# ILP PROJECT REPORT

s1945293

December 3, 2021

**Abstract**

This project is to help the School of Informatics create an autonomous airborne drone service where students can place a lunch order for sandwiches and drinks and get these delivered to them at several locations on Main campus. The drone will travel to shops in the University Central Area and collect the lunch items in an order. It will then fly to a location chosen by the student to deliver these items to the person who placed the order.

However, there is a "no-fly zone" consisting of several buildings. The drone will therefore plan its routes so that it does not fly over any buildings in the no-fly zone. Furthermore, the drone only carry's one order at a time and the drone can only fly for a limited time before its battery will run out. Therefore, before it runs out of charge, it needs to be back at its start position so that it can be recharged.

## 1 Software Architecture

My application consists of six classes. These classes are 'LongLat', 'Menus', 'WhatThree-Words', 'Database', 'Drone', 'Delivery' as well as the 'App' class which acts as as a main gets the drone to work using the classes in the software. This is where you can change the date of the lunch-time order day that you want to deliver.

### 1.1 LongLat Class

The 'LongLat' class contains methods of all the basic functions about the drone. These functions are vital as they are used repeatedly in the other classes. It allows us to see the position of the drone on a map as it gives us the position of the drone in longitude and latitude coordinates. Furthermore it helps fill the deliveries table as it gives us the coordinates of the next position of the drone if it makes a move as well as showing us when the drone is hovering down to give a student an order. It also allows us to see the zone that the drone is confined to and distances between two points on a map. Lastly, it allows us to see if the drone is close to a point on the map which is important as the drone is allowed to have an error of 0.00015 when it is delivering an order, picking up from a shop or returning to Appleton tower.

## 1.2 Menus Class

The Menus class is all about the Menus Json file from the web server and being able to obtain and use the information within in. First the we parse the web server and read the menus in the Json file that is on the website. From there, we can calculate the delivery cost of an order by going into the Json file and looking at the price of a particular item. We then repeat this until we have the overall price of an order and add 50 pence to this price for the delivery cost of the order. It does the same for the shops but instead of looking up the cost of an item, it finds the shops that the drone needs to go to to collect all the items in an order and returns the locations of these shops.

## 1.3 WhatThreeWords Class

This class changes the location details of a shop that are of the form 'what three words' into a longitude and latitude. Again this is done through the web server as it holds all the words as well as their corresponding longitude and latitude coordinates. I thought it would be easier to make a separate class for this as it is more clear for the reader and it is used throughout many of the classes.

## 1.4 Database Class

This class does everything to do with the Derby database file given on the web server. It allows us to connect to this database and therefore read the tables already inside. These tables are called 'orders' and 'orderDetails'. From these tables we can collect all the information we need about a students order. From the order number, the date it has been ordered on, what the order needs to deliver to where it needs to be delivered to. This information is then used by all the other classes to know where the drone needs to go and when. This class also creates new tables called 'deliveries' and 'flightpath' before every lunch delivery service i.e. for every new day, the table is dropped and remade. Therefore, the tables start off empty and get filled by information throughout the process of a days lunch delivery service. The 'deliveries' table uses information from the other tables in the database. Then, the drone uses this table to create a flight path for the drone. For every move that the drone does on this flight path, the details get recorded and added to the 'flightpath' table. So by looking at this table, we can see where the drone went for each order during that specific day.

## 1.5 Drone Class

This is the biggest class in the app. It initialises the drone and collects all the information needed for the drone to move without actually moving it. Firstly it parses the geojson 'landmarks' and 'no-fly-zones' files on the web server. From the 'landmarks' file, we get the coordinates of the two landmarks so that we can use them in our flight path if necessary. In the 'no-fly-zone' file, we read the geojson map to get the 5 polygons which represent the buildings that we are not allowed to fly over.

