ASE 479W: Path Planning

Victor Branea

March 24th, 2023

1. Introduction

This laboratory assignment focuses on developing the code to find the most efficient path between two points, knowing the map. Additionally, the path is smoothened in order to make it flyable by the quad using polynomial smoothing.

2. Results

Depth-First Search algorithm

As can be seen in figure 1, the depth-first search algorithm found a path between the two given points. The grid used is one of the provided grids: "test_grid_medium". The found path is clearly not the most efficient, it is 29 nodes long as opposed to the most optimal path of 12 nodes. This is however to be expected, as the DFS algorithm is only guaranteed to find a path between two nodes if the path exists, it makes no guarantees with respect to the length of the path.

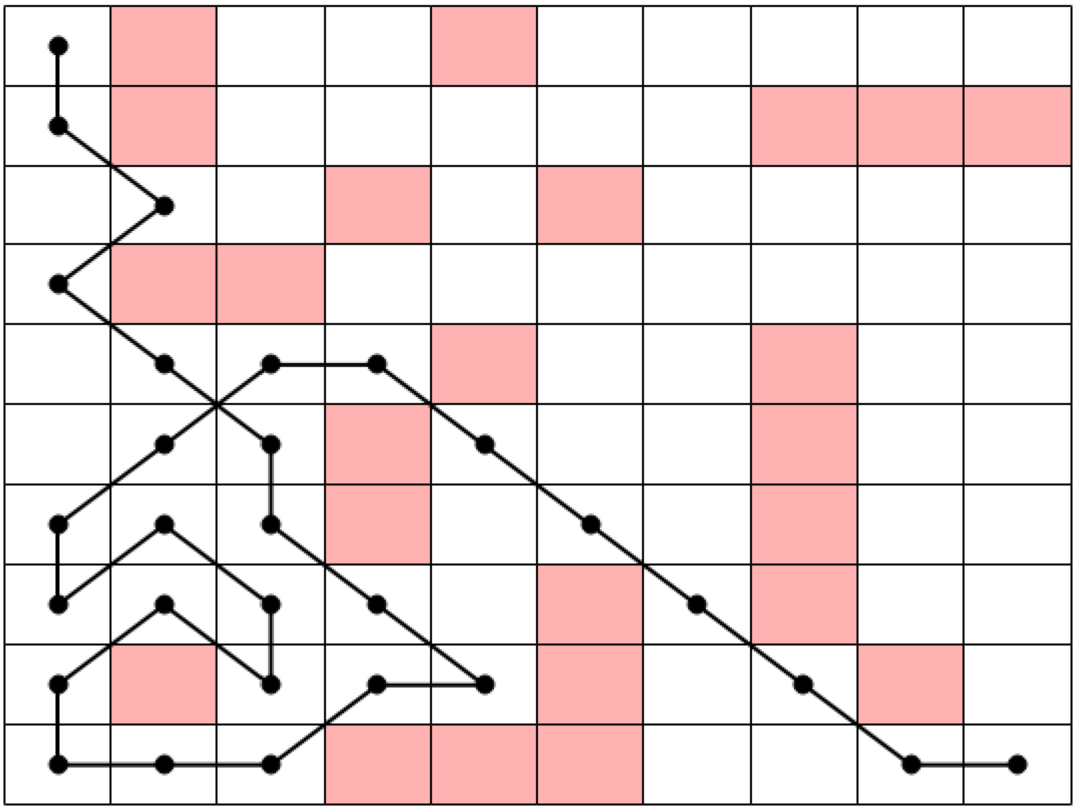

Figure 1 DFS for test_grid_medium starting at (0,0) and ending at (9,9)

Dijkstra’s algorithm

Dijkstra’s algorithm was also run using the same grid that was used for the DFS algorithm depicted in figure 1, “test_grid_medium”. The resulting path can be seen in figure 2 and is guaranteed to be the shortest possible path between the start and end points.

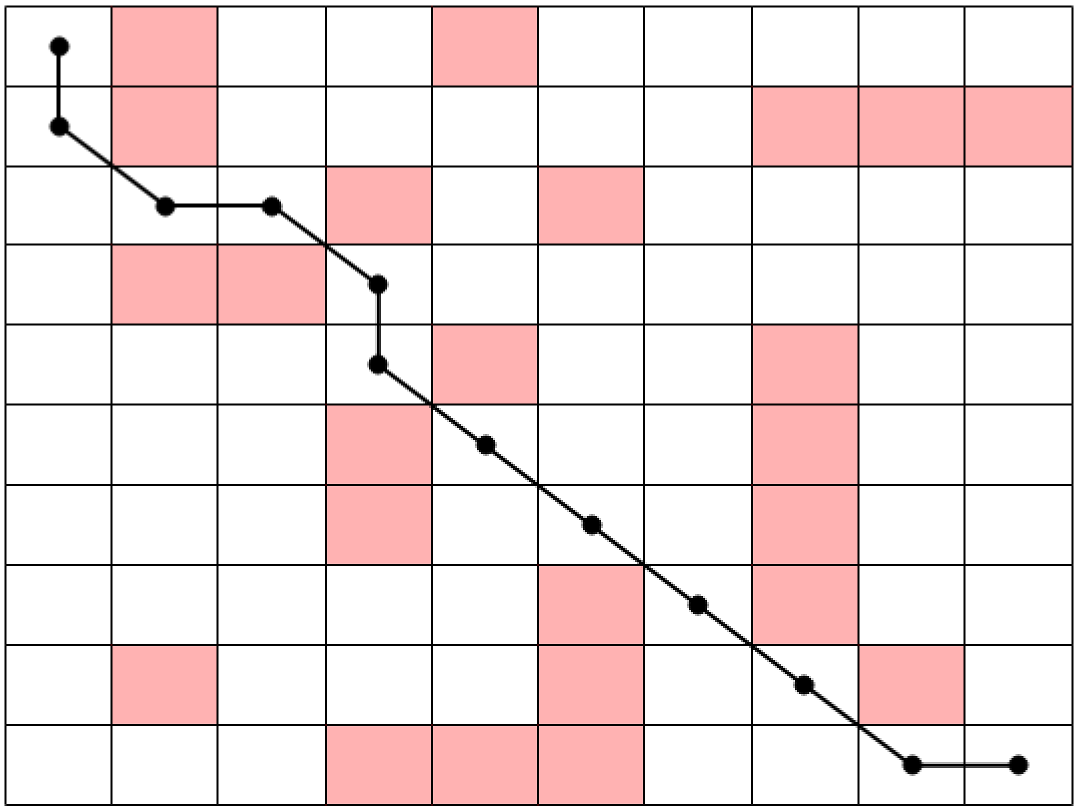

Figure 2 Dijkstra's for test_grid_medium starting at (0,0) and ending at (9,9)

Dijkstra’s method finds the shortest path, which is in this case, compared to DFS, a whole lot shorter at 12 nodes compared to the 29 nodes path found by DFS. However, in order to find this path, Dijkstra’s had to explore 74 nodes compared to the 30 nodes explored by DFS. This directly translates to the run time of both algorithms, where Dijkstra’s runs for 178 µs, which is around 7 times longer than DFS (which runs for 24 µs). This data can be easily visualised in the table in figure 3 below.

Search algorithm	Depth-First Search	Dijkstra
# of nodes explored	30	74
# of nodes in path	29	12
Run time	24 µs	178 µs

Figure 3 Comparison between Depth-Fisrt Search and Dijksra's algorithm

A* algorithm

For a heuristic of the A* algorithm to be admissible, it must always be smaller or equal to the true cost of reaching the end node from the current node. If a heuristic is as close as possible to the true cost it will make the algorithm more efficient than if it is just arbitrarily smaller. At the same time, as can be seen in the data tabulated in figure 4, a heuristic that can potentially be larger than the true cost will make the algorithm less efficient by making it explore additional nodes. In the case of the inadmissible heuristic analysed in figure 4, this happens because the algorithm will prioritise nodes that are further from the end point, causing the exploration of additional nodes.

Heuristic	$\sqrt{\Delta x^2 + \Delta y^2}$	$\Delta x + \Delta y$
# of nodes explored	20	75
# of nodes in path	12	12
Cost of path	13.8995	13.8995
Run time	29 µs	439 µs

Figure 4 Comparison between an admissible heuristic and an inadmissible heuristic for the A algorithm*

If the heuristic in the A* algorithm is set to 0, it can be noticed that the algorithm becomes exactly the same as Dijkstra’s algorithm. This, as the data from figure 5 shows, is visible due to the identical path and number of nodes explored. The only difference in this case is that Dijkstra’s algorithm has a slightly shorter run time. However, when running the algorithms multiple times, the run time for Dijkstra’s and A* fluctuated a bit, so I wouldn’t feel comfortable stating that this difference is due to the small additional step of handling the 0 valued heuristic.

Search algorithm	Dijkstra’s	A* with heuristic = 0
# of nodes explored	74	74
# of nodes in path	12	12
Cost of path	13.8995	13.8995
Run time	178 µs	191 µs

Figure 5 Comparison of A with heuristic = 0 to Dijkstra's*

As the drone additionally requires the velocity, acceleration, and yaw time histories to be known in order to fly, the A* algorithm could be modified to also generate the velocity and acceleration time histories. This could be done by defining the velocity as $\frac{\Delta x}{\Delta t}$ and the acceleration as $\frac{\Delta x}{2(\Delta t)^2}$. However, this is not done in practice because the path outputted by A* is not the path flown in practice, due to it not being smooth enough for the drone to fly. This is why polynomial smoothing is done on the output from A*, as will be discussed later.

Polynomial smoothing

Below, in figures 6, 7, 8, 9, and 10, 5 different paths are shown aiming to complete the path (0,0) -> (1,0) -> (1,1) -> (0,1) -> (0,0). The difference between the 5 paths is the order of the derivative whose difference is being minimised at these waypoints. This causes the path to be increasingly smoother as the order of the minimised derivative increases. This increase in smoothness ensures that the quad will be able to track the given path as closely as possible. Minimising the 4th derivative already changes the path significantly less than the change seen between the smaller order derivatives, which leads to the conclusion that there is not much to be gained from minimising a higher derivative than 4 in this laboratory's case.

