CP34 - Optimal Path for Drone Delivery

Dylan Duplessis
Nicholas Hui
William Wu
Derong Liu
Justin Lee
Yufan Xu

Tutor: Abdallah Lakhari

School of Computer Science
Faculty of Engineering and IT
The University of Sydney

25 October 2020

Executive Summary

Project Title: CP34 - Optimal Path for Drone Delivery

Team Members:

- Dylan Duplessis
- Nicholas Hui
- Ivan Xu
- Jimmy Liu
- William Wu
- Justin Lee

Summary:

There is a rapid growth in the technology industry opening many opportunities for experimental drone development.

This project assumes we are flying drones at various altitudes where it they are constantly calculating battery level, nearest charging stations, and distance to the next way-point. The final goal of the project is to allow a fleet of drones to autonomously deliver parcels to destinations in the most optimal way considering different factors such as weather conditions, battery life, and accurate distances.

This project is not concerned with object detection, but rather the algorithm to compute the shortest path given a number of parameters previous mentioned. However, collision avoidance between drones and objects will be considered.

Contents

Executive Summary 2
1 Introduction 4
1.1 Aim 4
1.2 Key stakeholders 4
1.3 Identification of resources and risks involved in the project 5
2 System Specifications and Design 5
2.1 Core Requirements 5
2.2 Constraints 6
2.3 Systems' Architecture 6
2.4 Remaining User Stories 6
3 Quality of Work 8
3.1 Testing plan 8
3.2 Acceptance Criteria and Tests 8
3.3 Relevant Testing Techniques 9
3.4 Use and Application of Discipline Knowledge 9
4 Quality of Group Processes 11
4.1 Development Tools Setup, Task Allocation and Management 11
4.2 Evidence of Collaboration 11
4.3 Allocation of Roles, Potential Risks, Constraints 11
4.4 Use of Tools 12
4.5 Client Interaction 13
4.6 Reflections and Conclusions 13

References **14**
5 Appendix 14
Wiki 14
Individual Report 14
User Stories 14
Meeting Minutes 14
Email History 15

1 Introduction

1.1 Aim

The project undertaken is the optimal path for drone delivery which requires us to implement autonomous drone flight with built in object detection and collision avoidance within simulated worlds. The goal is to compute and navigate the shortest path from a starting position to a number of way-points landing at charging stations along the way if necessary.

Using the Gazebo simulator, a couple of flight stacks such as PX4 and ArduPilot, and help from various online communities, we have tackled a variation of the infamous travelling salesman problem and set out to create a product our client will find useful.

The problem to be addressed is the drone being controlled autonomously, flying to avoid obstacles and reach way-points without running out of battery in the shortest time possible. This project is not too concerned about object detection but rather focused on the algorithms for path finding where collision avoidance between drones and buildings is considered.

In order to resolve this problem we have planned to use a reduction of the Dijkstra algorithm which outputs the shortest path given way-point and charging station locations. The goal of autonomous drone flight is broken down into subtasks such as algorithm design, simple drone navigation, and collision avoidance.

Each week a pre-determined milestone will be reached by working on each subproblem and the smaller tasks associated with them and as each is solved, the solution is tested and documented. This process ensures a higher quality of work and motivates us to continuously improve.

1.2 Key stakeholders

1.2.1 Who are they?

Stakeholders include, but are not limited to:

- Client
- Delivery companies
- Relevant communities (e.g. PX4, ArduPilot, Gazebo)
- Development team

1.2.2 What do they do?

In the modern world, delivery companies would benefit from an updated delivery system. By using drone with an optimal path for delivery, companies can achieve a cost efficient and an improved delivery time. As drones are not limited to ground traffic they would be more efficient in performing door to door deliveries especially considering that multiple drones can be deployed. Additional benefits would be the reduction of manual labour as drone delivery will be automated, and the increase of job opportunities as drones will require mechanics and software engineers.

With the growth of the drone technology, communities and development teams can focus more on the automation of the drone development and path optimisation. Being able finding optimal path for drones considering external factors such as weather conditions and obstacles will greatly benefit the drone community and delivery companies.

