is included in Tkinter by default, which meant there was no need to install other libraries.

Although PySimpleGUI was an option, PySimpleGUI moved to a closed-source distribution around the start of 2024. Since the project is expected to continue, having less restrictions on the usage of the GUI is preferable. In comparison, Tkinter remains open-source and it is a Python library, will remain so in the foreseeable future. This adds another reason for choosing Tkinter over PySimpleGUI.

2.3 Feature Breakdown

2.3.1 Connecting To the Drones

Overview of Method

As per specifications 1 and 2, the GUI must be able to connect to the drones to send software to the drones and run the software. To connect to the drones, Secure Shell (SSH) and Secure Copy (SCP) are used. SSH allows the user to remotely connect to the drones and control it via the terminal, while SCP can send the files or a folder to a drone.

Going to SSH to connect to the drones makes the user have a terminal to communicate with the drones. Eventually, this method will be superseded by other functions of the GUI. However, being able to SSH to the drones allows the user to take direct control of the drone and debug important information.

SCP was used because it is a simple protocol that allows the user to send a file or folder. The location of the file is specified in the form of a path to the directory or file that is to be sent. Other alternatives include rsync to send files to the selected drone. However, SCP was used over rsync because of my familiarity with SCP over rsync.

Figure **21** below shows the Drone select screen. 'Connect to Drone via SSH' SSH's into the drone. Upload files in directory and Upload to Drone uses SCP to send a directory/file to the drone.

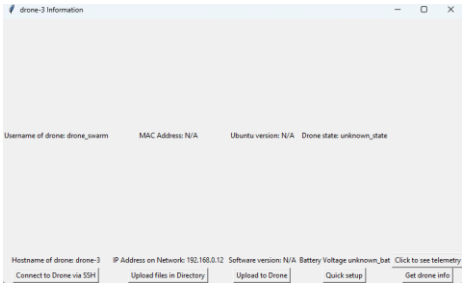

Figure 21 - The Drone selection screen

2.3.2 Specific implementation

SSH and SCP are usually typed into the terminal by the user. For the GUI to help decrease the amount of manual intervention required, the GUI needs to provide a faster way of running the commands. For this reason, the GUI uses subprocess, an inbuilt python library, that runs the shell commands passed to it. The subprocess call is enclosed in a function, which can then be called by Tkinter's callback functions. Subprocess was chosen over the older os.system library by Python.

Os.system is a library which is in the process of being superseded by subprocess. Per preference requirement 2, subprocess is the preferred library to run the SSH and SCP commands because it will be developed and supported by python in the near future.

2.4 Connecting to members in the swarm

The GUI is able to send files or folders to all of the drones at once, on the main page, as shown in Figure **22** below. The GUI can also send to a group of drones selected by the user under the 'Send files to multiple drones' button. These features were added in order to speed up setting up a large swarm of drones by letting the user configure a group of drones. By letting the GUI automatically send to all members in a swarm, a swarm can be configured with the same effort as setting one drone.

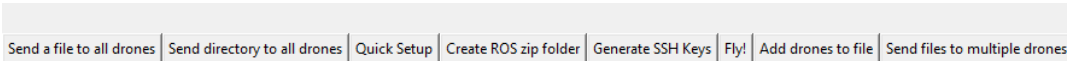

Figure 22 - Send a file and send directory send to all drones on the network. Send file to multiple drones sends to the group of selected drones.

2.5 Assisting Setup of the GUI

2.5.1 Background: How Drone Information Is Stored

Internally, there is a text file called *droneconfigs.txt* that holds of the important information about all of the drones controlled by the GUI. Each text file contains the hostname, username, ip address, mac address, Operating System (OS) version, Robot Operating System (ROS) version, battery level and current drone state. The text file is then read by the drone, parsed and then passed to other functions in the GUI.

Other ways of storing information, such as JSON or CSV were considered. However, when the text file was being developed, my Python skills were not yet able to read more complex file types. To speed up development, reading from a simple text file was chosen over implementing the ability to read from more complex file types.

2.5.2 Future Proofing of Setup

Although the current GUI only reads from a text file, it can be adapted to parse other file types in the future. Adapting the GUI to read other file types is possible because the processing of the data in the file is separate to parsing the data in the file. If a new file type were to be read from, the parser only needs to make the output data the same but can read from any variety of inputs. This lets the GUI adapt to different file types that can hold information in a more human-friendly way.