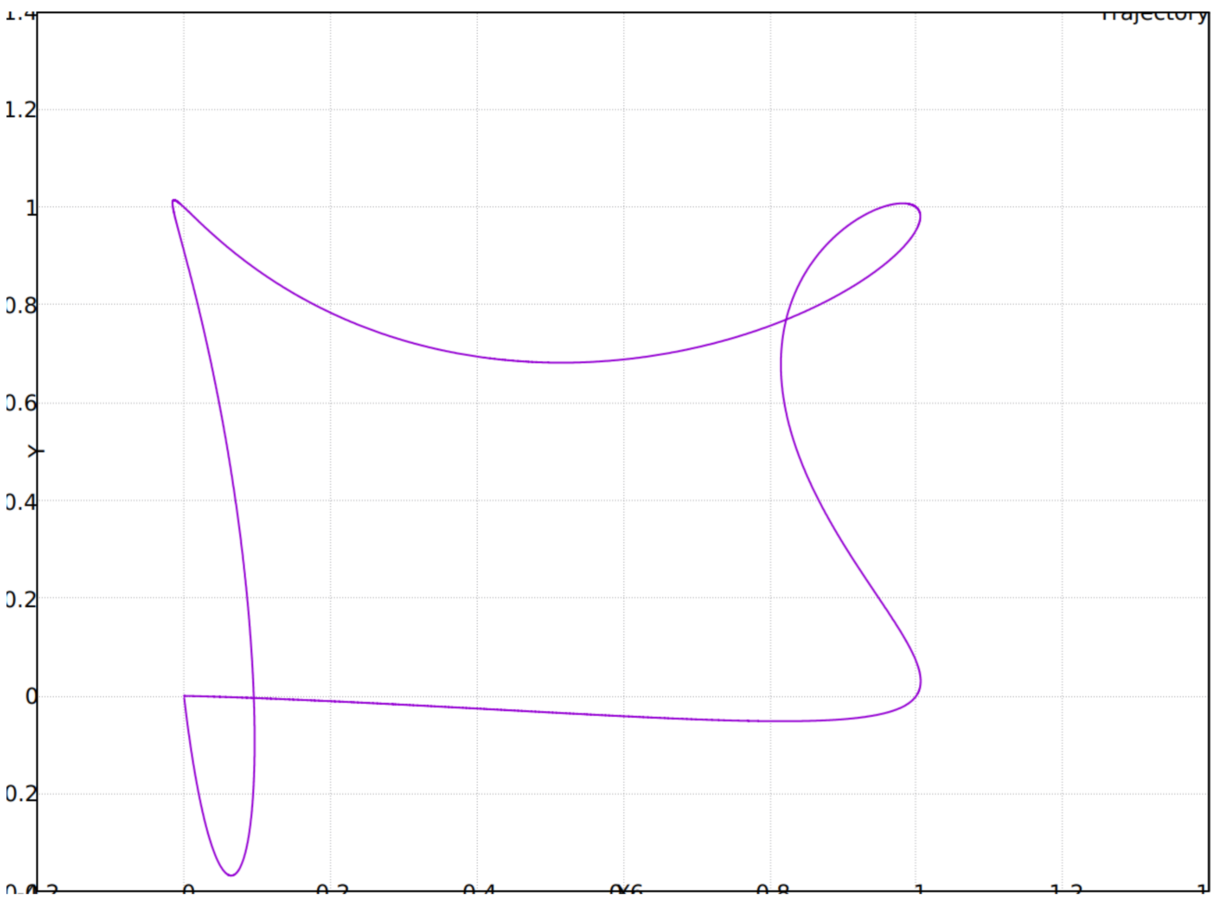

Figure 6 Square path minimising 0th derivative

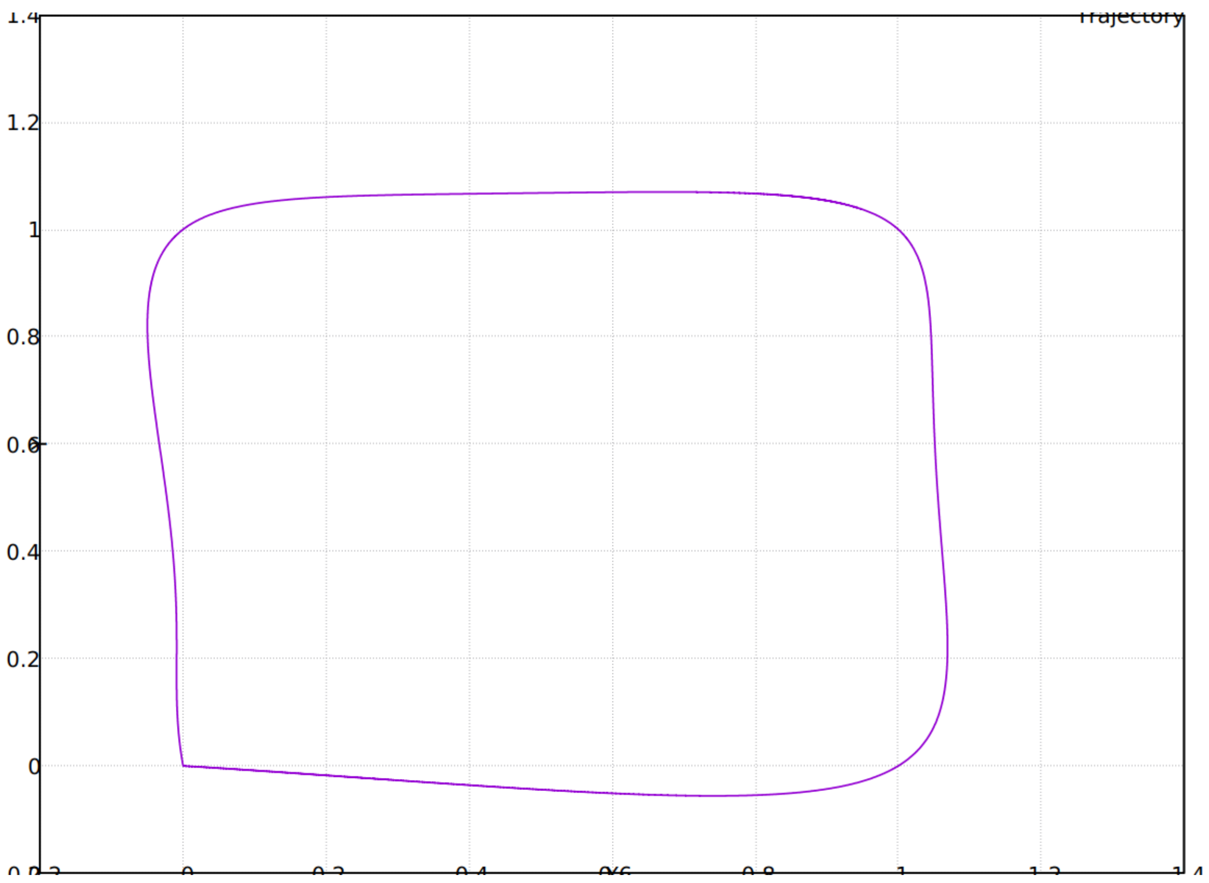

Figure 7 Square path minimising 1st derivative

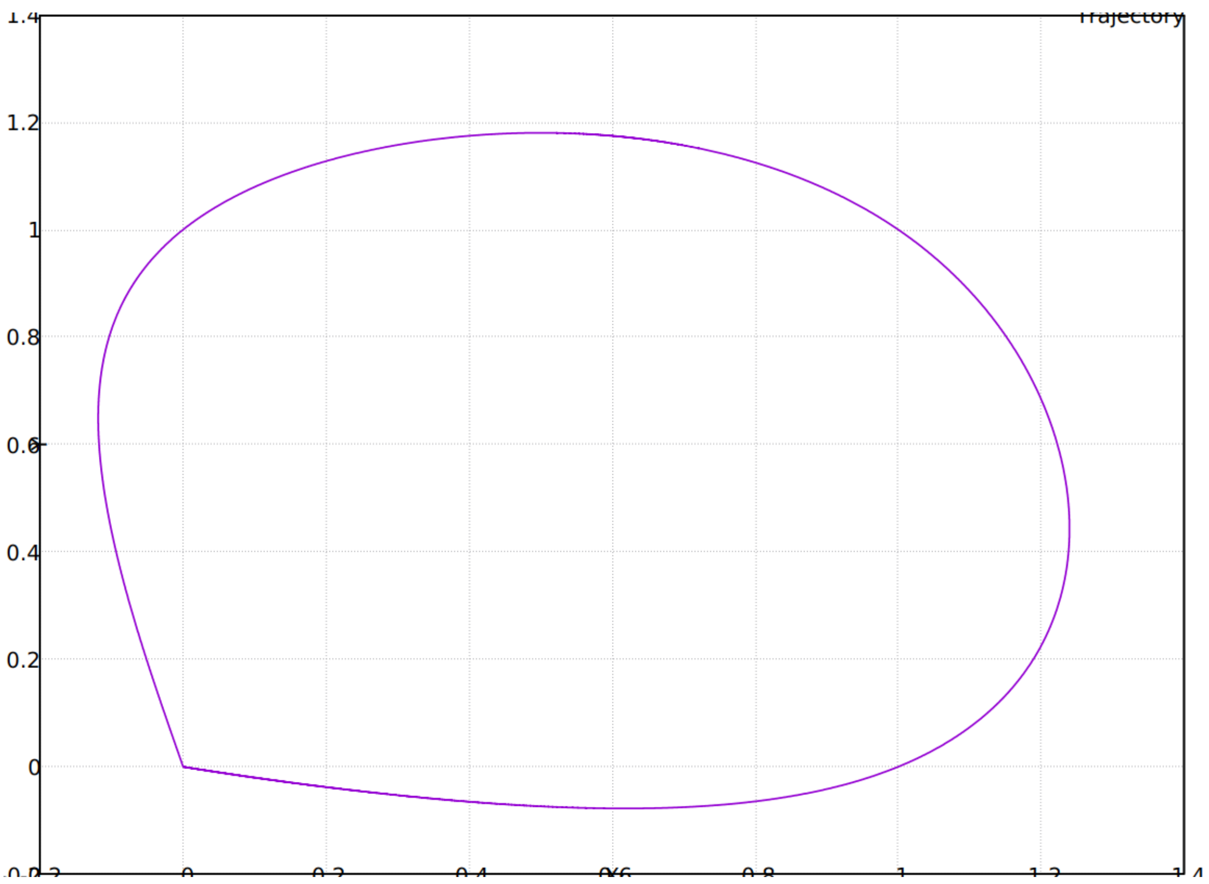

Figure 8 Square path minimising 2nd derivative

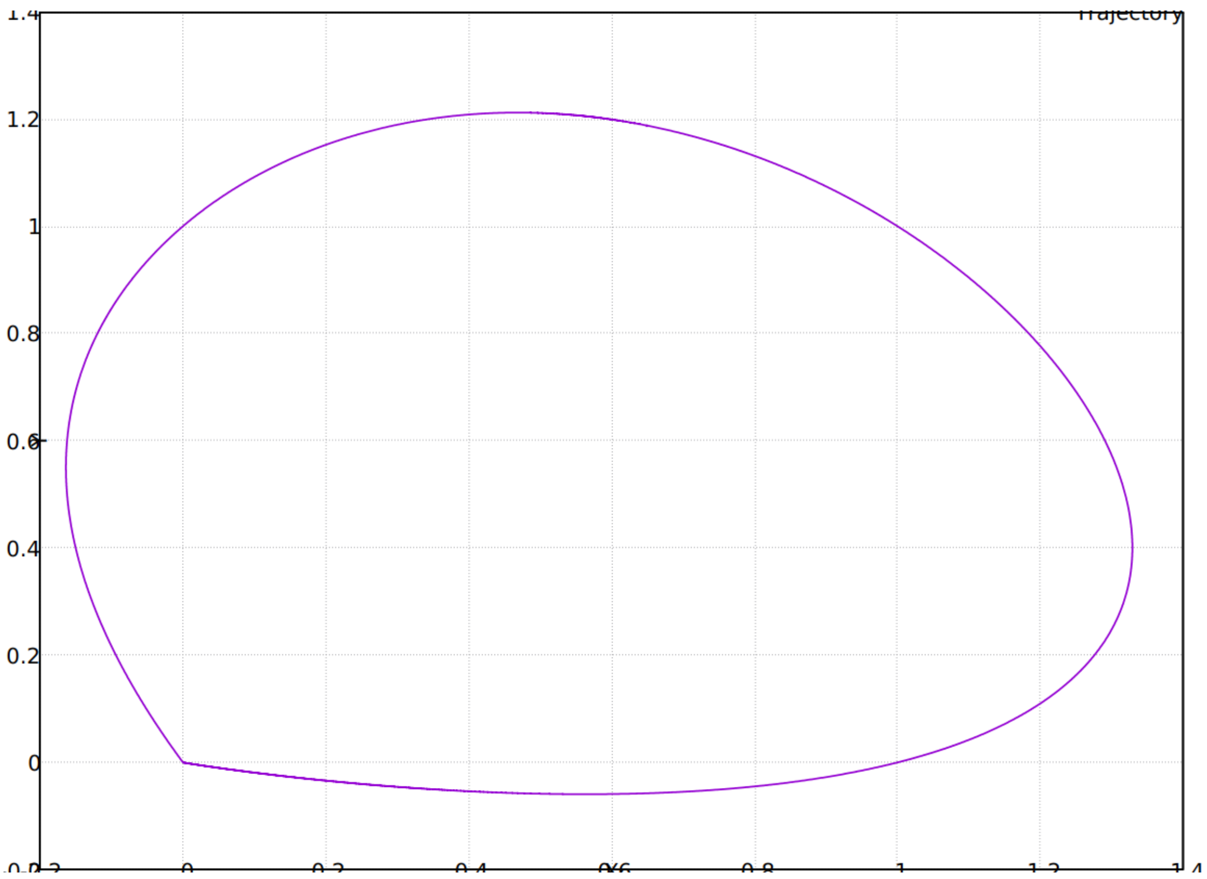

Figure 9 Square path minimising 3rd derivative

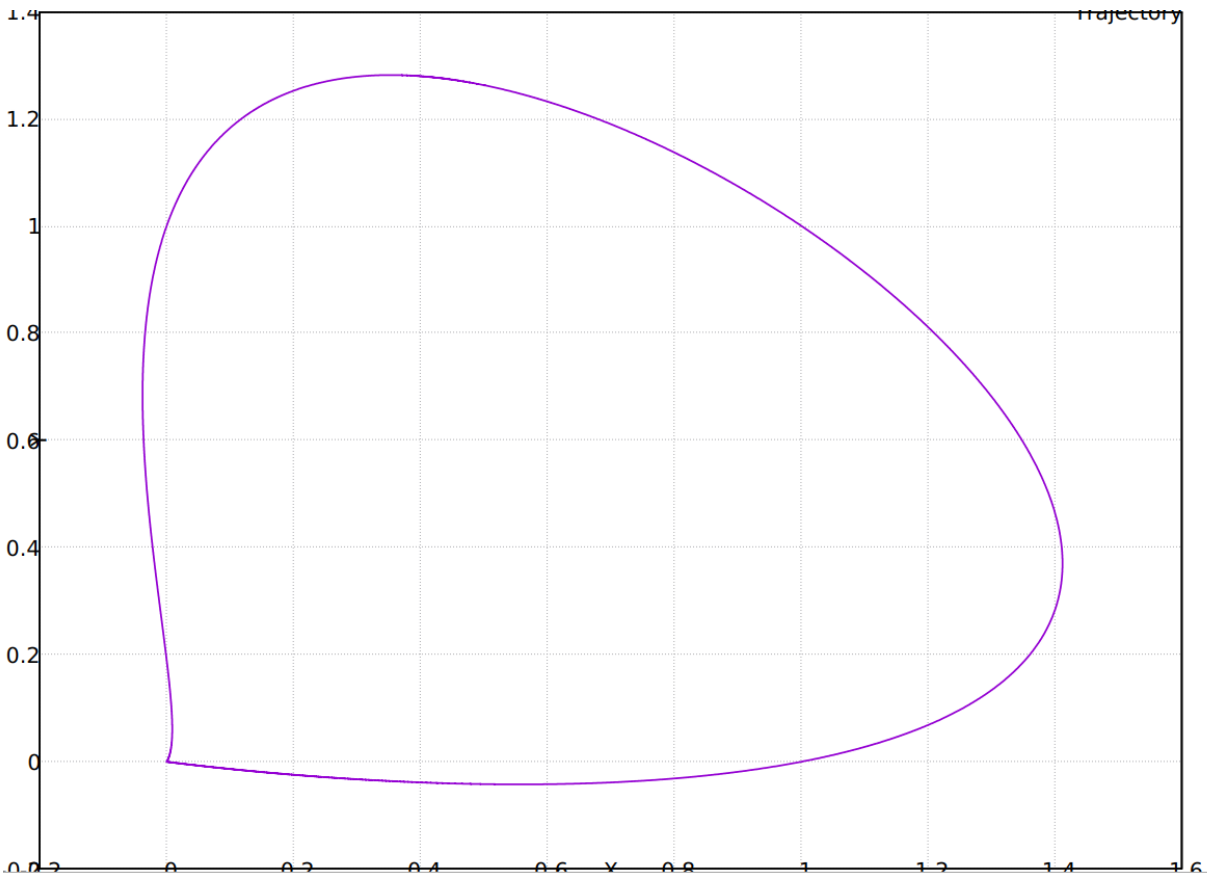

Figure 10 Square path minimising 4th derivative

Another factor that plays an important role in the smoothness of the path, and an even more important role in the values of the velocities and accelerations over the path, is the time allowed for the quad to fly between waypoints. As can be seen in figures 11-15, when a reasonable fly time of 1 second is allocated for traversing each waypoint, the path is smooth and the velocities and accelerations over the path are reasonable. However, when the time between waypoints is unreasonably short (0.1 second), as depicted in figures 16-20, the path becomes slightly discontinuous, and the velocities and accelerations become unreasonably high. Similarly, when time between waypoints is unreasonably long (10 seconds), as shown in figures 21-25, the changes in velocity and acceleration become too fine for the quad to fly.

