Automated Drone Delivery Report

EWAN BLOOR
S1948228

Section 1

Software Architecture Description

The software architecture approach was chosen to follow a few key principles, decoupling, encapsulation, and abstraction. Following these would be a good indicator for a successful architecture. For instance, prioritising decoupling, and increase the readability as such, allowing for more expansive and clear coding from future developers, and expanded ideas to be implemented in a sustainable way with the blueprint already in place. This project lends itself to that due to the confined nature of its current state, with the ability of expansion in the future being clear, and code maintenance almost certainly being required by future developers, producing a less interwoven system that can be adapted and changed with much fewer knock-on effects in different areas of the code. High quality encapsulation is also important in a similar vein, when thinking of future developers, allowing them a level of safety when restricting access and providing clears bundles of data. Abstraction is also a key aspect I choose to try to implement here. Allowing coding methods and classes to be completely individual and hiding their complexity. Inheritance is less of a principle I have chosen to put to the side when considering this architecture as there are mixed reports on its usefulness and likely an implied detriment to encapsulation.[1]

To start with my demonstration on how I implemented these key principles are the simplest building blocks of my code structure, the classes: Database and Website.

These classes are simple and efficient, used to collect data from both the database and the website respectively. They are used to exclusively work with the website and database directly. These are the only classes that this has been done. Producing highly cohesive classes where each method within each has one task and one task alone, with the most complex method in website simply converting a piece of data retrieved into a more easily usable value. This allows for any future developer to easily understand and use these methods and classes in the future and even expand upon them. If for instance an expansion upon the database was required, the blueprint of how and where this must be done is clear and easy to follow meaning ideally a better following of these key concepts in the future.

The two classes however do require the use of two parsing classes, Shop, and What3Words. These are used exclusively for parsing and hence don't have huge implications on the architecture however due to the values returned there were an optional two more passing classes that could have been created: Item, LngLat. However, for the sake of reducing improving abstraction, these classes are nested within their respective Gson parsing classes. This allows a more readable code design. And with the use of a static class method, meaning a nested class can exist without the class, this reduces association and hence improves a key attribute, decoupling.

Next, we will proceed to look at likely the most used class within the entire project, LongLat. This is our representation of a location in longitude (x) and latitude (y). This class is how many values from aforementioned classes are stored (e.g getLandmarks which produces all landmarks for the system to use). The decision to produce this class allows for me to construct a highly abstracted design with which allows many future required values to be defined and operations to be performed on them.

This is a completely discrete class providing all the required methods upon which a LongLat object can be operated. This allows for high level of object orientation and provides an important steppingstone to defining the algorithmic functions further down the line.

