# Search and Rescue Swarm

|  |  |  |
| --- | --- | --- |
| Tyler Baldwin | Dominic Calabria | Jacob Wood |
|  |  |  |

### Project Sponsor

### BAE Systems

### Faculty Coach

Scott Hawker

## Project Overview

Search and Rescue Swarm (SARS) is a project that comes from the need to search a large area quickly for a missing person. If a child is lost in a large park then it becomes critical to search and locate the child as quickly as possible so they can return to safety. Manually searching for the child would take many people looking, would risk going through tough terrain that is hard to navigate on foot, and would be time consuming. Through the use of multiple drones it is possible to automatically perform the search as efficiently and quickly as possible. The project contains two key components to accomplish this task. First, an operator interface used by a single operator to connect to the drones, define and start the mission, and monitor the status as the mission progresses. Second, the onboard mission software of the drone that does path planning, navigation, and target identification. With these two components working together it will be possible to define an area for drones to search, launch the drones, and wait for the drones to identify the child.

The project has a few major challenges that need to be addressed in order to accomplish the goals. First, there were financial constraints that prevented us from purchasing drones with onboard computers. That functionality has to be simulated to preserve the distributed nature of the project. The second challenge comes from the distributed nature of the project. Communications between the drones will need to address the “two generals'' problem where messages between two points can not be guaranteed to be delivered. A final challenge again relates to the communication and is that the system needs to be fault tolerant in the event a drone crashes, loses connection, or the battery dies.

The success of this project will be tracked by regular flight tests that show improving results over time. By the end of the project the system should be able to define a search area on a map and have the connected drones evenly split the area between them to search. The drones should then launch and fly their individual paths all while using computer vision to search for a target. When identified, a picture of the target will be sent back to the operator for verification. If a drone is disconnected from the system then the other drones should perform a re-planning operation and divide the remaining area between themselves. With all of these tasks accomplished it will demonstrate the challenges previously talked about have been successfully mitigated.

## Basic Requirements

The operator interface is the only software that accepts input from the user. It must accept the IP addresses of the drones that are part of the system as well as a set of GPS points that define a polygon search area. Other inputs to the system include the button to start the program, a button to recall the drones and abort the mission, as well as a button that appears when the drone thinks it found the target to confirm or reject that image. All of this input comes from the operator of the system. The input is then passed along to the mission control software so the drones can react appropriately to it. There are no external outputs to this software, but while the software is running there will be an image captured of any targets the drones identify and sent back to the operator interface for confirmation.

The operator should be able to set up any number of drones with the operator interface. The only limitation will be the number of computers used and the number of USB ports each computer has since each drone is operated by a controller that must be plugged into the laptop running the software. The operator is also limited by the maximum range of the drone controllers, but will be allowed to select the area that best defines their mission. In the event a drone can no longer be reached, the system should be able to detect that and reconfigure itself so the drones that are reached can split the work lost by the lost drone. When a drone becomes lost, it is the responsibility of the operator to attempt to recall it using the drone controller return to home function.

While running, the drones will first agree on a way to split the area N times for as many drones are in the system and then assign each drone its own path. While on these paths the drones will stream video and status back to the operator so the status of the mission can be monitored. Upon detection of a human, the drone will send an image back to the operator asking for confirmation. This confirmation message will be used to determine if the drones need to keep looking or not. This whole time the drones must be in constant contact with each other and update when they move to a different location. This is how the status of each drone can be maintained.

## Constraints

As of right now, only Parrot ANAFI drones are supported by our software. However, our design is extensible, enabling implementations with heterogeneous drones. The Parrot ANAFI does not support onboard compute, so it is mocked for our project. The drone controller’s (SkyController) range is only up to ~4km line-of-sight, so the software does not work beyond that. The SDK wrapper component of our software must run in Ubuntu as required by the Olympe SDK.

An internet connection is required to initially load the map tiles of the operator interface. This can be done before going to the mission area.

The target detection algorithm is CPU-intensive, so its performance is heavily reliant on the processor being used. In our testing, we ran the full software stack on an Ubuntu VM using 12GB of RAM and an i7 processor.

