ABSTRACT

Title: PLATFORM DEVELOPMENT FOR THE IMPLEMENTATION AND TESTING OF NEW SWARMING STRATEGIES

Sara Celidonio, Nathaniel Hoffman, Helen Kent, Paul Motter, Matthew Patsy, Kevin Postlethwait, Henry Tuit Farquhar

Directed By: Dr. Anil Deane

Institute for Physical Science and Technology

Swarm robotics--the use of multiple autonomous robots in coordination to accomplish a task--is useful for mapping, light package transport, and search and rescue operations, among other applications. Researchers and industry professionals have developed robotic swarm mechanisms to accomplish these tasks. Some of those mechanisms or "strategies" have been tested on hardware; however, the technical requirements involved in fielding a drone swarm can be prohibitive to physical testing. Team SWARM-AI has developed a platform that provides a starting point for testing new swarming strategies. This platform allows the user to select vehicles of their choosing- either air, land, or water based, or some combination thereof- as well as define their own swarming method. Using a novel decentralized approach to ground control software, this platform provides a user interface and a system of computational "units" to coordinate drone swarms with a centralized, decentralized, or combination architecture. Additionally, the platform propagates user input from the master unit to the rest of the swarm and allows each unit to request sensor data from other units. The user is free to edit the processes by which each drone interacts with the environment and the rest of the swarm, giving them freedom to test their swarming strategy. The software system is then tested with a swarm of quadcopters using Software in the Loop (SITL) testing.

PLATFORM DEVELOPMENT FOR THE IMPLEMENTATION AND TESTING OF NEW SWARMING STRATEGIES

By

Team SWARM-AI

Sara Celidonio, Nathaniel Hoffman, Helen Kent, Paul Motter, Matthew Patsy, Kevin Postlethwait, James Tuit Farquhar

Thesis submitted in partial fulfillment of the requirements of the Gemstone Honors Program, University of Maryland, 2020

Advisory Committee:

Dr. Anil Deane, University of Maryland, Mentor

Dr. Yancy Diaz-Mercado, Discussant

Dr. Evan Golub, Discussant

Dr. Michael Otte, Discussant

Dr. Huan Xu, Discussant

Team SWARM-AI

Sara Celidonio, Nathaniel Hoffman, Helen Kent, Paul Motter, Matthew Patsy, Kevin Postlethwait, James Tuit Farquhar

Acknowledgements

Team SWARM-AI would like to thank our mentor Dr. Anil Deane for everything he has done to support our research and serve as a mentor to all of us for these past three years. We would also like to thank Dr. Kristan Skendall for her work in keeping the Gemstone program running and helping out our team with logistical support and advice for our research. We would also like to recognize our librarian, Celina N. McDonald, for aiding and supporting us in the initial stages of writing and research in our project. In addition, gratitude is extended to the former Gemstone Director Dr. Frank Coale and all of the Gemstone staff for making our research possible and supporting the Gemstone honors program. Finally, we would like to thank our discussants for contributing their time, and providing their feedback for this project.

Table of Contents

List of Figures

I. Introduction

With an increasing use of autonomous vehicles in society, and a desire to be able to accomplish tasks as quickly as possible, the ability to easily create a swarm is more prudent than ever. A swarm consists of multiple units that interact in a cooperative manner to complete a task. In the context of this research an autonomous vehicle refers to one primarily controlled by an autopilot to complete its mission. These autopilots control the movement of these vehicles by following a defined path of waypoints. Autopilots interact with a ground control station, which allows the user to input waypoints to set a "mission", or defined path that they want the vehicle to follow. However, ground control stations currently lack the ability to easily and efficiently allow users to control multiple autonomous vehicles at once and to define their behavior for experimental swarming strategies.

Team SWARM-AI has developed a software platform that will allow users to test novel swarming strategies for autonomous vehicles. In an effort to appeal to all potential users, the platform developed is usable for most autonomous vehicles, although during its development an emphasis was placed on multirotor drones.

The proposed system takes the place of a more traditional ground control system with one that is purpose-built for drone swarms and trades user provisions for more flexibility. This is accomplished by decentralizing the responsibilities of a ground control station into several different processes that can be run on hardware throughout the swarm- ground-based computers or the drones themselves. This allows each piece to

react individually to the environment and control the drones accordingly. Users are then able to define the behavior of each piece individually, which allows for the implementation of novel drone swarming strategies.

II. Literature Review

A. Introduction

Swarm robotics has many useful applications, such as package delivery, underground mapping, aerial displays, and search and rescue, among others. Researchers have developed various swarming methods to accomplish these tasks, but testing these swarming strategies is sometimes difficult due to technical requirements involved in operating a drone swarm. The user may not have the physical space or proper hardware to test a swarm. Different types of drones, such as aerial (UAV), aquatic (UUV) or land drones (UGV), each have different requirements and physical limitations for testing. Drones use a variety of sensors for both data collection and navigation that are essential in implementing a swarming pattern, and whose data needs to be communicated between drones in a swarm as well as to the user. This is so that crashes can be prevented between drones and their flight paths can be observed by the user. Team SWARM-AI plans to create a software that can test swarming strategies independent of physical hardware, but can also be used to control real world-drones' flight paths and monitor telemetry and sensor data.

B. Potential Applications

A platform capable of implementing different swarming methods and architectures could have many different applications, the set of which spans public safety, commercial uses, and beyond. Two such potential uses are listed here.

1. Package Delivery

An emerging market for drone swarm technology is that of package delivery services. Companies such as Amazon are already using drones to quickly deliver packages to customers [1]. Aside from online purchases, delivery drones are also being utilized for healthcare packages. Drones can transport medicines and vaccines as well as retrieve medical samples into and out of regions that are remote or would otherwise be inaccessible [2]. Since aerial package delivery would involve potentially thousands of daily flights in the same geographic area, a software that could control the flight plans of these drones would be useful to prevent conflicts between drones navigating similar or intersecting routes [3].

2. Underground Exploration

A drone swarm could also have the potential to aid in underground mapping. Inaccurate maps of underground mines can pose a great risk to workers' safety. There have been instances in mines where accidents could have been avoided if the workers had a more accurate map of where they were working [4]. A drone swarm could build off of existing technology for mapping underground mines and could complete this task completely autonomously, removing any risk to human safety. The maps could then be

made more accurate by having multiple drones map the same regions allowing for error correction and greater map accuracy. Additionally since a swarm uses multiple drones, the maps could also be made more efficiently as each drone would only be required to survey a small section of the underground area, thus saving time for the mining company.

C. Background on Drones and Drone Swarms

1. Drone types

A robot can be classified as a drone if it does not have a human operator, operates remotely or autonomously, or carries a payload. An aerial drone is referred to as a UAV, or unmanned aerial vehicle. The three most common types of UAVs can be considered fixed-wing aircraft, rotary-wing aircraft, and bio-inspired designs based on flapping wings [5]. All three classifications present individual benefits and limitations. Fixed-wing aircraft are able to fly efficiently, but cannot typically hover. Rotary-wing aircraft can hover and are quite maneuverable, but are less efficient in forward flight. They are ideal for surveying work [5] and are among the most common drones (70% of civilian manufactured drones in 2014) [6]. Bio-inspired designs can be more easily scaled down in size and may be useful in swarm technology but bring fluid-mechanics-modelling and control challenges [5].

Besides aerial drones, there also exist unmanned ground vehicles (UGVs) and unmanned underwater vehicles (UUVs). UGVs are vehicles that operate unmanned while in contact with the ground and are used in situations where it may be inconvenient or hazardous to have a human operator present. While UGVs do not have classifications like UAVs do, their design generally includes the following components: platform, sensors,

control systems, guidance interface, communication links, and systems integration features [7]. On the other hand UUVs can be classified as either remotely operated underwater vehicles (ROUVs), which are controlled by a human operator, and autonomous underwater vehicles (AUVs), which operate indirectly through human input [8].

The platform that Team SWARM-AI is developing is drone agnostic, so the user would be able to use the software with the drone of their choosing. Drones are not the only vehicles that would be able to use this platform; any land or water based vehicles that meet the prerequisites would be capable as well.

2. Basic sensors

Drones use a variety of sensors both for data collection and navigation [9], [10]. In order to fulfill these two functions, any type of software for drones will require a method in which sensor data can be passed to both the individual drone units and the base computer. In order to try and fulfill this function, there must be an understanding of the most common drone sensors and the type of data that they collect. Some of the most common sensors seen on drones for navigation are GPS devices and inertial measurement units [11]. Inertial measurement units (IMU's) work by combining accelerometers and gyroscopes to collect data on the drones positioning and acceleration. GPS units provide information on the drone's coordinates. Other sensors commonly found on drones include camera systems, LIDAR, thermal sensors, and ultrasonic sensors [12]. LIDAR, thermal, and ultrasonic sensors all provide the drone information on its immediate surroundings. This information can then be used for either navigation/obstacle avoidance or data

collection, but overall only a small amount of data needs to be transmitted for these three sensors. Cameras on the other hand, while they can be used for similar functions, require a lot more data to be transmitted if the images are going to be processed on another drone unit [13]. This can be handled in three ways: either the image data can be saved to the drone unit to be processed later, the image can be processed immediately by the drone unit, or the image data can be sent to the base computer for immediate processing.

3. Ground Control Stations and MAVLink Overview

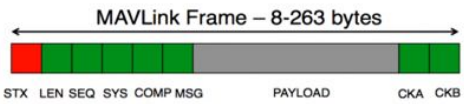
Ground control stations are software systems that are used to communicate with a user's drones through telemetry. A ground control system will have a graphical user interface (GUI) that allows the user to control the drone, both before flight and while in flight. The user can utilize the ground control station to change the mode that the drone is currently in, as well as change parameters and upload new mission commands or waypoints [14].