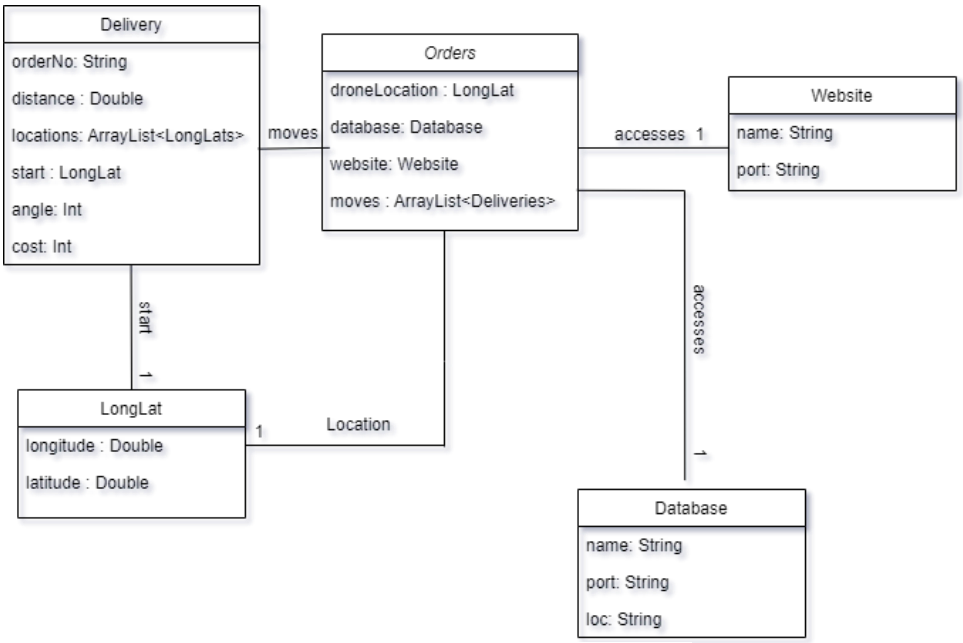


Figure 1, UML class diagram of select classes

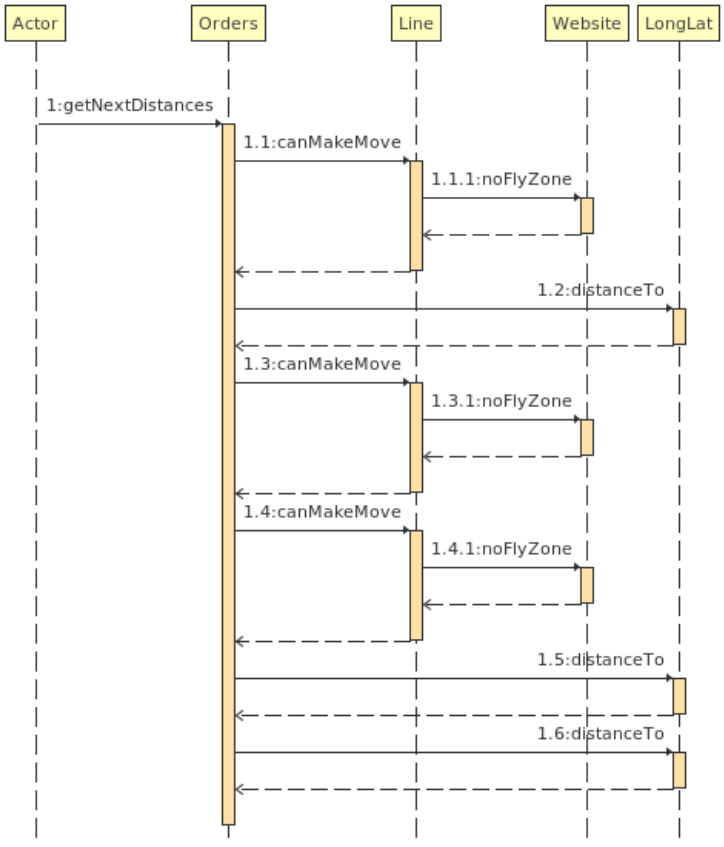


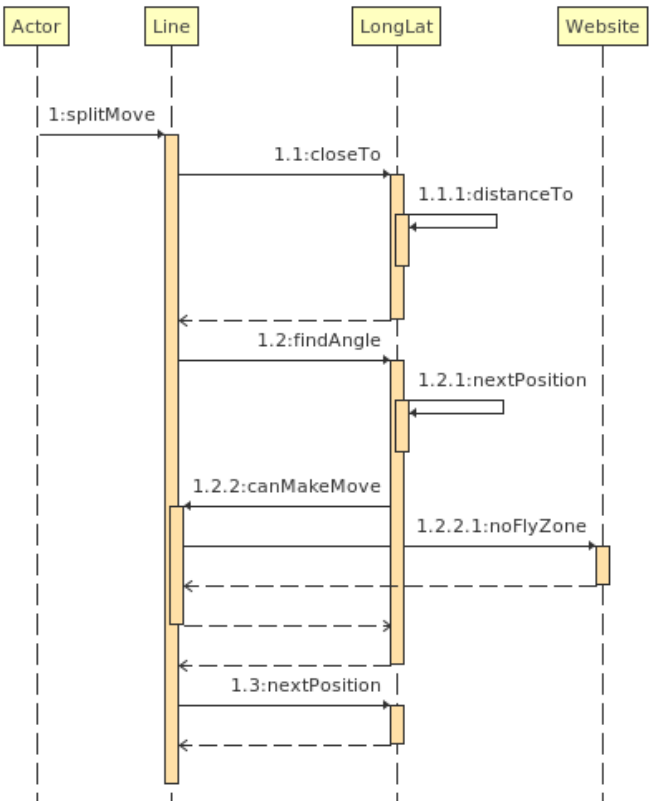
Figure 2 "getNextDistance Sequence Diagram