Under the FAA regulations, any education projects require a Part 107 license to fly the drones outside. Therefore, all Part 107 regulations are required to be followed when flying the drone, such as when flying at least one drone, you need a minimum of two people: a FAA licensed pilot and a visual observer.

## Development Process

From the start of the project to the end, we used Feature Driven Development (FDD) as our main process for the project. We chose FDD because it allowed us to break up the main features into small blocks of work for us to work on. FDD also allowed us to abstract most of the agile process into a more concise way, which was better for a smaller team. The extra time that was saved from abstracting away the process was spent coding and working on MVP work. FDD enabled us to perform frequent flight tests of completed features, providing high visibility to the sponsor.

Our process was not mandated but approved by the sponsor. For communication with the sponsor, we had weekly meetings on Thursdays that lasted for about 45 minutes. These meetings ranged from a weekly 4up chart, monthly feature and metric check ins, feature discussions, and assisting us with design challenges. If any emergency needed to be discussed, we had a Discord server for all of us to talk.

Team roles were identified in the early weeks of the project. These roles came from the work that was assigned to us at the start, and the work we naturally picked up. The following were the roles for each team member.

* Tyler
  + 4up slides, all software process material, project manager, and lead software tester
* Dominic
  + Sent out weekly meeting notices with agenda, developer
* Jacob
  + Lead developer, managed repo and project website

### Project Schedule: Planned and Actual

Back in September, we sat down as a team and brainstormed all the potential features that needed to be done before the entire project was officially started. We then created a Gantt chart to identify dependencies on the features. With all of the dependencies mapped, we then created a schedule with all of the features mapped by month. After mapping all of the features by month, we identified two major milestones in our project: Using the software with one drone (End of February) and using the software with multiple drones (end of March) . Throughout the schedule, we planned all process meetings into account when creating the schedule.

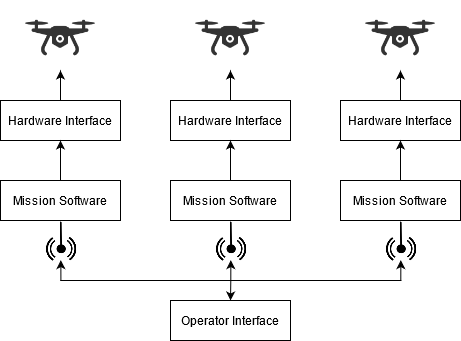
The actual schedule was mostly on par with the plan. Throughout the semester, we only had three instances where we deviated from the schedule. The first one was the timing of the Part 107 license. In October, we had funding issues related to the license, so we were required to delay it till November. However, by the end of the semester, we were caught up and did all required for the semester. The other instances where we did not follow the schedule was the CV and Search Plan trade studies. Due to the unknown complexity of them, as a team we took a risk and removed a week of testing and we pushed the planned schedule out a week for the rest of the Spring semester. This caused us to remove one week of testing towards the end of the semester. This did not affect MVP and allowed us to submit a product that met the sponsor's needs.

## System Design

### High-Level Architecture

The core component of the drone search and navigation system is highly modular. The design allows for one instance of the core system to run per drone hardware - a key aspect of mocked onboard compute. A number of subsystems support the system in communicating with the swarm and operator, executing the search algorithm, and monitoring sensor data. Two separate components exist within this architecture: the core mission software, and the drone SDK wrapper. These components are executed as separate processes, bridged by the Hardware Interface subsystem. However, if an SDK implementation permitted direct usage of Java, these components would be merged into a single process.

