UAV-to-UAV Communication Establishing a Leader-Follower Formation

Daniel Cross

***Abstract*—The UAV-to-UAV communication project aimed to research, implement, and review a communications structure capable of establishing a leader-follower formation between multiple unmanned aerial vehicles. With the intention of creating a scalable and extensible system with consumer grade devices to create an accessible swarming solution this research and development process explored and implemented methodologies of drone swarm control that allow operators greater command over groups of UAVs whilst only communicating with a single access point. This lens was chosen to increase the accessibility of drone swarm solutions for enthusiast and entry level consumers who require the technology but are unable to make large investments into more costly options which provide similar behaviour natively. As such, the system explored is composed of multiple consumer grade UAVs paired with Raspberry Pi devices which form a common managing mesh network while providing processing power for supplied in flight commands. These mesh network nodes are combined with a developed ground controller application that transmits commands into the network to be broadcasted and actioned across connected devices to allow for the intended formation flying. This framework achieves the goal of departing from the standard operation of individual connections between drones and corresponding ground controllers whilst maintaining the desired accessibility and affordability compared with other contemporarily available solutions.**

I. Introduction

A. Project Overview

The rapid development, growing availability, and increasing affordability of unmanned aerial vehicles (UAVs) has massively expanded the viability for commercial, militaristic, and enthusiast applications to take advantage of this expanding technological field. This evolution has allowed UAV implementations to propagate through the market with utilisation increasing rapidly in a massive range of situations from military operations to photography, search and rescue scenarios, cinematography, and simply into individuals' hands [1]. These applications have also seen an increasing desire and capability appearing in formation flying, which can expand the utility of unmanned aircraft systems through one-to-many control schemes that allow single operators command over swarms of craft. [2]. However, typical deployments of UAVs still remain in a fixed one-to-one model where a vehicle is connected to and wirelessly communicates with a single ground controller. This standard unmanned aerial system consisting of the UAV, ground control system, and communication interface between the two does offer an immense number of flexible use cases with increased safety over traditional aircraft due to the much smaller size of devices, lack of human operator, and much lower initial investment. Despite this, one-to-many systems hold new opportunities for industry and military that have seen the development of swarmable devices increase as these systems can expand search and rescue operations with automated devices scanning grids from overhead, form light displays that replace fireworks, and generally increase the flexibility of single operator capabilities [1], [3]. A massive issue does remain in that many of these systems are still prohibitively expensive to both commercial and enthusiast consumers, with one-off mesh light shows, for example, estimated to start at around 99,000 United States Dollars (USD) for 200 drone systems, which works out to 495 USD each for just a single use [4]. Other implementations available may also require refinement as large-scale systems are typically complex, follow pre-set paths due to difficulties in computation required, or are difficult to implement due to few consumer-grade UAVs holding simple but extensible ad hoc out-of-box swarm capabilities. These limitations in accessibility and adaptability to manual live input are constraining factors in the usability of UAVs in more complex commercial and consumer-grade scenarios that must be overcome to enable further adoption. As such, this project focused on the development of a framework that allows for the creation of a scalable and extensible system using consumer-grade devices whilst enabling an ad-hoc leader-follower UAV formation to assist in increasing the availability and accessibility of multi-drone unmanned aerial systems.

This project was supervised by Jyoti Sahni, Winston Seah, and Alvin Valera.

B. Environmental and Sustainability Considerations

The primary environmental and sustainability considerations this project and its solution worked towards improving were the United Nations (UN) 8th and 9th sustainability goals [5]. Goals 8 and 9 are defined as to "promote sustained, inclusive and sustainable economic growth, full and productive employment and decent work for all" and to "build resilient infrastructure, promote inclusive and sustainable industrialization and foster innovation," respectively. This project partly addressed each of these goals through the concepts of promoting inclusive economic growth, promoting productive employment, promoting inclusive industrialisation, and fostering innovation. Each of these points has attempted to be improved by enabling an affordable and accessible drone-swarming solution that can allow individuals and small businesses to implement and begin utilising more UAVs in their processes. Inclusive economic growth and inclusive industrialisation have been assisted by the affordability of the solution created and the accessibility and wide availability of the devices used, allowing for implementations to occur at all levels of business rather than just large corporations. Fostering innovation benefits from these traits, too;

however, it also benefits from the flexibility in control schemes and onboard processing that this solution has implemented, which massively increases the potential programmability and specialisation for use cases that the devices are capable of compared to their factory-provided state. Promoting productive employment has been assisted by enabling greater swarming control and aiding in providing more swarming opportunities, which allows drone operators to be far more efficient when needing to manage one-to-many UAV systems. Addressing both of these goals has been a vital driving force behind the restrictions and direction of this project, so managing to partially fulfill them and improve these noted regions has been a primary consideration and benchmark for the undertaken development process.

C. Project Requirements

The primary requirements of this project were guided by the principle desires of increasing the accessibility of swarming, providing the ability to follow live manual input, and maximising the extensibility and scalability of the created solution. These considerations align with the UN sustainability goals 8 and 9 through promoting inclusive economic growth, promoting productive employment, promoting inclusive industrialisation, and fostering innovation as the project is working towards providing these improved behaviours within an affordable and replicable solution.

Based on these considerations, the following basic requirements for the solution were defined:

1) The developed solution must only require a ground controller communicating directly with one drone member at a time, that is in turn responsible for informing all other members of received commands. This responsibility must be able to be undertaken by any of the system members the operator chooses when establishing the system for use. Whichever member is currently responsible for this is to be referred to as the current system leader while it handles this operation.
2) The developed solution must offer scalability by being able to accommodate for and handle the addition of new devices quickly and safely once they are configured to the settings the system requires.
3) The developed solution must be extensible through providing a system that can be further configured and developed for usage specialisation by operators.
4) The developed solution must be able to handle the replication of live flight command communications across all included drones swiftly and accurately.
5) The developed solution must be replicable and accessible to enthusiasts and small business implementations.
6) The developed solution must not introduce excessive management overhead that prevents prompt and real-time actioning of commands across devices to ensure the system is responsive and safe for flight.
7) The developed solution must have a safe handling of communications failures in the event the ground controller is no longer able to feed commands into the system or in cases where the current system leader and its followers are unable to contact each other.

D. Project Outcomes

1) Developed Solution Overview: The principle desires of the project requirements led to the design and development of the produced solution towards a mesh network containing drone nodes and a ground controller application.

Mesh nodes in the system consist of paired Raspberry Pi Zero W devices and Tello Talent drones, with each Raspberry Pi being responsible for managing its respective drone and communicating with neighbour nodes. This responsibility is achieved through each Raspberry Pi Zero W having an additional wireless interface added alongside its inbuilt capabilities to allow the device to connect to a common Wi-Fi network the mesh is hosted in whilst also being able to broadcast its own wireless access point that its drone and the ground controller can connect to. Communication between the managing Raspberry Pi and its drone is achieved through broadcasting commands in plain text via User Datagram Protocol (UDP) packets in accordance to the Tello 3.0 Software Development Kit (SDK) specifications [6] while communication between Raspberry Pis, and therefore nodes, is achieved through the usage of the Better Approach to Mobile Ad-hoc Networking advanced (B.A.T.M.A.N. advanced) utility which uses ad-hoc Wi-Fi networks to form a mesh network of Linux devices [7].

The ground controller application has been developed with a React and TypeScript front-end application communicating with a Node.js back-end server that broadcasts and receives plain text flight commands and responses via UDP. These communications are undertaken with the network member it is currently connected to via the Raspberry Pi-hosted Wi-Fi network. When connected, the ground control software is able to alert the connected member that it is the new leader, which will then inform the mesh network of its elevated status for command and response routing.