The four most commonly used ground control stations are Mission Planner, APM Planner 2.0, MAVProxy, and QGroundControl. All four of these software platforms share the same basic functionality, while also containing unique features for their users. Mission Planner is one of the most commonly used ground control systems, but is not compatible with the Linux operating system. APM Planner 2.0 does not offer all of Mission Planner's features, but it is runnable on Linux. MAVProxy is unique since it utilizes a command line interface unlike the other ground control stations. QGroundControl is the only ground control station out of these four that can be run on both a desktop and mobile device [14].

All four of these ground control stations utilize MAVLink as a communication system. MAVLink is used to communicate with drones running Ardupilot or PX4 autopilots. These drones include fixed wing, rotary wing, rovers, and submarines. MAVLink, or Micro Air Vehicle Link, is a protocol that sends messages between the ground station and drones. MAVLink messages can be sent over serial or IP, therefore drones using MAVLink often use serial radios or wifi for communication. MAVLink is frequently used because it can be sent over a number of serial connections, making it a versatile communication platform. Companion computers, i.e. computers onboard the drone, as well as the ground station must regularly check for a MAVLink message because they are not guaranteed to be delivered.

In order to determine where the message came from, the sender fills out a "System ID" and a "Component ID". There is a unique "System ID" for each drone and ground station in the environment. If there is another device on a drone that can receive MAVLink messages, it shares the same "System ID" as the companion computer, while having a unique "Component ID".

A MAVLink message can contain up to 263 bytes. The first 6 bytes are used to set up what the message is and allows the receiver to know where the message is coming from. The majority of the rest of the message is composed of the data that is being transmitted. The last two bytes are made up of a checksum [15]. A summary of message composition can be seen in figure 1 [15].



Byte Index	Content	Value	Explanation
0	Packet start sign	v1.0: 0xFE (v0.9: 0x55)	Indicates the start of a new packet.
1	Payload length	0 - 255	Indicates length of the following payload.
2	Packet sequence	0 - 255	Each component counts up his send sequence. Allows to detect packet loss
3	System ID	1 - 255	ID of the SENDING system. Allows to differentiate different MAVs on the same network.
4	Component ID	0 - 255	ID of the SENDING component. Allows to differentiate different components of the same system, e.g. the IMU and the autopilot.
5	Message ID	0 - 255	ID of the message - the id defines what the payload "means" and how it should be correctly decoded.
6 to (n+6)	Data	(0 - 255) bytes	Data of the message, depends on the message id.
(n+7) to (n+8)	Checksum (low byte, high byte)	ITU X.25/SAE AS-4 hash, **excluding packet start sign, so bytes 1..(n+6)** Note: The checksum also includes MAVLINK_CRC_EXTRA (Number computed from message fields. Protects the packet from decoding a different version of the same packet but with different variables).	

Figure 1: MAVLink message composition

4. Autopilots

For a drone to operate autonomously it will need an autopilot. There are many different kinds of autopilots available in the market, each with its own capabilities. In the case of aerial drones, some autopilots only function to keep the vehicle stable in the air. However the most common autopilots tend to have more functionality. Autopilots are almost always connected to RC receivers which allow for drones to be remotely piloted. Autopilots that are used for autonomous navigation take in data from the drones sensors such as the GPS, gyroscope, etc., and use that information to determine where the drone is located, and what adjustments are needed to get the drone to the next waypoint. The autopilot is then responsible for making the subsequent adjustments to the drone's motor

controls. Ardupilot is a good example of one such autopilot. Ardupilot is a free open source autopilot which supports a wide variety of hardware [16]. An Ardupilot flight controller can perform the basic tasks of flying an autonomous drone such as flying level, taking off, landing, and navigating to waypoints [16]. Ardupilot flight controllers can be connected to a companion computer [16]. A companion computer can read sensor data from the flight controller and send new waypoints to the flight controller for the drone to fly to [17]. The use of a companion computer enables more complex behaviors than would be possible using only a flight controller.

5. Typical Swarming Methods

Swarms of multiple agents functioning as an entity to accomplish a specific task is not unique to robotics. Many animals--most often insects--function in swarm colonies. The properties of these swarms, such as centralized or decentralized communication and use of pheromones, have been used in multiple robotic swarms. Before investigating these bio-inspired swarms, several terms must be defined. A taxonomy proposed by Dudek et al. categorizes swarms based on their organization (centralized vs. decentralized), size (number of units), composition (homogeneous vs. heterogeneous), and communication type (broadcast vs. unicast) [18], [19], [20]. Gerkey and Matarić [19], [21] categorize the tasks that robot swarms can attempt to accomplish. This outlines multi robot task allocation (MRTA) in terms of single task (ST) robots versus multi task (MT) robots, single-robot tasks (SR) versus multi-robot tasks (MR), and instantaneous assignment (IA) versus time-extended assignment (TA) [18].

One type of bio-inspired swarm algorithm is based on bees. This algorithm, called the optimized Distributed Bees Algorithm (DBA), is a distributed swarm that aims to fulfill single robot and multi robot tasks with instantaneous assignment [20]. To test this swarm, a two-dimensional arena is used that contains a predefined number of targets with differing importance and a preset number of robots. Each robot can only be assigned to one target, but each target can have multiple robots assigned to it. The efficiency of the algorithm is measured by the amount of distance traveled by robots when they locate targets. In practice, the DBA is tested by having all robots start at a randomized starting location. They will search for a target, and when a robot finds a target it will transmit the quality of the given target to other robots, as well as its distance and orientation. Other robots use this to determine if they are needed at a target, or should keep searching. Two parameters, the distance between targets and quality of targets, were investigated for their impact on total distance traveled by the algorithm. Using this algorithm, a team was able to get viable results, but at the cost of higher distribution error. However, the team determined that the slight improvement per robot is significant for the robot swarm as a whole, especially when resources are limited [20].

Other distributed swarm algorithms are based on trophallaxis (exchange of regurgitated fluids) of insects, which is the exchange of fluid with direct mouth-to-mouth contact. While not using direct contact, this method has robots communicate by exchanging information with their closest neighbor. To test the feasibility of this algorithm, one research team, [22] used the Large Robot swarm Simulator (LaRoSim) with 300 robots. The goal of the swarm was to move a pile of "dirt" to a specified "dump"

spot [22]. The testing was altered multiple times by adding obstacles in the simulation arena. The actual communication is similar to how bees distribute nectar amongst themselves. Bees with more nectar will give some of their nectar to bees with less nectar. Similarly, the robots will distribute the work amongst themselves with unburdened robots taking some responsibility from burdened robots. This strategy allows for a robust self-organizing swarm that can explore an area efficiently and quickly aggregate to an area of importance. Also, it allows for lower level sensors to be used, because the algorithm efficiency compensates [22].

Another bio-inspired strategy uses a digital pheromone based communication. A heterogeneous swarm consists of two air drones and four ground robots controlled by a stigmergic algorithm. Stigmergic is a biological term used to describe information exchange through a shared environment, like pheromones. This method is used for its simplicity, robustness, and scalability. The digital pheromones used act like biological pheromones by depositing information (information fusion and aggregation), evaporating over time (truth maintenance), and propagating over an area (information diffusion and dissemination). To ensure coverage of an area, “attractive” pheromones are placed in an environment. This “scent” is either stored in a central computer and relayed to individual units or stored in some sort of emitter device placed in the environment. When a drone from the swarm searches a region or subset of that environment, it places a “repellent” pheromone to deter other drones from repeating the initial search. Using this method, the team was able to successfully provide surveillance over a target area. The demonstration showed the possibility of scalability using a fairly simple pheromone-based algorithm

and the possibility to adapt this approach to a survey application. The team suggested that further improvements could be made by adding new pheromones, combining current pheromones, and fine tuning the algorithm [23].

A more systematic approach to searching an area is called a Virtual Mesh Network. The basic idea behind this network is that all the robots in a small area create a triangle lattice (mesh) by treating each robot as a node and the edges between them as virtual springs with similar properties of real springs. By creating virtual spring forces, the robots understand when they are getting too close or too far from their neighbors and are pushed away or pulled towards the neighbor depending on their relative location. After many iterations, the robots will eventually reach an equilibrium in which their distances from one another are relatively equal and similar to that of the natural spring length [23], [24]. One paper found three main problems with the Virtual Mesh Network method- exploration of unknown areas, complete coverage of unknown areas within a certain range, and obstacle avoidance—which they aimed to remedy through their research. To solve the first and second problem, each edge robot was given a force in addition to that of the virtual spring forces acting upon them, called the force of exploration. This compelled the robot to move towards the edge of the predefined boundary. The springs have dynamic properties so that they can adapt to the size of the area to keep the system at equilibrium once the robots have covered the predefined space. Lengthening or shortening the natural spring length lowers or raises the density of robots per meter. On the third problem, the researchers found that by treating obstacles as nodes in the system they are able to force the robots around the object in question without

breaking the lattice [24]. As most environments have a variety of different obstacles it would be beneficial to have a swarm that can adapt to nearly any challenge it finds in its path. After various computer simulations, they found that their algorithm was very efficient at covering predefined areas and working around obstacles. An important note is that more robots did not always correspond to better coverage, but rather that each area has an optimal number of robots based on the situation at hand. Another important finding is that adding a moderate amount of wireless interference caused the robot swarm to collapse as the individual robots thought they were further from their neighbors than they were. This caused effective coverage to drop to only around 25% as opposed to the 90%-100% being achieved without such interference [24].

In prior research, additional functionality has been added to QGroundControl for the automated creation of missions for multiple UAVs. This allowed the user to define tasks that they wished to be completed by the drones, as well as to define the vehicles that would be used to complete said tasks. The Multi-UAV Cooperative Mission Planning Problem (MCMPP) is used to assign each task and order of each task to the vehicles. The MCMPP takes into account properties such as time constraints, fuel constraints, and sensor constraints when determining tasks and task orders for each vehicle [25]. Due to the MCMPP being a Multi-Objective Optimization Problem (MOP), there are numerous variables to optimize. Due to these variables- including but not limited to flight time, flight risk, number of vehicles, and distance travelled- there are often a number of solutions. This is where preferences of the operator are extremely important.