High Level Interaction Diagram



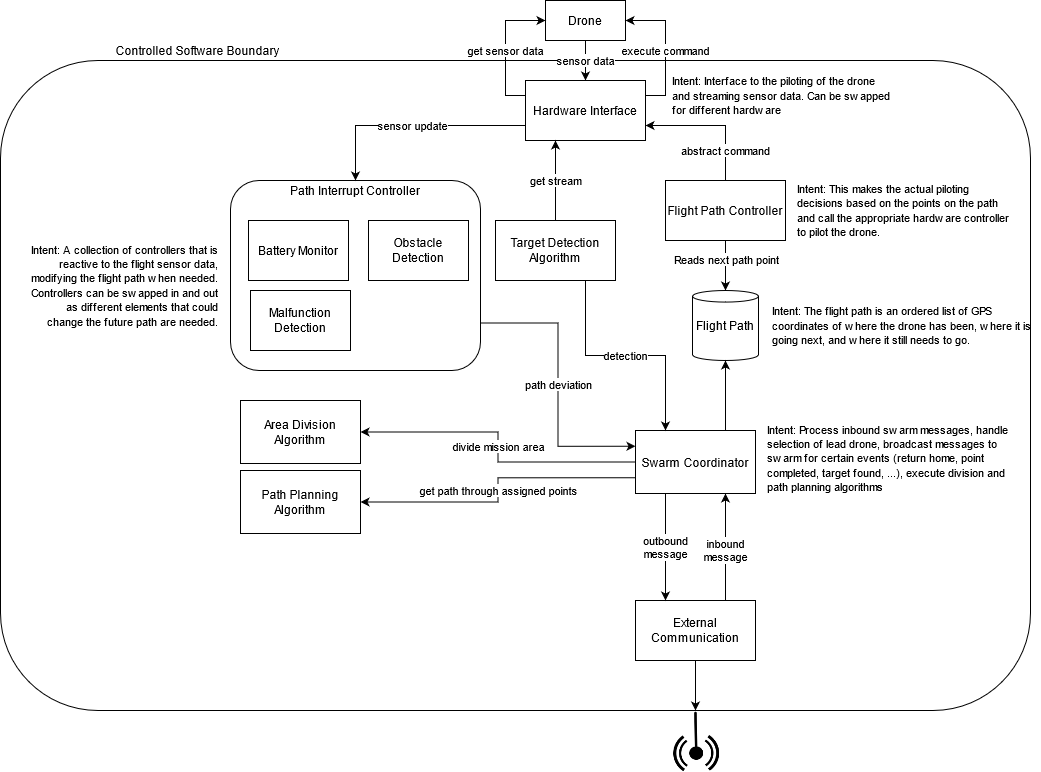
The core mission component is a reactive system that is responsible for area division consensus, swarm status monitoring, path planning, target detection, and path execution.

At the beginning of the mission, the swarm elects a lead drone that is responsible for decomposing and distributing the mission area. Each drone will then plan and execute a route through its assigned area points. During execution of this path, a stream of the drone’s downward-facing camera is fed to the target identification algorithm. If a target is detected, the drone’s current state is recorded and sent to the operator console along with the associated full video frame.

During the mission, the swarm continuously monitors the status of each drone. If a drone is detected to be down, its remaining points are re-distributed by the leader to the remaining drones, which will then update their flight paths accordingly. If the unreachable drone was the current leader, a new leader is selected.

The operator may request a detection to be focused on, in which case a new task will be created and distributed by the lead drone - once again resulting in the drone’s updating their flight paths as needed.

Mission System High Level Architecture



### Subsystems

#### Flight Path Controller

The Flight Path Controller handles flight plan management and execution. At startup, a waypoint-based flight plan is established for the drone’s assigned points using the search algorithm. During flight, this path can be modified if the drone is assigned new points, such as in the case of another drone becoming unreachable. The flight plan is executed through interaction with the Hardware Interface. The Flight Path Controller does not directly ingest any sensor data - the determination of when a flight command has been fully executed is handled by the Hardware Interface.

**Path Interrupt Controller**

The Path Interrupt Controller is responsible for monitoring sensor data and changing the flight plan based on this data when needed. The subsystem is composed of a set of Path Interrupt Modules that operate independently for extensibility. Currently, the only implemented module is the Low Battery Monitor. Future modules could include Obstacle Avoidance and Malfunction Detection. These modules are intended to be *reactive* - it is expected that the Hardware Interface will stream sensor data as opposed to the modules needing to continuously query sensors.