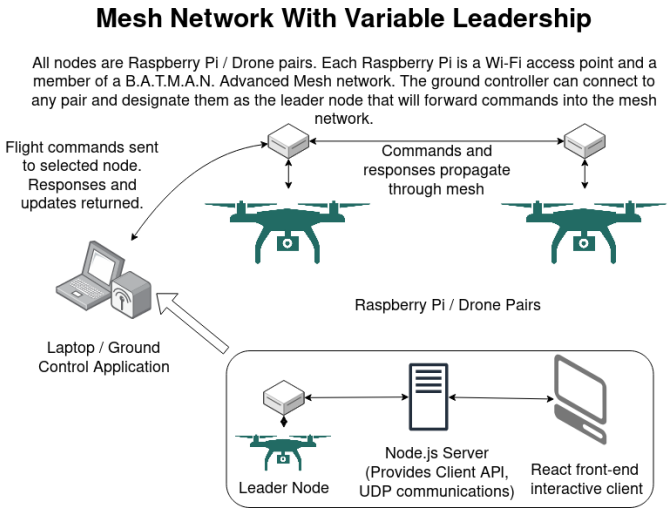

Fig. 1: The Developed Solution Topology Overview

2) Performance of the Developed Solution: The solution topology outlined in Fig. 1, defined as a mesh network with variable leadership, overviews the constructed solution relationships. Producing this structure fulfilled many of the project requirements, with operators only needing to direct messages to the currently designated leader device to control both of the two configured nodes the project has prepared in live and at-will flight. Despite only having two nodes currently configured, the developed solution provides the desired scalability with the capacity available to extend this current implementation with only minor configuration required from the repository settings as defined in Appendix A, "Configuration of a Mesh Networked UAV-UAV Communications Structure" whilst also providing the extensibility of the system through each node having processing power available on its assigned Raspberry Pi Zero W. This scalability was ensured through the aforementioned project requirements guiding the solution to designate the current system leader the responsibility of learning the details of its follower devices on demand and forwarding communications and responses received accordingly to permit the swift replication of flight commands. The requirements for replicability and accessibility in the designed solution drove the decision to use the widely available and reasonably affordable Tello Talent drones and Raspberry Pi Zero Ws as the base of the solution, which provide a current base price of the core components of 453 New Zealand Dollars (NZD) [8], [9]. This costing excludes the variable pricing of the selected Wi-Fi adapters and additional cables, as these are changeable based on implementation requirements alongside the mounting solution costs, which were not able to be finalised during the project, as will be discussed in the hardware implementation section. These item choices worked in tandem with the sustainability focus of the project, as using such relatively affordable and widely available parts and devices would allow for the replicability of this system for a theoretical enthusiast or small business implementation. Choosing the Tello Talent also allowed for the natural fulfilment of the 7th project requirement as the devices have multiple helpful inbuilt safety features, including an automatic landing capability when no commands have been received in the last 15 seconds when in the utilised SDK mode when the battery level is too low, or when the connection to the managing device is too weak or lost [10]. These native behaviours allow for the safe handling of flight issues or failures in the system, which is vital for potential cases where devices which may only be indirectly managed are no longer commandable.

Flight performance of the solution was intended to be undertaken through measuring connectivity metrics, flight performance, and failure resiliency; however, issues in the final implementation, not including an in-flight power and mounting solution for the Raspberry Pi Zero Ws to be attached to their Tello Talent have prevented some test data being gathered. Despite this, tests were able to be undertaken with a temporary unmounted solution which showed that actions were able to be replicated across the mesh network reliably, safely thanks to the inbuilt behaviours, and swiftly without undue overhead being added in system end-to-end communications. This is shown through the average round trip communications time increasing by an average of 6.3 milliseconds for the leader drone and 25.8 milliseconds for the follower drone across hover and movement tests against the baseline one-to-one performance of 28.85 milliseconds. Overall, the performance of the system was found to be acceptable for an initial implementation; however, clear areas for improvement in further developments and testing have been identified.

3) Project Tools and Methodology: The development of the final implementation of this system has seen a wide range of programming languages, hardware, libraries, and tools utilised. The programming languages included in the developed solution are Python for the Raspberry Pi onboard flight management software, TypeScript with React for the ground controller front-end software, and Node.js for the ground controller back-end server. Python was selected for the onboard flight management software following initial development in C++ as it enabled much faster and safer development and future extensibility despite the notable performance disparities in the languages [11], [12]. This trade-off overall benefited the project design as Python enables a much lower barrier to entry for extensible development with its emphasis on readability alongside an increased safety in development due to the automatic memory management the language holds that C++ does not. This is vital considering the low Random-access memory (RAM) of 512 Megabytes (MB) held onboard the Raspberry Pi Zero W [13] and the risks of memory mismanagement potentially causing issues for the system in flight. For the ground controller software, the TypeScript+React front-end and Node.js back-end were chosen for the widespread support of the systems, the potential modularity for redevelopment React brings, and the powerful libraries that exist within the ecosystem that allow for swift development.

In the course of development in these languages, four primary system-enabling libraries were vital for enabling the major functionalities of the system. These were React, Express.js, dgram.js, and Python's socket library [14]–[17]. React is a common front-end library that enabled the quick development of the interactive portion of the ground controller software that was developed with Vite to allow for easy starting and live refreshing of the development version of the application as changes were being made. Express.js is a web framework for Node.js that made the development of the back-end server quick due to the ability to quickly create an Application Programming Interface (API) the React client would be able to communicate with. This heavily benefited the design of the system as it made the communications of commands through POST messages to the Node.js server a quick and flexible task. These commands are processed and then sent to the leader device using the dgram.js library, which enables the quick and direct integration of the UDP communications functionalities required for the flight

command control structure. Python's socket library is used in the flight control software to enable the receiving and sending of commands and responses from the Raspberry Pi Zero W devices, which is vital for being able to process and redirect content in the system as required. Together, these libraries provided the key functionalities that the system required and enabled development to continue on the logic and processing of the control system without the need to redevelop these existing structures.

The Raspberry Pi Zero W and Tello Talent were chosen for the base solution hardware used because of their flexibility and widespread availability. Raspberry Pi Zero Ws, as complete single-board computers, opened up many design and development avenues for the system compared to what other more limited processing devices would have permitted. Despite extra potential issues in processing timings being affected by operating system activities, the potential to run a complete Linux installation in Raspbian [18] opened many options for software, programming languages, and utilities used in the solution design. This was especially useful for enabling the usage of the B.A.T.M.A.N. Advanced, dnsmasq, and hostapd utilities that enable the mesh network functionality, domain name system caching, and Wi-Fi network hosting capabilities that this entire solution relies on [7], [19], [20]. Without the flexibility of the Raspberry Pi platform and these utilities, the system would have required a dramatically different design. The Tello Talent hardware choice benefited the solution design due to the development capabilities inherent to the product, the inbuilt safety measures, and the accessibility of the model. The Tello Talent SDK [6] accessibility and flexibility, especially with being able to convert the drone from access point host to access point member, enabled the development of the system as complete control could be achieved through the simple hosting of a Wi-Fi network on the Raspberry Pi devices. The inbuilt automatic landing safety measures benefited the system design immensely with such a vital project requirement being implemented by default in a reliable and definite way that software control would have had issues replicating compared to the firmware-defined actions.

II. Related Work

A. UAV Development and Implementations

UAVs, as with many areas of technology, have historically seen development driven by militaries for warfare, reconnaissance, or intelligence gathering [21]. A renewed interest arose in the 1970s and 1980s, which led to widescale military development programs; however, in the past decades commercial, industrial, and enthusiast interest in UAVs has expanded as the technology has matured and become more accessible for small-scale deployments.

Historically, these developed UAV systems have been configured in one-to-one schemes with ground control systems being linked to single air vehicles. This format is a powerful tool in a range of situations, but the usability of swarms, formation flying, and drone-drone communication is a massive field that is dramatically expanding potential UAV applications. Search and rescue operations can be expanded across rough or remote terrain with autonomous UAVs, drone-based light shows can replace fireworks, and swarms can even be used in surveying and agricultural settings [1], [3], [22]. Many of these implementations are, however, restrictive in terms of scalability or controllability, have to follow preprogrammed routes, or are expensive to implement and, as such, are prohibitive for individuals or independent companies that may not have the financial backing that militaries, governments, or large corporations have.

Current commercial drone swarming solutions are available for the public to purchase with systems such as Teal's 4-Ship multiple drone controller and out-of-box functionality in the DJI Tello EDU and Tello Talent devices; however, these systems each have disadvantages in their implementations that this project aimed to rectify.

B. Existing Commercially Available Swarming Solutions

Teal produces a range of drone devices with a focus on producing US DoD-compliant devices such as the Teal 2, but has also produced the Golden Eagle [23]. Teal's 4-Ship control software allows a single operator to control four Golden Eagles simultaneously and provides a focus on surveillance, reconnaissance, mapping, or surveying applications [24], [25]. This implementation can reduce the costs and times required to a quarter of a single vehicle due to the ability to have four UAVs working together simultaneously from a single operator and, as such, fulfils some of the needs of this project's focus; however, the implementation and costs of this system are prohibitively expensive for most applications. With a single Golden Eagle costing more than ten thousand United States Dollars (USD), completing a kit of this scale is completely out of scope for enthusiasts and many companies. Instead, Teal's solution focuses on governmental departments, defence, and security clientele who have the desire and capability to invest massively into their swarm technology. As this project aimed to instead provide an accessible solution at a fraction of the cost of Teal's 4-Ship control system, this product fills a market and technological implementation space far outside of what was in the scope of development.