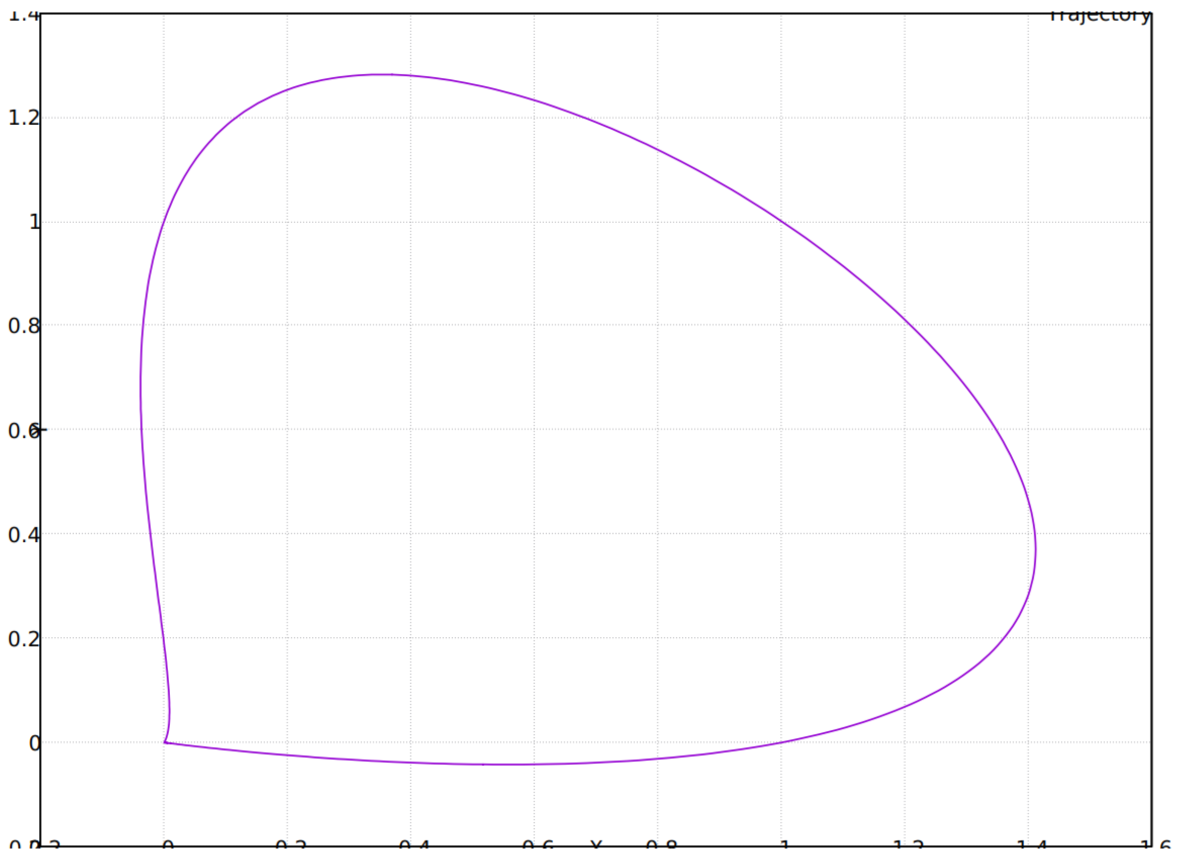

Figure 11 Square trajectory with reasonable time

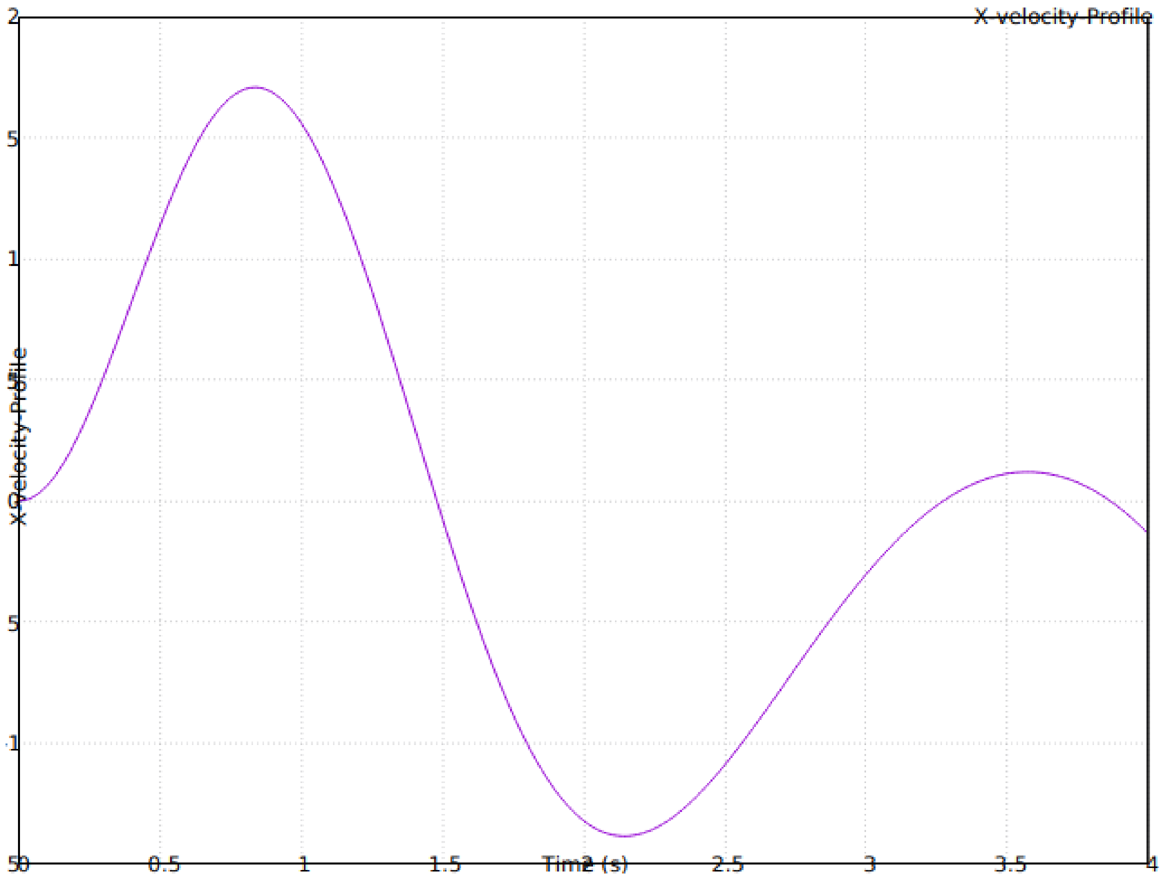

Figure 12 X-axis velocity for reasonable time

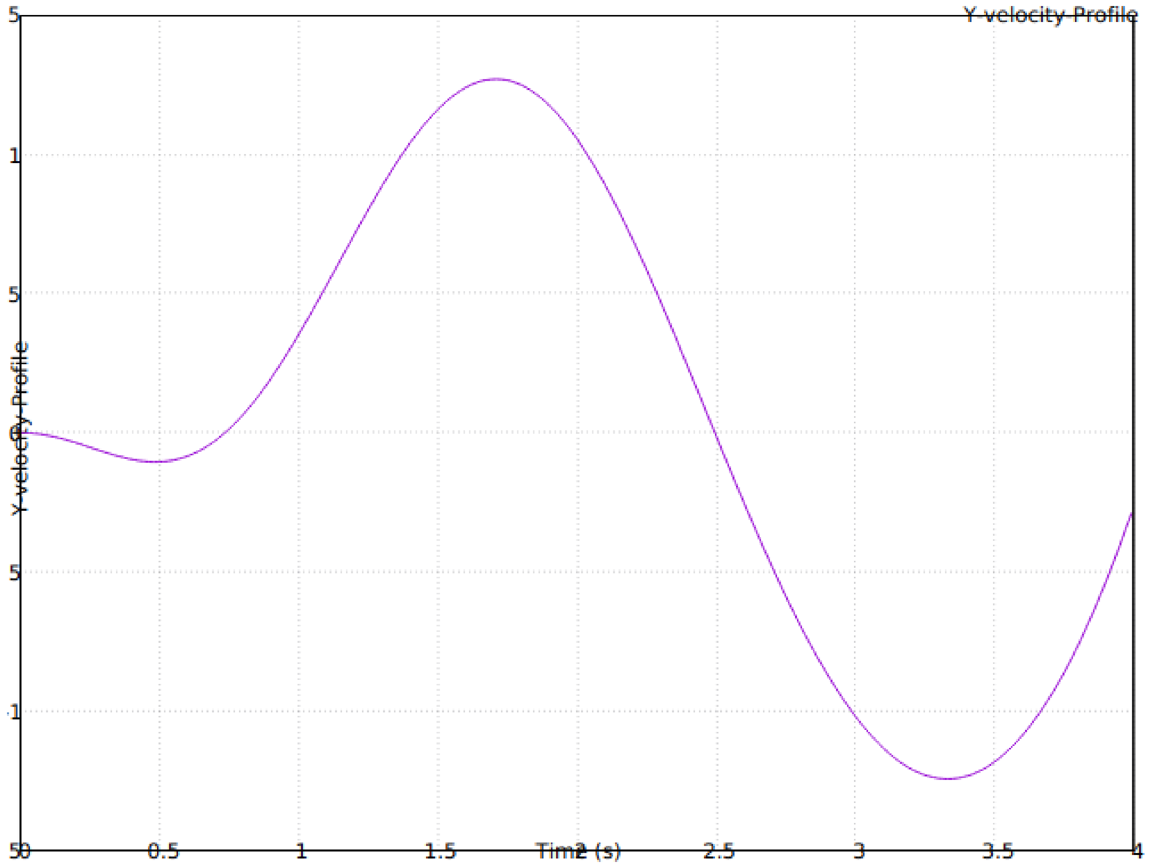

Figure 13 Y-axis velocity for reasonable time

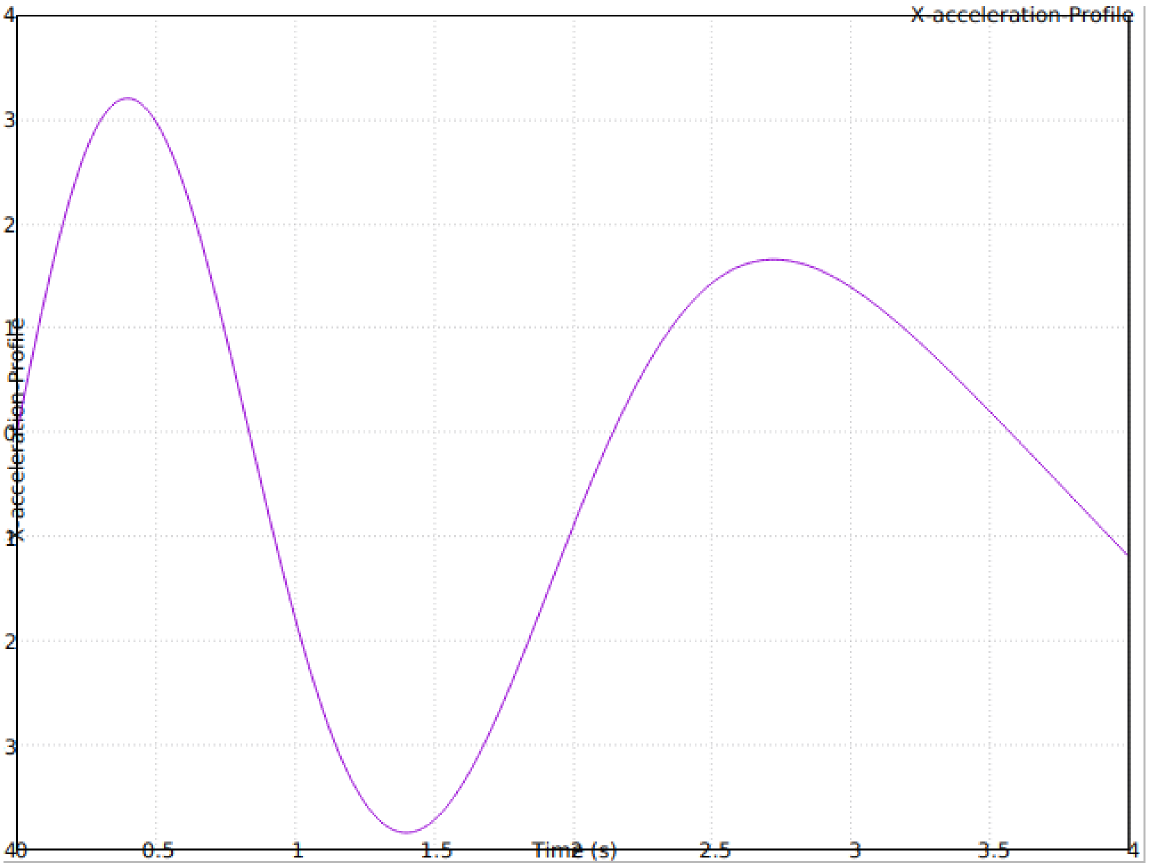

Figure 14 X-axis acceleration for reasonable time

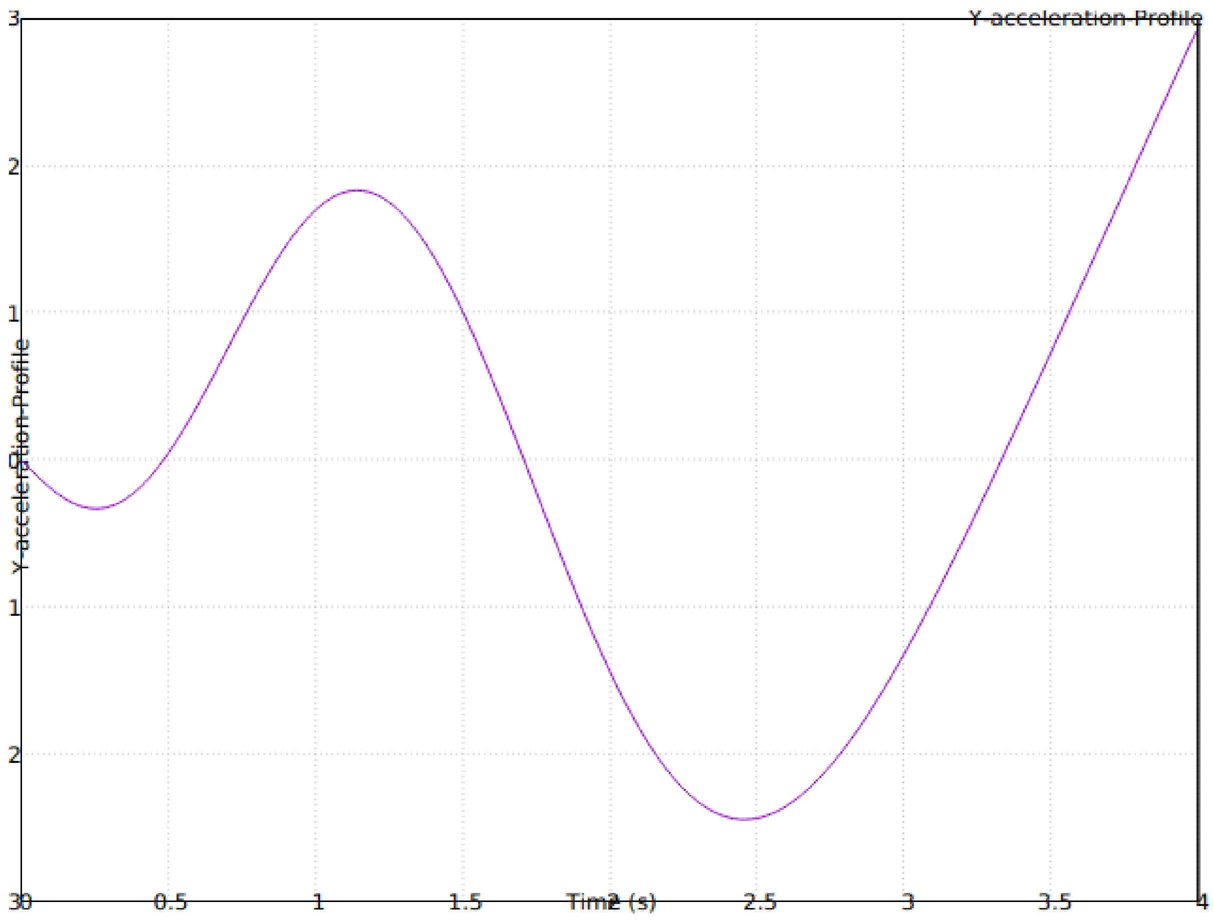

Figure 15 Y-axis acceleration for reasonable time

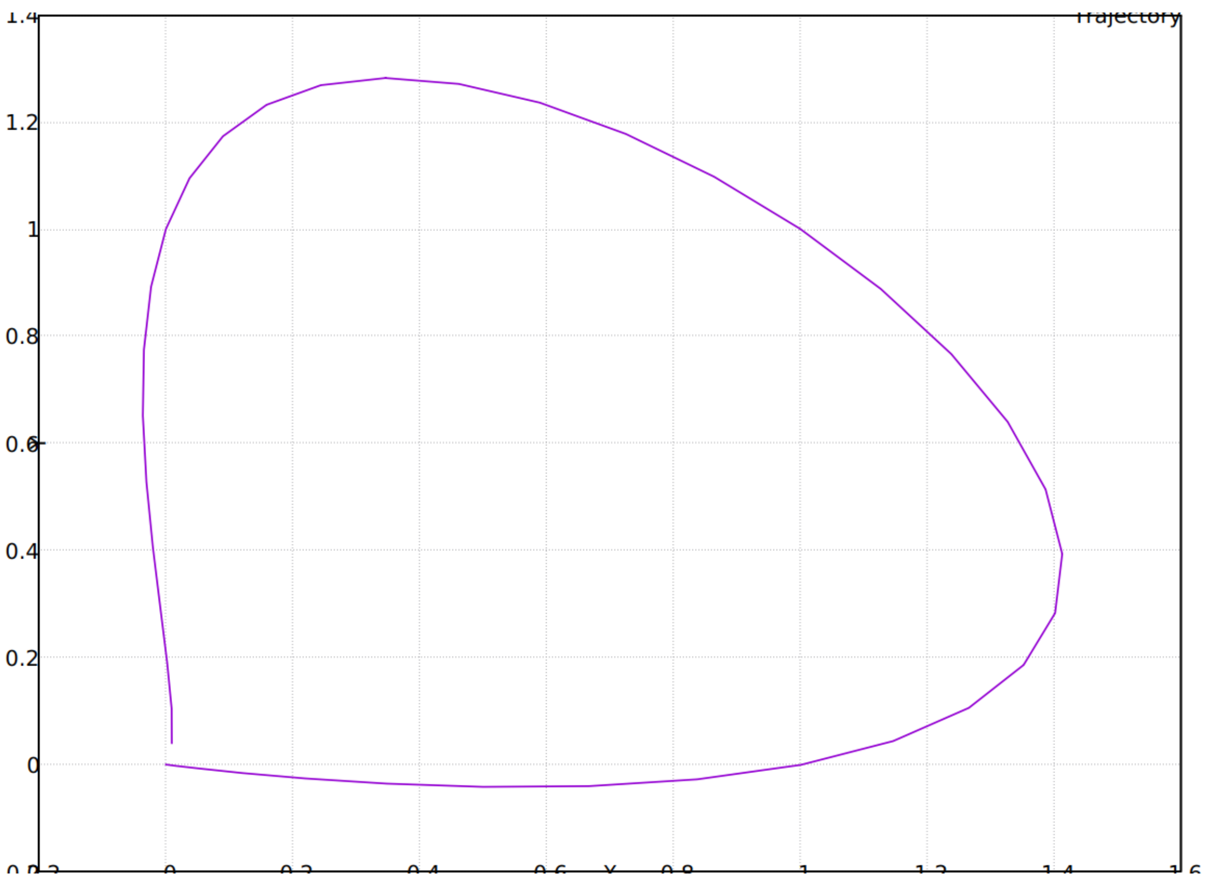