Secondly, this is where we check whether a line between two points intersect with any of the 5 buildings. This is used in other methods in this class which find the flight

path for a days lunch delivery service. Whilst finding the flight path, the number of moves made by the drone is taken into account so that the drone is sure to be back at Appleton tower before it runs out of battery.

Lastly, this class writes a new geojson file into the ilp folder. This new file contains the all coordinates of the flight path of the drone which allows us to see it's trajectory when we plug it into the website geojson.io.

### 1.6 Delivery Class

The 'Delivery' class finds the optimal delivery order for all the orders needing to be delivered. It then calls functions from other classes to get the drone to move and deliver all the orders for that day. Hence in the App class we only call the Drone class to initialise the drone and then the delivery class does everything else.

## 2 My Algorithm

### 2.1 Website and Database

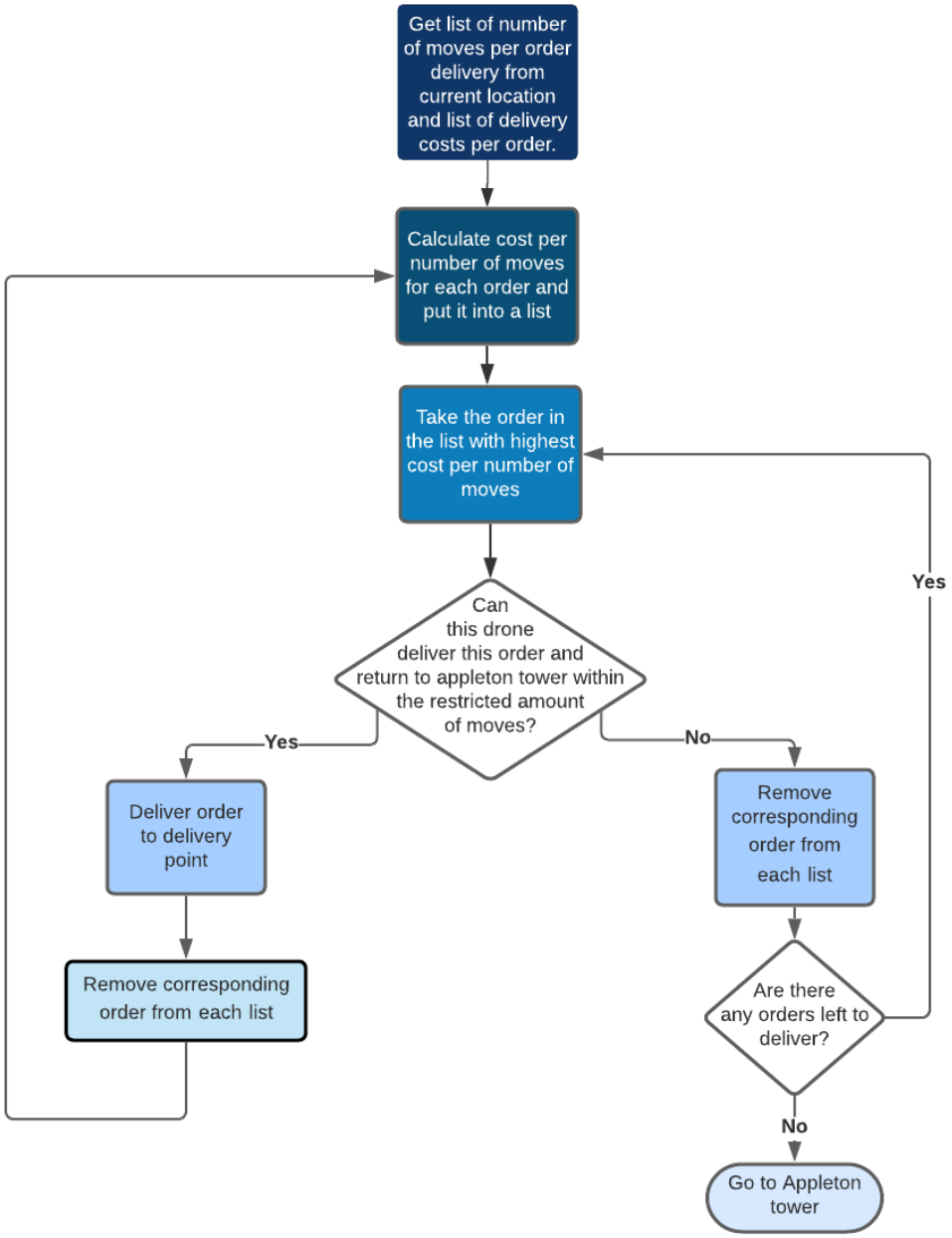
The first step that was taken to implement the drone service was to connect to the web server created by the school of Informatics. This contains all the data needed for the drone to provide lunchtime deliveries. Firstly, it contains all the confinement zones which allows us to tell the drone where and where not to go. Therefore, we can sure that the drone never fly's over certain buildings as well as not leaving the confinement zone. Secondly, it contains the Menu for each shop that the students are allowed to order from. This was useful to to get the prices of orders and to find where items that students had ordered where located. I.e which shops the items are from and the location of these shops.