The DJI Tello EDU and Tello Talent are devices that were deemed feasible for usage within this project. They are available at a much more accessible price point from DJI of 219 and 389 Australian Dollars (AUD) per device, respectively [9], [26]. This massive price difference stems from the sizing, functionality, and intended usage differences between the Tello and Teal devices; however, the Tello devices provide the core functionality required to form a swarm. This fits within an acceptable price point for enthusiast and commercial applications, and also both are capable of swarming out of the box. There are two major issues this designed swarm capability does have, however. These are the typical implementations requiring fixed, preprogrammed

commands and the lack of adaptability in terms of deployment scalability. The Tello SDK 3.0 declares the capability for devices to be communicated with through text commands via the Wi-Fi UDP protocol [6]. This SDK defines flight control commands; device information read commands, and expected behavioural responses for usage in development when the aircraft is connected with the UDP command setting. These capabilities allow users to connect their devices to defined Wi-Fi networks where they can then use the same network to communicate with multiple devices at the same time through code. Typically, this code involves pre-setting the exact drone information and sending commands to specific devices individually, with swarming occurring from repeating the same command information to all devices the user adds code for. This solution is very limited in terms of scalability and control complexity, which are features this project aimed to solve as the swarming structure is limited in the ability to handle changing swarm sizes with each device needing to be found and programmed for whilst also being limited to a range of the host network. The implemented solution instead allows for dynamic sizing with the replication of commands across any Raspberry Pi and Tello pair that are accessible within the mesh network. The accessibility and count of devices do not need to be known for the ground controller to function as the flight management software on the devices in the B.A.T.M.A.N advanced mesh network can locate neighbours and adapt with the changing node topology, which allows for command replication to be automatically scaled as the network expands. This capability is combined with the range expandability that utilising a mesh network of nodes brings through, allowing devices to communicate with each other even if they are not direct neighbours. This permits the theoretical expansion of the system across wider physical areas where node Wi-Fi networks overlap to extend the swarm range in a much more effective way than a single access point control system the Tello devices are typically used in at the cost of longer response times from devices further from the leading node. Utilising this methodology of command communication also allows for more effective ad-hoc flight controls as the added onboard processing available to the user through the usage of Raspberry Pis massively expands the capability and potential for pre-set or extended actions compared to normal operation. This is because the Tello devices are limited to the SDK commands available for flight actions; however, independent processing capabilities running for each pair unlocks an avenue for larger user-defined routine functionalities and specialised control logic to be built into the system. This increased device independence can reduce ground controller workload whilst providing much more complex control systems that leverage the SDK capabilities and onboard processing of device statuses and commands. These expansions to the capabilities of the Tello devices that this project focused on massively improve the potential usability, flexibility, and viability of swarm usage for minimal further investment compared to upgrading to more expensive commercial devices or creating custom UAVs.

Custom UAV systems are another existing solution that would be able to permit a similar performance to the solution developed in this project. This is especially the case with guides available for Raspberry Pi and Pixhawk (or similar flight controller) systems that would enable similar B.A.T.M.A.N. advanced mesh control systems, complex on-board software development, and more precise flight and control management than the developed system offers [27]. Despite the shared capabilities, these systems have a higher barrier to entry with the complexity of construction required alongside a higher entry price with kits being available from around 899 USD [28]. Sourcing the required components individually is possible; however, this added complexity in purchasing and assembling devices is a problem that this project design reduces as the Tello devices come pre-built and ready for flight with the only additional component construction in designing and implementing a mounting and Raspberry Pi power solution.

III. Design

The design of the developed solution was heavily guided by the aforementioned project requirements. As such, the decisions regarding what design choices should be made when there are multiple possible development avenues rested heavily on which solution would better fulfil the requirements and sustainability goals of the project. These considerations kept in mind will be justified through this design section whilst outlining the selected system design.

A. Design Overview and Sustainability

The final system design centered on creating a mesh network of affordable nodes consisting of a drone and a managing device that commands the drone and controls communications of flight commands and responses between neighbour nodes. This system was to be connected to an interactive ground controller that would be used by the operator to broadcast flight commands into the mesh and receive responses and status updates from all connected nodes. This ground controller software was only to communicate live flight commands to one mesh node at a time, with the currently selected pair acting as the current leader and distributing commands to all followers within the network. Enabling this communications structure required that each mesh node managing device have the capability for running an on-board flight management software that would process and forward received commands, keep track of the system topology, and provide extensible development through being easily modifiable to add specific routines or complex management methods.

These concepts drove the design to call for Raspberry Pi Zero W devices mounted atop Tello Talent drones running software to provide in-flight management of both the connected UAV and communications with neighbouring Raspberry Pi devices. These were designed to be achieved through the Raspberry Pi hosting and connecting to two Wi-Fi networks - the first being a hosted Access Point that the paired drone and ground controller software could connect to

and communicate with and the second being an interface for the mesh network solution. This design decision also drew from the UN sustainability goal considerations the project aimed to address. Both of these items are widely available and at a much cheaper price point than the other discussed existing solutions, which enables inclusive economic growth opportunities for operators of this system comparatively. Additionally, this design kept technical sustainability in mind through selecting the Raspberry Pi platform, which provides a free and open-source implementation of the Debian operating system in Raspbian [18], [29]. This means that the system is sustainable across a longer timescale due to the tools and updates available in free community libraries and the option to run and modify the system software as needed, even if the support of the operating system for the devices used is discontinued. Additionally, the long-term sustainability of the devices utilised in this specific system implementation has been considered and reassured due to the Raspberry Pi obsolescence statement available on many of their products. For the Raspberry Pi Zero W, there is a stated minimum lifespan of production until January 2026, while the Raspberry Pi Zero 2 W, which could be used in place of it as they are the same size, has a stated minimum production lifespan of January 2028 [13], [30]. These dates provide certainty to the long-term sustainability of this project design as new items can be purchased through to at least these stated dates, device support will be ongoing accordingly, and the system has been designed in a way where the mesh solution is able to be drone model agnostic. Although the system calls for the Tello Talent drone, any new device that provides an open SDK, communicates via Wi-Fi, and provides the lifting power and shape a mount can be designed for would be able to be used in the system with compatibility changes made in the flight control software. These features of the selected items together show how the designed solution promotes economic and technical sustainability in line with the sustainability goals of the project.

B. System Topology

In designing the system topology, two primary approaches were considered in concepts based around a single leader with fixed followers and a mesh network with variable leadership.

Initially, the single leader with fixed followers structure, as outlined in Fig. 2, was considered for perusal. Implementing this would enable swarming and flight command replication by creating a single Raspberry Pi-Tello pair that would act as the leader device. The Raspberry Pi mounted to the UAV would be a Wi-Fi access point to which all devices within the topology would connect. This design would allow for the ground controller to communicate with the leader Raspberry Pi, which would then forward received commands to all connected UAVs through UDP packets. This solution would fulfil the desire of this project to allow a single ground controller to control multiple drones with only needing to communicate with a leader device; however, it lacks swarm sizing extensibility due to the limited control range offered, and it provides less complex control processing opportunities.

Swarm sizing extensibility is vital in ensuring that the created solution can expand from the limited development implementation created within this project to a theoretical larger system with more aircraft. The main factor that affects this explored solution's capability in this regard is the issue of control range caused by the inherent limitation found in all devices connecting to the leader Raspberry Pi's access point rather than allowing for the network to extend across devices. As all devices in this solution would connect to a single Wi-Fi network, the range of usable flying would be permanently limited to what is produced from the leader device. This severely restricts the usability of this implementation as practical usages of swarms need the flexibility to spread craft further than what a single Wi-Fi network may provide. Complex onboard processing is a major restriction that this solution would hold due to the practical limitations of controlling each connected device from a single Raspberry Pi. Although logic can be handled at a small scale, as in this project's implementation, the requirements of managing larger networks on a single low-powered Raspberry Pi Zero may not be practical for what is required to keep a flying formation safe. Instead, the ability to process commands and system activity onboard each device was preferable to reduce the peak workload on any given device whilst also increasing the overall command or processing complexity possible within the system.