Additionally, as technology progresses, interest in drone technology and the technology itself will become more prevalent and thus online drone communities and companies would reap benefits from this project as they can build upon the work completed in this project and explore new avenues.

1.2.3 How do they interact?

For delivery companies, they can utilize our project to help simulate a world resembles their area of interest and test out the optimal path for drone delivery, by following our documentation on how to work with the simulator, flight control software and the algorithm we designed.

For relevant communities, can use our project as reference for further research and development of optimal path finding for drone delivery, or utilize drone for other usage for other tasks that requires path finding.

1.3 Identification of resources and risks involved in the project

The drone in the simulator in the project used is an iris drone model, if the key holders have different drone model, the party will have to change the drone model in the simulator and the flight control software. As a different model of drones will have different constraints what will affect the path finding algorithm.

The Civil Aviation Safety Authority (CASA), where the drone safety rules and regulations will have to strictly followed by the key holders while using the project simulations as the laws do not allow for anyone to fly drones without a license. As for personnel associated with the university cannot fly a drone without a remote pilot license, since the university is a registered organisation with CASA.

Resources available in the project include but are not limited to.

- Feedback from the tutor.
- Feedback from the client.
- Discord channel, has industry experts and students other group doing similar capstone projects.
- PX4 and Ardupilot forum, community.
- Github repository from last year's team.
- Edstem discussion forum.

2 System Specifications and Design

2.1 Core Requirements

(1) As a user, I want to be able to use both ArduPilot and PX4 drones for delivery so I have a broader choice of drones.
(2) As a user, I want the Gazebo simulator to be fully functional and configured with PX4
(3) As a user, I want the Gazebo simulator to be fully functional and configured with ArduPilot
(4) As a user, I want to be able to control the drone by doing coordinate based or point to point navigation in the simulator
(5) As a user, I want to be able to have different weather conditions in the simulator so that I can simulate the delivery in different environments
(6) As a user, I want to have a plan of how to achieve the optimal path and include base stations and proof of concepts in the simulator.
(7) As a user, I want to have a charging station model in the Gazebo simulator so that I know what a charging station looks like
(8) As a user, I want to be able to control the drone using code/scripts so that I don't have to manually control the drone
(9) As a user, I want to be able to create worlds in Gazebo with variable spawning locations for charging stations and delivery way points
(10) As a user, I want autonomous drone flight to be possible so that the whole delivery process is automated

Functional Requirements

- Modelling objects such as the charging station and worlds,
- Documentation,
- Autonomous drone flight,
- Randomised locations of charging stations given specific parameters,
- Object detection and decision making

Non-Functional Requirements

- Readability: Code should be well documented and re-factored such that
- Scalability: The user should be able to render larger worlds with more drones and have the algorithm work as effectively.
- Usability: The system should be intuitive and easy to use such that an user without technical expertise would be able to quickly learn and utilise the delivery system.

2.2 Constraints

- The compatibility of the script between the flight control software ArduPilot and PX4, as they run on different version of Python and the structure of the code are different in the two flight control software.
- Gazebo, ArduPilot and PX4 work best with Ubuntu18.04, as it has the most reliability among the other OS.

2.3 Systems' Architecture

The system consists of flight control software as well as ground control software. The flight control software are PX4 and Ardupilot, which are autopilot systems that controls the drone. The ground control software are QGroundControl and Dronekit. QGroundControl displays the drone's telemetry data and can provide manual control of the drone, whereas Dronekit, a Python library, can be used to send flight instructions/flight plans to the flight control system. The flight control and ground control software communicate through MAVLink, which is a messaging protocol that allows for the sending of flight instructions and receiving of drone telemetry data. The Gazebo simulator, which deploys a custom environment, represented as an xml file, simulates one or more drones flying in the real world as well as physics, wind, terrain and objects include buildings, obstacles and charging stations.

The implementation of this system involves generating the Gazebo world environment and passing the location of the charging stations and delivery way-points as input to the optimal path algorithm. The outputted optimal path is converted into a flight plan and uploaded to the drone to be executed via the navigation script created using Dronekit. These implementation are all done in Python, using relevant Python libraries where needed.