D. Drone Swarm Control and Communication

1. Available Software Platforms

In order to distribute sensor data from the GUI to the drone swarm, a robotic middleware must be used. Robotic middleware is a collection of software frameworks used for robot software development, such as in drone operation. The software used needs to be able to communicate main points and regions as well as points and regions from individual drones, along with MAVLink commands.

One such software is the Robot Operating System (ROS), which is a flexible framework for writing robotic software. It is a combination of free and open-source tools, libraries and conventions that can be used to create robot behavior [26], [27]. A benefit of ROS is that it is modular as well as distributed. This means that, with over 3,000 user-contributed packages in the ROS ecosystem, users have a wide range of choices and can pick and choose the packages that are useful to them.

The main alternative to ROS is Dronekit, which is a service that provides a software development kit (SDK) and developer application programming interface (API). While ROS can be used for various types of robotic programming, Dronekit is designed specifically to make applications for drones.

Currently there are very few ground control stations that allow for the opportunity to control multiple vehicles simultaneously. Those that do exist require additional hardware, or are often described as “experimental”. The Robsense SwarmLink telemetry radios enable the user to connect multiple drones to a ground control station with only a single radio on the ground control side [28]. The ground control station EasySwarm,

developed by Robsense, must be used for this to be successful. EasySwarm allows a user to create custom swarming methods and dynamic waypoint selection as well as providing real-time updates on vehicle location [28]. The Robsense SwarmLink telemetry radios are significantly more expensive than the telemetry radios typically used on drones. For five drones, the Robsense SwarmLink radios cost $1,199 and standard radios would only cost $300 [29], [30]. The extreme cost difference is an inhibiting factor in why these telemetry radios and this ground control software are not widely used.

There has been some experimental success in using Mission Planner as a ground control station to control multiple vehicles at once. This method requires a pair of telemetry radios for each vehicle being operated, as well as a flight controller running antenna tracker firmware [31]. The antenna tracker is used to consolidate the telemetry data to send back to Mission Planner. This method has been proven to work in some scenarios, but is still considered to be experimental [31]. Outside of the experimental nature of this method, the biggest downfall is it requires a telemetry radio for each vehicle on the base computer running the ground control station.

QGroundControl also allows for multiple drones to be connected at a single time. When drones connect to the ground control station the user will have the option to switch to “Multi-Vehicle” mode, where the user would then see a list of all the vehicles that are connected to the ground control station at that time [32]. However, with QGroundControl the user cannot interact with all of the vehicles as a swarm, having to instead interact with a single vehicle at a time.

2. *Distributed Systems*

A distributed system, or distributed computing as it is sometimes referred to, is a system using multiple computers on different machines that all operate concurrently and communicate and coordinate their actions in order to appear as a single computer to the end user [33]. In a distributed system a problem is divided into many smaller tasks or computations, each of which is solved by one or more of the distributed computers. There are different types of distributed systems, but for the purposes of this research the focus will be on systems where each machine is contributing to the completion of a singular task, and that appear to the end-user as one cohesive system.

Distributed systems are ideal for drone swarms due to their horizontal scalability and reliability [34]. Horizontal scalability means that additional units can be added to the system in order to increase the system's processing power or efficiency. This is in contrast to vertical scalability wherein the individual computers are updated with better hardware to complete more complex computing. This is key in the creation of a platform for drone swarms, where each user will have their own set number of drones, and therefore machines must have the ability to be easily added or removed. Distributed systems are also not as susceptible to single-point failures, meaning that if one of the machines is removed, the system as a whole will continue to run. This is helpful in case drones lose their line of communication with the master unit or undergo some kind of failure, collision, etc., that takes them offline.

However, there are many challenges that occur with distributed systems in their development and implementation. Distributed systems can be very complicated to set up,

as many protocols are required to manage the system as a whole and to design the system architecture [34]. An entire communication scheme needs to be developed such that each of the units can communicate with the master unit and share and receive information. Protocols need to be developed for inconsistencies or conflicting information sent from separate units. Many problems can occur in the initial development stages and can be difficult to troubleshoot, as it is sometimes difficult to locate the problem across multiple systems.

3. Graphical User Interfaces (GUI)

Graphical User Interfaces (GUIs) are extremely useful at providing a user with visual information from a computer software. In regards to drone flight, GUIs are often utilized to allow a user to design a flight path. The researcher Perry [35] uses a GUI that has different modes- Planning Mode and Execution Mode. This allows the user to plan a test flight, and observe predicted results based off of weather charts that are being read by the software. This gives the user the opportunity to alter their mission if the predicted results are not favorable. The Execution Mode provides similar capabilities, but the user is instead looking at real time data and is capable of altering the mission in real time [35].

Further developing a ground control station (GCS) gives the user the ability to surround themselves with more information while in flight. When dealing with swarms, the user is no longer responsible for just one vehicle, they must be able to monitor a series of vehicles. As illustrated in figure 2, the GCS that was developed has four main components: UAV selector, UAV information, Map Area, and a Tab Widget [36]. The UAV selector allows the user to choose what vehicle it wants information from. By

selecting a UAV in this area (1), all of the information for that specified vehicle is displayed in area (2). The map area (3) displays the current location of the UAVs, and any tasks they might have been assigned. The tab widget (4) contains all other widgets. These widgets would have various sensor data that the user is interested in.

Figure 2: [36] Ground control station. (1) UAV selection, (2) Vehicle information, (3) Map display, (4) Other widgets

4. Robot Operating System (ROS)

Robot Operating System (ROS), is an open-source framework that provides some of the basic tools and libraries needed to create robotics software. ROS is organized such that individually designed "Nodes" run processes in a distributed framework. These Nodes can then be grouped into "Packages" and then "Stacks." ROS also has a framework by which ROS Nodes can communicate with one another via messages. ROS can use messages in two different ways: topics and services. Topics create a publisher

that continuously sends messages which can be received by subscribers. Services create a client which sends request messages and a server which sends back response messages [12]. ROS's communication frameworks are useful for this project because one of the challenges in developing drone swarms is creating a communication network. Drones in a swarm either need to be able to communicate with each other, with the base computer, or both. The ROS system is also set up so it can handle "many-to-many" communication, which is ideal for some swarming strategies. The only downside of utilizing ROS is that it requires that Nodes communicate over IP. The goal of ROS was to create a flexible framework that made it easier for researchers to share their software. ROS is widely used in research with a large online code base that others can then build off of [34].

5. Software in the Loop (SITL) and MAVProxy

Software in the Loop (SITL) is a method of testing a program that allows the user to run a vehicle that uses Ardupilot without using any physical hardware. The vehicles that can be simulated include multi-rotor aircraft, fixed-wing aircraft, ground vehicles, and underwater vehicles, as well as some optional sensors [36]. It is possible to run an SITL environment in both Linux and Windows. The sensor data used in the simulation is based off of a flight dynamics model from a flight simulator.

The communication system for running SITL is illustrated in figure 3 [37]. The numbers in the boxes refer to the port numbers that were used in this example, and will not be the port numbers used in all cases. The communication system relies on both Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). TCP is used to communicate with the ground control system that is being used as well as MAVProxy

to communicate telemetry data. This telemetry data is then sent to the ground control station. As seen below, MAVProxy can be used as the ground control station, but it is not required. Just like when using hardware, this communication needs a serial connection. UDP is used for the simulated environment that SITL created to interact with the physics simulation.

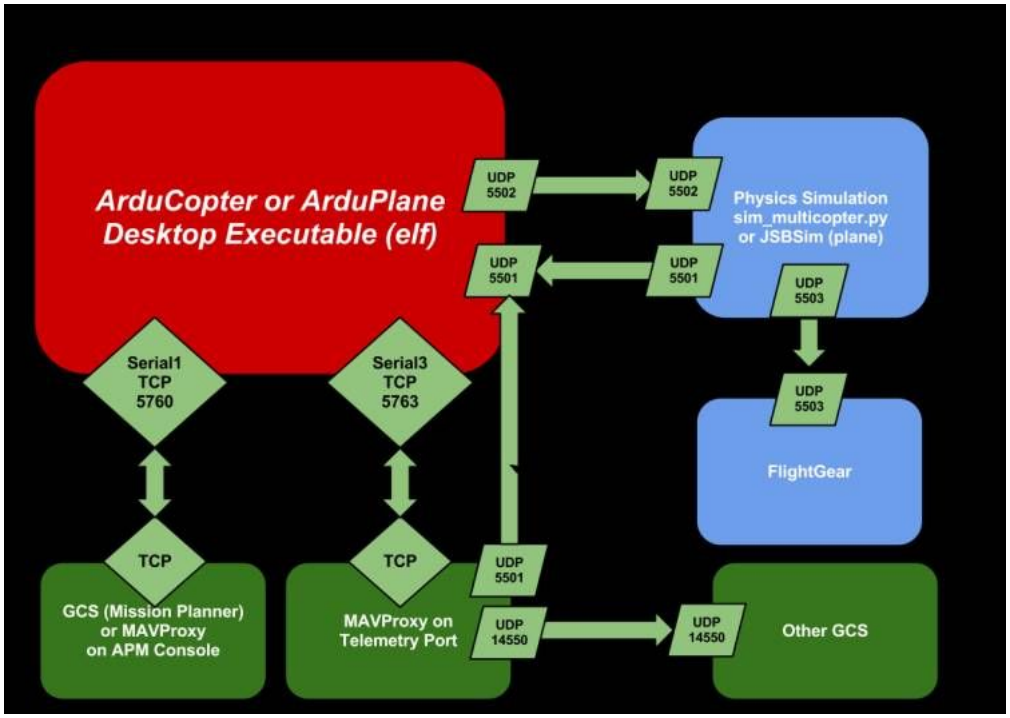


Figure 3: Example SITL communication structure

TCP is a protocol outlining how a network communication system will be created and sustained so that data can be exchanged between applications. TCP is used in scenarios where desired error detection is minimal. This error-free data transfer is possible because the client computer waits for all packets of data to be received, and then requests missing packets to be sent again before the packets are constructed together for the application receiving the data. Due to this process, it is expected for a TCP data

stream to experience latency [38]. SITL utilizes TCP for communicating with the ground control station and for sending telemetry data because it is crucial that none of this information is lost while being transmitted, and a certain amount of latency is acceptable for this application.