**Low Battery Monitor**

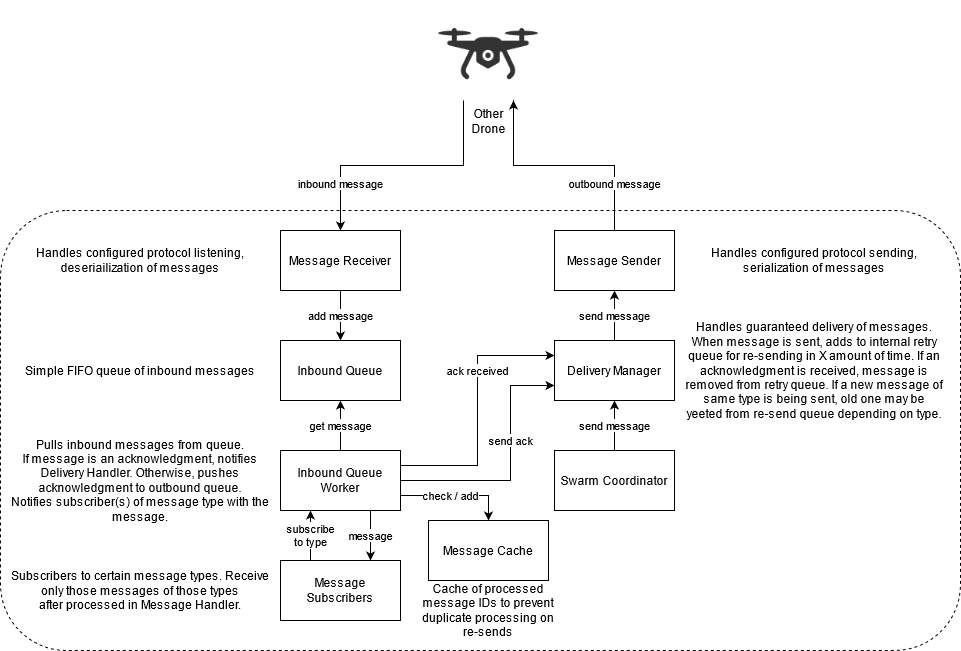
To ensure a drone will not die mid-flight, the Low Battery Monitor checks its battery percentage. Should the percentage drop below a configured abort amount, the Flight Path Controller is instructed to immediately fly the drone back to its home position. This has a side-effect of the drone informing the swarm that its remaining points need to be re-allocated.

This functionality is also implemented natively by some drone manufacturers, such as a 5% return home threshold for the Parrot ANAFI.

#### External Communication

The External Communication subsystem is responsible for drone-drone and drone-operator messaging. This is the only major subsystem that is leveraged both in the mission and operator software components.

Custom queueing and guaranteed delivery mechanisms were chosen for extensibility. We assessed that our use case of an IP network may be unlikely in non-mocked onboard compute scenarios, so we avoided usage of traditional IP-based solutions like ZeroMQ. With this design, different protocols can be easily supported by adding implementations for the Message Receiver and Message Sender components. Elsewhere, the underlying communication protocol is completely abstracted away.



**Implementation: Parrot ANAFI**

Onboard compute is mocked for our Parrot ANAFI implementation, so communication occurs between computers on an IP network. As such, UDP was chosen as our underlying protocol implementation. This choice was made based on the unwanted network overhead of TCP.

#### Hardware Interface

The Hardware Interface provides a common bridge between our mission software and drone-specific control software. This interface includes both the ability to send commands and queries to the drone, as well as listen for drone events (i.e. sensor data). A key principle of this system is that it abstracts away drone-specific knowledge to provide extensibility. Currently, the only implementation of this interface is for the Parrot ANAFI, but implementations for other drones would be easily supported without any modification to the rest of the system.

**Commands**

Commands are executed synchronously. For IPC-based implementations, a “Command Completed” event is provided to permit blocking until command execution has completed.

**Drone Events**

It is expected that each drone event will be automatically generated by the Hardware Interface at a reasonable interval. This enables the system to be reactive as sensor data does not have to be continuously queried. However, these events can also be triggered by one of the above queries (e.g. Get Orientation results in an Orientation Status event).