2.5.3 Adding New Drones

The GUI also adds the ability to add new drones to *droneconfigs.txt* by pressing 'Add drones to file'. When the user inputs the information required for the drone and saves it, the drone information is saved to the text file. Being able to add more drones that the GUI knows is important because it helps the user quickly set up a drone when desired, per requirement 1.

In Figure **23** below, clicking on the 'Add new Drone' creates the screen in Figure **24**, where the user can type in drone information.

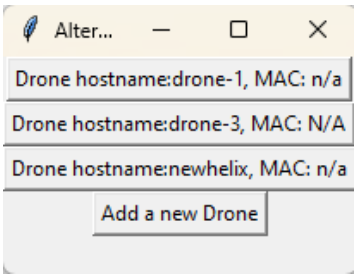

Figure 23 - After clicking on 'Add drones to file', the user is sent here

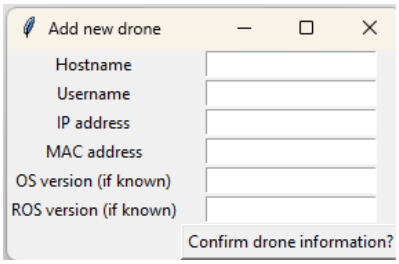

Figure 24 - They can then input the required information

2.5.4 Extensibility

Being a continuation project, the GUI should be extensible. For this reason, although the current GUI is only able to add new drones, other features can be added. Other features such as deleting drones and editing current drone information can be added by adding more features to the Entry class in the DroneConfigAlter file.

2.6 Miscellaneous Features

2.6.1 Addition of Tests

There are unit tests added for each of the five classes present in the GUI. These unit tests test each method in the class. Adding these unit tests was a way to practice test driven development, with the tests being developed first, and then the code being written to fulfil that test. By doing this, the tests specify the method's purpose, and what is must do for a given input. Another benefit of adding unit tests is that when the method is implemented, a passed test means that the method works as expected. Finally, the last benefit is that changes to the code can be done because the tests can check if the methods have been broken due to something added.

2.6.2 Extensibility of code

There are three classes which deal with connecting to the drones. These classes are SwarmActions, Drone, Swarm and DroneList. Adding more features to the GUI will involve adding more methods to the four classes. By using classes, the code remains extensible because code does not need to be

modified, only *added* to add new behaviour. By being extensible, the GUI future-proofs itself for new features which may be desired in the future.

3. Discussion On Work

3.1 Specification 1 -> Success

The GUI completes specification 1, because the GUI is able to send files to the drones, updating the software on the drone to the most recent version, helping the user update drone firmware without having to type in the commands manually. Another benefit is that the GUI can detect if a transmission has failed and notifies the user if this is the case.

3.2 Specification 3 -> Partial success

The GUI can send files, which, when run, will collect the information about the drone. This includes the MAC address, and the OS version of the drone. Other information can quickly be gathered by the GUI by adding them to the collect_info file. The GUI can command the raspberry pi to provide this information, but the GUI cannot automatically update the stored information file. As a result, the result is just printed to the terminal and the user must manually update the drone information file.

Despite the partial success, the GUI has provided a framework to receiving the information from the drone. Future work on the GUI, as a result, will be focused on updating the current information instead of having to collect it. Another benefit of this GUI is that if other information needs to be collected, it can quickly be added to collect_info.

3.3 Specification 2 -> Unmet

The GUI cannot complete specification 2, which is running the software on the drone. This is because the setup for the drones remains unstable, with commands that tell the drone to start flying unexpectedly failing due to QGroundControl not detecting the drone. Other issues include having to configure the drones manually using QGroundControl following a reset, as the files cannot be sent wirelessly. Thus, since the launch process requires significant manual intervention, there is no purpose trying to automate the process by sending commands to run software.

Because being able to run software remotely on the drones is important, the GUI is able to meet specification 2 by calling a new function, run_command, that allows the user to run commands which are passed to the function.