Figure 16 Square path for unreasonably short time

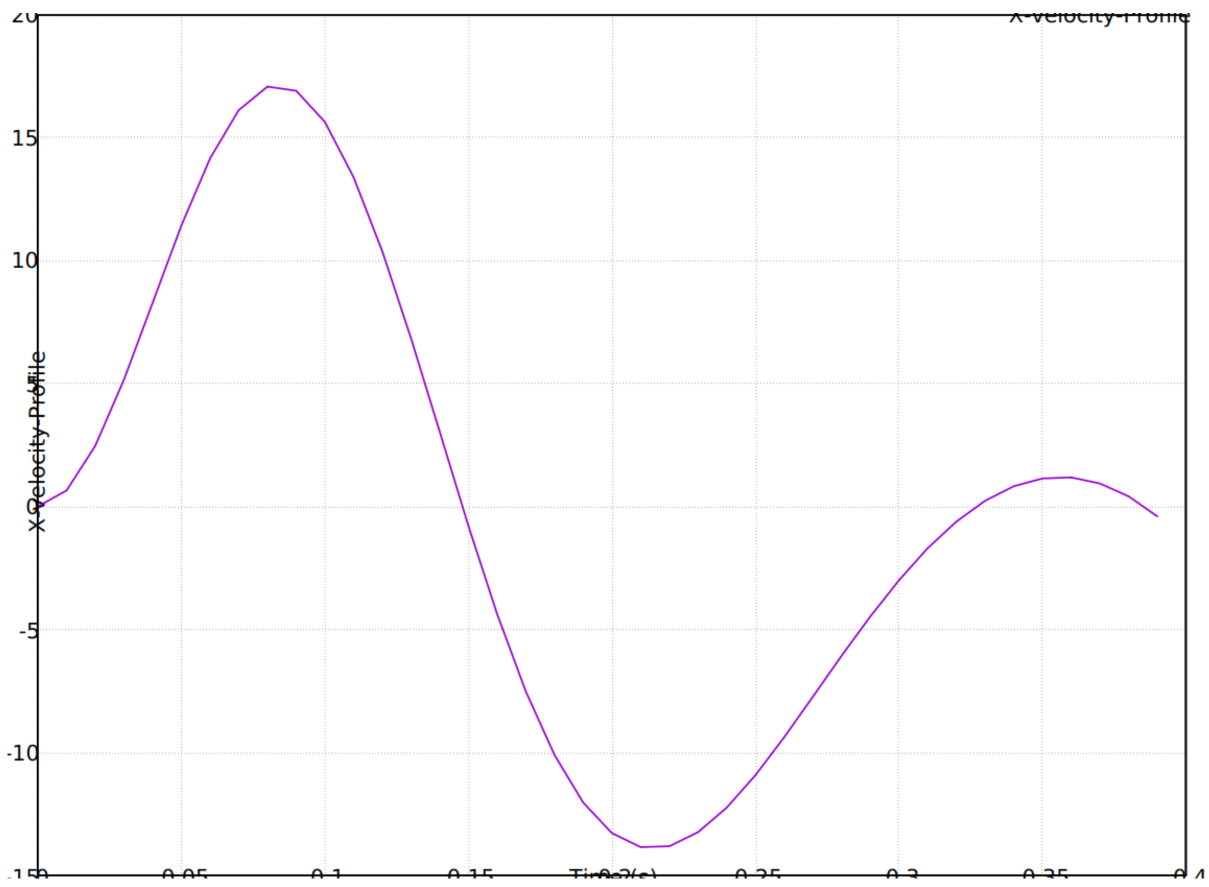

Figure 17 X-axis velocity for unreasonably short time

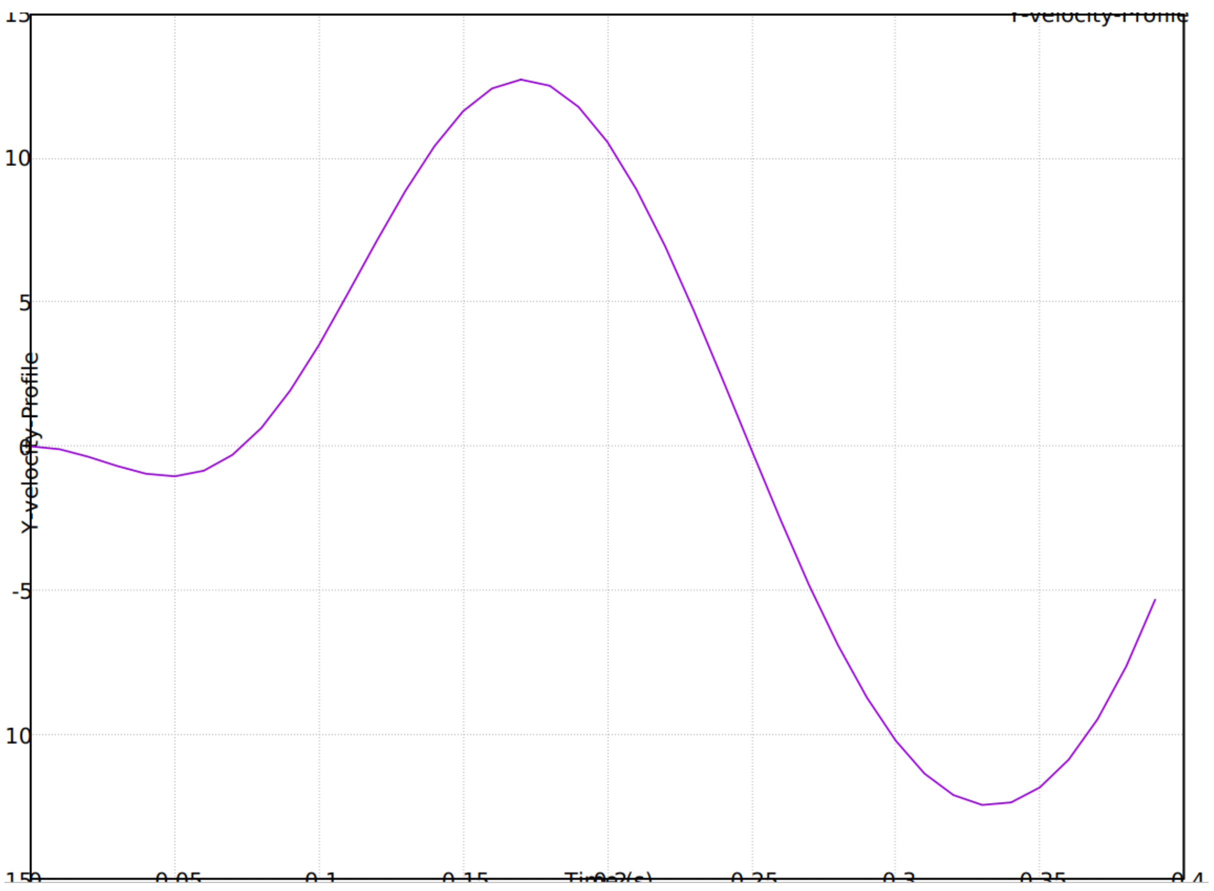

Figure 18 Y-axis velocity for unreasonably short time

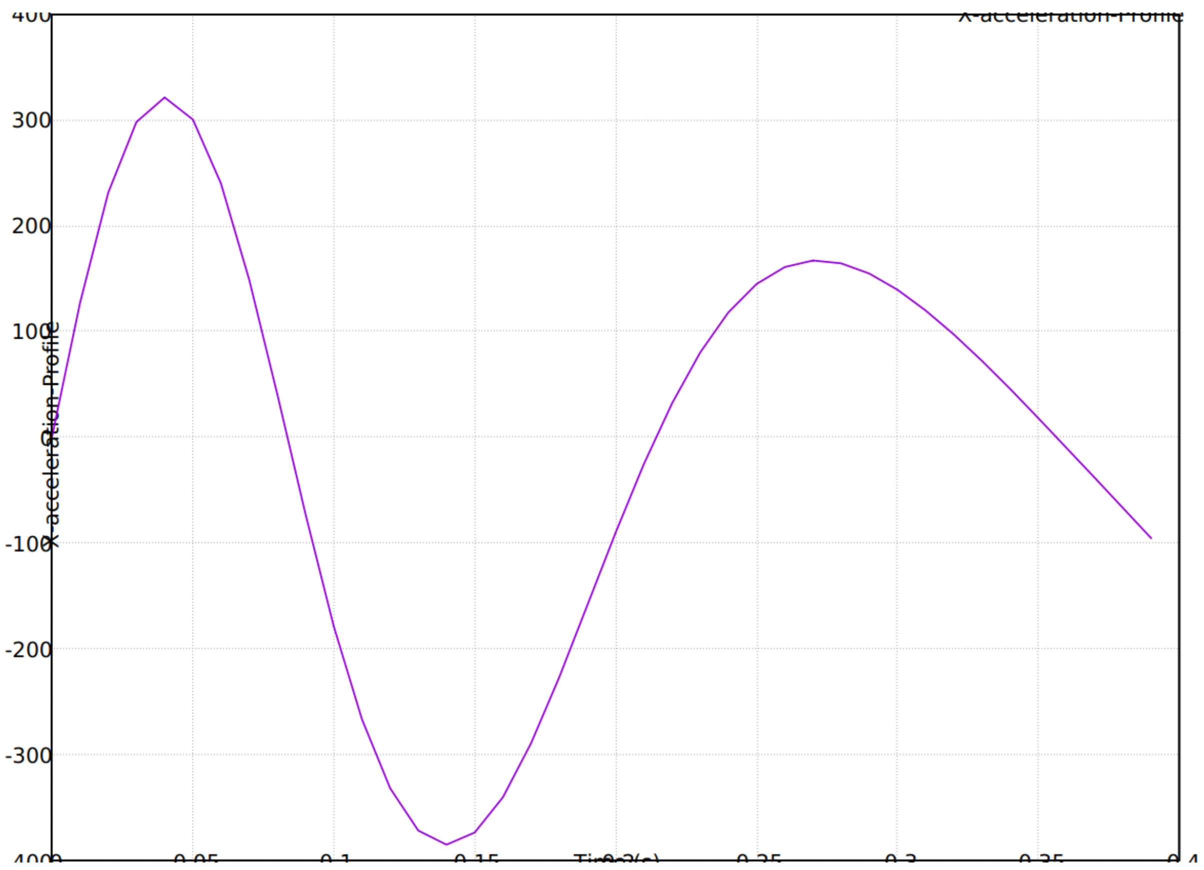

Figure 19 X-axis acceleration for unreasonably short time

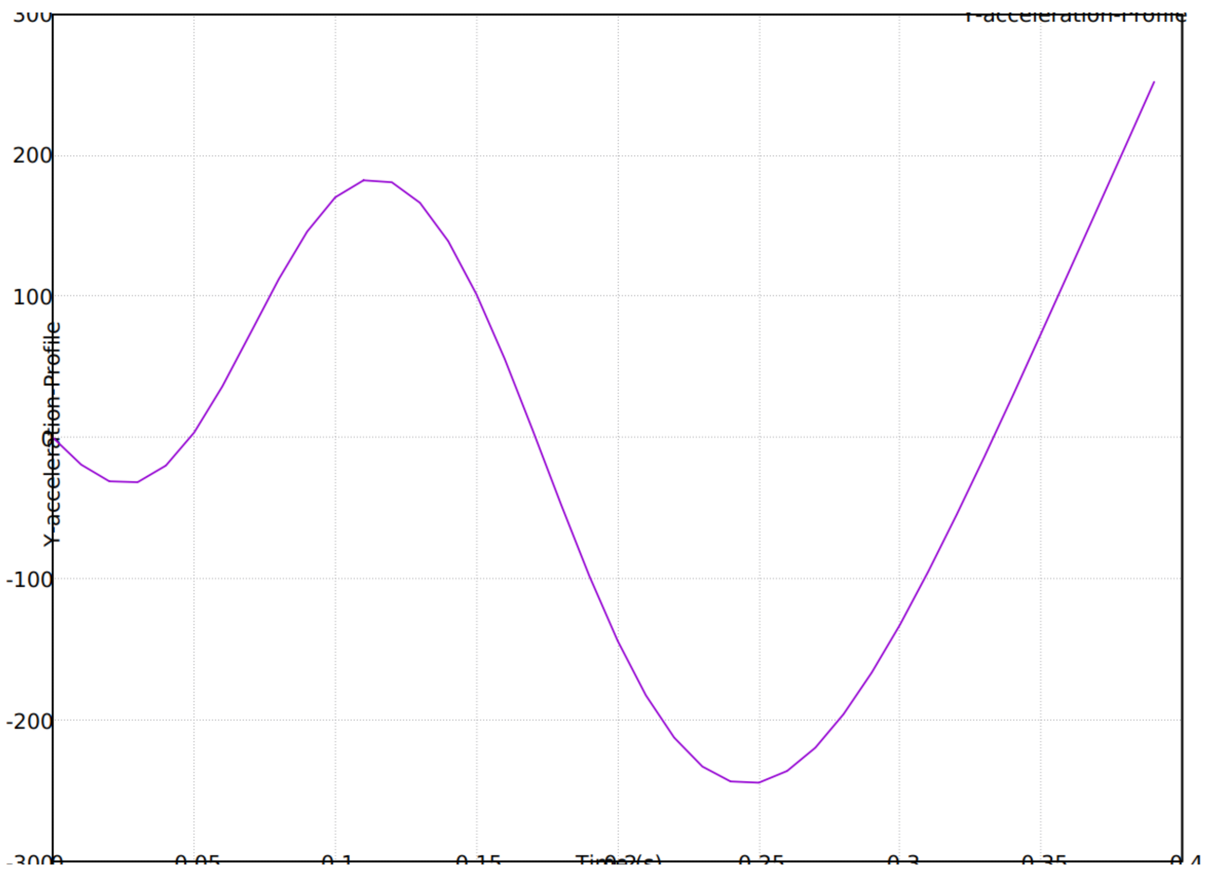

Figure 20 Y-axis acceleration for unreasonably short time

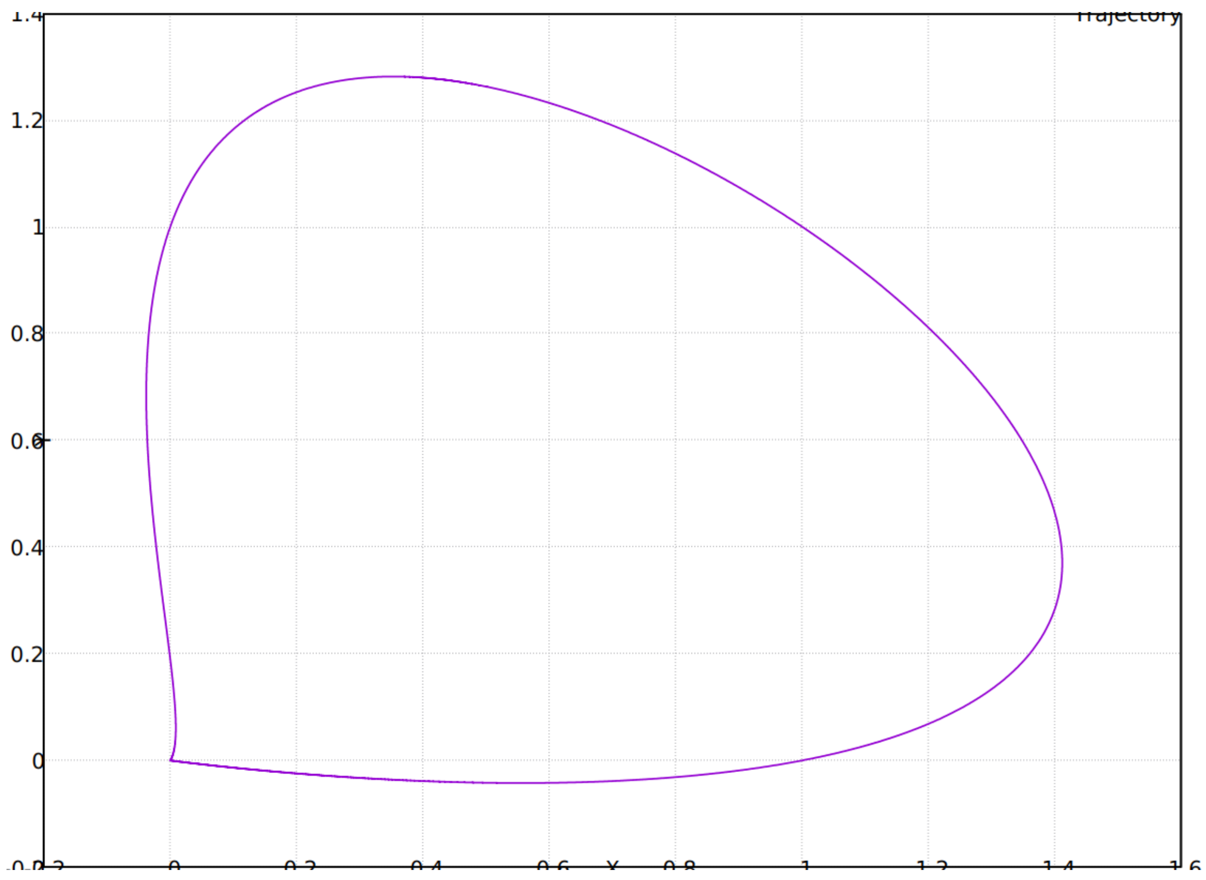