**Implementation: Olympe**

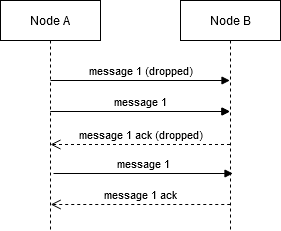
The [Olympe SDK](https://developer.parrot.com/docs/olympe/) is used to interface with Parrot ANAFI drones. Because Olympe is strictly a Python SDK, a cross-language bridge was needed to support it. This bridge is implemented using [ZeroMQ](https://zeromq.org/) [PAIR](https://learning-0mq-with-pyzmq.readthedocs.io/en/latest/pyzmq/patterns/pair.html) sockets with Protobuf message serialization. Events are automatically generated by Olympe and streamed back to the Java-based Hardware Interface client. Similarly, the Python server uses the Olympe SDK to execute piloting and camera adjustment commands.

**Swarm Coordinator**

The Swarm Coordinator is responsible for selecting and maintaining a lead drone, monitoring the status of each drone in the swarm, maintaining the swarm coverage state, and executing the area division and path planning algorithms. This subsystem handles most inbound messages and generates most outbound messages. The responsibilities of the subsystem make it a key component to the distributed and autonomous aspects of the system, as it enables the swarm to respond to changes without the need for operator interference. For example, if a drone needs to return home due to low battery, the Swarm Coordinator will inform the other drones of this, resulting in the lead drone re-allocating the returning drone’s remaining points.

### Algorithms

**Guaranteed Delivery**

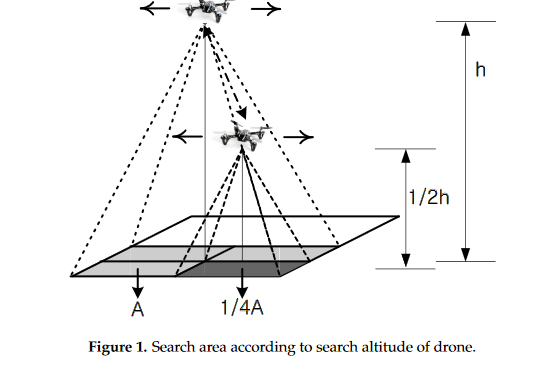
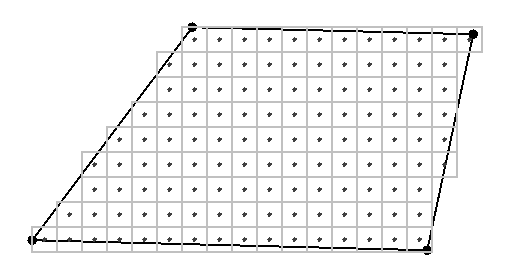


The guaranteed delivery algorithm ensures that external messages are delivered. This is necessary due to the inherently unreliable communication between drones.

The flow of this algorithm is rather simple: when a message is received, an acknowledgement message is returned. Each message will be continuously re-sent after some delay until an acknowledgment message is received. However, acknowledgement messages are not re-sent unless the same message is received again.

**Area Division**

The search area polygon is first approximated by grid cells. The side of these grid cells is determined by the horizontal field of view of the lead drone’s camera and the minimum operating altitude for the mission. The center of these grid cells are used as the distributable search points.

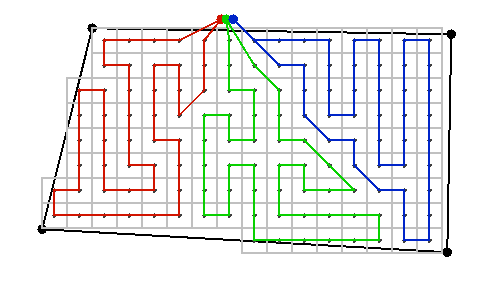
  
Example of grid cell approximation Demonstration of grid cell sizes varying with altitude

These points are then divided using each drone’s current position as inputs to the MILP optimization load balancing algorithm defined in [2]. The battery status of each drone is *not* taken into account by this algorithm.

This division algorithm is repeated in the event that points need to be re-distributed or added to the mission.