3.4 Specification 4 -> Inability to display runtime data

The *Data Dissemination Protocol* cannot be integrated this year. For this reason, the GUI cannot collect the runtime data and visualize it. A possibility of receiving runtime data is to collect the information via the webserver, by downloading the information from the webserver on the drone. However, the webserver has not been implemented this year, with the webserver creating a simple webpage that only allows for manual downloading of files.

It was unexpected that QGroundControl is able to display telemetry data, including position, drone state and current connection status, live on the app. Because QGroundControl can display more information, in real time, specification 4 is redundant for the GUI. This was brought up to the sponsors, who were surprised by QGroundControl's ability to display data. Afterwards, specification 4 was emphasized less in favour of developing other features.

3.5 Conclusion

The GUI meets specifications 1 well, has a partial success on specification 3 and is unable to complete specifications 2 and 4. The GUI cannot meet specification 2 because manual setup is currently unstable, and so automatic setup cannot be relied upon to work correctly. The GUI cannot meet requirement 4 because the Data Dissemination Protocol and the webserver have not yet been integrated or fully developed. Furthermore, QGroundControl is able to display live telemetry already, making specification 4 redundant.

Overall, the GUI is a mixed success. Although it meets some key features asked for, it is lacking on other features. Instead, the GUI provides a framework for future groups to improve upon by extending the code. By providing a framework, future development can be focused on implementation instead of research and defining problems, which should help future groups develop the GUI quicker.

Sustainability

Understanding the impacts the drone swarm may have on the environment and society, is important for its further development. Being informed of the negative impacts of the swarm gave the team the opportunity to mitigate and minimise these aspects during the development process. In the sections below, both the environmental and societal impacts the drone swarm technology has at the current stage of development and the potential impacts it may have once commercialised have been considered.

Environmental Impact

Assessing the lifecycle of the drone swarm requires analysis of the inputs and outputs to the system. Looking at the environmental effects of the drones before, after, and during active duty will help us gain an understanding of their environmental impact. Climate change potential and the depletion of minerals are the primary environmental indicators for the drone swarm.

Material Extraction and Manufacturing

The primary material for the drone body is carbon fibre [31], as this boasts a high strength-to-weight ratio. The production of carbon fibre requires significant energy usage emissions. The weight of each drone is 610g [31], assuming roughly 500g of this is the carbon fibre, each drone can cost 10-13.5 kgCO2/drone for the carbon fibre alone (assuming hydro energy)[32].
The connections for the frame are made from fibre-reinforced nylon connectors [31]. Nylon is made from a synthetic polymer derived from petroleum. The production of nylon is energy-intensive and releases Volatile Organic Compounds into the atmosphere [33].
LiPo batteries are lithium-based batteries. Lithium extraction is particularly water intensive, requiring 500,000 gallons per ton of lithium [https://hir.harvard.edu/lithium-triangle]. The manufacturing of the battery also uses harmful chemicals, such as solvents and heavy metals, that can create toxic waste.

Swarm Energy Consumption

Investigating the climate change potential, we can analyse how much energy it takes to operate the drone swarm and convert this it to its CO_2 equivalent emissions. At this stage, flight time of each drone is short. It takes ~18 minutes using 5000mAh LiPo batteries, for a drone's battery to deplete. This is fine while the drone swarms are still being developed and are in the prototyping phase. However, once the drone swarms are used in the field, frequent recharging throughout missions will be inconvenient and require more energy consumption.

CO_2 emissions will vary depending on the energy mix of the electricity grid. Using New Zealand's emission factor of 0.101 kg CO_2-e / kWh from [34], one year of operation using a ten-drone swarm, completing one hour-long daily flights, would result in ~90 kg of CO_2-e emissions (the calculation of this can be found in Appendix C). This is only an estimation as losses in charging have not been accounted for. But, to put this in perspective this is less CO_2 than would be produced driving from Wellington to Auckland in a conventional petrol car [35]. The total energy consumption for a single swarm is minimal, regardless, Efforts must be made during the

project to ensure that software developed does not use excessive amounts of energy, draining the batteries faster and increasing the energy required to operate.