### 2.2 Order of delivery

The drone's start point is Appleton tower. When the drone is given a set of orders for a day, it will filter through all the orders in the database table 'orders' to collect all the orders that need to be delivered for a specific date. In this table it collects the order numbers for this date as well as where the order needs to be delivered to. It will use 'orderDetails' table. and filter through all the order numbers collected in the previous 'orders' table and create a list of order items for each order. The delivery cost of each order is then calculated to obtain a list of delivery costs of each order. All these values get added to a new 'deliveries' table. We then calculate the cost per path of each order starting at the current location of the drone. This is so that we can get the most money whilst trying to use the least amount of moves. We check if there are enough moves to deliver the order as well as being able to return to Appleton tower. Even though it does not need to go back to Appleton tower after each delivery, we need to ensure that it will be able to get back before executing an order as if it delivers the order and is then unable to return before it runs out of battery, this will be an issue. If there are has enough moves to complete the order and return to AT, the order with the most optimal cost per path will be executed. This order will then be removed from the order list

delivery cost list made. If it does not have enough moves, we will check if the next order in the list does. We will repeat this process with the all orders left. The process will be repeated until all the orders have been delivered or we do not have enough moves left to deliver any more orders. This process is clearer in the flow chart below.

Delivery Order flow chart

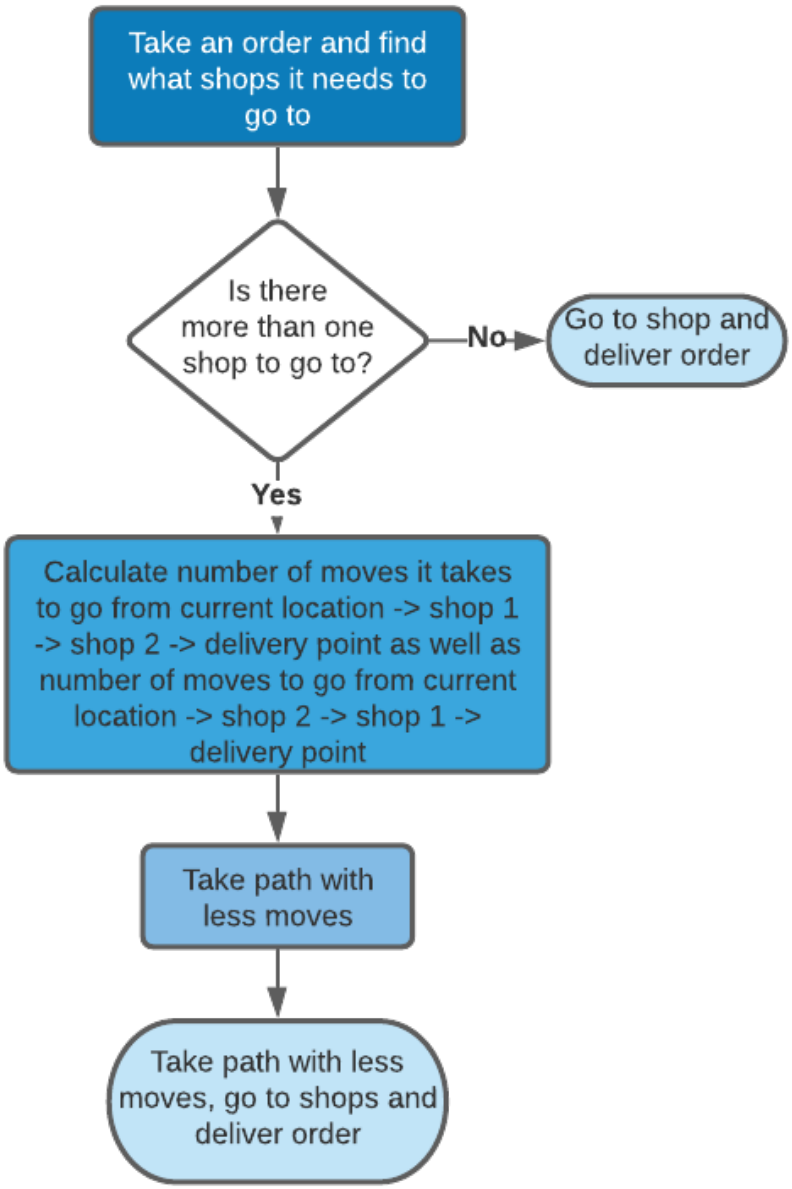



## 2.3 Finding overall path

To find the overall path of each order, the drone looks for the shortest path possible to complete the order. If there is only one shop to go to in the order, the drone takes the path to get to this shop directly and then delivers the order to the location specified by the student as there are no shorter paths. However, if there are two shops, it checks

the number of moves it would take to go to the first shop, followed by the second shop as well as the delivery point and vice versa. The path with the least moves is chosen and the drone delivers the order. This is repeated for every new order that the drone has to deliver.

Finding Path flow chart



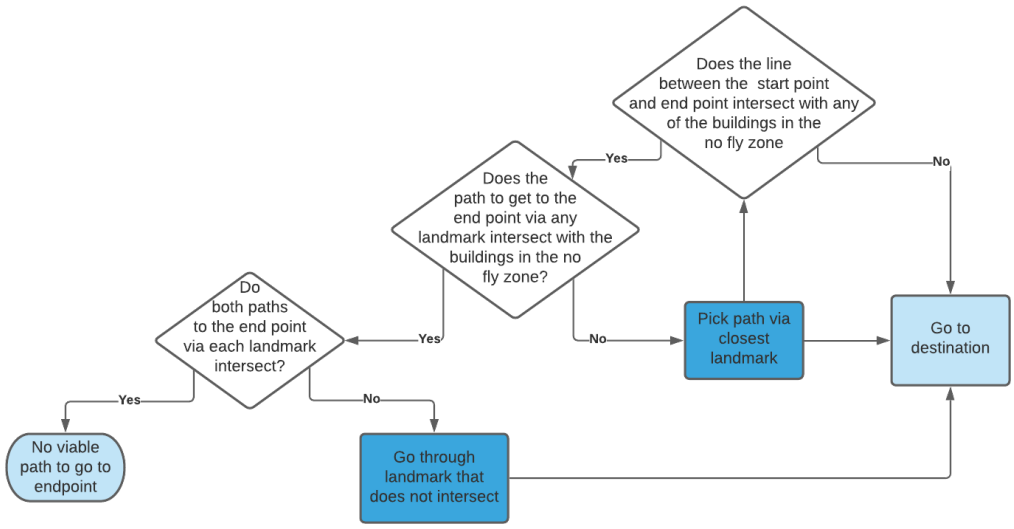

## 2.4 How to get to next point in path

To decide how to get to the next point in the overall path found for each delivery above, we first checks if a straight line between the current position of the drone and the next point it needs to go to (i.e. a shop, delivery point or Appleton tower) intersects any of the buildings in the no fly zones. If the line does not intersect, it takes this path to get