Unlike TCP, UDP does not have any system for error control. This leads to lower latency levels, but does allow for some data loss. A contributing factor to the lower latency levels is that the packets of data can be sent out of order due to packets being able to take more than one path between the sender and receiver [39]. The data loss stems from the multiple paths that packets can be sent through. SITL uses UDP due to the need for the physics simulation to be processed as fast as possible. The simulation would greatly suffer if the physics and the flight simulator began to lag.

E. Conclusion

After review of available literature, it can be seen that there is currently a wide variety of drones available in the market today that mirror the amount of available drone applications. Drones can each have a unique system architecture and sensor set-up to fulfill its own specialized purpose. From the literature it can also be seen that drone swarms have many available applications, and a number of swarming methods of various complexities have already been tested such as the pheromone method and the Virtual Mesh Network. However, due to the physical limitations in complexity and cost, drone swarms can sometimes be underutilized. After reviewing the literature on available methods and technologies for controlling drones and drone swarms (distributed systems,

GUI, ROS) team SWARM-AI decided to posit the question, “Could a platform be created that makes it easier for users to design and test novel swarming approaches?” The platform should set-up some of the basic functions that all swarms use such as a basic communication scheme, and data collection while maximizing user customization for their unique swarm functions.

III. Methodology

A. Platform Development

1. Formulating Software and Communication Scheme

Through the development process, the goal is to create a platform that uses a decentralized approach to the creation of drone swarming strategies. Since many different types of drones communicate with MAVLink, this platform also facilitates the deployment of heterogeneous drone swarms. With this decentralized approach, the drone need not be controlled from the ground, but can instead be controlled by software called the “drone unit” which can run on any computer on the same network as the drone. Each drone unit is allowed to have its own internal logic for how it processes waypoints and is capable of controlling the drone’s autopilot as well as communicating with other units.

In order to control a user defined number of drones N, the software creates N+1 “units.” There is one unit for each drone, and one “master unit”. These units are individual processes that communicate via a multitude of ROS nodes. Each drone unit sends commands to the drone’s autopilot via MAVLink and uses ROS to send telemetry data to the master node. The drone units could be run on a computer on the ground or on

a computer on a drone. The master unit communicates telemetry data to a GUI and sends geographic data (points and regions), collected from the GUI to each of the drone units. The GUI's main task is to present data about the swarm to the user and allow the user to input commands to the drones.

2. ROS Communication Protocol

For communication, the developed platform will use ROS topics to exchange messages between ROS nodes. As was mentioned previously, ROS nodes are essentially executables, and topics are the means by which ROS nodes exchange messages. Nodes can communicate with other nodes by either publishing or subscribing to a topic, where the publisher node sends information that the subscriber nodes then read from the topic. Since topics are designed to only send information in this one direction, the platform will use multiple topics so data can be sent from the master unit to the drone units, and telemetry data can then be sent back from the drone units. The way ROS sends messages via these nodes uses a type of queuing system, which means that should a drone fall out of range of Wi-Fi, when it comes back into range it should theoretically receive all the transmissions it missed in the proper order.

ROS is divided into two major versions, ROS 1 and ROS 2. The most relevant difference between these two versions is how nodes discover topics. In ROS 1, publishers are registered with the ROS master and subscribers must get this information from the ROS master to subscribe. In ROS 2, nodes are able to discover published topics on their own. This difference allows ROS 2 to better operate in environments with poor communication, and in applications with multiple decentralized agents. This difference

only applies to registering topics, as nodes still communicate with each other in the same way. While ROS 2 has some advantages for decentralized communications, the overall benefit provided by ROS 2 is minor as ROS 1 is also able to support multi-agent decentralized systems while also being more widely used. Potentially in future applications of the platform, if a user wanted to use a large number of drones and scalability became a significant factor, ROS 2 could then be revisited as an improved method of communication.

3. Handling of Basic Sensor Data

Since the software being developed is designed for heterogeneous swarms, it needs to be able to communicate data from a variety of common sensors. Therefore, data from common sensors such as telemetry, IMU, GPS, and cameras will be transmitted with a provided ROS topic. However, in order to accommodate for different sensors the user could potentially utilize, a ROS framework will be provided to transmit arbitrary bitstreams. This way, by providing a structure for sensor data transmission, the user can define an arbitrary framework to send data from an arbitrary sensor through the swarm.

The developed software provides an example function that utilizes a custom ROS message and a helper function to publish sensor data to a ROS topic. The custom message contains an array of arbitrary length allowing for the transfer of whatever sensor data is required. This example function may be used as the primary method of data transfer for users less familiar with ROS, however the suggested method of data transfer is for the user to write their own messages, publishers, and subscribers to provide a more robust and better tailored solution for transferring their desired data.

4. Graphical User Interface (GUI) Development

The user interacts directly with the GUI, whose main tasks are to take commands for the drones from the user and present data about the swarm. This includes but is not limited to typical drone data such as altitude and the GPS coordinates of each drone. The GUI would also display telemetry data from each individual drone unit. This data will be displayed along with superimposed graphics of each drone on a map, allowing the user to see exactly where each drone is located. This information is updated at least every half second, so the user would be able to track the position of each drone in real time. During flight the user would only be tasked with interacting with the GUI.

The user creates their swarm by determining points and regions of interest. These are selected by clicking on the map view in the GUI. These are the areas that the user wants their drones to visit. These points and regions will be able to be numbered as well as divided into sets. The numbering determines the order of priority of the areas of interest. This gives the user even more control on how the swarm is operated. The GUI is initialized by indicating how many drones it will be talking to for that mission. When a drone is powered on, it will publish a ROS topic for telemetry data. This ROS topic is named according to what number drone it is in the swarm, as well as clearly indicating it is for telemetry data. Each drone unit will be launched by receiving an IP address, and through communicating with Ardupilot, the drone unit will be numbered similarly to how the physical drone was numbered within the swarm.

In this scenario, points are observed as 3D GPS coordinates. Regions are defined by a set of vertices. This is created by connecting the dots that were selected by the user

interacting with the GUI. The user will be able to select a point on the map by selecting the “Point” tool and clicking on the map. When creating a region the user can either make a series of clicks in “Region” mode, or by dragging a rectangular shape over a region on the map. A ROS topic will handle this data being sent to the master unit.

5. Master Unit

This master unit directly sends telemetry data to the GUI, and sends data received from the GUI to the drone units. The data is made up of the points and regions that were defined by the user. This master unit makes a ROS topic available for the main points and regions specified by the user, as well as a ROS topic to each drone unit specifying specific points and regions only to be examined by that drone. The master unit can also send MAVLink commands to each drone unit. Each drone unit sends these commands to the drone’s autopilot via MAVLink and sends telemetry data to the master unit through ROS. Due to ROS being internet based and MAVLlink being serial based, these units are able to communicate on any hardware as long as there is a local area network, or LAN, across the drone swarm, or there is a 915MHz serial radio connected to each drone unit.

MAVLink is often used to send commands to drone autopilots. MAVLink is a two-way communication system with telemetry data being sent from the drone autopilot to the drone’s computer. Commands such as waypoints, flight modes, and auxiliary commands such as setting camera parameters, can all be sent to the drone through MAVLink messages [15]. Each drone unit will connect to its autopilot through a serial MAVLink connection or through an IP connection. The master unit sends MAVLink data

to each drone unit via MAVROS, a ROS implementation of MAVLink. Telemetry data sent from the drone units back to the master unit is also sent via MAVROS.

To maximize functionality, the user will have ROS topics available to them to enhance their platform, but they are not necessary for basic operation. MAVLink commands to individual drone units are another form of optional data output that would enhance the use of the platform.

6. Drone Unit

There will be a drone unit for every drone connected to the system. The user will be able to add functionality to each drone's controller by adding Python code to the provided script. This allows the user to further customize their swarm, enabling them to complete their desired mission. However, the master unit will be able to control each drone unit without additional code. Each drone unit can communicate with other drone units through ROS topics, the master unit through ROS topics and MAVLink, as well as with the autopilot on the drone through MAVLink. These MAVLink commands can be sent through either IP, 915 MHz serial radios, or through USB connections. The drone unit takes in the points and regions from the Master Unit.

B. Testing Design and Set-up

1. Test Hardware

There are several factors to consider when purchasing a drone that can vary based on the needs of the project. Since this project is focused on creating a platform that can be utilized for multiple types of drones, the options for testing are flexible. However, the drones still needed to be simple to use by researchers and could easily send telemetry

data back to the GUI. The main considerations when picking drones for this research project are type of drone, cameras, data transfer, and power.

The first decision is in regard to the type of drone. As mentioned previously, three main types of UAVs exist: fixed-wing, rotary-wing and bio-inspired aircraft. Since rotary-wing drones are the best at hovering and maneuverability, quadcopters were chosen for this project. Using quadcopters would make controlling the drones as simple as possible, since the main focus of the project is testing the efficacy of the platform.

Cameras are also a consideration for drone choice. Many drones can come with or without a camera, and some have better camera quality than others. A feature of the platform is the ability to send image data to the GUI, so cameras were needed for the planned testing but camera quality was not a major concern.