Resource Depletion

The HolyBro X500 drones consist of carbon fibre tubes and plates fastened using plastic connectors. The drones are not particularly robust. However, they are completely modular. All components can be replaced from the flight controller to the rotors. The plastic casings can be recycled, however, there are currently no commercially available methods for carbon fibre recycling. The batteries themselves are lithium-ion polymer batteries. LiPo batteries typically last 300 to 500 charge cycles [36]. Depending on the number of batteries purchased and charging habits, each battery may last 1- 3 years. These batteries are recyclable, but less than 5% are collected [37]. Additionally, if the batteries are damaged or overcharged, they can ignite posing risks to public safety. It is worth investigating potential alternatives.

Social Impact

These drone swarms ultimately will be used to track endangered native insects for environmental preservation. However, adaptive drone swarms have many other applications from search and rescue, military use, and surveillance. If this drone swarm technology was to be used in these sectors, they would have a major impact on society. Three social impacts surrounding the use of commercial drones were identified in [38] to be:

- Safety and Security – Personal and of property.
- Rights to Civilian Airspace – How drones interfere with civilian aviation.
- Privacy and Ownership – In regard to data collection by drones

Relating these back to the drone swarm in development, the safety of the public when the swarm is active will be a concern. Currently, each operating drone in the swarm must have a pilot ready to take manual control, however, when these restrictions are removed in the future a crash could damage people and property. This ties into the rights to civilian airspace. Currently, all drone operators must sign a form saying they understand the Federal Aviation Associations regulations for recreational flying, before being allowed to operate the drones. These regulations can be found in [16].

In regard to privacy concerns, at this stage the drones within the swarm do not share any sensitive data. Although, once the variable dissemination protocol is integrated with the drone swarm, each drone will have the capability to share a wide range of data throughout the swarm, which will pose security threats. All these social concerns must be taken in consideration throughout development of the drone swarm to ensure public wellbeing is maintained.

Conclusions

This project aimed to aid the conservation efforts of the Wireless Research Centre by continuing the development of target-tracking drone swarms compatible with their breakthrough harmonic radar technology. The overarching objective was to extend the previous drone swarm implementation to complete a successful multi-drone flight test, which was successfully achieved. Key requirements included improving the scalability and performance of the swarm, implementing a comprehensive target system to act in place of the insects, creating an in-house drone management and visualization application, and integrating the Drone Coordination Protocol (DCP).

A common theme across the updated Controller and Lifecycle software was that, although implementation time increased, the system's extensibility was significantly enhanced. This enhancement was beneficial to our group and is expected to aid future project teams. The drone formation and path planning were successfully decoupled and redesigned, vastly improving the scalability and performance of the swarm. Part of this performance improvement resulted from porting the software to C++. Including the Lifecycle in the C++ port was highly beneficial. Working in the same language streamlined collaboration and the joint development of the Controller and Lifecycle, which became crucial as the software began to integrate. The new event-based system of the Lifecycle simplifies the understanding of mission operations and facilitates their extension into more complex missions.

Although only partial success was achieved with the Drone Management and Visualization Application, a framework for further development was provided with solid and tested backend principles.

By creating the controller software in a target-agnostic format, the target was developed in isolation, ensuring that each target could be seamlessly switched. A robust target was implemented using GPS, which not only integrated with the controller but also allowed for simple and in-depth post-mission analysis.

Hardware integration was one of the most significant hurdles and remains an aspect that requires further investigation. More work is needed to refine the PX4 FCU configuration, as new issues are discovered with every flight test, such as multi-drone flights landing due to internal safety features that are not immediately apparent.

Ultimately, this project was a success. The overarching goal was achieved while meeting many of the sub-requirements. Careful attention was given to ensuring clean and extendable code to streamline future development. Although several features could be improved to enhance the project further, the system's potential is significant, being designed to be adaptable to any sector.

Recommendations/Applications

Controller Recommendations

Listed as follows are the recommended tasks that should be completed in the project's continued development:

- Overhaul FCU configuration – Remedy issues causing drones to land as a precaution.
- Collision Detection – Periodically check neighboured drones are outside a safe range and apply collision prevention if not.
- Manual Swarm Control – Allow the reference point to be moved freely without a target, using the virtual target as a starting point.
- Mission Events – Implement more events for the lifecycle states
- Mission Extension – Implement a mission that includes target tracking, path following, and manual swarm control.
- Heading Control – Test and further develop the control of the swarm's headings.
- Events Class – Could add an Events class, or at least module, that centralises and standardises all the events of lifecycle.