Figure 21 Square path for unreasonably long time

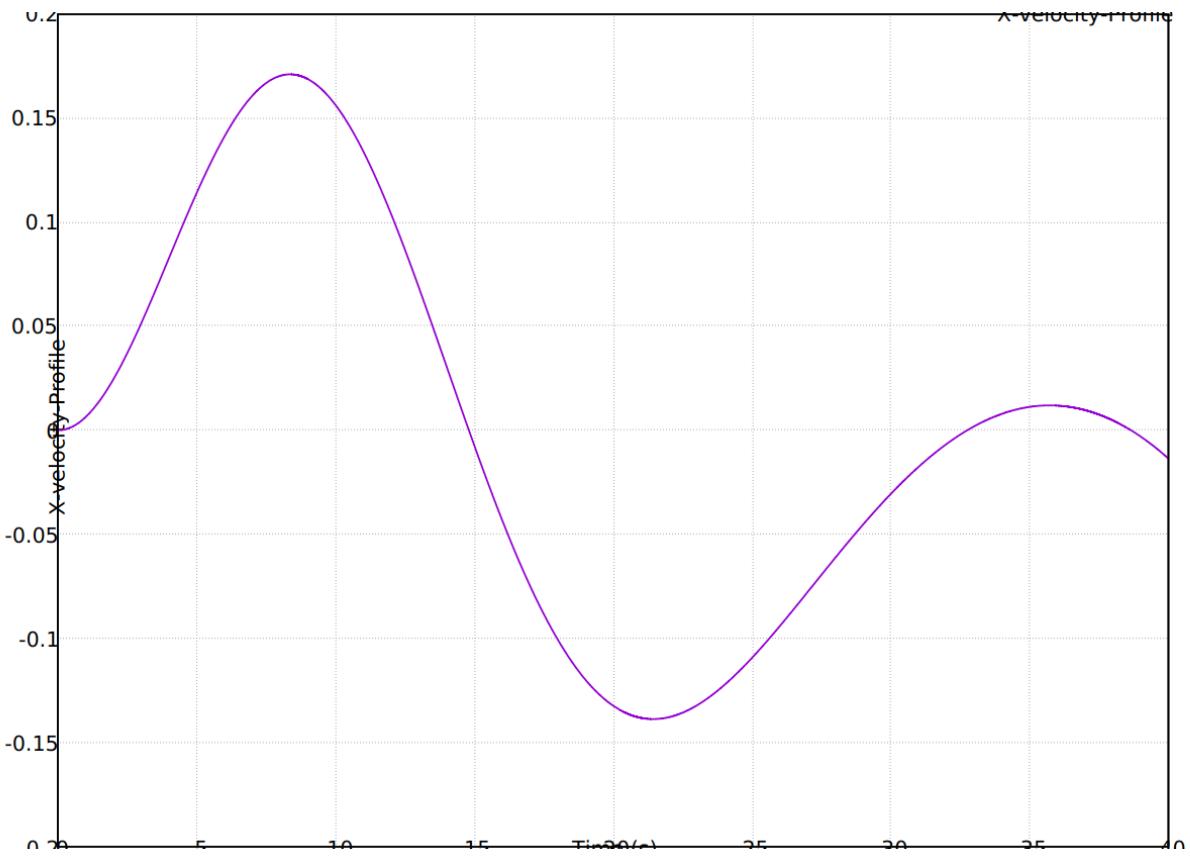

Figure 22 X-axis velocity for unreasonably long time

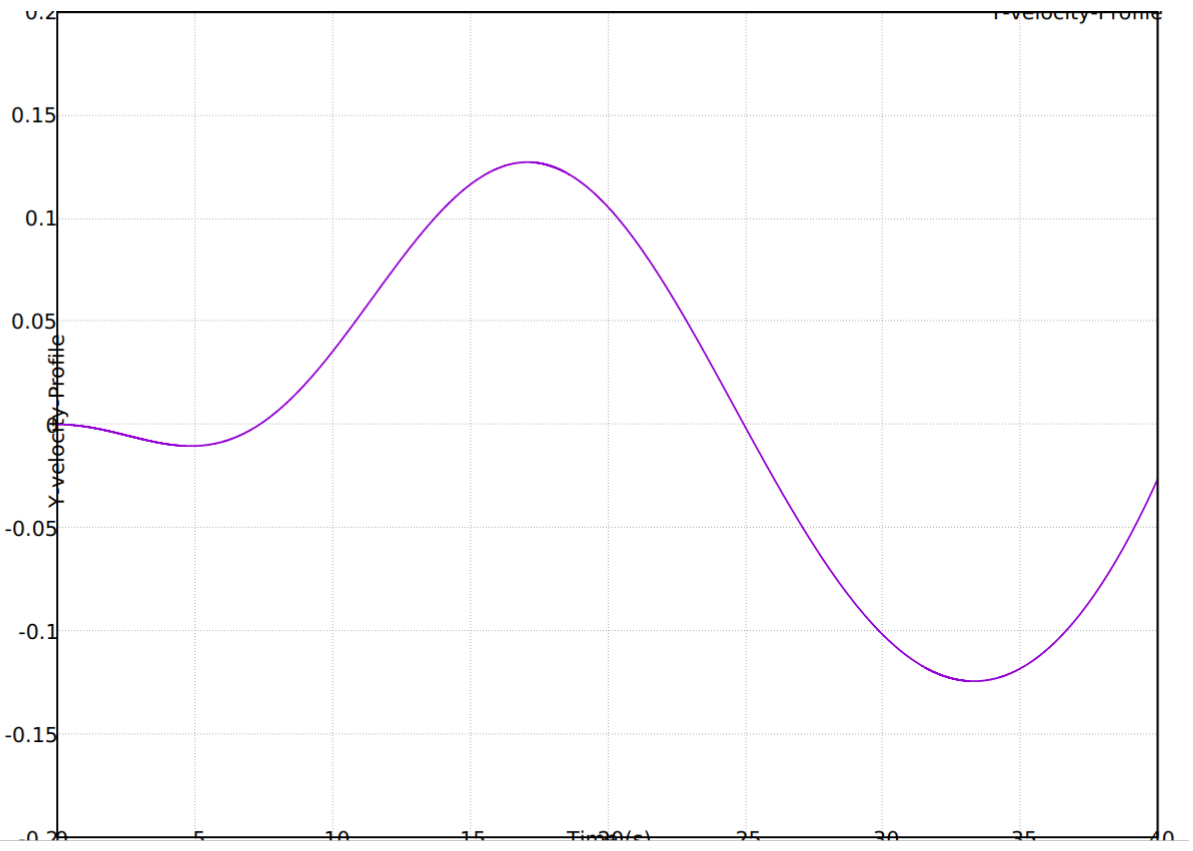

Figure 23 Y-axis velocity for unreasonably long time

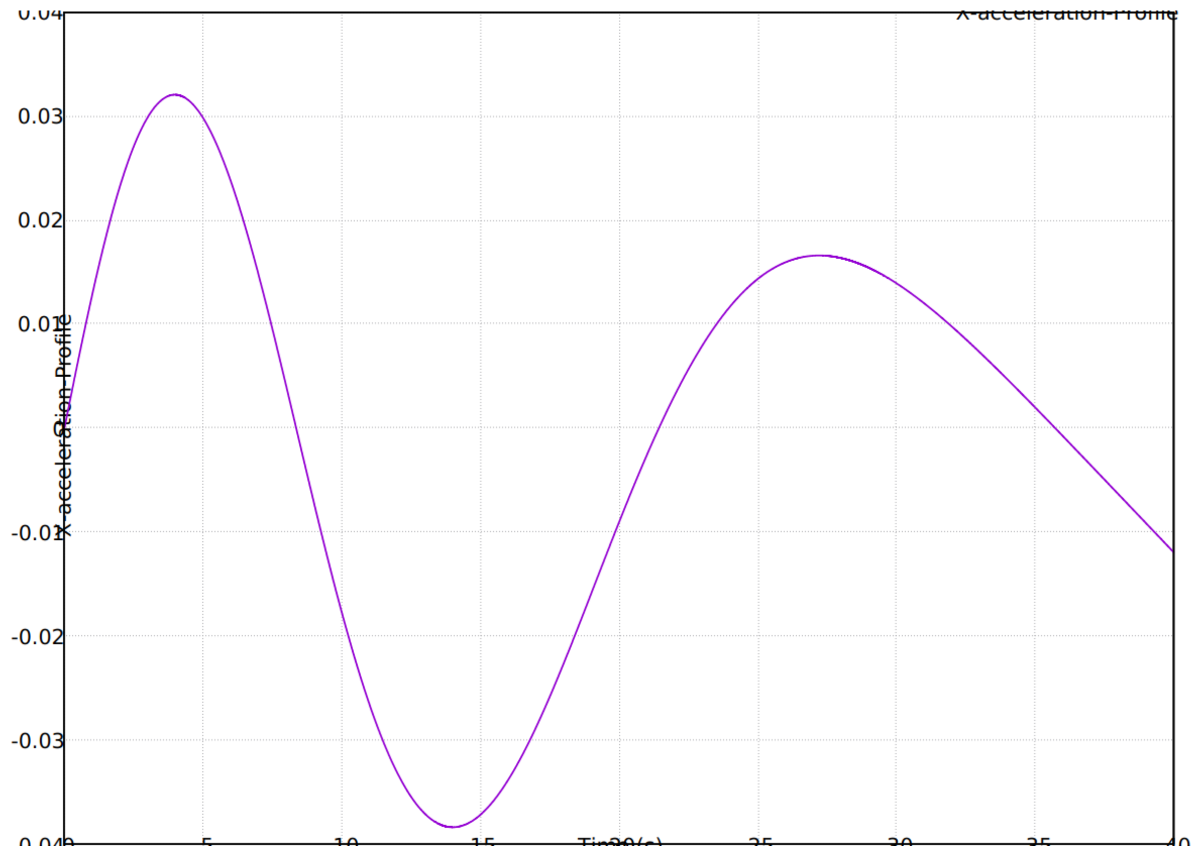

Figure 24 X-axis acceleration for unreasonably long time

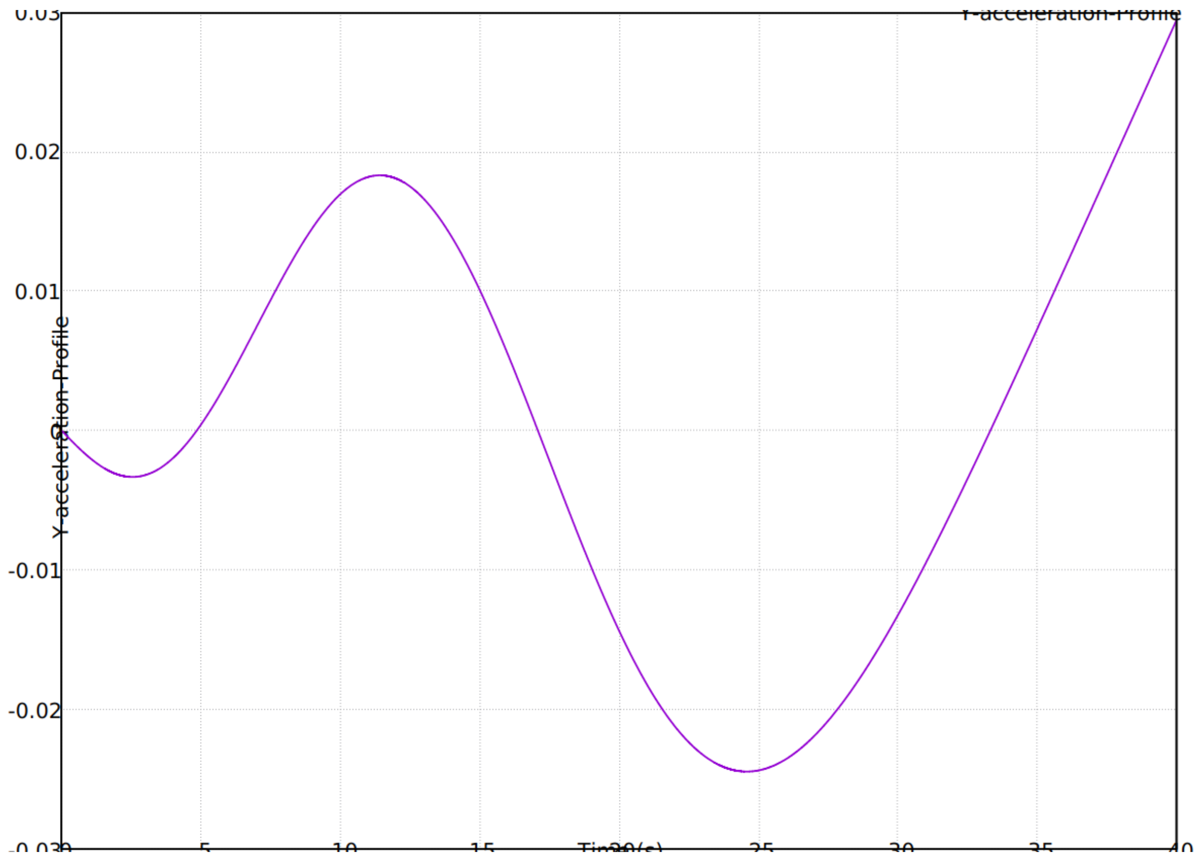

Figure 25 Y-axis acceleration for unreasonably long time

When generating a circle using evenly space waypoints, the number of waypoints plays an important role in how smooth the circular path is. As can be seen in figures 26, 27, and 28, a number of waypoints of 50, 100, and 150 are reasonable choices to generate an overall circular path, but the path plotted with 100 waypoints is visibly smoother than that with only 50. This is because the longer the distance between waypoints, the harder it is to find a limited order polynomial (in our case an 8th order polynomial) to closely match a circular arc.