For data transfer, the main choice was between telemetry radios and WiFi. The method of transferring data back and forth between the drones and the GUI is dependent on several different factors, including cost, simplicity and signal range. While WiFi is a cost effective option, it is only useful for short range requirements. Since radio does not require the use of an existing wireless system and is not limited by signal range, 915 mHz USB radios were chosen to facilitate communication from the drones to the GUI.

Lastly, the power requirements for the drone were determined. One type of drone- the Erle Copter- which will be discussed in the following paragraph, came with a 4s (4-cell) lithium battery pack. Upon testing, the team discovered that this 11,000 mAh battery pack was not sufficient to lift the drone off the ground, so a 5500 mAh 3s LiPo battery was purchased as a replacement, and was sufficient in power requirements for

liftoff. 3000 mAh 4s LiPo batteries were also purchased for the team's self constructed drones.

2. Drone Construction

Based on these considerations, two types of drones were chosen for testing. The purpose of having multiple types of drones was to test the efficacy of the swarming platform in controlling multiple drones that are different in nature.

The team purchased two Erle-Copters by the company Erle Robotics. These drones are Linux-based smart quadcopters with support for use with ROS. The Erle-Copter has a modular design, which makes it ideal for testing with different sensors. These two quadcopters each came with an "Erle-Brain"- a Raspberry Pi equipped with Linux autopilot, as well as a GPS system, integrated camera, 4s battery, and a 915 MHz telemetry radio.

Components for another two drones were also purchased for testing. Unlike the Erle-Copters which came pre-assembled, the frame, speed controller, motor, props, and battery had to be purchased separately and assembled in the lab. A Raspberry Pi and camera were also purchased for each drone assembled. These drones are the same type of drone often used for drone-based research.

3. Proposed Testing

Once platform development was finished, the system would then need to undergo testing to demonstrate proof of concept and usability. In order for testing to occur a swarming algorithm would then need to be developed to test on the software platform. For the purposes of testing the team decided to go with the lawnmower method, whereby

the drones perform sweeps back and forth across their designated area until the whole area has been covered. This method is both easy to create and deploy and also one of the most common methods used in drone missions to survey an area. Once the algorithm is developed, the first test to be performed will be SITL testing wherein a mission would be attempted in simulation using the developed GUI and platform. SITL testing is one of the few methods where tests can be conducted in-lab that are as close to testing on real hardware as possible as both use MAVLink and run ArduPilot, which responds to these MAVLink commands. Should the mission be successfully carried out in simulation, it would serve as a proof of concept for the platform design. For this test, the goal is simply for the simulated drones to complete the survey of the designated region and return to base while their movements are accurately captured by the GUI. If the swarming algorithm developed is successful in passing this test, it could later be provided to the user as example code.

Once the platform is able to successfully complete a mission in simulation, hardware testing is the next step. This test is to be broken up into two phases: human factors testing and hardware testing. During the first phase, a test would be set-up wherein a person is asked to try and implement the given example code and set up a mission with as minimal assistance as possible. The person would need to have knowledge of drones and their operation, but minimal knowledge on how drone swarms are set-up. They would then be observed and any difficulties or errors in the set-up would be noted for later improvements to the platform. Once the swarm is set-up correctly, testing would move to the next phase: hardware testing. In this phase the two Erle-Copter

drones, and the two assembled drones would be used to complete the drone mission in a controlled netted research facility. Again the drones as well as the GUI will be observed, and qualitative data would be taken down to later be used to improve the platform. For safety reasons, drone pilots would also be standing by to take over manual control of the drones if needed. Ideally, both phases of the testing would be completed at a minimum of ten times, with at least two observers taking thorough notes of observations each time.

IV. Final Software Functionality

A. System Operation

The drone units, master unit, and GUI are, in the proof-of-concept solution, separate executables. To launch the system, the team currently launches each section individually. It is conceivable that the drone units could be executed as part of the startup procedure on a companion computer, or as part of a script that launches the GUI and master unit, but this has not been tested.

On launch, the master unit creates a ROS topic with the main points and regions defined in the GUI. The master unit then creates a ROS topic for each individual drone, onto which points/regions for each individual drone can be posted in accordance with the user's desired swarm strategy. Finally, the master unit then subscribes to telemetry topics for each drone unit.

Each drone unit automatically connects to an autopilot, then subscribes to the master points and regions topic and its individual points and regions topic, and publishes its telemetry data to another topic.

The minimal network of ROS topics and nodes for 4 drones is shown in figure 4. In the figure, nodes are represented as ovals and topics are rectangles. The drone topics are connected to each respective autopilot over MAVLink. Each drone can also subscribe to telemetry data from each other drone at the user's preference.

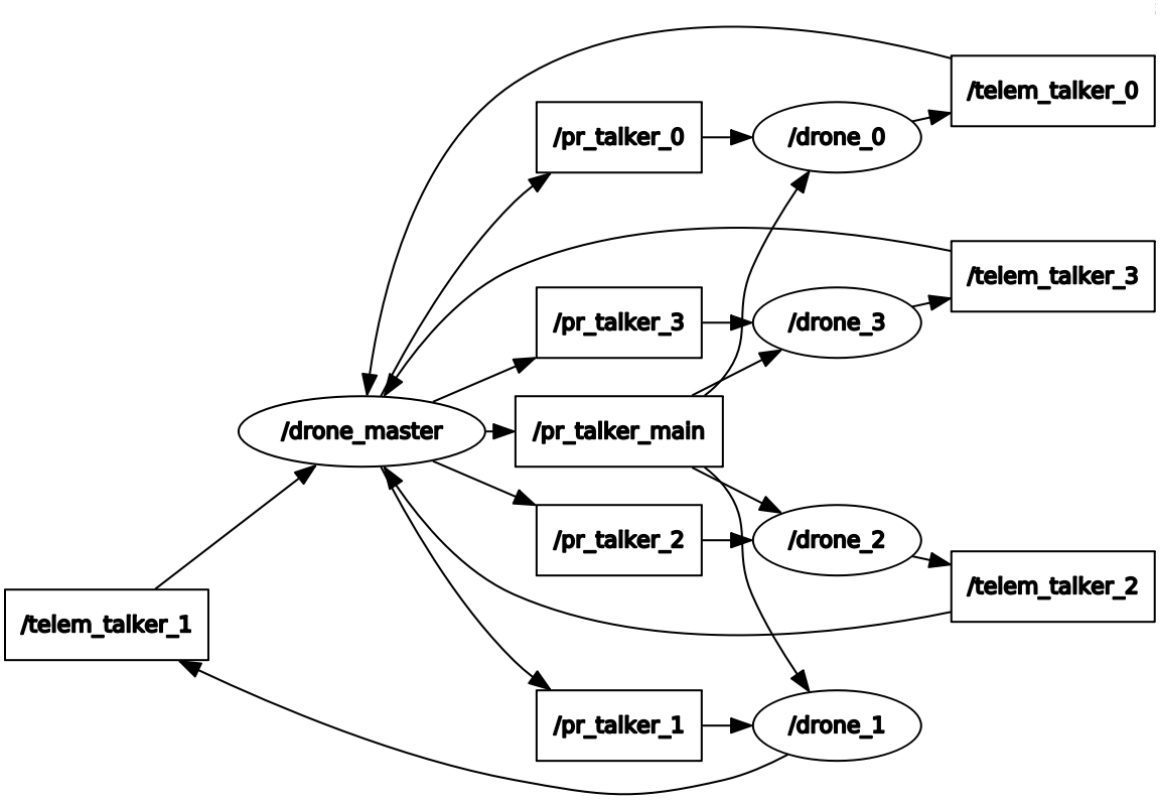


Figure 4: Auto-Generated Graph of example ROS Topics and Nodes for Four Drones.

Once the startup phase is completed, the user-defined code is called to handle the information presented by the system. This code decides how to send points and regions to each drone based on information from the drones and the points and regions submitted via the GUI.

The GUI is web-based, which maximizes system compatibility and contributes to decentralization (see figure 5). It uses Leaflet to display a map where the user can select

points and regions that they want to investigate with their swarm. Leaflet is an open-source JavaScript library for creating maps [40]. The GUI will request data from a Python server that is running on a computer in the network. The drone units will send telemetry data consisting of GPS coordinates to the GUI. This will allow the GUI to display icons on the map displayed showing the movement of the drones.

To describe spatial data with the GUI, a user defines groups of regions, regions, and a list of points for each region. Groups, regions, and points can be reordered within each of their parent groups and be moved between groups. When a user presses "save," the current points and regions list, ordered as they are in the panel on the right of the GUI, is sent to the master unit. When "Start!" is pressed, a launch command is sent to the master unit. The button then toggles to "Stop," which will send a RTL (Return To Launch) signal to the master unit.