the the next point it needs to reach. If the line does intersect with any of these buildings, the drone will not take this path, it will go to a landmark. The drone will then check if the path to the landmarks intersect. If both paths to the landmarks intersect, there is no viable flightpath. However, this should not happen as the reason we have the landmarks, is to be able to go around the no fly zone. If they do not intersect, the drone will calculate the distance of the drones current location to each of these landmarks. It will then pick the closest landmark as the path it wants to take.

This repeats for every point that the drone has to go to to deliver an order. It is then repeated for each order that needs to be done that day.

How to get to next point in path flow chart

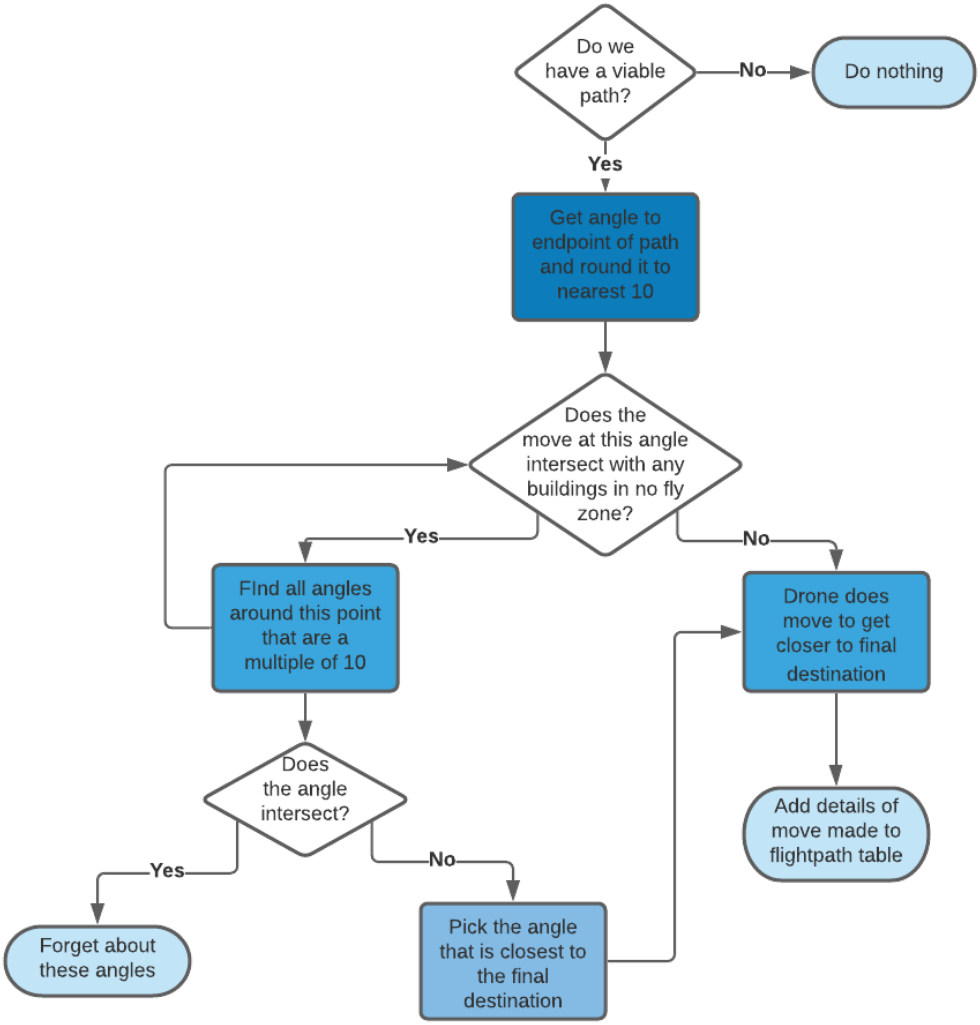



## 2.5 Moves

This section builds on the path we have found above and now showing how to execute going from one point to another in moves. The drone has to do several moves