2.4 Remaining User Stories

- As a user, I want to have multiple drones in the simulator at the same time so that I can use multiple drones for delivery
- As a user, I want a primitive detection/model working on 4 signs (stop, turning, parking, traffic lights)
- As a user, I want the drone to be able to do autonomous flight with detection active so that it does not crash into anything
- As a user, I want documentation for placing objects such as signs, traffic lights, objects so that I can place objects myself

The remaining user stories mainly involves object detection, which is well documented in the drone community and the team will be undergoing further research in order to implement these features. The other user story concerns flying multiple drone in the simulator, which is also well-documented and involves using Gazebo ROS, which is a tool the team has already

explored before. Whilst the team did not have time to implement these stories in the first few sprints, the team has a good idea about how to complete them.

2.4.1 Design Patterns

The goal of this algorithm is to find an optimal path for drone to fly from a starting point to destination while taking multiple factors into consideration. These are identified factors:

- Charging stations
- Drone battery
- Reserved battery
- Weather
- Obstacles
- Stand by drones

The core of this algorithm is Dijkstra's algorithm for finding the shortest path between nodes. Nodes are used to represent starting point, destination and charging stations, and edges connected these nodes are the paths between each location. To measure how factors affect the drone, we will convert the effect into a change in length between nodes. Drone battery is transformed to distance it can travel before next recharge. The design of the algorithm are based on the following assumption:

- Use three dimension point (x,y,z) to encode the location of starting point, destination and charging stations. This can be used to calculate the distance between them.
- The location data will be in floating-point numbers.
- The hardware will have sufficient capacity and computational power to run algorithm. Space and time complexity analyze will be provided.
- The algorithm will not be required to control the drones, only giving an optimal path.
- Battery status does not impact the performance of drone.
- The starting point as well as destination of a drone is not a charging station

2.4.2 High-Level System Architecture Diagrams

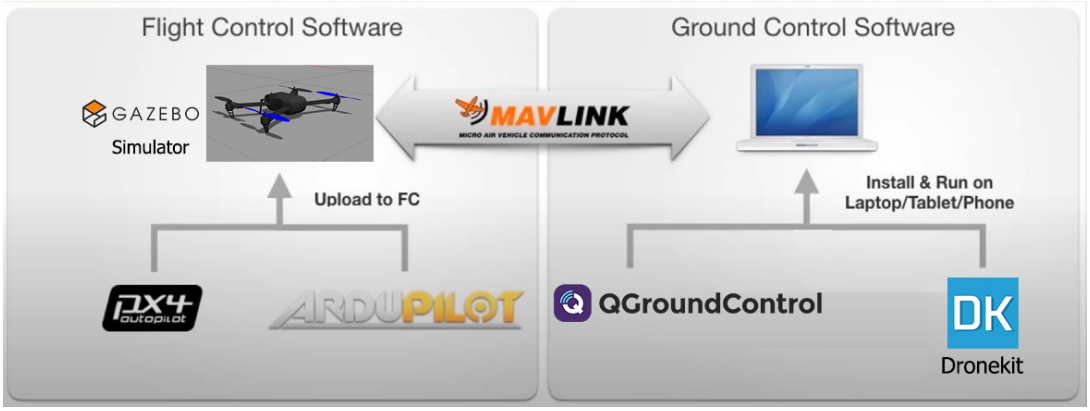

Figure 1: High-level system architecture diagram

2.4.3 High-level system component diagrams

Our three components added to the system are

- Gazebo World Generator
- Optimal Path Algorithm
- Automated Navigation Script

The user would be able to either choose to generate a Gazebo world through the generator and then send the world data to the optimal path algorithm or directly pass the location of way