A class which Is key to the final product is that of Orders, providing a strong algorithm to produce the best orders for the day given the date from the website, this begins to bring together and provide the link between all the aforementioned classes. As shown by figure 1. It brings together elements of the classes and provides a strong link that is simple and means that one command will return a required result, this however does have its negatives. As can be seen by figure 2 and figure 3, within each single use case, there is not an emphasis on decoupling, with many elements required from many different classes, this can sadly lead to one small change within any of the constructor arguments requiring the code to be almost drastically altered[2]. However, within its current state, this is not too large of a problem to worry about, and with already reduced decoupling by using my aforementioned approach I have ideally not made maintenance of this system beyond control. Nevertheless the positive to the use of this class method allows for a high level of abstraction. With all functions, except one, being set to private this also means that there is much improved encapsulation and maintenance.

With a simply “press play and it runs” function, it allows the code to be simply used, with only one argument required and the website and database ports and the starting location of the drone, this highly abstracted function produces a clear result. This result is in the form of the class Deliveries. This class is used similarly to the previously mentioned, LongLat, storing objects and providing methods to obtain them, using a high level of encapsulation. In this case it is used to store our Orders so that they can be used further down the line and without compromising on the cohesion of the class. This method of using a separate class for this task improves decoupling as no one change will likely affect more than just a few small aspects of the Orders class and no change to any class or method used further on will likely affect Orders either. This means that maintenance on the project will be easier and when we wish to use the values returned in Deliveries to populate a database, there will have no problem expanding it if required to change what it is populated with.

The final class unidentified is that of Line, used to do all the work on the coordinates turned into lines, from one LongLat object to another. Keeping these functions here, even if there are only currently two, creates good abstraction as these functions have a clear purpose along with each only performing one task. For instance, we can see in figure 3 here we make good use of other classes, allowing us to provide key data for out database without compromising too much on our aim of decoupling data.