This design can be used for more than just tracking endangered insects, with manual swarm control implemented the design could also be applied to numerous other tasks such as search and rescue, wildfire management, and infrastructure monitoring.

Network Protocol Recommendations

Once completed, DCP should be integrated with the control system as outlined in Appendix B or similar. DCP should be analysed in a physical flight test once implemented to identify any issues and or performance deficits with the protocol. Upon which potential solutions should be developed.

Target Recommendations

As the target system has not been tested alongside the drone swarm, it is recommended that this is to be done first. However, a working version of the implementation means that this part of the subproject is complete and don't need to further work. An optional implementation to improve the flexibility of the target is to create different target vehicles which act to stress test the drone swarm's behaviour.

References

[1] Holybro. “Pixhawk 5X”. holybro.com. https://holybro.com/collections/autopilot-flight-controllers/products/pixhawk-5x (accessed Oct. 17, 2024).

[2] PX4 Autopilot. “PX4 System Architecture”. PX4 Autopilot User Guide (v1.14). https://docs.px4.io/v1.14/en/concept/px4_systems_architecture.html (accessed Oct. 17, 2024).

[3] A. Koubâa, A. Allouch, M. Alajlan, Y. Javed, A. Belghith and M. Khalgui, "Micro Air Vehicle Link (MAVlink) in a Nutshell: A Survey," in *IEEE Access*, vol. 7, pp. 87658-87680, 2019, doi: 10.1109/ACCESS.2019.2924410.

[4] S. Macenski *et al.,* “Robot Operating System 2: Design, architecture, and uses in the wild”, *.Sci. Robot*, vol.**7**, no.66, May. 2022, doi:10.1126/scirobotics.abm6074.

[5] A. Willig, “Drone Coordination Protocol (DCP) and VarDis Protocol Specification”, UC, Christchurch, Mar. 1, 2024.

[6] Crostars. “Cross Star Drone Lightshow”. Crostars.com. https://www.crostars.show (accessed Oct. 9, 2024).

[7] R. Fulbright, “Swarm-Based Firefighting Drone and Mass Aerial Drop System and Method”, U.S. Patent 11565813B2, Oct. 1, 2019.

[8] E. Kayacan and R. Maslim, "Type-2 Fuzzy Logic Trajectory Tracking Control of Quadrotor VTOL Aircraft with Elliptic Membership Functions," in *IEEE/ASME Transactions on Mechatronics*, vol. 22, no. 1, pp. 339-348, Feb. 2017, doi: 10.1109/TMECH.2016.2614672.

[9] PX4 Autopilot. “Controller Diagrams”. PX4 Autopilot User Guide (v1.14). https://docs.px4.io/v1.14/en/concept/px4_systems_architecture.html (accessed Oct. 17, 2024).

[10] H. Seliem, R. Shahidi, M. H. Ahmed and M. S. Shehata, "Drone-Based Highway-VANET and DAS Service," in *IEEE Access*, vol. 6, pp. 20125-20137, 2018, doi: 10.1109/ACCESS.2018.2824839.

[11] L. Shi, N. J. H. Marcano, R. H. Jacobsen, “A review on communication protocols for autonomous unmanned aerial vehicles for inspection application”, Microprocessors and Microsystems, vol. 86, 2021, doi: 10.1016/j.micpro.2021.104340.

[12] J. Bloch, “How to design a good API and why it matters”, In Companion to the 21st ACM SIGPLAN OOPSLA, Association for Computing Machinery, New York, NY, USA, pp. 506–507, 2006, doi:10.1145/1176617.1176622.

[13] M. Reddy, “Introduction”, in *API Design for C++*, 1st ed., USA, Elsevier, 2011, ch.1, sec.1.3, pp. 6-11.

[14] PX4 Autopilot. “Standard Safety Configuration”. PX4 Autopilot User Guide (v1.14). https://docs.px4.io/v1.14/en/config/safety.html#traffic-avoidance-failsafe (accessed Oct. 17, 2024).

[15] Gazebo. “Gazebo Simulation”. Gazebosim.org. https://gazebosim.org/home (accessed Oct. 17, 2024).