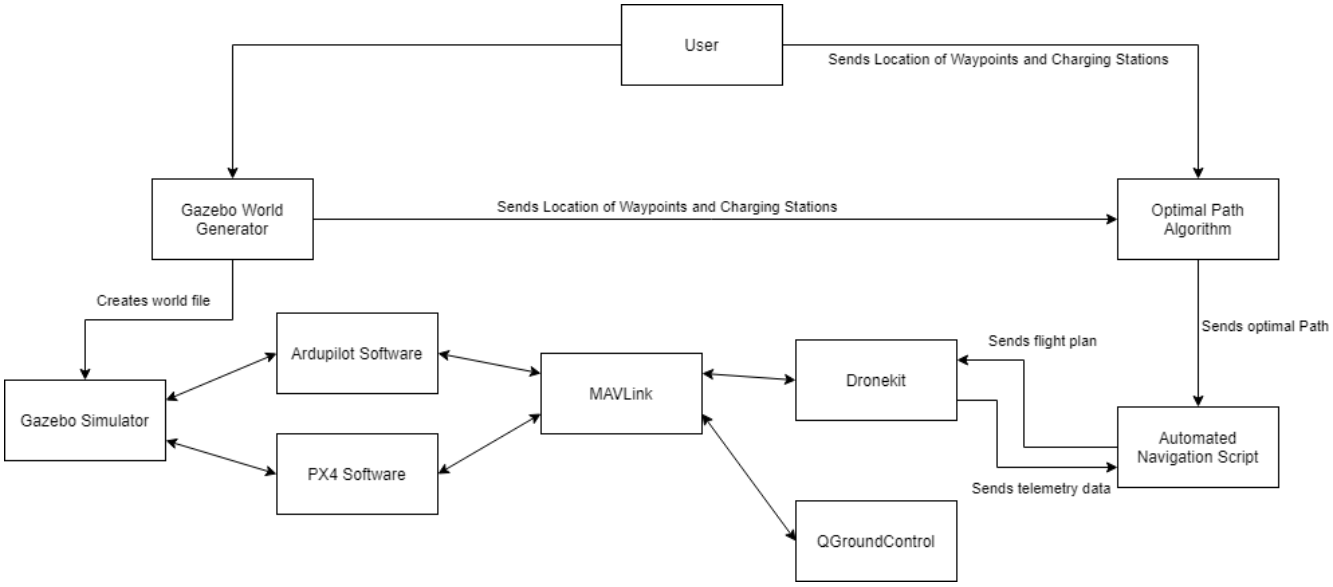

Figure 2: High-level system components diagram

points and charging stations of an existing Gazebo world. The output of the algorithm would then be converted to a flight mission and uploaded to the drone.

3 Quality of Work

3.1 Testing plan

Unit testing of the 3 main components will be conducted to make sure that the components will work by itself. This is done through creating test cases and examining the expected outcome vs the actual outcome.

Integration testing will be conducted on these components to ensure that the components are communicating with each other correctly by outputting appropriate and expected data for the other components to use.

- Gazebo will be able to load the world file created by the Gazebo World Generator
- The Gazebo World Generator will create charging stations and way points and pass it to the Optimal Path Algorithm
- The Optimal Path Algorithm will output a JSON file with a list of way points that is the optimal path, and the automated navigation script will be able to read this file and create a flight mission based on this path, upload it to the drone and execute the mission.

System testing will be conducted to ensure that all the components implemented will be able to work with all other system components.

3.2 Acceptance Criteria and Tests

The tests considered so far are:

- User stories 1-3: These are related to the installation of the simulator there acceptance criteria is that the team can get the simulator environment configured and working.
- User stories 4: Acceptance criteria - the drone is able to go to a user specified coordinate. This is tested by uploading a coordinate and have the drone navigate there
- User story 5: Acceptance criteria - the weather in Gazebo can be modified. This is tested through loading a world with different physics.

- User story 8: Acceptance criteria - the code/script is able to control the drone. This is tested through manually seeing if the drone behaves in the manner specified by the code/script.
- User story 10: Acceptance criteria - the drone is able to autonomously go to numerous specified coordinates and land at each coordinate, simulating a delivery pattern. This is tested through some basic coordinate combinations, as well as boundary cases with only one or 40 coordinates specified. Abnormal cases of having multiple identical coordinates consecutively are also considered.