Figure 3, “splitMove” Sequence Diagram



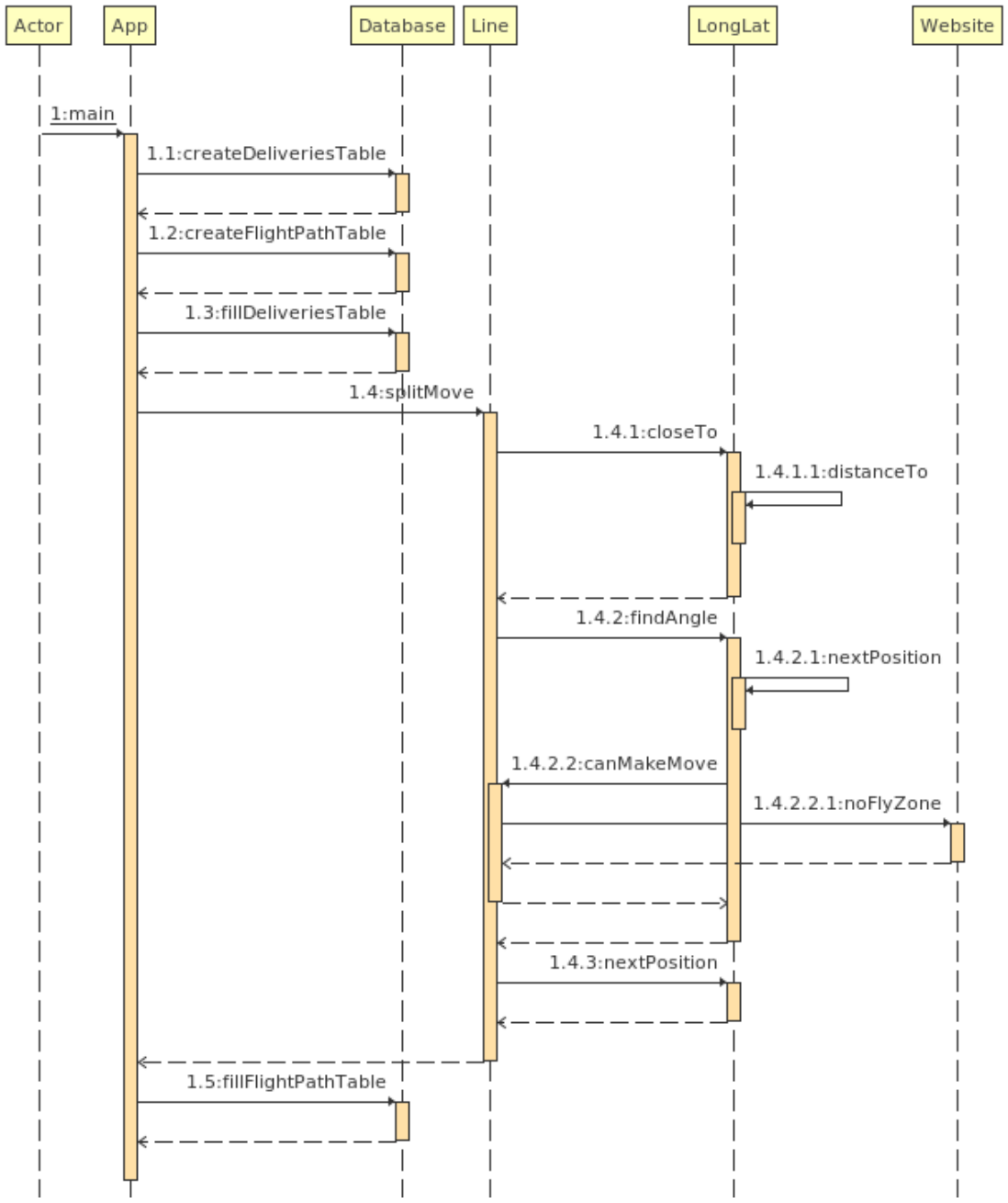


Figure 4, "Main app" Sequence Diagram

Now it is important to look at the purpose of bringing this all together, looking at figure 4 and figure 1 demonstrates well the key aspects outlined and their attempted following of this within our approach. Calling this function is the purpose of this project, and it is important that throughout we have provided a level of maintainability suitable by following the key principles I have laid out. When running the main function, it demonstrates a high level of decoupling. Throughout I have kept methods and constructors private where possible, I have provided a high level of encapsulation and attempted to keep a system with a reduced level of abstraction to allow for small changes to be made without effecting large part of how the system operates. This allows for future developers to have a significantly easier time when working with this code.

Section 2

Drone Control Algorithm

To produce the best possible algorithm there must be two conflicting aims, ideal route calculated and runtime. With a time constrain of 60 seconds to run the code and a total of 1500 moves it is important to find as small of a compromise in both as possible.

This algorithm is a variation on a greedy best first search algorithm[3].

First, I collate all the deliveries required to make within a day, using the order numbers. Then I access each order placed, one at a time. For each order I get the shops it must travel to to collect the items. For each of these shops the drone must visit I calculated the distance it will take the drone to get there from the drones starting location.

This is done by checking to see if it can directly fly there. To do this a line is created (from the drone to its next location), and it is checked to see if it intersects with any of the lines of any of the polygons defining the no fly zones. If it does not intersect, the drone flies directly there. However, if it does intersect (hence can't fly directly from its location to a shop) we then locate all the landmarks gathered (landmarks provided and midpoints of the confinement zone and the center of the confinement zone) and see if the drone can move to them and if from them it can then directly move to the shop, all the while checking if it crosses the no-fly zone doing these moves. All these options for the drone to get from its current location to its next delivery/pick up point are collated as possible routes, and the route that's the shortest distance is selected. Hence, we now have a route from the drone to a shop it must go to complete the order number.