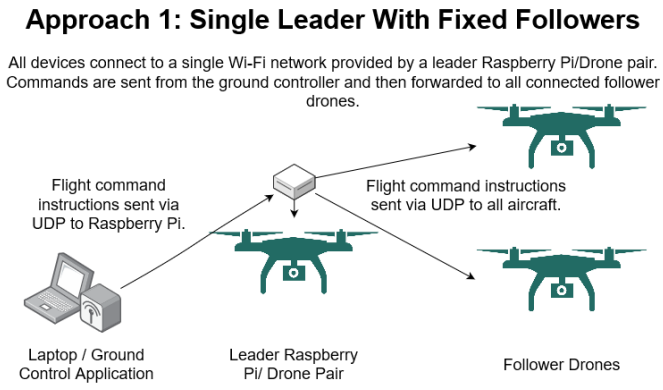

Fig. 2: Explored Single Leader Topology

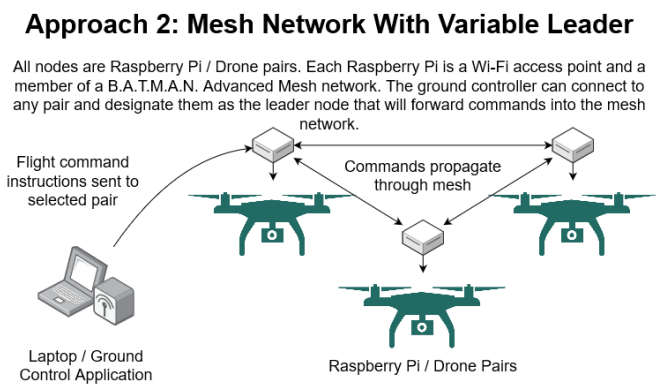

Fig. 3: Explored Mesh Network Topology

Considering these limitations, the mesh network with variable leadership design, outlined in Fig. 3, was preferred as this solution solved the major flaws the single leader with fixed follower design held. This structure involves each UAV being placed in a Raspberry Pi-Tello drone pair with the mounted Raspberry Pi Zero W acting as a wireless access point for the drone it sits atop with one wireless interface whilst also being a member of a B.A.T.M.A.N. Advanced mesh network on a second wireless interface. This allows each drone to have a constant connection to its controlling network and device while permitting flight command replication by sending information within the mesh network. In this situation, the ground controller can be connected to any node in the network by joining their active Wi-Fi network. Connecting to a Raspberry Pi-Tello pair designates that device as the leader of the swarm, and all commands will be sent through and replicated through this bridging system. Commands in this structure are thus sent from the ground controller to the currently selected Raspberry Pi, which then sends the command via UDP to the Tello it is mounted atop whilst also forwarding the content to its neighbour nodes by flooding the network. These neighbours can then forward the same control messaging to their relevant drones, return drone responses and, through B.A.T.M.A.N. advanced, rebroadcast the commands to neighbours allowing the command to propagate through the network [31].

This solution solves the extensibility of the system by allowing for any configured Raspberry Pi-Tello pairs to be automatically added to the network by simply powering them on as the B.A.T.M.A.N. Advanced mesh network will detect them through nearby neighbours and begin broadcasting commands to the new devices. This process simplifies the network extension immensely, as no central device needs to be modified to accommodate all new devices. Instead, only the nearest neighbours will have overhead in searching and providing communications for the mest to the new devices whilst the flight control processing will be handled on the new pairs dedicated Raspberry Pi.

Utilising a B.A.T.M.A.N. Advanced network also solves the range issue identified in the single leader solution. As long as a device has a neighbour in reach that has received the broadcast command, it will be in range of the network, and as such, the system can be chained across a much larger distance than a single Wi-Fi network would allow. This improves the expandability of the network usability immensely compared to the initial design, as the member UAVs would be able to cover a much greater distance whilst still only requiring the control commands to be sent to the current designated leader device.

Having each aircraft hold a dedicated Raspberry Pi expands the onboard processing capabilities of this system immensely, as the limited power each device holds is only dedicated to managing its paired drone. This can allow for the development of more complex tasks each Raspberry Pi can run. For example, this power could be used to design and define multiple command actions, which involve taking in a single custom command and then converting this into a series of SDK-compliant commands for example. Primarily, though, this processing power availability allows for an overall much safer system as it can permit the development of future system safety and monitoring tools that could improve the overall quality of network performance, such as flight health monitoring tools.

C. Flight Control Software

Once this distributed processing-capable mesh solution was selected, the onboard flight control software design trended towards the concept of an identical control script running across all Raspberry Pi Zero Ws in the system. Running the same software with the inbuilt capabilities of adapting the instance to behave as either a leader or follower on system start as required was a core requirement for enabling the variable leadership model that the design desired. This functionality was to be achieved through the flight control software having the ability to flood the mesh network with a message informing all devices of its elevated status to ensure they replicate commands sourced from it through flight and return drone response messages accordingly. This developed software was thus designed to enable communicating along both wireless interfaces available to the Raspberry Pi Zero Ws in the system to allow for the required communications within the B.A.T.M.A.N. advanced mesh network through neighbouring nodes and within the configured hosted access point. Enabling this functionality in an extensible, safe, and swift way was a core consideration for designing the specific implementation details of the flight control software, as the minimisation of added communication overhead and system processing requirements had to be balanced against the ability for the software to be redeveloped swiftly and safely for expanded capabilities. As such, the initial design concept for the flight control software called for using C++ to create the system; however, Python was chosen for the final design.

C++ was initially chosen for its known speed afforded to it by being a powerful compiled language with much lower overhead than other potential languages. This afforded speed and minimal processing requirements would be ideal for an in-flight system that must permit swift communication throughput to maintain operator control over devices they are indirectly managing; however, this system does come at the cost of ease of use. The added strict language requirements, manual memory management, and high barrier to entry in terms of writing good code that C++ faces heavily limit the potential extensible of the software for potential users who want to modify the software for their specific needs. Although the same basic management structure the design created could be recreated by another use, this adds a barrier to the innovation potential this solution aimed for as the excess work required reduces potential productive development time. Additionally, the potential challenges in writing safe and reliable C++ code in flight, particularly around memory management on a system with only 512

MB of RAM available for an entire operating system, raised concern about the viability of providing a safe platform for potential users to develop from. As such, the decision was made to seek alternative design options for the flight control software solution.

Python, despite its slower performance when compared to C++ [11], [12], offers a far lower barrier to entry for potential future system users through its simpler syntax, native presence of a garbage collector for memory management [32], large standard and community library, and inbuilt extensibility [33]. These quality of life and usability improvements contained within Python made it a clear choice for the final flight control system design as it enables both swift and accessible development for both this project and any future potential work whilst still maintaining the complex development opportunities and potential user-determined speed and functionality improvements. The extensibility and greater refinements Python offers are particularly powerful for any users who have the confidence or need to refine the system as they can turn off automated garbage collection [34] and extend the system with modules written in faster languages [33] as required if they are seeking performance improvements. Python offering these more complex options alongside providing a user-friendly system for less confident users of the system, which can enable safe or improved speed, drove the final decision in opting to use the language for the flight control software.

D. Ground Control Software

The ground control software's initial design concept followed the simple structure of providing a visual interface that a system operator could use to command the mesh network of devices whilst receiving updates on the system status alongside mesh updates and drone responses through communicating with the flight management system instance running on the leader drone. This led to the initial basic design concept outlined in Fig. 4.

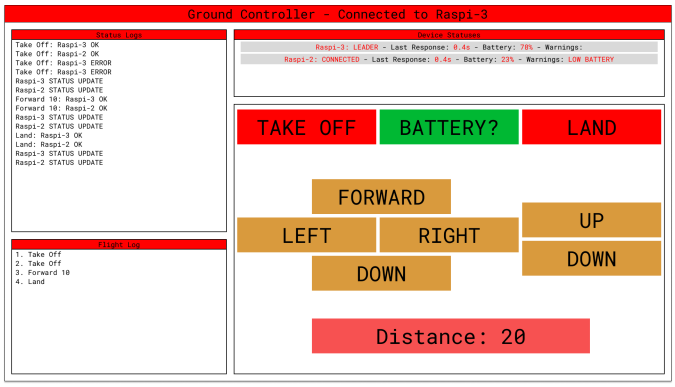

Fig. 4: Initial Planned Ground Control Software Design

This solution was designed to offer a clear and basic command scheme that permitted the user to undertake basic flight actions whilst receiving drone responses in the status logs section, basic device information in the device statuses section, and a recording of the flight in the flight log section. These sections were selected for the information they offer, being valuable for both in-flight and post-flight analysis of the system. The status logs section is vital for identifying the current command's completion status alongside notifying the user of any errors encountered with commands. The flight log section allows for a much more visible display of the commands and paths undertaken in flight to ensure that they are correctly followed. The device statuses section gives a quick overview of the discovered nodes, the relevant drone's battery status, and the last response time to ensure the user can quickly identify if there are any power or communication faults. However, the need for these sections to contain so much information drove a large design change through this project planning. The initially planned system aimed to use a cross-platform framework such as Flutter to develop the ground control software with both desktop and mobile functionality in mind. This was intended to allow the operator to select either mode for managing the system as preferred; however, the amount of information planned to be displayed alongside the space required for the control scheme rendered this design unviable. Instead, the decision was made to limit the software to desktop usage as the screen real estate and sizing of laptop and desktop devices would much more easily permit all the required information and controls to be displayed. This decision limited the expected flexibility of the system; however, the design decision had to be made to ensure all information was legible, and controls were able to be interacted with correctly and consistently, which is vital for ensuring the safe operation of such a system.