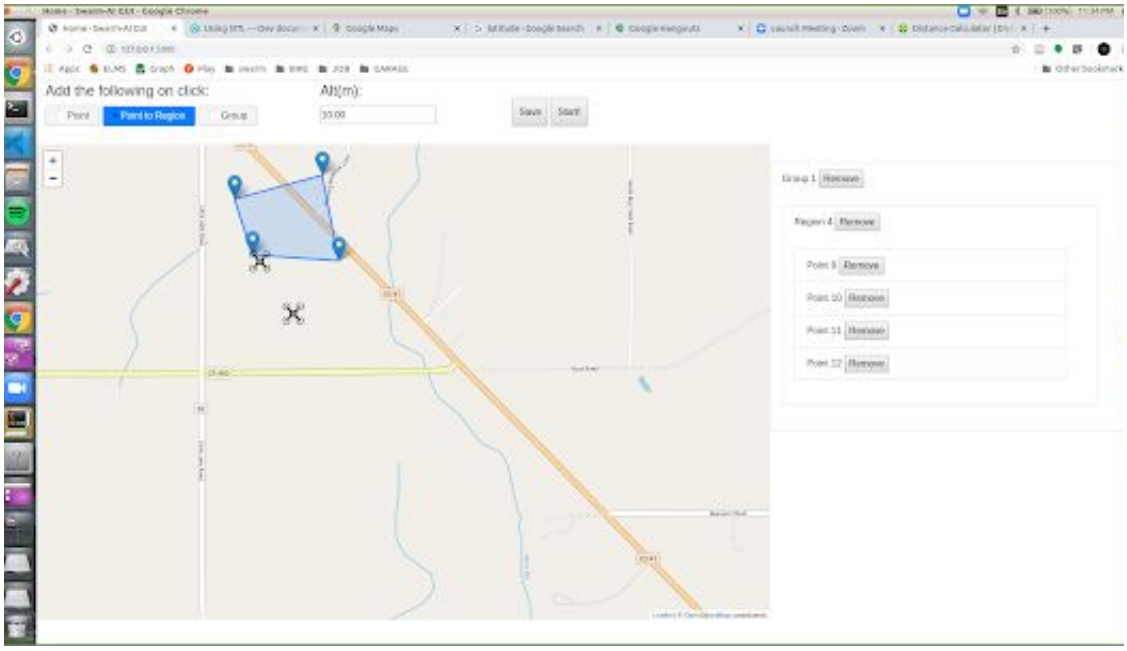


Figure 5: A Screenshot of a Functional Proof-of-Concept GUI

B. User Requirements

The final platform design provides several support services for the user's created drone swarm, while still requiring the user to provide some software and hardware components. First, the user is provided with a GUI interface in which they can initialize the program and monitor drone telemetry data while the drones are in operation. For hardware, the user themselves must provide the drones, the base computer on which the GUI can be viewed, and the autopilots for all the drones. For software, the user must provide their own custom code for sensor handling, as well as user-defined code outlining commands to the autopilots to fit their own mission parameters and swarming strategy.

C. Individual Customization

The three biggest areas for customization are the drone type, swarming method, and sensor data. Since the user decides what drones they want to use, they can pick the drones most suitable to their mission's purpose, whether that be UAV, UGV, or UUV. The user can also customize the method by which the drones search their assigned areas by developing their own swarming algorithm. In order to do this, the user has to edit either the logic by which points and regions are assigned and distributed by the master unit, or by editing the logic used to assign drone waypoints, or both. These files are all written in Python, a commonly known software language, and are available for the user to change so they can create their own swarming method with no knowledge of ROS required. As part of the platform the user and each drone unit also has access to all of the

data being collected by the entire swarm's sensors and thus the user can also implement that data into a swarming strategy. Finally, the user can also customize the sensors equipped to the drone as well as the processing of those sensors data. For example, the user can choose to have their drones collect data and then process and analyze that data when the drones return from the mission, or they can have the drones collect data and immediately send it to the base computer for processing. The only exception to this is that no matter what, the telemetry data for the drone will still be sent to the GUI during the mission, but the user still has access to this data should they also want to use it for another purpose.

D. Limitations

While the platform was intended to be accepting of all system setups, there are a few limitations on the software and hardware components of the system. In terms of hardware, the autopilots equipped to each of the drones must have MAVLink support, since this is how the platform is set up to communicate. This does not end up being too prohibitive as many autopilots use MAVLink, even ones designed for non-aerial drones. To use the platform the drone unit must also have access to the local area network the master node is connected to. This is needed so that the drone units can connect to the ROS topic from which they are getting waypoint data. This requirement could in some ways limit the platform's application in locations where a semi-reliable internet connection is not feasible. Finally, sensor data handling needs to be completely user defined, especially in the case of cameras, because the platform is not yet set up with

ROS nodes to handle the large amount of data. This allows for complete user customization but is also more time intensive.

V. Testing Overview

Due to the current situation surrounding COVID-19, at the time of this writing, the team has not been able to test the system outside of simulation. The system was tested twice--once with the robotics simulator Gazebo and once without--on the same mission. The user selects a region on the GUI and presses "Save." This sends the region to the master unit, where a region just large enough to encompass the defined region is split into 4 areas, once for each drone/drone unit, and each area sent to the respective drone. Each drone unit then defines a series of waypoints which is sent to its respective drone. These waypoints traverse each drone's region in a "lawnmower" pattern, or a pattern that snakes back and forth over the region. The majority of these tests were conducted on a Dell XPS 9550 laptop with an Intel i7-6700 CPU running at 2.59 GHz, 16 GB of RAM, and a Nvidia GTX 960m. The system was running Ubuntu 16.04. Additional testing was conducted on a desktop workstation for improved performance while recording footage. The desktop workstation consisted of a 16 core, 32 thread AMD Threadripper 2950X running at 3.5 GHz, 64 GB of RAM, and a Nvidia RTX 2080 on driver version 440.82.

VI. Results

For each test, an area of roughly 0.1 km^2 was defined on the GUI. The test without Gazebo took 6 minutes and 50 seconds to complete from startup to mission

completion, and 5 minutes and 50 seconds to complete from launch command to mission completion. While running a screencast tool, the computer displayed high clock speeds on all 8 virtual cores, but there was no noticeable lag. The test ran perfectly, with no further remarks. During a few attempts when the member of the team conducting the tests did not start the screen capture program correctly, the Stop function in the GUI was tested and worked well.

