**Informatics Large Practical Report**

**Section 1: Software Architecture Description**

**Section 2: Drone Control Algorithm**

The Algorithm employed by the drone is a greedy approach.

**Pre-Flightpath sorting**

Firstly all the valid orders made on a certain day are sorted, in ascending order, by distance from the starting position of Appleton Tower to the restaurant the pizzas are to be picked up from. This means the orders that are initially closest and would require fewer drone moves are prioritised and delivered first, to greedily increase the number of orders made by the drone within the limited number of moves.

**One-way flightpath generation**

Once the orders have initially been sorted, then the algorithm iterates through all of the valid orders, in this sorted arrangement. For each order, the drone generates a part of the flightpath only going one way; from the initial starting point, which will be close to Appleton Tower, to the restaurant the pizzas in the order have been ordered from.

The algorithm generates this section of the flightpath on an individual move by move basis, adding the move to the partial flightpath before generating the next move. The algorithm calculates the next move by firstly iterating through all possible angles in which the drone could move. This is all the angles between 0 and 337.5 degrees, in increments of 22.5 degrees, moving clockwise, where an angle of 0 degrees means moving straight north.

Then this move is checked to see if it is legal, which involves checking 3 separate criteria: intersection with the no-fly zone, leaving the central area, and repetition of moves.

**No-Fly Zone Intersection**

A drone move is only valid if it does not cause the path of the drone to intersect the perimeter of any of the areas within the no-fly zone. We check this by firstly considering each polygon within the no-fly zone. We then split this polygon up into the vertices in the corners of the polygon and then form a set of the line segments formed between each adjacent pair of these vertices. This set is then the set of straight lines each along the perimeter of the polygon.

We then consider whether the line segment between the position of the drone before and after the move intersects any of the line segments in the set we have obtained from our polygon.

A solution to this line-line intersection problem is calculated by viewing the two line-segments in terms of Bezier parameters.[[1]](#footnote-1) Firstly we represent the line segment between the two points and as

And the second line segment between the two points and as

We then calculate where along each line segment the intersection point of the two lines will be by:

We note that if the denominator of these two terms is 0 then these values will not exist, and so these two lines are parallel and will never intersect, therefore we check the value of the denominator is non-zero first. Then if the denominator is non-zero we calculate each of these values. If they are both between 0 and 1 inclusive then we note that the two line segments do intersect each other. Otherwise the two line segments will only intersect each other if at least one of them is extended past its start or end point, hence our two finite line segments defined between 2 points do not intersect.

This solution is calculated for each line segment formed by our polygon. If none of these line segments intersection then the drone move stays away form this part of the no-fly zone. We then check this for every polygon within the no-fly zone. If none of these polygons are intersected, the move is valid in terms of the no-fly zone.

**Staying within the central area**

The drone is not allowed to move in and out of the central area during the delivery of an order. To stop this the algorithm does not consider moves that would result in the drone leaving and re-entering the central area. As the algorithm uses extensive backtracking, the departure the central area is considered from the reverse viewpoint – if the drone has left the central area on its way to the restaurant the drone may not then re-enter the central area as backtracking this path would result in leaving and re-entering the central area on the return journey. So only move pairs where the following logical statement is true: if the drone’s position before the move is not in the central area, this implies the drone’s position after the move is not in the central area.

**Move repetition**

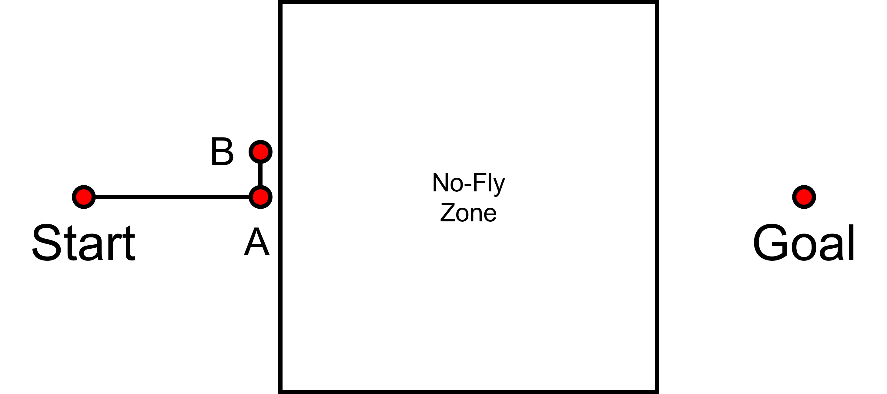
A drone move is considered invalid if it has been repeated on the way to a specific goal. This is because allowing repeated moves may lead to the drone being stuck at the edge of a no-fly zone. We can see this in Figure 1. Once the drone enters point A, it cannot then go through the no-fly zone, so it will enter point B. However once at point B, the point closest to the goal in terms of Euclidean distance will then be point A again, as going around the no-fly zone will temporarily increase the Euclidean distance between the position of the drone and its goal, in comparison to point A. So the drone will return to point A. We can then see that the drone will simply oscillate between points A and B and will never reach the goal, as the drone is required to go around the no-fly zone.

Figure 1: Drone gets stuck when allowing repetition

Thus, we avoid repeating moves. To implement this, at the drone’s starting position we initialise a list of points visited by the drone. Then after every move made by the drone, we add this point to the list of visited points. For every move, we discard any moves that would result in the drone re-entering a point already visited, and only considering moves that would enter new points. So our flightpath from Appleton Tower to the restaurant will include no repeated positions. We also note that when we backtrack and reconstruct the path to return to Appleton Tower we obviously allow positions to be repeated from the original flightpath to the restaurant, and observe that this section of the flightpath when returning will also have no repeated points. Then when we have returned to Appleton Tower the list of repeated points will be cleared, as we only consider moves to be repeated if they are made whilst attempting to pick-up the same order.

Then after this, the path to the restaurant is reconstructed in reverse order, with the angle of the move reversed by adding 180 degrees modulo 360. This subsection of the flightpath is then added to the drone’s overall flightpath, and the order would be delivered on that day if the number of moves performed by the drone would not exceed 2000. Otherwise, that subsection of the flightpath is discarded, and not performed by the drone, the order is marked as valid but not delivered and the algorithm then moves onto the next order to see if it is possible to deliver.

1. https://en.wikipedia.org/wiki/Line%E2%80%93line\_intersection#Given\_two\_points\_on\_each\_line\_segment [↑](#footnote-ref-1)