E. Hardware

The hardware design settled quickly on the concept of mounting an Arduino or Raspberry Pi device onto each drone used to form the desired mesh solution. These options were considered due to the additional logic that can be added to them and the meshing capabilities the platforms hold. This choice was also influenced by the design considerations around drone devices, as the Tello Talent comes packaged with an Arduino ESP32 controller [9]. Despite this, the uncertainty in the design stage around whether the Tello Talent or Tello EDU device would be selected for purchase based on university resourcing and potential future usage meant that the usage of this device was not a definite answer for the solution. As such, the design had to consider either of the system design viabilities.

This decision focused instead on the potential fulfilment of the project requirements each platform had and the potential shown in each solution from existing similar solutions. The Raspberry Pi-based design was preferred in both of these considerations due to the much wider potential for development and innovation available on the complete computer system for users, despite the added operating system overhead, the immediate compatibility it offered across both devices and the existing community Raspberry Pi mounting that offered guidance on the viability of the solution. The

existing onboard Raspberry Pi solutions and mounts designed by the Tello community [35], [36] supported the viability of the desired solution as owners had been able to power and connect Raspberry Pi devices to their Tello and Tello EDU devices in flight. As such, the final hardware design of Raspberry Pi Zero Ws mounted atop and powered in flight by the micro Universal Serial Bus (USB) slot available on the Tello Talent or Tello EDU devices was selected. These existing open and available resources also allowed for adapting the mounting and system solution to accommodate the additional USB Wi-Fi adaptor that the Raspberry Pi design required for providing an additional wireless interface.

IV. Implementation

A. Overview of the Delivered Implementation

The implemented system was able to match most of the design expectations with a few modifications and finalisations in terms of the framework used for the ground controller system and in the final Raspberry Pi mounting status. The mesh network system, outlined in Fig. 1, has been completed, with the configuration of the devices, ground controller software, flight control software, and hardware implementations outside of the in-flight Raspberry Pi power solution completed. This final implementation currently permits the management of two nodes, as seen in flight in Fig. 5, due to difficulties in sourcing compatible Wi-Fi adaptors for the Raspberry Pi Zero W's second WLAN1 wireless interface. However, has the capacity to be used with far more devices with this issue solved.

Fig. 5: Mesh Network Controlled Tello Talents in Flight

The final implemented system consists of unmounted mesh nodes of Raspberry Pi Zero W devices paired with Tello Talent drones through the UAV connecting to the Wi-Fi network hosted on its managing Raspberry Pi. Each Tello Talent drone does, however, have an additional unpowered Raspberry Pi Zero W attached in flight to simulate the mounting solution's performance impact on their flight abilities. The powered Raspberry Pi Zero Ws both run the completed python based flight control software, which can be set to either act as the system leader or follower based on the ground control software, built in React and TypeScript with a Node.js back-end, running on a laptop connected to the selected leader's broadcast Wi-Fi network. This ground control software implements a system starting process that informs the currently designated device of its elevated status to begin the communication of leadership to the follower through flooding the B.A.T.M.A.N. advanced mesh network the devices are connected to on their WLAN0 interface. Once this status message has been broadcast, the message "command" is sent through the mesh, as per the Tello 3.0 SDK [6], to prepare all nodes for flight by switching the drones into SDK mode. Once this command is broadcast, the flight control software begins receiving drone status updates, which are emitted multiple times a second from the Tello Talent devices. These updates begin to populate the device status information available in the ground controller software, which indicates to the user that the system is flight-ready with the discovered nodes. Commands can then be sent into the system solely through the leader node and then replicated across both drones through the constructed indirect management whilst responses are returned from all devices.

B. Mesh Network Overview

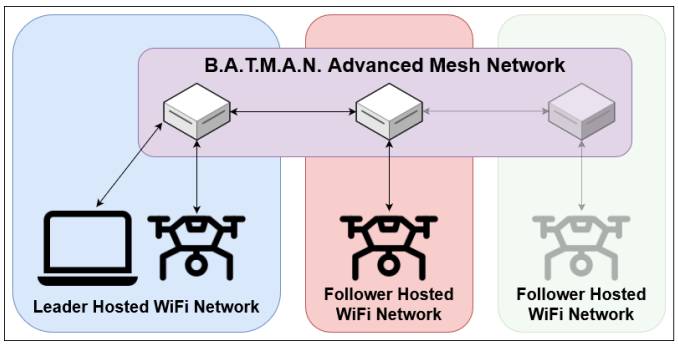

Fig. 6: Mesh Node Network Relationships

The mesh network has been implemented with the planned B.A.T.M.A.N. advanced utility running on each managing Raspberry Pi Zero W drawing from the Innes and Walicki guide [37]. This system is accessed through adding a virtual "bat0" interface on the systems WLAN0 interface alongside setting the default WLAN0 network to start on an ad-hoc "raspi-mesh" network. With each device set to these ad-hoc network settings, they automatically detect neighbour nodes. Additionally, each device has a unique static Internet Protocol (IP) address set in the 192.168.1.X/24 range through the device's dhcpcd settings file, which is used to provide a static address the flight control software can communicate with. The wireless interface added via USB provides a WLAN1 interface to each Raspberry Pi. This interface is set to act as a Wi-Fi access point through the hostapd utility, which allows both the Tello Talent paired with the device and the laptop running the ground controller software to connect to the device. These interface interactions are shown in Fig. 6. This diagram shows the distinct networks being broadcast from each node on the WLAN1 interface with the leader and follower-hosted Wi-Fi network sections whilst the common ad-hoc network

used on WLAN0/bat0 is shared across devices. Fig. 6 also shows, through the right-most section, the potential scalability capacity this solution has as further nodes can be added to the system with the same configuration setting applied to managing Raspberry Pi devices. Once this is done, the B.A.T.M.A.N. advanced mesh members would begin detecting the new neighbour node if it is in Wi-Fi range which then permits the utilisation of the node in the network. This functionality is assured through the flight control software flooding the address space through an informed leader broadcasting on 192.168.1.255, meaning that all devices in this potential range can detect and begin following the elevated node.

C. Flight Control Software

The flight control software was ultimately implemented as a Python run on the Raspberry Pi Zero W devices. The software successfully interfaces with the drone the node manages via the WLAN1 hosted Wi-Fi network, neighbouring nodes via the WLAN0/bat0 interface, and the ground controller when selected to be the current system leader running the ground controller. The script communicates to all of these systems using plaintext UDP packets, as seen in Fig. 7, containing either a command message meant for its drone and followers or a response message directed toward the ground controller. These messages are plaintext to provide simple communications and to match the Tello 3.0 SDK, which requires commands to be sent through this format to port 8889 on the controlled drone [6]. This flight management script takes in arguments containing the ground controller and drone IP addresses alongside the IP address the node occupies in the mesh space, such as in "i.e. python3 flight_control.py 192.168.10.4 192.168.10.6 192.168.1.10". These addresses are taken in as arguments to manage the variability in the IP address assignments given to the ground controller and drone when connecting to the Raspberry Pi Wi-Fi network alongside preventing hard coding from being required in each software instance each time the addresses change. This information provides the base settings of the management system until it receives information on whether its instance is a leader or a follower.

D. Ground Control Software

The ground control software did follow the final design concept of being a desktop system rather than a multi-platform solution; however, the selected language and structure of the application changed from the design phase. The initial Dart+Flutter selection was abandoned for the developed React front-end and node.js back-end solution. This decision was made due to React's web app nature more easily fitting the new structure alongside the fast developing and scaling solution that the library offers [38]. These features, alongside the much wider library opportunities available through JavaScript/TypeScript, led to the React front-end being created with Vite and TypeScript to produce the basic ground controller application shown in Fig. 8. This front-end application interfaces with the back-end Node.js server that runs uses the dgram library [16] to broadcast and receive the plaintext messages the flight control software and drones use via UDP. This interaction between the front-end and back-end, shown in Fig. 9, is enabled through the Node.js server running express [15], which provides API endpoints that the front-end posts the appropriate flight command content to as directed by the on-screen buttons. In turn, the Node.js back-end emits server-sent events to update the content shown to the user through the React front-end when it receives updates from the system.