This process is then repeated from the shop it has found the shortest route to, to the next destination it must travel to complete the order - if it has only one shop in the order then it goes to the drop off point, else it goes to the next shop in the order. This process is repeated until a route has been created. If, however there are two shops in the order, as we have just performed this with one of the shops being the first destination, we must now perform it with the other shop as the first destination. Hence the whole process is repeated again but with the other shop as the first place the drone must move. This will leave two route options, one with the drone going to one shop first then completing the delivery, the other route being the drone going to the other shop first; the smallest distance of these values is selected. This whole process will create a drone route for one order number, and this route method, using the landmarks, will more often than not produce an ideal route for this one drone move.

This process is then repeated for the following orders on that day. Producing a list of all the best possible options for each order numbers on that day from the drones current starting location.

For each of these orders we gather the total delivery cost of each, and with this we perform a ratio of each, orders distance/monetary value. This allows us to maximise the amount of money we make per move on average as we will now select the smallest value as that is the best one to perform for this first delivery. This is the delivery we perform first.

Overall, the smallest distance for each delivery to the final destination is calculated and divided by its monetary value. The best one of those is then selected as the best route and added to our list of routes.

This order is now completed so is removed from the list; the current drone position is now wherever the previous package was delivered to. This whole process is performed again with the updated list and from this new location; until no order numbers are left. This produces a list of orders which the drone will perform, from the drones first starting location to its final drop off point, significantly a list of lines that the drone must follow to perform all the orders going via landmarks if necessary. The total distance of this drone move is collected, and compared to the total distance the drone can travel, if this total distance is greater than a predetermined value signifying when the drone must go home, the final order is removed off the end, if the distance travelled is still too great another order from the end of the list is removed, this is repeated until the total distance the drone must travel to complete these orders is under the predetermined value. Then a final move from the drone's current location back to Appleton Tower is added on. This is now the final list of deliveries upon which we will base our moves. From best route first monetarily to the worst.

To get the exact moves I gather this list and split it up into the lines, following each line I calculate the angle of it from east and plot my course using multiples of 10 degrees as my angle until I am close too the end of the line, when I am, I check if the end of the line is a landmark. If it is not a landmark it is subsequently hovering. It then will follow the next line in the list and repeatedly until all lines are completed and it returns to Appleton.

If at any point during this, it attempts to cross into the confinement zone the angle is changed and attempted again to avoid errors in the calculation when performing a close to.