[16] *Exception for limited recreational operations of unmanned aircraft*, 49 USC 44809, 2024. [Online]. Available: https://uscode.house.gov/view.xhtml?req=granuleid:USC-prelim-title49-section44809&num=0&edition=prelim#

[17] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, “ROS: An open-source Robot Operating System,” in *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Software*, Kobe, Japan, May 2009.

[18] ROS Documentation. “End of Life Announcements.” [Online]. Available: https://index.ros.org/doc/ros2/Releases/#end-of-life-eol-of-ros1-distributions (accessed Oct. 17, 2024).

[19] ROS 2 Blog. “ROS 2 Dashing Diademata Released.” [Online]. Available: https://index.ros.org/doc/ros2/Releases/ (accessed Oct. 17, 2024).

[20] mavlink/mavros. “MAVROS.” GitHub.com. [Online]. Available: https://github.com/mavlink/mavros (accessed Oct. 17, 2024).

[21] PX4 Autopilot. “PX4 User Guide.” *PX4 Autopilot User Guide* (v1.14). [Online]. Available: https://docs.px4.io/v1.14/en/ (accessed Oct. 17, 2024).

[22] EProsima. “Micro XRCE-DDS Documentation.” [Online]. Available: https://micro-xrce-dds.docs.eprosima.com/en/latest/ (accessed Oct. 17, 2024).

[23] Object Management Group. “Data Distribution Service (DDS) Specification.” [Online]. Available: https://www.omg.org/spec/DDS/ (accessed Oct. 17, 2024).

[24] mavros. “MAVROS Documentation.” [Online]. Available: https://mavros.readthedocs.io/en/latest/ (accessed Oct. 17, 2024).

[25] University of Canterbury, “School of Forestry,” [Online]. Available: https://www.canterbury.ac.nz/study/academic-study/engineering/schools-and-departments-engineering-forestry-product-design/school-of-forestry. [Accessed 19 10 2024].

[26] GAO RFID Inc., “433 MHz Active RFID Coin Tag,” GAO RFID Inc., [Online]. Available: https://gaorfid.com/product/433mhz-coin-id-active-rfid-tag/. [Accessed 19 10 2024].

[27] PBTECH, “GlobalSat BU-353N5 USB GNSS Receiver,” GlobalSat, [Online]. Available: https://www.pbtech.co.nz/product/HHAGST0355/GlobalSat-BU-353N5-USB-GNSS-Receiver?qr=GShopping&gad_source=1&gclid=Cj0KCQjwjLGyBhCYARIsAPqTz1_Z2s-RX9oID-rlq_R8u3VNPCjj012yY4fgMQYkBOMdaWuNkh8OTZwaAog4EALw_wcB. [Accessed 19 10 2024].

[28] VisualGPS, “VisualGPS Software and Services,” 2020. [Online]. Available: https://www.visualgps.net/. [Accessed 19 10 2024].

[29] YIC, “Multi-Constellation GNSS Receiver (G-Mouse) GU504-GGB,” [Online]. Available: https://www.digikey.com/en/products/detail/yic/GU-504GGB-USB/18717324. [Accessed 2024 10].

[30] man7, “ssh(1) — Linux manual page,” man7, 11 10 2023. [Online]. Available: https://man7.org/linux/man-pages/man1/ssh.1.html. [Accessed 20 10 2024].

[31] Holybro, "PX4 Development Kit X500 V2," Online 2024. [Online]. Available: https://docs.holybro.com/drone-development-kit/px4-development-kit-x500v2.

[32] F. Meng, "Environmental and cost analysis of carbon fibre composites recycling," Online 2017. Accessed: 2024/05/26. [Online]. Available: https://www.researchgate.net/profile/Fanran-Meng/publication/321794490_Environmental_and_cost_analysis_of_carbon_fibre_composites_recycling/links/5b055c80a6fdcc91ed8b0dde/Environmental-and-cost-analysis-of-carbon-fibre-composites-recycling.pdf

[33] I. Chrysafi, N. M. Ainali, and D. N. Bikiaris, "Thermal Degradation Mechanism and Decomposition Kinetic Studies of Poly(Lactic Acid) and Its Copolymers with Poly(Hexylene Succinate)," *Polymers,* Online vol. 13, no. 9, p. 1365, 2021. [Online]. Available: https://doi.org/10.3390/polym13091365.