3.3 Relevant Testing Techniques

Taking into consideration the early stages of the project was plagued with installation issues, there was little time allocated to testing.

3.4 Use and Application of Discipline Knowledge

3.4.1 Quality Constraints

Within this project, a plethora of skills and knowledge are required from a variety of disciplines in order to produce deliverables to a satisfactory degree. The implementation of the system required knowledge of the Linux and Windows operating systems. This discipline knowledge was applied by all team members and includes but is not limited to: Installing an entire operating system (In this case, Ubuntu 18.04 LTS) via dual-booting, using the command line, running system updates and using built-in software unique to Linux such as text editors like vim.

As the project becomes more technical, application of skills and knowledge from the Computer Science discipline grow more common. Programming skills are essential to this project, they are the backbone of the software developed. Strong knowledge of programming techniques and idioms as well as algorithm design and analysis enhance the quality of the code written. Additionally, through this enhanced understanding, a more detailed documentation is produced; within the code through comments and "docstrings", and separate documentation reports written to describe the high level aspects. These applications improve the overall performance and readability of the code.

Software engineering afforded testing skills to ensure all corner and edge cases are covered, agile methodologies, and strong background knowledge on the software development life cycle providing guidance and a development structure.

Above all, methods drawn from project management are utilised more than anything. These include:

3.4.2 Research and use of relevant algorithms and design patterns to solve problems during development

During the development of the program which randomly generated charging stations within a given world, many problems arose. To combat these problems, research into algorithms which generate random points within geometric shapes was required. Much was learnt about the cumulative distribution function and the inverse transform sampling method in order to generate random points within a circle.

Within this program, there were numerous design patterns utilised. Creational patterns such as prototypes were relied upon extensively as the generation of charging stations required a blueprint; the size, physics, and other properties, and a simple python function was used to fill in the generated locations with respect to the drone.

Extensive research was undertaken regarding numerous existing DroneKit programs and documentation, and it was all pieced together in order to create an automated navigation script tailored to our team's needs. This involved using relevant algorithms such as converting a way

point's GPS coordinates to an offset in metres relative to the drone's location, as well as refactoring existing DroneKit programs such that it is compatible with both PX4 and ArduPilot.
Research for algorithm: The root problem is to figure out an optimal path for the drone delivery. At the beginning, two possible algorithms are presented, one is called postman algorithm which finds a shortest path when a drone is required to go to multiple destination. This algorithm has high complexity, comparing to Dijkstra's, which finds an optimal path from a starting point to an end point.

For the ease of implementation and reduction, we will do Dijkstra's first. However, it is not guaranteed in the real world that all drones carry a single item at a time. It is possible that a drone can carry multiple items at one go. Thus, we need to enable a quick switch or improvement between algorithms in the future. This possibly requires a set of abstract classes and design patterns which we do not employ in our current algorithm code base, this might be taken into consideration in the future.

The core idea behind the reduction is to convert everything into distance, which can be easily fed to Dijkstra's. The initial distance can be calculated using 3D coordinates. And then, for weather/obstacles/altitude, we modify this distance respectively to reflect the influence of them. In this way, using some polymorphism, we will able to easily add multiple features. Finding the impact requires effort in simulator.

3.4.3 Complexity analysis of designed algorithms

Phase 1: Reduction to Dijkstra's

Abstracting the problem to a drone with a single start point and single destination letting n be the number of charging stations in between.
First we analyze the space complexity required to reduce the drone problem to Dijkstra's. The three steps to generating paths are:

- connect start point to the destination $O(1)$
- connect start point to all charging stations $O(n)$
- connect stations to one another (classic handshake problem) $O(n^2)$
- connect stations to destination $O(1)$

Space complexity In total:

$$nodes : O(n)$$

$$edges : O(n^2)$$

The complexity of Dijkstra's is known: $\Theta(n^2)$ since the number of edges are already analyzed above. Hence the running time for algorithm is

$$\Theta(n^2)$$

Phase 2: Battery Consideration

For each edge, before it is added to the graph, check the following conditions:

- for outgoing edges from start point, no longer than existing range of battery
- for edges between stations, no longer than maximum range
- for incoming edges to destination, make sure a drone has sufficient battery left

This add $O(c)$ to each edge creation process. The running time for algorithm is still

$$O(n^2)$$

but theoretically, there should be fewer edges which help reduce the actual time and space complexity.