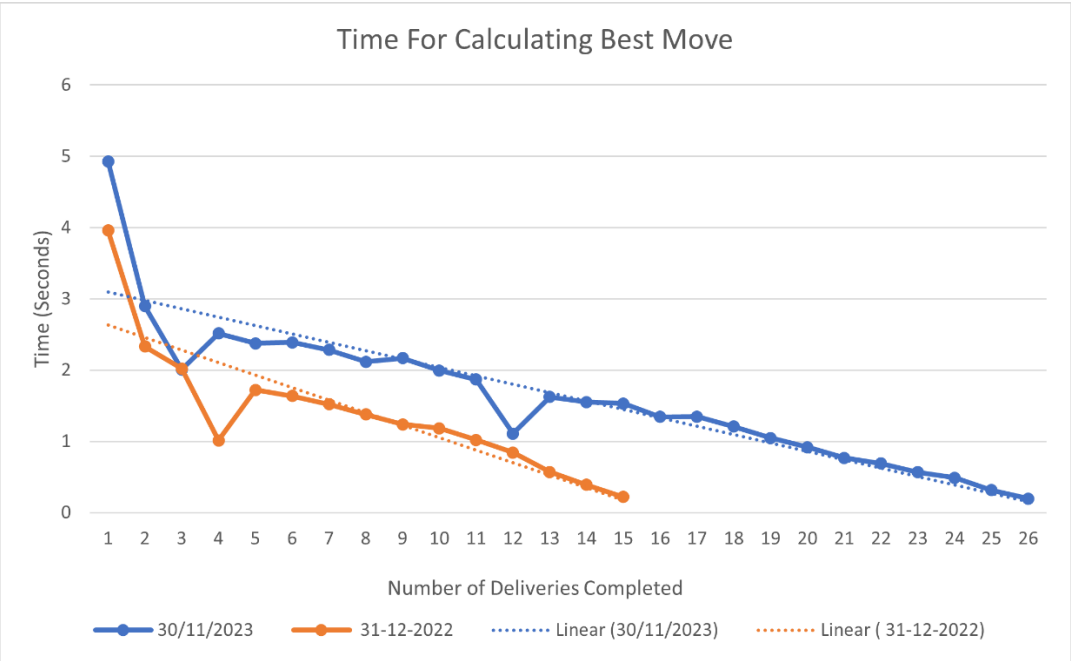


Figure 5: Time Of Running "setNextMove" for Each Iteration

Significantly this algorithm does not differ much from a greedy best first search algorithm, picking the best value every time with no regard for what comes next. This does ideally produce a very strong result with most nearly every delivery being completed, however not producing a perfect algorithm to save computation time and power. Nevertheless, it does have a very significant time issue. As shown by figure 5 we can see that the time it takes to complete the algorithm is significantly longer the more stops there are to calculate. This algorithm also slows down significantly if there are too many landmarks - the cut off point for 60 seconds seems to be 8 landmarks when running on 6gb of memory and a 3.6GHz CPU hence only 7 landmarks are used as a cut off with a little room provided for deliveries to be performed without going over the limit. These are time constraints that most likely are a limiting factor when running on a small system however when scaling up, this algorithm would likely perform well given a larger memory and higher CPU frequency. But considering this if this project is expanded and the processing restraints are less restricting due to this, it may be important to consider a more thorough greedy algorithm that doesn't just consider what directly the next route should be but looks further ahead by at least one or 2 more moves, this would mean the algorithm would be slower however produce a higher quality resulting route.

The results of this algorithm I have chosen are shown in figure 6 and 7.

Within these figures you can see clearly demonstrated a priority in performing the shorter distance orders, as can be seen in the bottom right of the graph where the 3 shops are located having a high density of the shop locations. You can also see that a landmark generated by the midpoint of the top line of the confinement zone is heavily utilised whilst traversing the map, reducing the overall distance travelled by the drone on each move required to get to the top left of the figures. Overall, no lines approach the confinement zone and the drone never enters even close to the no-fly zone.