of length 0.0015. Since we already know the path we have to take, we simply take the angle between the current position and the next position we need to go to in the path. We round the angle to the nearest 10 since the moves are only allowed to move in 10 degree increments. Once the angle has been rounded, we can now check whether or not this singular move intersects with any of the buildings in the 'no fly zone'. If it does not, we can move the drone. If it does, we need to pick a new angle which is a multiple of 10 that does not intersect the buildings. To do this, we find all the angles around this point which are a multiple of 10 that do not intersect. We take the angle that is the closest to our destination. We then add the details of that move to the 'flightpath' table in our database.

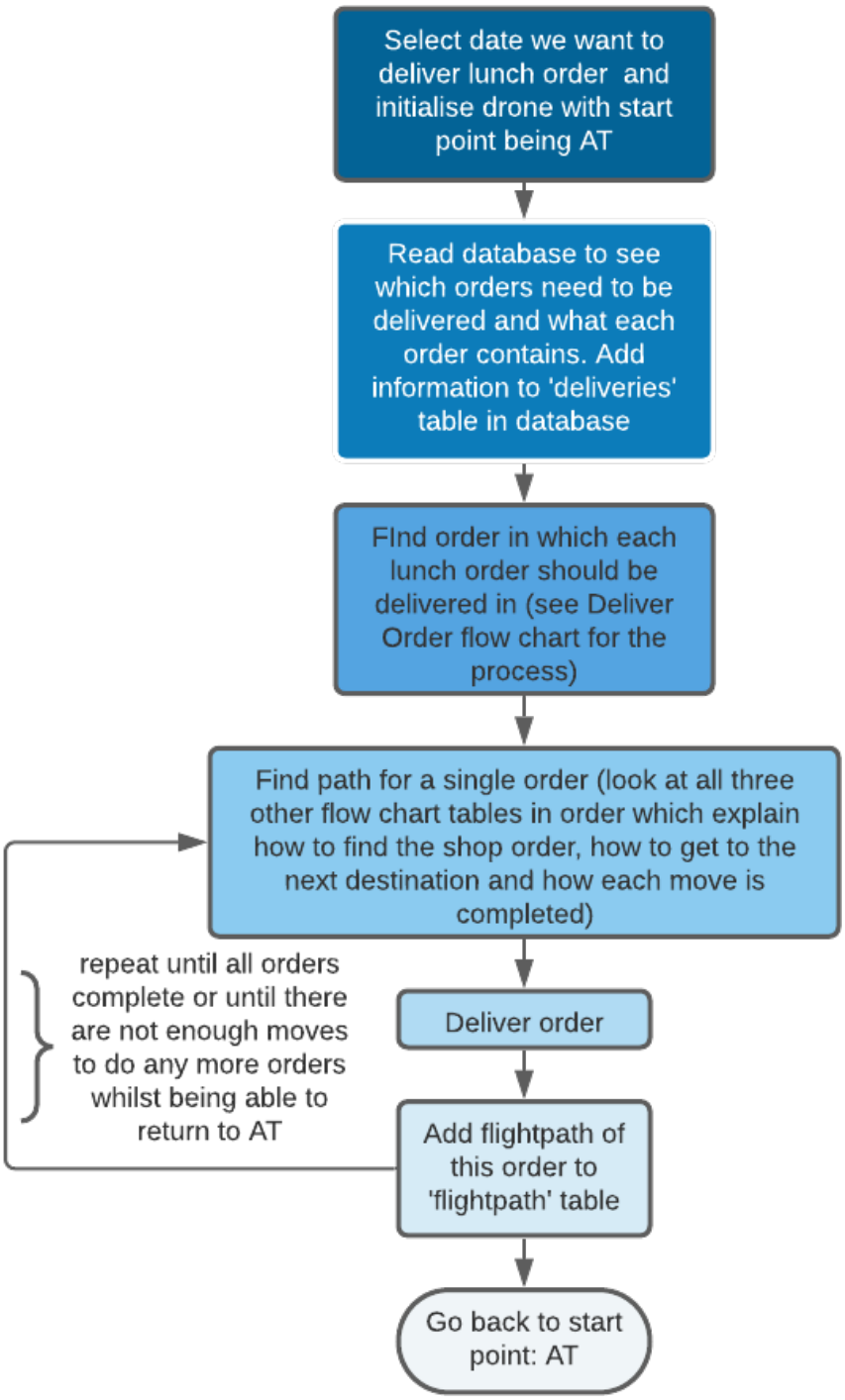
The flow chart below shows the process we take to make a singular move. This process is repeated until we have completed the path each point of every order.