4 Quality of Group Processes

4.1 Development Tools Setup, Task Allocation and Management

Setup of Tools

The team installed the Gazebo simulator in order to simulator the world and drone(s), open source flight control software such as ArduPilot and PX4, and flight planner software QGroundControl. These work best with a Linux operating system thus requiring installation of Ubuntu in order to work effectively with the simulator tools and components.

Allocation of Tasks

As mentioned, the project is broken into two main challenges and problems. These are simulator tasks and the optimal path for the drone delivery and as a result the team is broken into two to reflect this. There are two team members assigned to working on the algorithm design for the optimal path for drone delivery, in this they are required to research and understand the different factors that will affect the optimal path. The other four team members are assigned to work on simulator tasks involving the simulated world and factors which may affect the optimal path, such as weather conditions, battery life, and obstacles. The members allotted simulator tasks will also have to work closely with the algorithm design team as the factors which affect battery life are imperative to the core functionality of the algorithm.

Management of Tasks

Through weekly team meetings, issues of currently assigned tasks are spoken of and dealt with considering the resources at hand. If ever there was a obstacle preventing further progression in one team's area, it was common to re-allocate tasks to distribute the work load more effectively. These scrums provided many benefits as they allowed the team to progress at a rapid rate whilst keeping track of resources such as time and provided insight into roadblocks and immediate team progression. This transparency and visibility aided the team in accurately identifying issues and predicting how things will go as the project progressed and improved the overall quality of the work produced.

4.2 Evidence of Collaboration

Evidence of collaboration is shown through the meeting minutes of client, team, and tutor meetings, the slack channel, and project status reports which are all bound by the group contract. The links for each of these evidence can be found in the appendices.

4.3 Allocation of Roles, Potential Risks, Constraints

Our group decide to keep fixed roles instead of rotating every week as we believe that stable roles will make us familiar with them more quickly and result in a higher efficiency. the rough distribution is shown below, while at the same time, we may take multiple roles according to each week's process and requirements.

Tracker	Manager	Customer	Programmer	Tester	Doomsayer
Jimmy	Dylan	Nicholas	William	Ivan	Justin

Although it seems to be perfect with the distribution, potential risks may also exist. Some of them may include:

- As each of us acts a specific role, if one fails to catch up with the progress, it may influence the work of the others. For example, if programmer couldn't push the new code on time, tester would have nothing to test.

- If one of the members fails to complete his work, the whole project may also be affected. The examples should be more than that manager is unable to manage the team or project progress well, the tracker forgets to track the meetings, and so on.
- Some of the roles may be overlapping work to be done, where team members will be doing repetitive tasks in XP terms, which is not optimal for the progress of the project.

Realized of risks above, we make a set of constraints to ensure the project development.

- We signed group contract at the beginning of the project and promised to fairly contribute the project. If one fails to come up with the process, the others will provide necessary help to both project staffs and things that bothers.
- We will stay in regular contact in daily base to ensure everyone catches up planned process. If one has difficult with specific work, he can point that out and the others will also help with that. If the problem is big, an additional team meeting may be operated to discuss and solve it.
- At the beginning of each week, a short meeting will be hold to communicate the work of following week. If one work is related to multiple roles, they may divide that into parts or decide to work together.

4.4 Use of Tools

The use of bitbucket and slack have been extensive thus far. The designated bitbucket master maintained the project wiki. This involved; regularly uploading the minutes relating to client, tutor, and team meetings, providing pdf versions of said minutes for extra readability, creating and updating links to provide easy access to important reports, videos, and information, and uploading all relevant documents to create a central point of access for all stakeholders. Also maintained was the source repository containing all information pertaining to the simulator and drone navigation. Viewers are welcomed with "readme" files describing the content within the repository. This included an installation guide, which all team members followed, ensure project reproducibility, documentation of pertinent programs, such as the charging station randomiser, and configuration files.