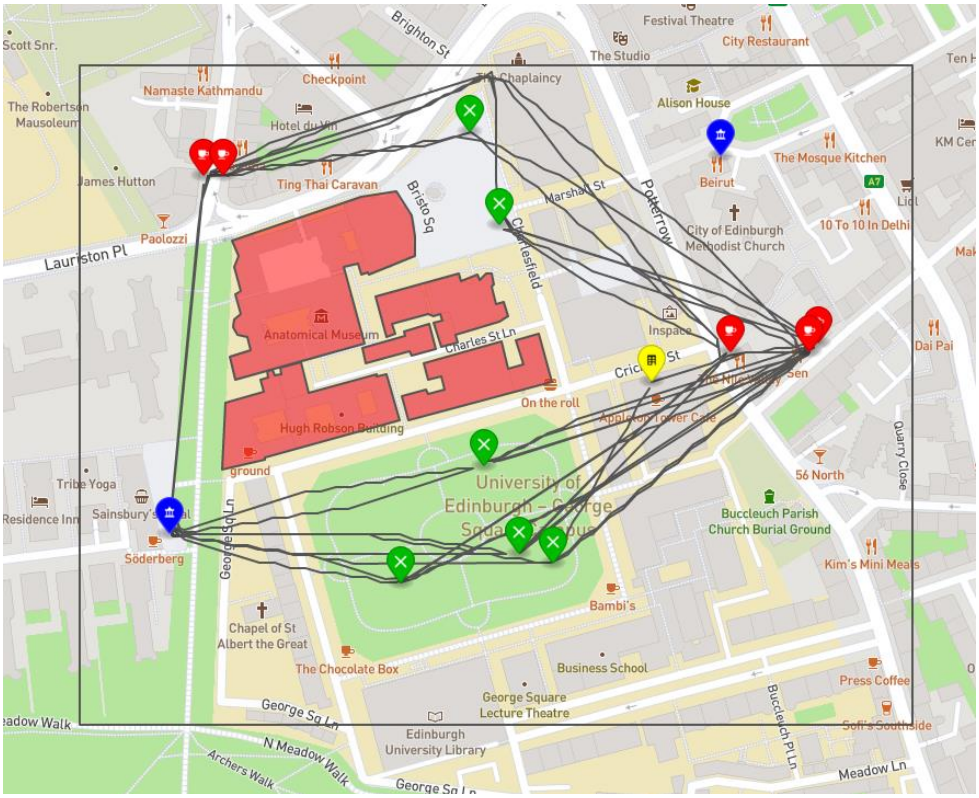


Figure 6: 30-11-2023 (25 deliveries completed) Flightpath (Time Taken = 46.8 seconds)

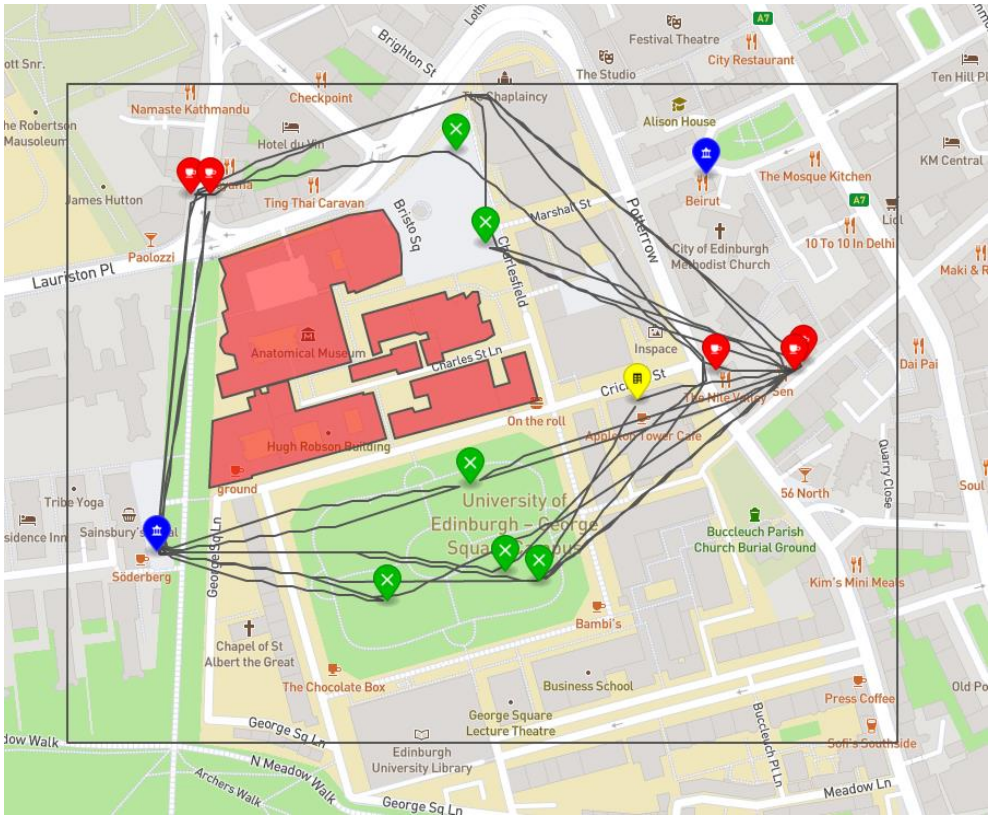


Figure 7 31-12-2022 (15 deliveries completed) Flightpath (Time Taken - 25.7)

Appendices

1. https://dl.acm.org/doi/abs/10.1145/28697.28702
2. http://www.coldewey.com/publikationen/Decoupling.1.1.PDF
3. https://www.cs.cmu.edu/afs/cs/project/jair/pub/volume28/coles07a-html/node11.html#modifiedbestfs