**Path Planning**

A modified Lin-Kernighan heuristic defined in [2] is used for path planning through a drone’s assigned points. The algorithm was implemented using Google [OR-Tools](https://developers.google.com/optimization). This algorithm takes into account the cost of turns to minimize flight time and energy cost.

  
Example of planned paths through divided area

**Target Detection**

The [Okutama-Action](http://okutama-action.org/) Pedestrian model is used for target detection. Streamed video frames are fed to this model through OpenCV and detection bounding boxes above a configured confidence threshold are processed by the system.

## Process and Product Metrics

The following are the metrics we tracked and the metrics at the end of the year:

|  |  |
| --- | --- |
| **Metrics** | **Value** |
| Defects after “release” | 4 |
| Number of features completed (Second Semester only) | 35 |
| Code coverage (unit tests, Total) | ~75% |
| Feature delays | 5 |
| Documentation for completed work | 100% |
| Average percent of requirements passed at test event | 91% |
| Number of flight tests | 13 |

Ultimately we did not find these metrics particularly valuable. The Feature Delays measurement showed some of our schedule slippage in the spring semester, and defects per 1KLOC increased as we began testing more frequently and the complexity of our implementation increased.

The metrics that we found to be the most important were the flight test metrics in the spring semester because they allowed us to verify how much of the system that is currently written was working correctly at that time.

Overall, we believe that our metrics showed that we were a successful team. Throughout the semester, our work consistently showed we were on target for everything. However, we wished we took the metrics more seriously, as there were times where we neglected the metrics and went with our instincts, leading to some schedule issues.

### Product State at Time of Delivery

As the project was written in the initial requirements document, the software is complete with no planned features missing. We have some known bugs throughout the software, and they have been fully documented. In the last weeks of the project, we added an additional beta feature that requests a drone to zoom-in on a detection location after the operator requests it, by request of the sponsor. This was the only feature that was not promised back in the Fall semester that we did not commit to. However, we were able to complete implementation of this beta feature. This feature was tested once; it was not successful due to an unrelated target detection issue. This implementation is included in the final code deliverable.

## Project Reflection

As a team, we believe that this project was a success! We completed all requirements, and even some stretch goals like computer vision and heterogeneous drones, or allowing the system to take in more than one drone type. During the last weeks of the project we were also able to start on a beta feature that allowed the drones to zoom into a potential target, viewing the target in more detail. All software, except for the beta feature, has been completely tested, both at a software and hardware level. One of our earliest achievements was Tyler getting his Part 107 drone license early enough that we were able to test the drones outside back in the Fall semester. In addition, our process of testing the system outdoors allowed us to have high visibility into the progress we made as a team.

However, not everything went as planned. We had some slight schedule issues in the Spring semester due to the unanticipated complexity of trade studies. We believed that working intently was vital to the success of this project because they give insights on what path we should take technically as a team. Unfortunately, due to Rochester weather, we had to cancel planned tests because of the weather, which limited project visibility. That being said, we wish we investigated the Parrot simulator more so we were not so dependent on nature. Our process was very loosely followed as our team of three felt it to be more of a burden than a tool. We used feature driven development to limit some of the process methodology at the start of the semester, but towards the middle of the Fall semester, we picked and chose what we felt was useful for us, which was very little. One thing we found to be very helpful with FDD was breaking down each requirement into smaller features for us to work on.

If we were to do this project again, we would’ve created a new process methodology that allows us to have more flexibility as a team of three. We would’ve also used metrics more to guide us into more decisions with software quality, because there were points in the project where our documentation and code coverage was low. In addition, we wish we worked on the trade studies back in the fall semester so we could’ve designed the system around them, rather than treating them like separate subsystems.

### References

1. <https://www.faa.gov/uas/media/part_107_summary.pdf>
2. Modares, J., Ghanei, F., Mastronarde, N., & Dantu, K. (2017). UB-ANC planner: Energy efficient coverage path planning with multiple drones. 2017 IEEE International Conference on Robotics and Automation (ICRA). doi:10.1109/icra.2017.7989732