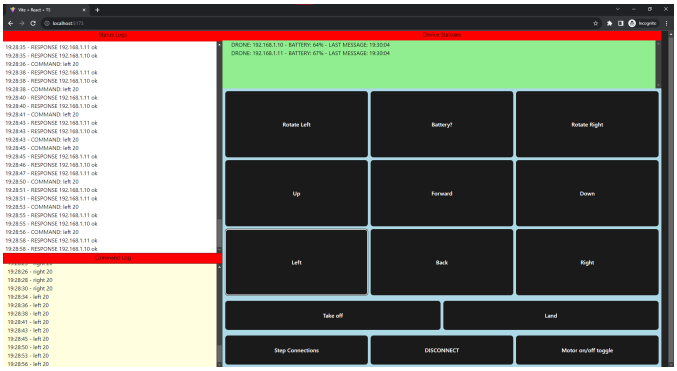

Fig. 8: Implemented Ground Control Software

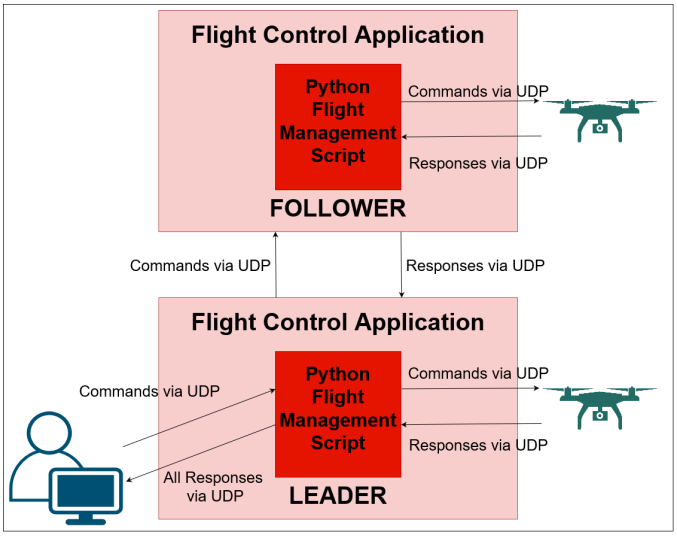

Fig. 7: Implemented Flight Control Software Design

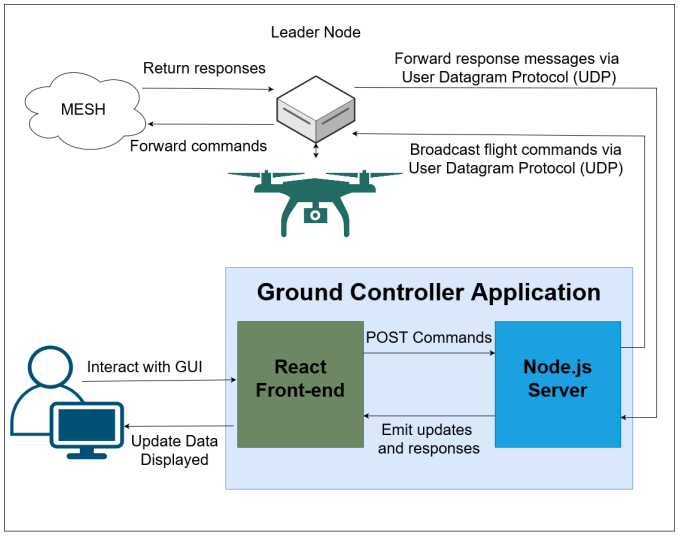

Fig. 9: Implemented Ground Control Software Design

E. Hardware

The solution hardware saw the biggest changes and challenges in converting the solution from design to implementation. This was primarily due to Linux driver issues with Wi-Fi adaptor chipsets and firmware changes across the Tello EDU and Tello Talent. The current solution, as mentioned, includes Raspberry Pi Zero Ws and Tello Talents performing as expected; however, the ancillary hardware in the Wi-Fi adaptor and in-flight management solution for the Raspberry Pis have caused issues in producing the expected design. Raspberry Pi Wi-Fi adaptor compatibility issues prevented against multiple Wi-Fi adaptor models being used in this project. Initial designs and productions were achieved through the usage of an on-hand TL-WN821N V4 Wi-Fi adaptor, which uses the RTL8192cu chipset [39]. This is a version well supported by the operating system, which led to the belief that the system should work with newer devices. This initial testing exposed itself as misleading with testing of RTL8188eu and RTL8192eu chipset-based devices, which are commonly available in new products such as the TL-WN725N V2/V3 and TL-WN821N V6 devices, respectively [40], [41]. Unlike the TL-WN821N V4, these products are not well supported by the kernel, so they rely on community drivers such as the lwfinger rtl8188e repository [42], however, these systems do not provide the same functionality as the TL-WN821N V4 adaptor and as such would have required major reworking of the system configuration to permit the same level of operation. This issue heavily meant that either the system design would have to be shifted or additional older adaptors had to be sourced to provide more functional nodes. Fortunately, another TL-WN821N V4 adaptor was located and attached to the second Raspberry Pi, as seen in Fig. 10, enabling the operation of the system at the expense of the current scalability of the implementation.

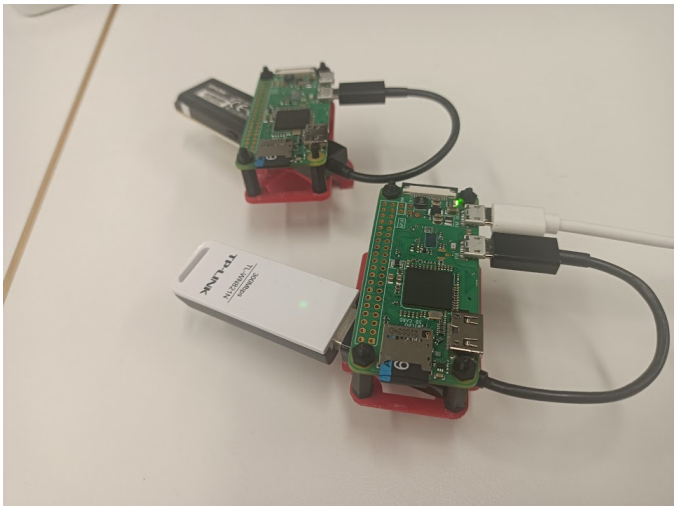

Fig. 10: Raspberry Pi Zero W Devices with Wi-Fi Adaptors

The second change between design and implementation is in the in-flight power solution for the Raspberry Pi Zero Ws. Although examples existed supporting the potential of using the micro USB port on the Tello drones to power a Raspberry Pi Zero W in flight as in [35], a firmware update released following the Tello Talent meant that powering a device from the USB port on either the Tello Talent or Tello EDU would result in the drone not being able to connect to Wi-Fi [43]. This is an incredibly damaging concept for this implementation as Wi-Fi is the intended communications method between the managing Raspberry Pi Zero W and its follower, and, as such, an alternative power solution must be developed in the future to provide the designed mounted control system. Without this solution, the current managing devices must remain unmounted despite the 3D printed mounts being available for usage, as seen in Fig. 11, which shows an early and unstable version, and Fig. 12, which shows the latest low-profile design.

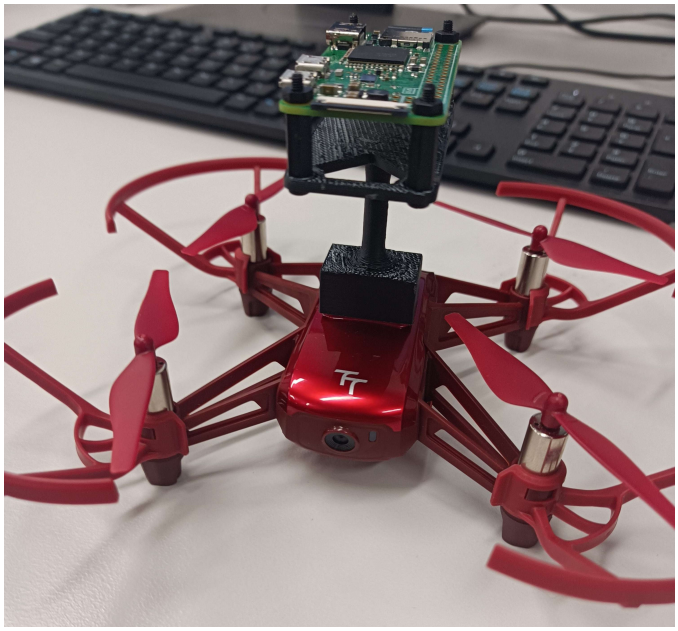

Fig. 11: Tello Talent with V1 Raspberry Pi Mount

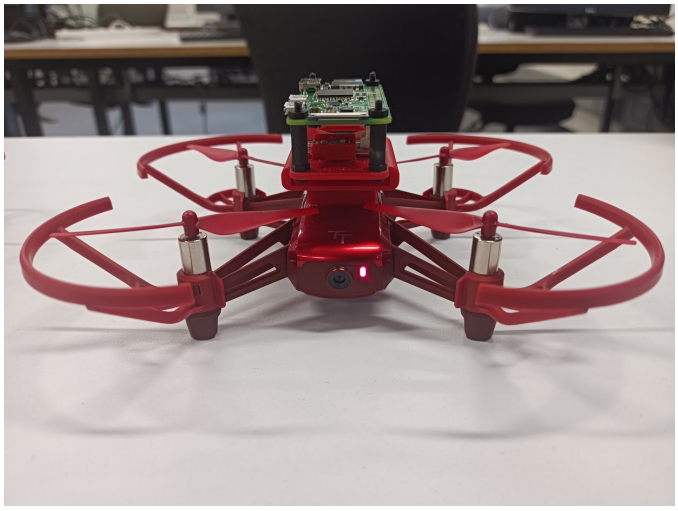

Fig. 12: Tello Talent with Latest Raspberry Pi Mount

V. Evaluation

A. Results

The suitability of the implemented solution and system was based on the outlined project requirements. These seven concepts drove the design and development of the system and, as such, are the best determination of how well the solution performs.