Another tool utilised was Jira. The bitbucket technology was used to track all relevant tasks through categorisation and assignment to team members. This allowed for improved team organisation, communication, and ability to meet deadlines more efficiently. These tasks were linked to commits to bitbucket which aided team members in tracking updates to the project and provided a basic timeline of commits allowing for enhanced project planning in regular team scrums.

The use of slack via team members was constant and consistent. As the main mode of communication, any and all project related information was discussed there. The use of separate text channels allowed for the important, or urgent, information, such as meeting times, to be prioritised and segregated. This feature proved extremely useful greatly improving team communication and thus overall organisation. Slack provided an easy method of communication with the client and related communities such as the PX4 slack work space. Utilising bitbucket integration through slack ensured that all commits were tracked in a separate, and easy to access location illustrating a timeline of commits. This timeline helped team members to stay on track and remain updated on other team member's progress. Additionally, this provided a simple system to update team members of current work being done by others.

Zoom was the primary mode of communication with the client. It was used for the majority of team, tutor, and client meetings. The screen sharing function welcomed methods of XP programming as members were able to collaborate and program/debug as one, watching a single screen.

The last tool commonly used was Discord. The streaming application allowed team members

to communicate with the other groups doing Capstone projects at the University of Sydney as there was a collective work space dedicated for those students which afforded new perspectives and help with debugging of common simulator issues.

4.5 Client Interaction

The team communicates and work with the client through a few platforms. We used email, discord and slack to raise issues and clarify project details. The scope and requirements document can be found here and this details the scope statement which has been confirmed by the client. Group interaction with the client can be reviewed in the slack channel as the majority of discussions are held there on the various text channels available. The email history with the client has been downloaded and uploaded here as pdf documents.

4.6 Reflections and Conclusions

This project has facilitated certain characteristics that has proven to display unique skills reflected by each member. This has has been advantageous in allocating tasks as each member of the team is more proficient and effective in a specific area of research and section of the project. As a result, the team was better equipped to organise priorities and assign work efforts. Perhaps the only deviation from the groups' processes that were initially set; is that team members became narrowly focused on selective issues causing a lack of communication, which of course, can be improved upon. Furthermore, it was difficult to manage user stories with a project of this nature, where it is very open ended and the team has freedom in exploring different options in development. This resulted in our inability to explicitly define user stories before starting the task. In addition, the user stories tend to be quite broad, take a lot of time to complete and needs multiple members working on the same story, which is against the XP principles. This may lead to repetitive work done, as some of the roles may clash with on and other. This also meant that test-driven development was very difficult to follow. Therefore, the communications among the team and the client are very crucial on the flow of the whole progress of the project.

References

[1] Random Points within Circles,
https://programming.guide/random-point-within-circle.html

[2] Generating Random values With a Given Distribution,
https://programming.guide/generate-random-value-with-distribution.html

[3] Cumulative Distribution Function,
https://en.wikipedia.org/wiki/Cumulative_distribution_function

[4] Software Design Pattern Classification,
https://en.wikipedia.org/wiki/Software_design_patternClassification_and_list

[5] Simulation Test Bed for Drone-Supported Logistics Systems,
https://dspace.mit.edu/bitstream/handle/1721.1/121281/McCunney_Cauwenberghe_2019.pdf?sequ

[6] Drones,
https://www.casa.gov.au/drones

[7] Getting Started with PX4,
https://docs.px4.io/master/en/getting_started/px4_basic_concepts.html

[8] Gazebo Simulation,
https://dev.px4.io/master/en/simulation/gazebo.html

[9] Github: Dronekit,
https://github.com/dronekit/dronekit-python

[10] QGroundControl,
https://docs.qgroundcontrol.com/master/en/index.html

[11] Dronekit,
https://dronekit.io/

5 Appendix

Wiki

See the wiki home page here.

Individual Report

See the wiki home page here.

User Stories

See a full list of user stories here.

Meeting Minutes

See a summary of all meeting minutes here.

Email History

See the collection of emails exchanged with the client here.