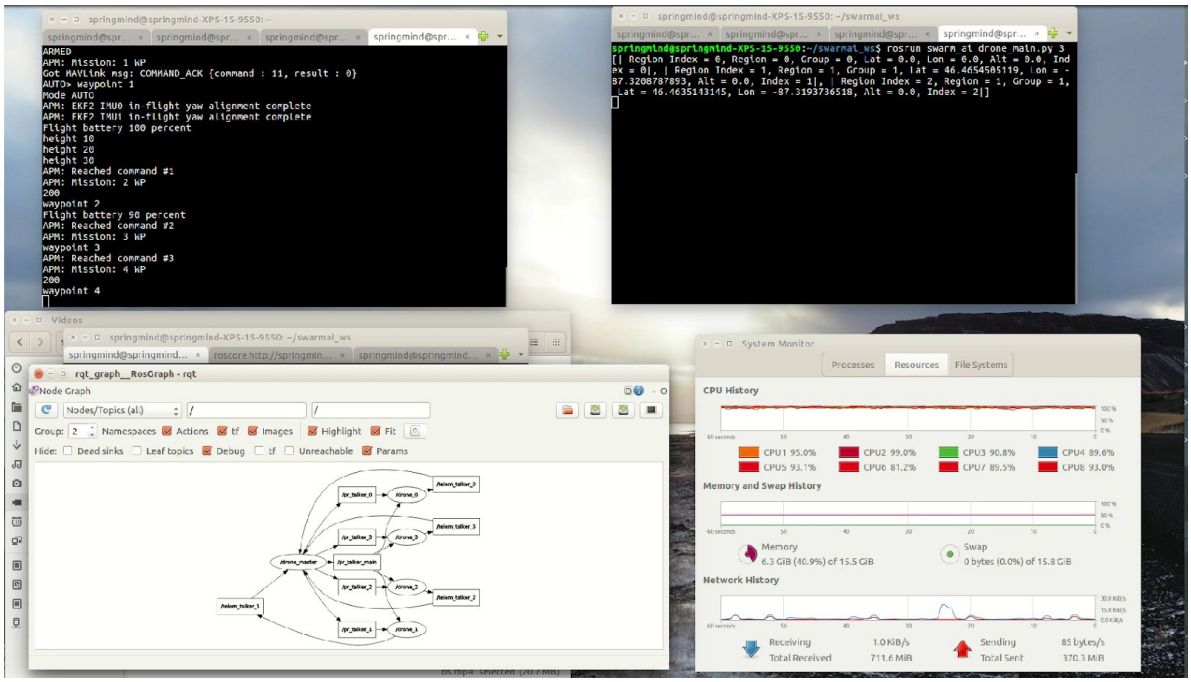


Figure 6A: Set-up for four SITL drones

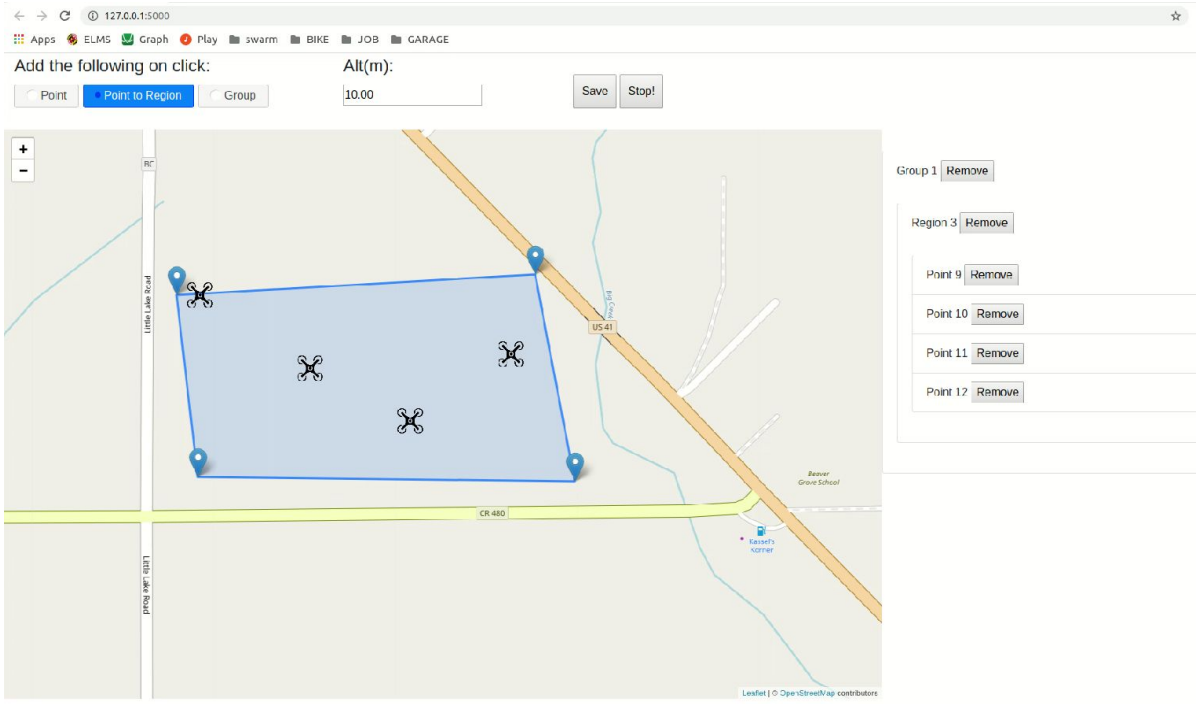


Figure 6B: GUI with four SITL drones

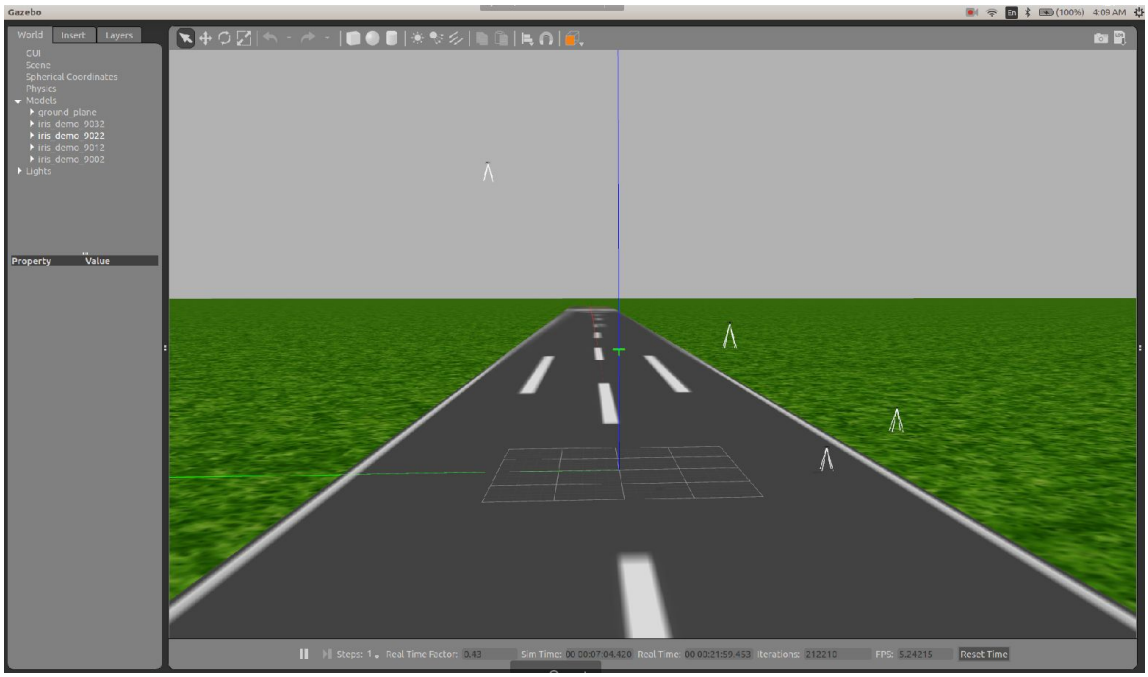


Figure 6C: Simulation with four SITL drones

With Gazebo, the simulation took 23 minutes and 4 seconds to accomplish a similar test from starting procedure to mission completion, and 18 minutes and 15 seconds from launch to mission completion. The increase of time was mostly due to an average simulation time reduction to 25 percent of wall time, and the increase in starting procedure time due to each sim drone's reluctance to obtain a GPS fix, as well as lag in the system. Again, the mission was formed correctly after a change in the waypoint publisher to include a takeoff command was made, which would be necessary in a "real-life" drone. It also uncovered a problem in the way take-off and RTL commands are sent, in which drones would immediately try to take off after connection if the waypoint file is not cleared. The above were conducted on the laptop described in Section V.

The recorded testing on the desktop workstation was conducted in Gazebo with the simulation setup as mentioned above. With Gazebo, the simulation took 11 minutes and 25 seconds from starting procedure to mission completion, and 8 minutes and 2 seconds from launch to mission completion. The difference in test time when compared to the simulations run on the laptop is primarily from an increase in Real Time Factor, which was as high as 1.07, meaning it would only take 1 second of real time to simulate 1.07 seconds of simulation time. It is theoretically possible to run tests with more drones and with a more complex system, but due to time constraints there have been no further tests at time of writing.

A secondary test was conducted to test the systems overall scalability. This test was run exactly the same as the previous verification testing except in this test missions were run with 10, 15 and 20 drones. The purpose of this test was to determine if the

system would still be able to operate and handle the data transmission of a larger number of drone units. In order to conduct these tests, a file was created that was able to launch a desired number of drone units and SITL instances automatically. In this way, the user only needed to specify the number of drones, instead of having to individually launch 20 drone units and SITL instances. The file was also able to terminate all the created instances once the mission was completed.

During the scalability testing the drones were able to run with no errors while using 10 drones, but encountered unexpected issues when run with 15 and 20 drones. In each of the tests there was a large increase in simulation time reduction, as was to be expected since greater processing was required from the computer. Theoretically this lag would not occur with actual hardware, as each of the drones would not need to be simulated on a single machine. During the 15 drone test, the Ardupilot instance for each drone received the correct waypoints. However while some of the drones were able to initialize and take off on their own, around 3 to 4 drones would need to first be given any command in their MAVProxy instance in order to get the drones to proceed as normal. Similar results were achieved in the 20 drone test, where all of the drones were able to receive their waypoints to map, but some of the drones did not initialize the flight mission until any command was run in their individual MAVProxy instance.

It was unclear what specific aspect of the simulation was causing this issue. Since all the waypoints were delivered to each drone unit and there were inconsistencies with the number of drones that would not respond to commands, it was theorized that the problem was due to the limitations of a single computer to simulate all of the drones, and

not any fault of the system itself. In order to further investigate this error, more testing would need to be conducted where multiple computers connected to the same network would be used to run subsets of the drone nodes. This would allow one to conclude whether the error is caused by the platform's scalability, or whether it was the limitation of a single computer to run the drone swarm. Should it be the case that the error was caused by the limitations of the one computer, this same testing method could then be used to determine the full limitations of the system's scalability, and tests could be conducted with a greater number of drones. However, until a full test of the platform can be conducted the limiting factor for scalability is yet to be determined.

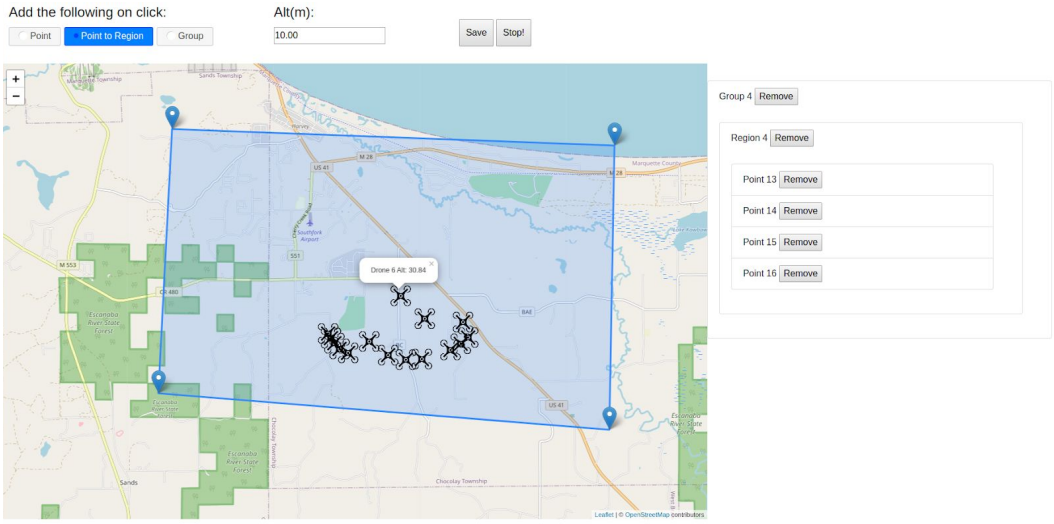


Figure 7: GUI with fifteen SITL drones

VII. Conclusion and Future Work

The team presents the concept for a decentralized control and data transmission system for swarm robotics. This system is novel in that unlike other popular ground control systems it was made to support drone swarming and that it splits the

responsibilities of a traditional GCS among different processes to communicate over IP. The team then presents a proof-of-concept implementation of the decentralized swarm concept and tests it in simulation.

Now that a proof-of-concept for the decentralized platform has been created, there are many possibilities for future improvements and testing of the system. The first task would be to fix the automatic take-off bug. From there, the GUI currently does not have as many functions as other available ground control stations. One important example is that other software platforms allow drone parameter tuning, which can be invaluable for real world testing. It would also be helpful for the user if the GUI could display waypoints from the drones. Similarly, the GUI could host other functions such as publishing video feed from the drones during the mission. The platform could also be improved by developing ROS nodes that are specifically made to process common sensor data such as IMU, GPS, or camera data. These ROS nodes could then make the process of data transfer for each of these unique sensors more efficient and easier to implement.

In addition, the platform could be tested with more complex swarm algorithms such as the pheromone method. In order to implement a new swarming method, additional functionality would be added to the drone node and the master node. For the pheromone method, for example, the drone node would publish a new topic that includes data on its pheromone drops. The master node would subscribe to this topic and track the locations of all pheromones. The master node would run a topic that would publish pheromone information relevant to the drone's location.