Project requirement 1 has the system meet expectations as the developed solution requires the operator to communicate only with the currently selected leader drone through the ground controller software interface outside of beginning the flight management software. Either node can take on this leadership responsibility, and commands are successfully replicated across to the follower device, reaffirming this success.

Project requirement 2 is partially successful as the system holds the capacity for scaling with new devices; however, the current issues in Wi-Fi adaptors and in-flight power systems mean that this system still has much room for improvement to permit the desired capability.

Project requirement 3 is successfully met through the utilisation of the open-source and capable Raspbian operating system on the device, alongside the usage of Python for the flight management software, which permits easy development for specialised usages.

Project requirement 5 is successfully met through providing a system with a current core component cost of 453 NZD [8], [9] compared to the higher 899 USD Raspberry Pi drone kit [28] and the much more expensive Teal devices.

Project requirement 7 is successfully met through the selection of the Tello Talent drones for the system. Their inbuild collision avoidance and automated landing when no management connection is detected or 15 seconds have passed without commands in SDK mode mean communication losses are safe failures even for the indirectly managed UAV.

Project requirements 4 and 6 had further flight testing to determine their success. This evaluation was undertaken by comparing the round-trip command time of a base one-to-one control scheme system to the mesh system's leader and follower device response times. The results, found in Fig. 13, show that the one-to-one control scheme had a lower response time in all of the ground, hover, and flight movement command situations; however, the performance difference is in an acceptable range. The leader device only saw a minor average response time increase of 6 and 6.6 milliseconds across the most important hover and flight movement situations where the drone's response time is vital for maintained safety. The follower drone did see a much higher jump of 24.95 and 26.65 milliseconds in round-trip response times across these tests. These results saw an average increase in response times of 21.83% and 89.42% for the two drones, which does have room for improvement with a drive for higher code efficiency or language usage; however, they fall well within an acceptable operating window of 100% increase for live flight management. This figure was selected due to the incredibly quick base response times that the system achieved in round-trip communications, and falling beneath it reassures that flight response times are acceptable in the developed solution. The command replication accuracy raises questions, however, due to the underlying nature of the communications structure the devices use. UDP, which the Tello 3.0 SDK requires for drone communications [6], is a "best effort" system that broadcasts without considering or requiring confirmation of message acceptance. As such, any messages that are dropped or take significantly longer to arrive in flight will not be detected natively within the system. This could cause issues if commands are not continually arriving, causing desynchronisation of the mesh node's drones, allowing them to exist in different direction states, and causing potential crash opportunities or control issues if the operator does not identify the problem. This potential safety issue brought about by the system does indicate that requirements 4 and 6 are partially fulfilled as the swift execution of commands and low overhead are achieved, but system safety is not guaranteed in regard to command replication accuracy.

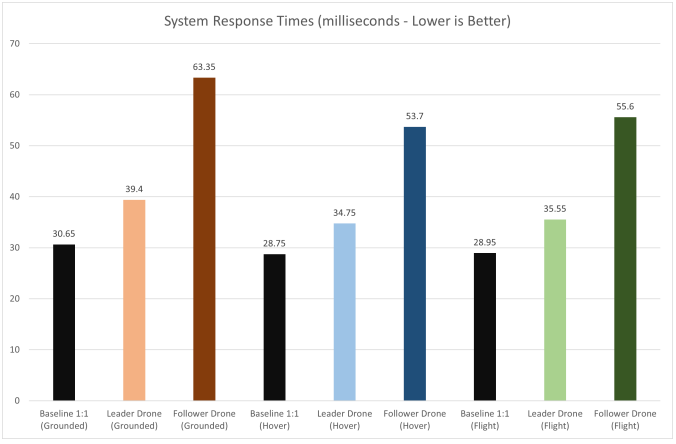

Fig. 13: Round Trip Command Response Times

B. Implementation Limitations

The produced artefact is a limited implementation with improvements available in multiple areas. The first opportunity for improvement would be to offset some of the potential problems UDP communications has on the system by converting the ground controller - flight control and flight control - flight control communication systems to use the much more sturdy Transmission Control Protocol (TCP) system for communicating commands and responses. This system requires acknowledgments of events occurring and so would allow the control systems to detect and rebroadcast possibly missed commands automatically through flight to allow for greater replication accuracy across the mesh. Although the flight control - drone communications are stuck with UDP in this situation, this added security in the rest of the system would be hugely beneficial to providing certainty

of command execution and identifying quickly if a device is no longer available. A potential improvement related to this factor would be the integration of flight health monitoring and flight safety tool software that could be added to the system. Having each aircraft hold a dedicated Raspberry Pi expands the onboard processing capabilities of this system immensely, which could the development of more complex tasks each device could be running to monitor the current system status. This could be used to utilise the connection strength, command delay, relative drone location from launch, or battery life of the commanded drone to begin landing routines or send commands back to the ground controller, for example, as the consistent connection and updates from an individually controlled drone open many opportunities for more complex analysis of its status.

The current hardware implementation problems are another massively limiting factor in the developed artefact. Although the system does work in providing mesh management of a leader and follower drone, the lack of an in-flight power solution and reliable new Wi-Fi adaptors being ready for integration means that the true potential of the system can not yet be realised. Solving these issues would allow for tests to be undertaken at greater distances and with larger swarm sizes to provide much richer evaluation of system performance metrics and would provide development goals for a more effective implementation accordingly.

A final limitation of the system that could be improved is the overhead of the developed solution. Although the system evaluation found the increased response times to be acceptable for a system of this scale, increased distances and swarm sizes would massively increase the workload and maximum node hops commands have to take. As such, redeveloping the flight management control system for efficiency or changing its language to something with faster performance would allow for a more scale-friendly solution compared to the currently produced project. Although C++ was written out of the design stage for extensibility purposes, swapping back to the language could provide the higher performance the system would require at a larger scale at the expense of operator usability.

VI. Conclusions and Future Work

A. Project Conclusions

Through this project, the desired goal of an accessible, extensible, and adaptable multi-UAV communications structure has been successfully formed. The mesh networked solution has enabled the replication of commands across devices with lower operator demand due to being able to indirectly manage follower drones whilst maintaining a swift end-to-end response time across a two-node system, allowing for precise and controlled live inputs, with per node costs of less than half of comparable Raspberry Pi drone kits. This successful implementation of such a system has shown that drone swarms can be an accessible concept for individual enthusiasts and small corporations alike with modern and affordable hardware that can still provide additional opportunities for innovation and development for specific needs operators may have.

B. Future Work opportunities

Following this system implementation, future work opportunities include adding a security protocol and verification system, potential flight monitoring software development, and redevelopment with future software and hardware updates.

The current implementation has no formal security checking outside passwords on the Raspberry Pi-hosted Wi-Fi networks. This was because the project focus did not place any focus on enforcing secure communications, and it means that any device that was to join the B.A.T.M.A.N. advanced mesh network, which is open, could begin to send malicious commands into the system. Developing a standard for source identification and confirmation would be a task that would massively improve the safety of this style of system.

Developing adaptive flight monitoring software based on drone update messages and external communication levels also offers an avenue for future work that could utilise this system base as a testing and data collection location. Software that would be able to smartly adapt and warn operators of potential upcoming problems in either their node or mesh region would offer valuable insight and expand the reliability and use cases of the system immensely.

Finally, as the system has been so restricted by hardware behaviour and availability, taking this mesh network concept and redeveloping it for alternate or future hardware iterations would be a massively beneficial task for the system. Updating it with the lessons in this developed solution would provide massive improvement avenues for individual and commercial users at a much quicker timescale.

VII. Acknowledgements

This project would not have been possible without the constant support, advice, and feedback from my supervisors, Dr Jyoti Sahni, Professor Winston Seah, and Dr Alvin Valera - thank you. Additional thanks to the Victoria University of Wellington Wireless Networking (WiNe) research group, whose members provided additional knowledge and valuable insight through this project and Jason Edwards who produced the mounting solution used in the project. Finally, thank you to all my remarkable peers whose friendship and camaraderie aided in every step of this process.