[34] *Measuring Emissions: A Guide for Organisations*, ME 1527, 2020. [Online]. Available: https://environment.govt.nz/assets/Publications/Files/Measuring-Emissions-Detailed-Guide-2020.pdf

[35] “Government moves on climate promises: Supporting Documents”, beehive.govt.nz. https://www.beehive.govt.nz/release/government-moves-climate-promises (accessed May. 21, 2024)

[36] A. Renewables, "Lithium Polymer Battery Guide," Online 2024. [Online]. Available: https://renewablesadvice.com/battery/lithium-polymer/.

[37] H. Bae and Y. Kim, "Technologies of lithium recycling from waste lithium ion batteries: a review," *Materials Advances,* Online vol. 2, no. 14, pp. 3234-3250, 2021. [Online]. Available: https://doi.org/10.1039/D1MA00216C.

[38] B. Rao, A. G. Gopi, R. Maione, “The societal impact of commercial drones”, in *Technology in Society*, vol. 45, pp. 83-90, 2016, doi:10.1016/j.techsoc.2016.02.009.

Appendices

Appendix A – RC Controller Configuration

Channel	Label	What Switch Does	Switch Position
1	N/A	Controls Roll	Continuous
2	N/A	Controls Pitch	Continuous
3	N/A	Controls Throttle	Continuous
4	N/A	Controls Yaw	Continuous
5	SA	Flight Mode	Away – “Land”
			Middle – “Position”
			Toward – “Stabilized”
6	SG	Arming Switch	Away – Unarmed
			Middle – Unarmed
			Toward – Armed
7	SD	Offboard Switch	Away – Off
			Middle – Off
			Toward – On
8	SF	Emergency Kill Switch	Away – Un-kill
			Toward - Kill
9	N/A	N/A	N/A
10	N/A	N/A	N/A
11	N/A	N/A	N/A
12	N/A	N/A	N/A
13	N/A	N/A	N/A
14	N/A	N/A	N/A

Figure A *– Futaba Controller Configuration for Flight Tests.*

Pre-Flight

Set all switches away from pilot and set throttle all the way down.

Power on and connect controller to UAV.

Run through Flight Mode selection, Arming/Unarming, and Kill/Un-kill, to ensure all are in working order (toggle status can be found in QGroundControl under Flight Mode in the vehicle setup).

Run through flight safety check: Arm, Kill, Un-kill, Disarm.

Launching

To launch the UAV, ensure the kill switch is in the un-kill position. Change the drone flight mode to ‘Stabilized’ (This allows automated flight to be initiated). To initiate the swarm, send a start command through the CLI or the GUI.

Landing

After a successful landing set the kill switch to the killed position before approaching the UAV.

Emergency Control Override

To take manual control of the drone in an emergency, the throttle must be lifted above 30% threshold. If this happens the drone will exit offboard mode switch to position mode, where it will hold position. Here it can either be manually landed, or the flight mode switch can be flicked up to land where it will do so automatically.

Kill Switch

When the kill switch is thrown into the kill position all the motors on the UAV stop immediately, beware that this will cause the UAV to fall from the sky most definitely breaking the drone frame. THIS SHOULD BE A LAST RESORT!!

This configuration was originally authored by Toby Smillie.

Updated by Benjamin Ireland

Appendix B – Protocol Integration Roadmap

Figure B depicts a potential integration strategy for the drone coordination protocol. Currently the flight controller gets flight information about other drones through ROS 2 broadcasts. This should be replaced by DCP.

To integrate the drone coordination protocol with the rest of the flight controller the following steps could be taken:

1) All flight control processes should be made 'Composable Nodes', this means they will not broadcast information to ROS 2 nodes that are not running on their own computer, using shared memory to communicate.
2) Allow the flight controller to make transmission requests using the state reporting protocol's API. Transmit mission critical information i.e., heading, position, altitude, etc.
3) Periodically query the neighbour table for information about neighboured drones.
4) Safety check information to check for potential collisions or other issues.