Moves flow chart




## 2.6 Putting it all together

Overall, we put all the above components together to get the drone to complete the task at hand. Firstly, we collect all the information from the website and database. We then select a date for which we want the drone to deliver orders on. Then, we can initialise the drone and set it off from Appleton tower. The drone decides which order it should deliver each lunch order in and sets off to do so. For each order, it has several places to go before reaching the delivery point. It checks for intersections and whether or not it needs to go through a landmark to get to each point. The drone delivers orders until all the deliveries or done, or until there are not enough moves left to complete a delivery and get back to Appleton tower. At the end of each day's deliveries, we output the earnings made that day as well as the number of moves made by the drone. Again the flow chart below shows the process of the whole algorithm put together.

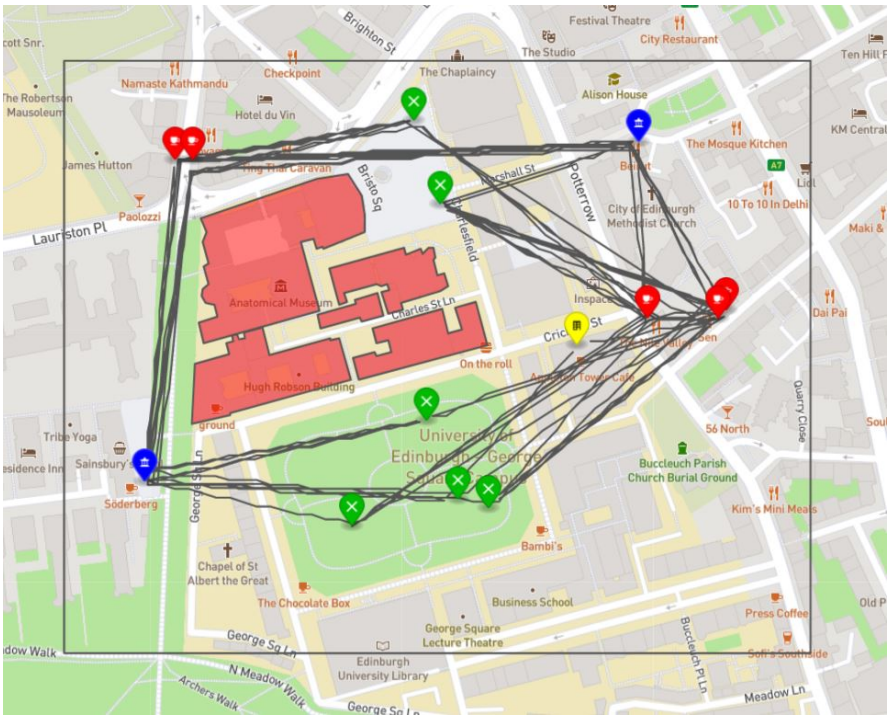



# 3 Results

## 3.1 Flight path for 29/11/2023

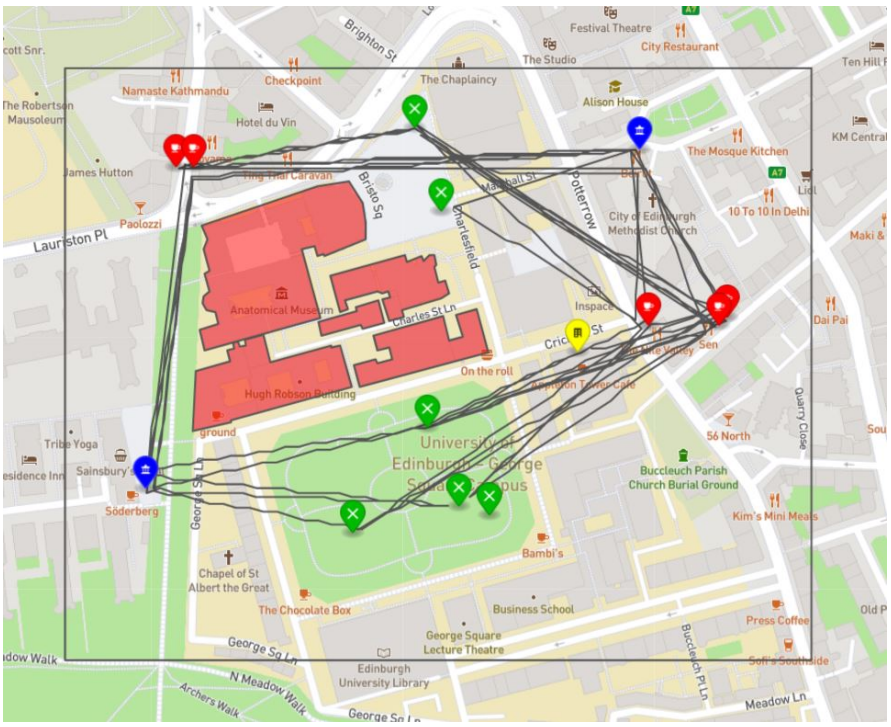
In this lunch delivery order day, the drone's earnings is 21175 pence (£211.75) and the number of moves made is 1440. There were 26 orders to be delivered on this day. The

drone managed to complete 24 of them before having to return to AT so that it does not run out of charge. Here is it's flight path:




## 3.2 Flight path for 27/12/2022

In this lunch delivery order day, the drone's earnings is 11030 pence (£110.30) and the number of moves made is 900. There were 15 orders to be delivered on this day. The drone completed all of them. Here is it's flight path:



# 4 Ideas I had but did not use

- Instead of using the flightpath algorithm explained above, I could have used the well known A-star algorithm. This would make the drone use less moves as it would get the drone to go around buildings and directly to the end point instead of using landmarks. This would allow the drone to get more orders done and ensure that more students obtain there lunches. However, I did not use these as I found the landmarks were important to keep the drone from going anywhere near the no fly zone. This will keep the schools in the University Central Area happy as they will not be worried about any potential crashes if the drone were to pass extremely close to any building.

- I could have used a convex hull to avoid flight paths from going into the hall ways in between the polygons. However, I did not do this as if the no fly zone was to be changed so that it could be used by another drone in a different area with a different no fly zone, there might be a more direct flight path possible. If the polygons were to be made more spread out, the drone could fly in between the polygons without having the issue of tight space and no direction possible.

# 5 Issues and Improvements

If the landmarks were to be removed, there would not always be a viable flightpath and the algorithm would not work anymore. To improve my algorithm, I could add in my own landmarks that always allow a flight path to be possible to make sure this error does not occur. However, if the program takes has a landmark file, then whoever is maintaining the code should understand that if the current landmarks do not do the job then they should add new ones that allow the drone to do this.