References

[1] R. D. Arnold, H. Yamaguchi, and T. Tanaka, "Search and rescue with autonomous flying robots through behavior-based Cooperative Intelligence," Journal of International Humanitarian Action, vol. 3, no. 1, 2018. [Online]. Available: https://doi.org/10.1186/s41018-018-0045-4 [Accessed: May 16, 2023]

[2] B. D. O. Anderson, B. Fidan, C. Yu, and D. Walle, "UAV Formation Control: Theory and Application," Recent Advances in Learning and Control, pp. 15–33, 2008. [Online]. Available: https://doi.org/10.1007/978-1-84800-155-8_2 [Accessed: May 16, 2023]

[3] Intel, "Queen of drones: Revolutionizing night sky light shows," [Online]. Available: https://www.intel.com/content/www/us/en/technology-innovation/article/queen-of-drones-revolutionizing-night-sky-light-shows.html [Accessed May 25, 2023]

[4] Sally French, "How much does a drone light show cost?"[Online]. Available: https://www.thedronegirl.com/2020/08/17/drone-light-show-costs/ [Accessed: October 5, 2023]

[5] United Nations Department of Economic And Social Affiars, "The 17 Goals" [Online] Available: https://sdgs.un.org/goals [Accessed June 3, 2023]

[6] DJI, "Tello Robomaster TT SDK 3.0 User Guide", [Online]. Available: https://dl.djicdn.com/downloads/RoboMaster+TT/Tello_SDK_3.0_User_Guide_en.pdf [Accessed: May 20, 2023]

[7] Open Mesh, "B.A.T.M.A.N. Advanced Documentation Overview", [Online]. Available: https://www.open-mesh.org/projects/batman-adv/wiki [Accessed: October 10, 2023]

[8] PB Tech, "Raspberry Pi Zero W", [Online]. Available: https://www.pbtech.co.nz/product/SEVRBP0412/Raspberry-Pi-Zero-W-1GHz-BCM2835-Single-Core-CPU-5 [Accessed: May 23, 2023]

[9] DJI Store, "Tello Talent" [Online]. Available: https://store.dji.com/nz/product/robomaster-tt [Accessed: October 7, 2023]

[10] DJI, "ROBOMASTER TT TELLO TALENT User Manual v1.0 2021.03" [Online]. Available: https://dl.djicdn.com/downloads/RoboMaster+TT/RoboMaster_TT_Tello_Talent_User_Manual_en.pdf [Accessed October 7. 2023]

[11] Lokesh Varman, "C++ vs. Python: A Performance Comparison using an Example". [Online]. Available: https://www.c-sharpcorner.com/article/cpp-vs-python-a-performance-comparison-using-an-example/ [Accessed October 10, 2023]

[12] Anthony Behery, "Python VS C++ Time Complexity Analysis". [Online]. Available: hhttps://www.freecodecamp.org/news/python-vs-c-plus-plus-time-complexity-analysis/ [Accessed October 10, 2023]

[13] Raspberry Pi, "Raspberry Pi Zero W", [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-zero-w/ [Accessed October 3. 2023]

[14] React.dev, "React - The library for web and native user interfaces", Available: https://react.dev/ [Accessed October 3, 2023]

[15] Expressjs.com, "Fast, unopinionated, minimalist web framework for Node.js", Available: https://expressjs.com/ [Accessed October 3, 2023]

[16] Nodejs.org, "Node.js v21.0.0 documentation - UDP/datagram sockets", Available: https://nodejs.org/api/dgram.html [Accessed October 3, 2023]

[17] Python.org, "socket — Low-level networking interface", Available: https://docs.python.org/3/library/socket.html [Accessed October 3, 2023]

[18] Raspbian.org, "Welcome to Raspbian", Available: https://www.raspbian.org/ [Accessed October 12, 2023]

[19] wiki.debian.org, "dnsmasq", Available: https://wiki.debian.org/dnsmasq [Accessed October 17, 2023]

[20] Jouni Malinen, "hostapd: IEEE 802.11 AP, IEEE 802.1X/WPA/WPA2/EAP/RADIUS Authenticator", Available: https://w1.fi/hostapd/ [Accessed October 17, 2023]

[21] P. G. Fahlstrom and T. J. Gleason, Introduction to UAV Systems, 4th ed. Hoboken, N.J.: John Wiley & Sons, 2012

[22] DJI, "Agras T40", [Online]. Available: https://www.dji.com/nz/t40 [Accessed May 25, 2023].

[23] Teal Drones, "Aircraft," [Online]. Available: https://tealdrones.com/solutions/aircraft/ [Accessed May 27, 2023]

[24] B. Crumley, "Teal to roll out four-drone, single-operator mini-swarm platform," DroneDJ, [Online]. Available: https://dronedj.com/2022/06/03/teal-to-roll-out-four-drone-single-operator-mini-swarm-platform/ [Accessed May 27, 2023]

[25] B. Crumley, "Teal's 4-ship multiple drone controller gets Tomahawk Robotics enhancements," DroneDJ, [Online]. Available: https://dronedj.com/2022/10/28/teal-4-ship-drone/ [Accessed May 27, 2023]

[26] DJI Store, "Tello EDU" [Online]. Available: https://store.dji.com/nz/product/tello-edu [Accessed: October 15, 2023]

[27] Sally French, "How to build a Raspberry Pi drone", September 15, 2023, [Online]. Available: https://www.thedronegirl.com/2021/04/05/how-to-build-a-raspberry-pi-drone/ [Accessed: October 17, 2023]

[28] Sally French, "Beloved Raspberry Pi drone training site launches ready-made Raspberry Pi Drone Kit", June 20, 2022, [Online]. Available: https://www.thedronegirl.com/2022/06/21/raspberry-pi-drone-kit/ [Accessed: October 17, 2023]

[29] Raspbian.org, "Raspbian Repository". Available: https://www.raspbian.org/RaspbianRepository [Accessed: October 20, 2023]

[30] Raspberry Pi, "Raspberry Pi Zero 2 W", [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-zero-2-w/ [Accessed October 3. 2023]

[31] Open-Mesh, "Broadcasts in B.A.T.M.A.N. Advanced" [Online]. Available: https://www.open-mesh.org/projects/batman-adv/wiki/Broadcast [Accessed: June 4, 2023]

[32] Python.org, "Garbage collector design", [Online]. Available: https://devguide.python.org/internals/garbage-collector/ [Accessed: October 20, 2023]

[33] Rizel Scarlett, "Why Python keeps growing, explained", March 2, 2023, [Online]. Available: https://github.blog/2023-03-02-why-python-keeps-growing-explained/ [Accessed: October 20, 2023]

[34] Python.org, "Garbage Collector interface", [Online]. Available: https://docs.python.org/3/library/gc.html [Accessed: October 20, 2023]

[35] martinpi, "Onboard Raspberry Pi", March 31, 2020, [Online]. Available: https://tellopilots.com/threads/onboard-raspberry-pi.5117/ [Accessed: October 21, 2023]

[36] zeroTM, "Raspberry Pi Zero Tello mount" December 05, 2019, [Online]. Available: https://www.thingiverse.com/thing:4022999 [Accessed: October 21, 2023]

[37] B. Innes, J. Walicki, "Create a Mesh Network over WiFi using Raspberry Pi," GitHub, [Online]. Available: https://github.com/binnes/WiFiMeshRaspberryPi [Accessed: April 20, 2023]

[38] BootesNull, "Flutter vs ReactJS: Which one to choose in 2023?", July 21, 2023, [Oline]. Available: https://medium.com/front-end-weekly/flutter-vs-reactjs-35a8abdf4ba3 [Accessed: October 23, 2023]

[39] techinfodepot, "TP-LINK TL-WN821N v4", [Online]. Available: http://en.techinfodepot.shoutwiki.com/wiki/TP-LINK_TL-WN821N_v4 [Accessed: October 20, 2023]

[40] techinfodepot, "TP-LINK TL-WN725N v2", [Online]. Available: http://en.techinfodepot.shoutwiki.com/wiki/TP-LINK_TL-WN725N_v2 [Accessed: October 20, 2023]

[41] techinfodepot, "TP-LINK TL-WN821N v6", [Online]. Available: http://en.techinfodepot.shoutwiki.com/wiki/TP-LINK_TL-WN821N_v6 [Accessed: October 20, 2023]

[42] Larry Finger, "Repository for stand-alone RTL8188EU driver", GitHub, [Online]. Available: https://github.com/lwfinger/rtl8188eu [Accessed: October 20, 2023]

[43] Hacky, "Pack a homebrew action-cam on top of Tello", Tello Pilots, [Online]. Available: https://tellopilots.com/threads/pack-a-homebrew-action-cam-on-top-of-tello.6188/#post-39724 [Accessed: October 20, 2023]