This testing would give a better idea of how the platform's decentralized approach affects the development of swarming strategies. Finally, once these changes have been implemented final testing with physical hardware could be carried out. This testing would serve as a final proof of concept and demonstrate any variable not accounted for in the simulations. Additional human factors testing with drone researchers and the platform would also allow for improvements in the ease of use of the software, and the addition of desirable features/functions for the target users.

Appendix

Appendix A: Code

The code for the platform can be found here:

GUI: https://bitbucket.org/swarmai/flask_gui/src/master/

ROS nodes: https://bitbucket.org/swarmai/master_node/src/master/

References

[1] "Amazon Prime Air," *Amazon*. [Online]. Available: https://www.amazon.com/Amazon-Prime-Air/b?ie=UTF8&node=8037720011.

[2] A. Raptopoulos, "Andreas Raptopoulos," *TED*. [Online]. Available: https://www.ted.com/speakers/andreas_raptopoulos. [Accessed: 01-Apr-2020].

[3] M. Gharibi, R. Boutaba and S. L. Waslander, "Internet of Drones," in *IEEE Access*, vol. 4, pp. 1148-1162, 2016.

[4] D. F. Huber and N. Vandapel, "Automatic Three-dimensional Underground Mine Mapping," Int. J. Rob. Res., vol. 25, no. 1, pp. 7–17, 2006.

[5] D. Floreano and R. J. Wood, "Science, technology and the future of small autonomous drones," *Nature*, vol. 521, no. 7553, pp. 460–466, May 2015.

[6] G. Cai, J. Dias, and L. Seneviratne, "A Survey of Small-Scale Unmanned Aerial Vehicles: Recent Advances and Future Development Trends," *Unmanned Systems*, vol. 02, no. 02, pp. 175–199, 2014.

[7] "GROUND ROBOTICS RESEARCH CENTER," *studylib.net*. [Online]. Available: https://studylib.net/doc/14032425/ground-robotics-research-center. [Accessed: 01-Apr-2020].

[8] US Department of Commerce and National Oceanic and Atmospheric Administration, "What is the difference between an AUV and a ROV?," *What is the difference between an AUV and a ROV?*, 01-Jun-2013. [Online]. Available: https://oceanservice.noaa.gov/facts/auv-rov.html. [Accessed: 01-Apr-2020].

[9] A. Hernandez, C. Copot, R. D. Keyser, T. Vlas, and I. Nascu, “Identification and path following control of an AR.Drone quadrotor,” *2013 17th International Conference on System Theory, Control and Computing (ICSTCC)*, 2013.

[10] T. Moribe, H. Okada, K. Kobayashl and M. Katayama, "Combination of a wireless sensor network and drone using infrared thermometers for smart agriculture," *2018 15th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, Las Vegas, NV, 2018.

[11] M. F. Sani and G. Karimian, "Automatic navigation and landing of an indoor AR. drone quadrotor using ArUco marker and inertial sensors," *2017 International Conference on Computer and Drone Applications (IConDA)*, Kuching, 2017.

[12] Y. Li, M. Scanavino, E. Capello, F. Dabbene, G. Guglieri, and A. Vilardi, “A novel distributed architecture for UAV indoor navigation,” *Transportation Research Procedia*, vol. 35, pp. 13–22, Dec. 2018.

[13] E. Yanmaz, S. Yahyanejad, B. Rinner, H. Hellwagner, and C. Bettstetter, “Drone networks: Communications, coordination, and sensing,” *Ad Hoc Networks*, vol. 68, pp. 1–15, Jan. 2018.

[14] “Choosing a Ground Station,” *Choosing a Ground Station - Copter documentation*. [Online]. Available: https://ardupilot.org/copter/docs/common-choosing-a-ground-station.html. [Accessed: 01-Apr-2020].

[15] “MAVLink Basics,” MAVLink Basics - Dev documentation. [Online]. Available: https://ardupilot.org/dev/docs/mavlink-basics.html. [Accessed: 01-Apr-2020].

[16] D. Turner, A. Lucieer, and C. Watson, “An Automated Technique for Generating Georectified Mosaics from Ultra-High Resolution Unmanned Aerial Vehicle (UAV) Imagery, Based on Structure from Motion (SfM) Point Clouds,” *Remote Sensing*, vol. 4, no. 12, pp. 1392–1410, 2012.

[17] Il-Kyun Jung, I.-K. Jung, and Lacroix, “High resolution terrain mapping using low attitude aerial stereo imagery,” in *Proceedings Ninth IEEE International Conference on Computer Vision*, 2003.

[18] G. Dudek, M. M. Jenkin, E. Milios, and D. Wilkes, “A taxonomy for multi-agent robotics,” *Auton. Robots*, vol. 3, no. 4, 1996.

[19] B. P. Gerkey and M. J. Matarić, “A Formal Analysis and Taxonomy of Task Allocation in Multi-Robot Systems,” *Int. J. Rob. Res.*, vol. 23, no. 9, pp. 939–954, 2004.

[20] A. Jevtić and A. Gutiérrez, “Distributed bees algorithm parameters optimization for a cost efficient target allocation in swarms of robots,” *Sensors* , vol. 11, no. 11, pp. 10880–10893, Nov. 2011.

[21] T. Gigl, G. J. M. Janssen, V. Dizdarevic, K. Witrisal, and Z. Irahhauten, “Analysis of a UWB Indoor Positioning System Based on Received Signal Strength,” in *2007 4th Workshop on Positioning, Navigation and Communication*, 2007.

[22] T. Schmickl and K. Crailsheim, “Trophallaxis within a robotic swarm: bio-inspired communication among robots in a swarm,” *Auton. Robots*, vol. 25, no. 1–2, pp. 171–188, 2007.

[23] J. Sauter, R. Matthews, H. Parunak, and S. Brueckner, “Demonstration of Digital Pheromone Swarming Control of Multiple Unmanned Air Vehicles,” in

Infotech@Aerospace, 2005.

[24] K. Derr and M. Manic, "Extended Virtual Spring Mesh (EVSM): The Distributed Self-Organizing Mobile Ad Hoc Network for Area Exploration," *IEEE Trans. Ind. Electron.*, vol. 58, no. 12, pp. 5424–5437, 2011.

[25] C. Ramirez-Atencia and D. Camacho, "Extending QGroundControl for Automated Mission Planning of UAVs," *Sensors*, vol. 18, no. 7, p. 2339, 2018.

[26] "About ROS," *ROS.org*. [Online]. Available: https://www.ros.org/about-ros/. [Accessed: 04-Feb-2020].

[27] K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. C. Wheeler, A. Y. Ng, and Morgan Quigley, "[PDF] ROS: an open-source Robot Operating System: Semantic Scholar," *ICRA 2009*, 01-Jan-1970. [Online]. Available: https://www.semanticscholar.org/paper/ROS:-an-open-source-Robot-Operating-System-Quigley-Conley/d45eaee8b2e047306329e5dbfc954e6dd318ca1e. [Accessed: 04-Feb-2020].

[28] "Robsense SwarmLink," *Robsense SwarmLink - Copter documentation*. [Online]. Available: https://ardupilot.org/copter/docs/common-telemetry-robsense-swarmlink.html. [Accessed: 01-Apr-2020].

[29] "Home," *Craft and Theory, LLC*. [Online]. Available: http://www.craftandtheoryllc.com/store/swarmlink-gateway-and-five-nodes-kit/. [Accessed: 01-Apr-2020].

[30] "mRo SiK Telemetry Radio V2 915Mhz," *mRobotics.io*. [Online]. Available: https://store.mrobotics.io/mRo-SiK-Telemetry-Radio-V2-915Mhz-p/mro-sikv2.htm.

[Accessed: 01-Apr-2020].

[31] “Multi-Vehicle Flying,” *Multi-Vehicle Flying - Copter documentation*. [Online]. Available: https://ardupilot.org/copter/docs/common-multi-vehicle-flying.html. [Accessed: 01-Apr-2020].

[32] D. Gagne, “v3.2 (Detailed),” *v3.2 (Detailed) · QGroundControl User Guide*. [Online]. Available: https://docs.qgroundcontrol.com/en/releases/stable_v3.2_long.html. [Accessed: 01-Apr-2020].

[33] R. Moller, “Distributed Operating Systems: Concepts And Design,” *IEEE Concurrency*, vol. 6, no. 2, pp. 93–94, 1998.

[34] G. Chmaj and H. Selvaraj, “Distributed Processing Applications for UAV/drones: A Survey,” *Progress in Systems Engineering Advances in Intelligent Systems and Computing*, pp. 449–454, 2015.

[35] S. R. Perry and J. H. Taylor, “A Prototype GUI for Unmanned Air Vehicle Mission Planning and Execution,” *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 12214–12219, 2014.

[36] D. Perez, I. Maza, F. Caballero, D. Scarlatti, E. Casado, and A. Ollero, “A Ground Control Station for a Multi-UAV Surveillance System,” *Journal of Intelligent & Robotic Systems*, vol. 69, no. 1-4, pp. 119–130, 2012.

[37] “SITL Simulator (Software in the Loop),” *SITL Simulator (Software in the Loop) - Dev documentation*. [Online]. Available: https://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html. [Accessed: 01-Apr-2020].

[38] M. Rouse, “What is TCP (Transmission Control Protocol)?,” *SearchNetworking*, 27-Nov-2019. [Online]. Available: https://searchnetworking.techtarget.com/definition/TCP. [Accessed: 01-Apr-2020].

[39] M. Rouse, “What is UDP (User Datagram Protocol)? - Definition from WhatIs.com,” *SearchNetworking*, 18-Sep-2019. [Online]. Available: https://searchnetworking.techtarget.com/definition/UDP-User-Datagram-Protocol. [Accessed: 01-Apr-2020].

[40] “an open-source JavaScript library for interactive maps,” Leaflet. [Online]. Available: https://leafletjs.com/. [Accessed: 01-Apr-2020].