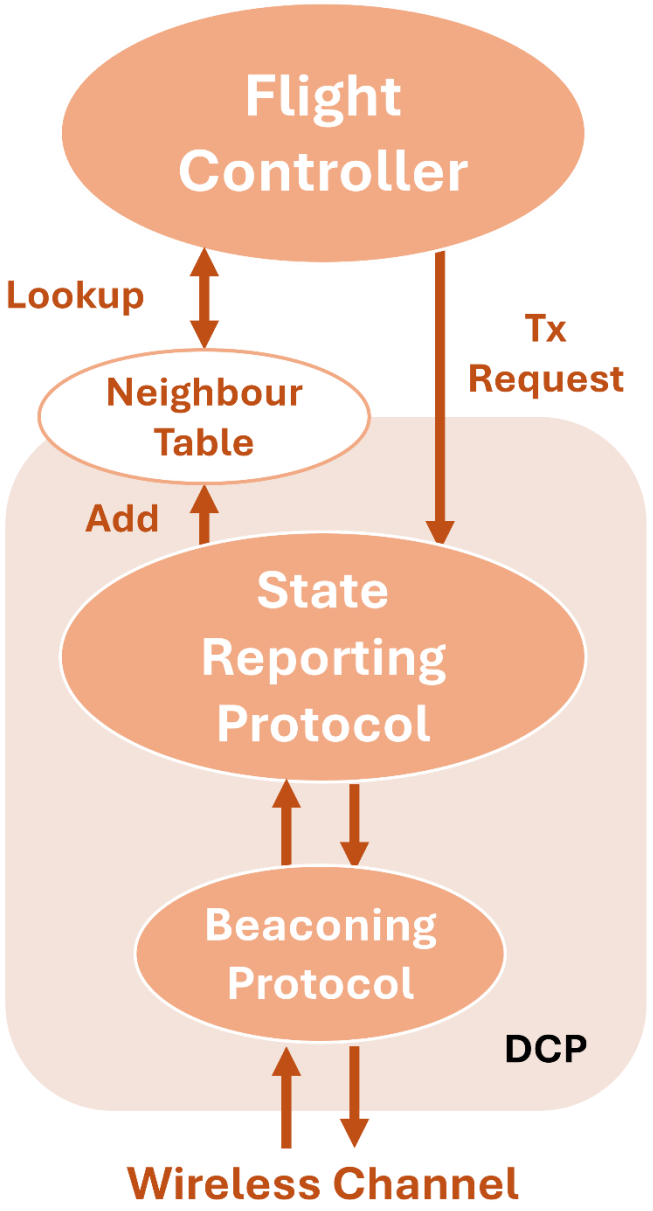

***Figure B** – Potential Integration of DCP with the Flight Controller.*

A potential issue with this implementation relates to the use of composable nodes. They should be used to allow provide security and computational benefits. However, doing this may prevent the Lifecycle node from communicating with the PX4 FCU. This may not be the case, if it is the lifecycle node will not be able to be used as a composable node and the network it operates on might have to be restricted to the companion computers local network.

Link to composable node documentation:

https://docs.ros.org/en/humble/Tutorials/Intermediate/Writing-a-Composable-Node.html

Appendix C – CO_2-e Calculation

Note: This calculation does not consider losses.

$$P = IV = 5A \times 16V = 80\,W$$

Using ten drones gives 800W per charge that occurs 3 times an hour,

$$800W \times 3 = 2.4\,kWh$$

Multiplying by an emission factor of 0.101 gives,

$$2.4kWh \times 0.101\,kgCO_2 - e/kWh = 0.2424\,kgCO_2 - e$$

Repeating this every day for one year,

$$0.2424\,kgCO_2 - e \times 365 = 88.5 \approx 90\,kgCO_2 - e$$

Appendix D – User Input Commands Subset

Format:

`{address}! --command {message}`

Valid Messages:

- **Logging Controls:**
  - `enable_log_dropouts`
  - `disable_log_dropouts`
  - `enable_log_coops`
  - `disable_log_coops`
  - `enable_all_logs`
  - `disable_all_logs`
- **Cooperative Control Resets:**
  - `reset_launch_coop`
  - `reset_formation_coop`
  - `reset_return_coop`
  - `reset_all_coops`
- **Input Only Mode Controls:**
  - `launch_coop_input_only_enable`
  - `launch_coop_input_only_disable`
  - `formation_coop_input_only_enable`
  - `formation_coop_input_only_disable`
  - `return_coop_input_only_enable`
  - `return_coop_input_only_disable`

Appendix E – Original Gantt Chart