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, each of which is detailed further below.

If the move is indeed legal, we then check to see if this move is an improvement on the current best move. We do this by storing the value of the current best move at that position and the Euclidean distance between the position after that move and the end goal, which is initialised to be something far greater than the largest feasible value for any move, for example, 1000 >> 2000 * 0.00015. If the move we are considering is an improvement on this distance we then select that move as a new best move. After all legal moves have been considered, we then put that move into the partial flightpath for the current order and consider the best move from the new position.

Backtracking

This process is then repeated until the drone is close to its goal of the restaurant, and the drone hovers at the restaurant to pick up the order. After this point the original section of the flightpath to this point is reconstructed in reverse order, towards Appleton Tower. This process is done by iterating through all moves in the flightpath, in reverse order, then taking the angle of that move add 180 (modulo 360) and then calculating the position of the drone after this new move. We then add this reversed move to the partial flightpath. As this is backtracking the drone's flightpath, we find that the position of the drone after this new move is almost identical to the position of the drone on the path to the restaurant, except for some small rounding error due to floating point arithmetic, which is insignificant when the co-ordinates are changed from a double to a float.

Once this backtracking process is finished, the drone will have returned to Appleton Tower where again the drone will hover to drop off the order. After this we have obtained a complete section of the flightpath for the specific order. We consider if adding this section to the current flightpath already obtained would cause the drone to have made over 2000 moves on the specified day. If this is the case, we discord this section of the flightpath and do not deliver the order, otherwise we add it to the full flightpath of the drone and that order is delivered. We then mark the order as valid but not delivered, or delivered accordingly and then repeat this process for the next order.

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. Firstly we represent the line segment between the two points (x_1, y_1) and (x_2, y_2) as

$$L_1 = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + t \begin{pmatrix} x_2 - x_1 \\ y_2 - y_1 \end{pmatrix}$$

And the second line segment between the two points (x_3, y_3) and (x_4, y_4) as

$$L_2 = \begin{pmatrix} x_3 \\ y_3 \end{pmatrix} + u \begin{pmatrix} x_4 - x_3 \\ y_4 - y_3 \end{pmatrix}$$

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

$$t = \frac{(x_1 - x_3)(y_3 - y_4) - (y_1 - y_3)(x_3 - x_4)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4)}$$

$$u = \frac{(x_1 - x_3)(y_1 - y_2) - (y_1 - y_3)(x_1 - x_2)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4)}$$

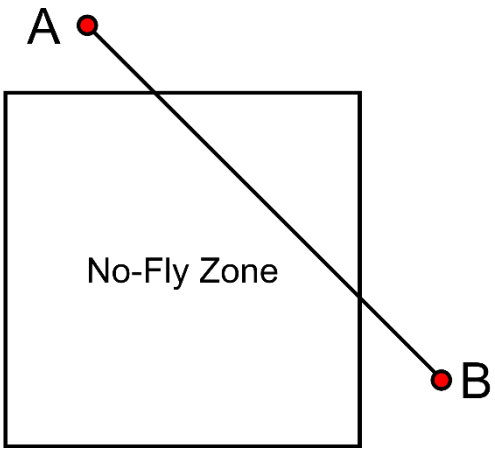

Figure 1: Bad approach - end points not in the No-Fly Zone, but the drone still goes through it

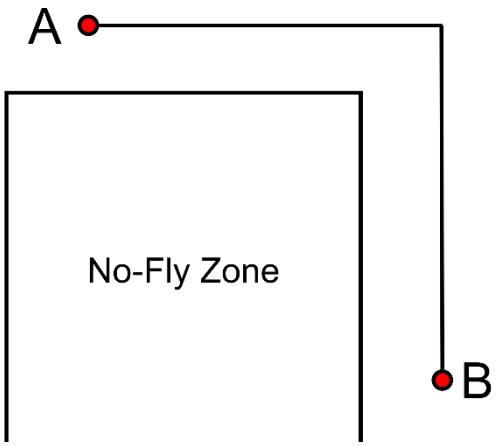

Figure 2: Good approach - Drone fully avoids the No-Fly Zone

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.

Before this approach was taken for intersecting the no-fly zone an alternative approach was taken where we simply consider whether the end point of the line segment formed by a move is in the no-fly zone. However it was noted that this approach was sometimes incorrect as it would allow moves which enter and then exit the no-fly zone, as shown in Figure 1: Bad approach - end points not in the No-Fly Zone, but the drone still goes through it. So we use the line-line intersection approach which, although less efficient, correctly identifies paths which fully avoid the no-fly zone, as shown in Figure 2: Good approach - Drone fully avoids 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 3: Drone gets stuck when allowing repetition. 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.

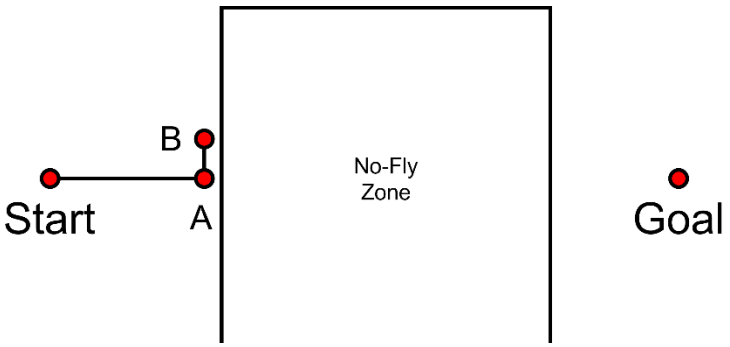

Figure 3: Drone gets stuck when allowing repetition

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.

Improvements

One improvement that could be made to the algorithm to improve optimality of the drone's flightpath could be to consider the reversed approach to the flightpath algorithm as well. That is, firstly consider the flightpath that is created by the greedy move-by-move algorithm starting from the restaurant and travelling towards Appleton Tower and then backtrack the path from Appleton Tower to the restaurant. Then swap the first and second halves of the flightpath, to create a new flightpath that may be different to the one created by starting at Appleton Tower. Then the drone would take whichever of the two flightpaths is the shortest. This new approach taking the best of 2 paths would mean the result of the flightpath would be at worst the same as the original approach, and potentially better producing a shorter flightpath which would allow more orders to be delivered.

A major flaw with this greedy approach is that it is not guaranteed to be an optimal solution. Seeing as the flightpath must go around no-fly zones the algorithm could be drawn to moving in a direction where the drone originally moves closer to the goal but must then take a longer route overall, compared to a path where the drone initially moves away from its goal but then takes a more direct path. To this end a provably optimal approach such as the A* algorithm was considered, however this approach has a significant drawback in terms of memory required. As there are potentially 16 possible moves at each point (including repetitions) and paths to and from a restaurant will very often over 100 moves long the number of nodes that would need to be stored in memory was deemed infeasible as the memory requirement for A* is $O(b^d)$, where b is the average number of moves available at a

point and d is the depth of moves in a solution. However some form of graph pre-processing before any moves are calculated or an implementation of a memory bounded algorithm, such as SMA* [1] might allow for a path to be created where there is a guarantee of optimality.

References

[1] S. Russell and P. Norvig, "Artificial Intelligence: A Modern Approach," 2009, p. 101.

[2] S. Russell and P. Norvig, "Artificial Intelligence: A Modern Approach 3rd Edition," 2009, p. 98.

[3] Wikipedia, "Line-Line Intersection," 21 11 2022. [Online]. Available: https://en.wikipedia.org/wiki/Line%E2%80%93line_intersection#Given_two_points_on_each_line_segment.