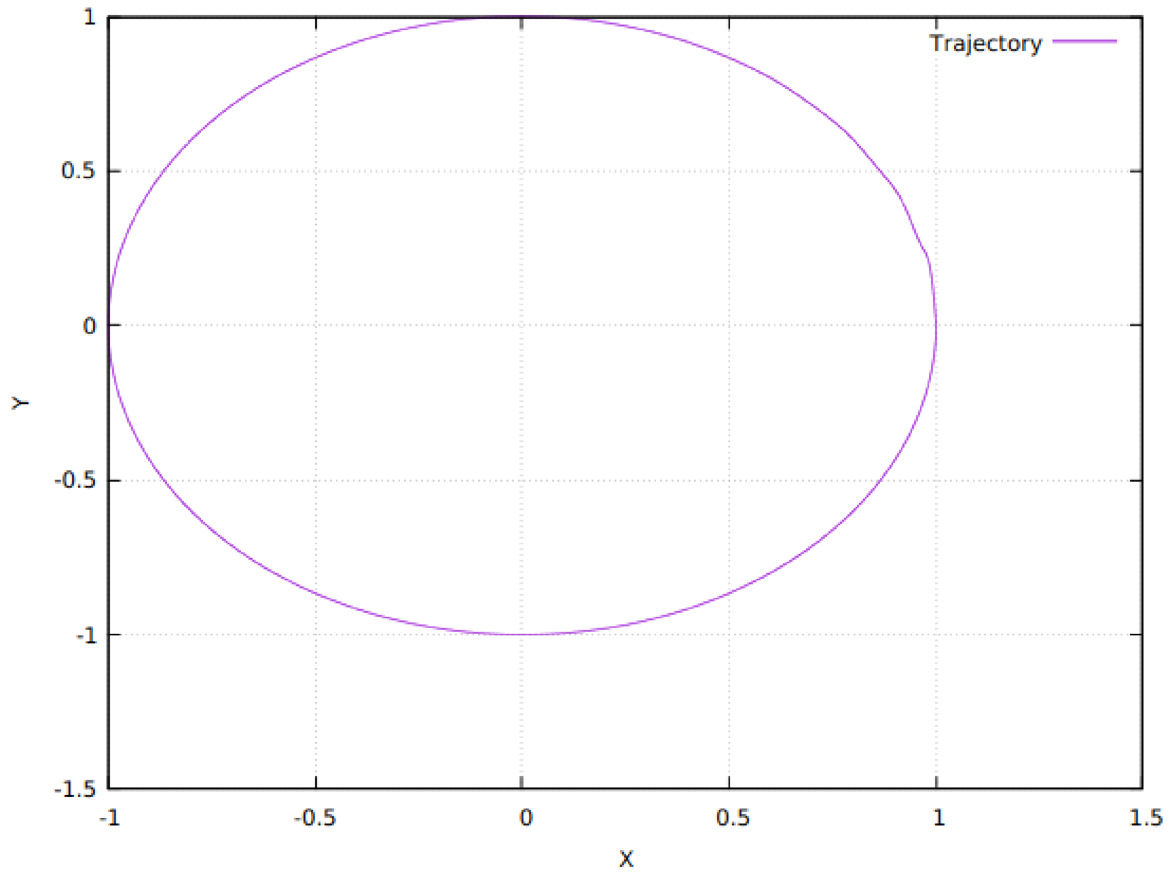

Figure 26 Circular path at N = 50 waypoints

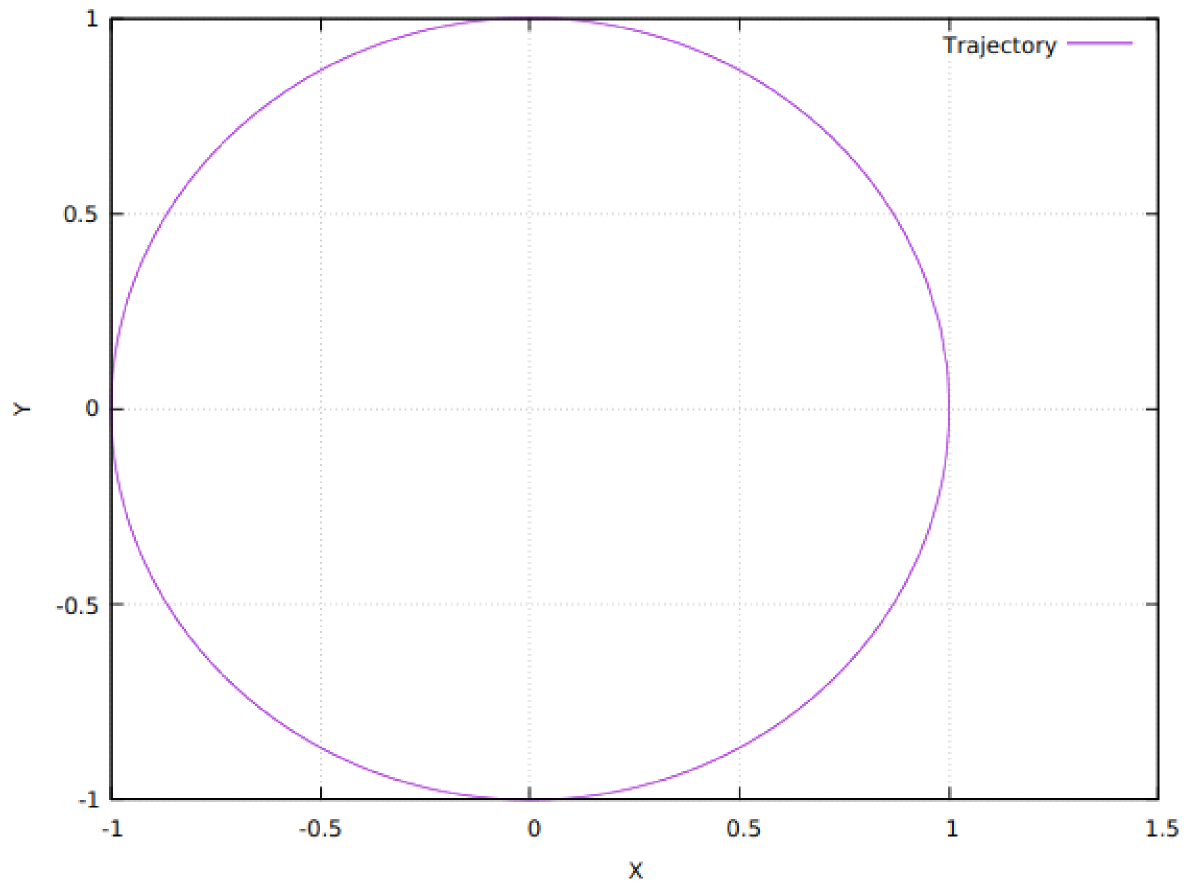

Figure 27 Circular path at N = 100 waypoints

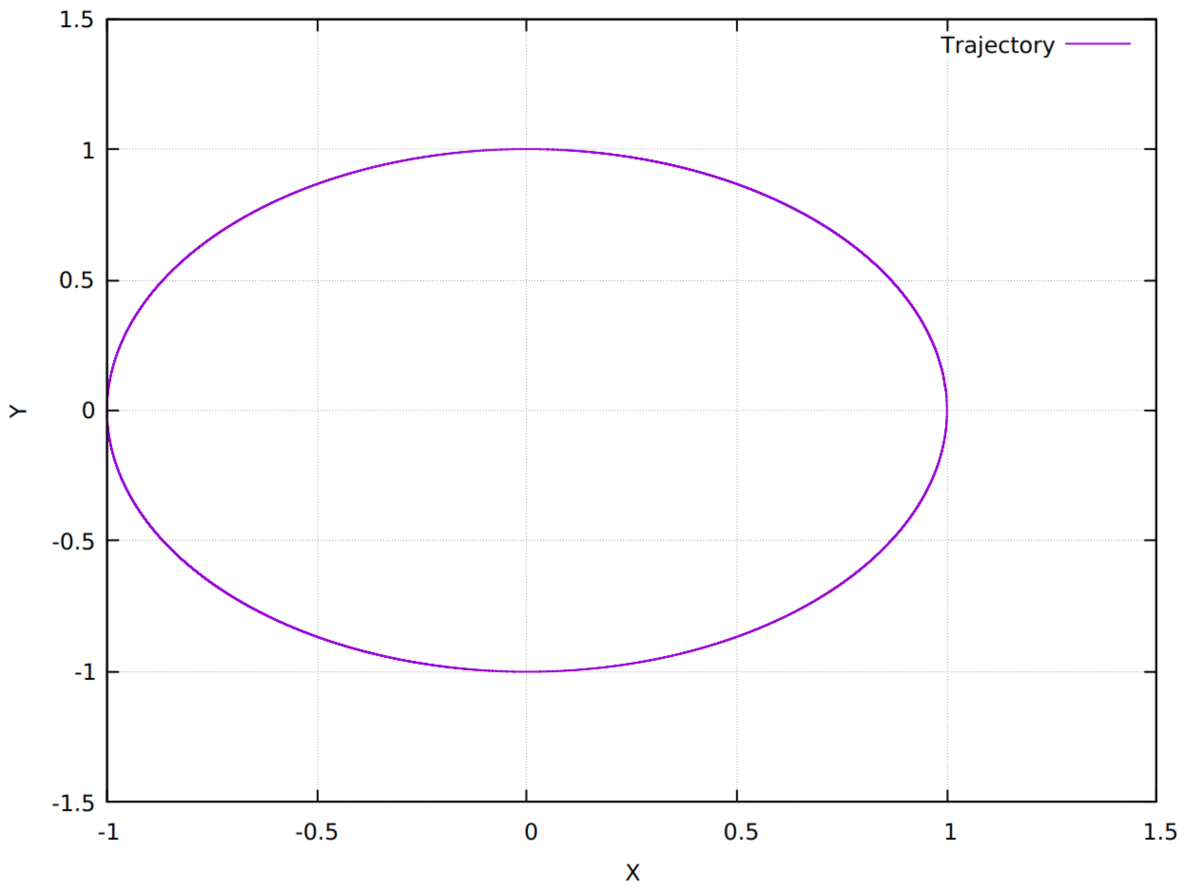

Figure 28 Circular path at N = 150 waypoints

Putting it all together

Using the same grid as was used to test the three different pathfinding algorithms, the optimal path between start point (0,0) and end point (9,9) was smoothened using the same method as outlined in the Polynomial Smoothing section of the report. There were 12 waypoints, one for each node in the found path. The path between them was generated using 8th degree polynomials continuous until the 4th derivative and minimised for the 4th derivative. This setting provided the path that can be observed in figure 29.

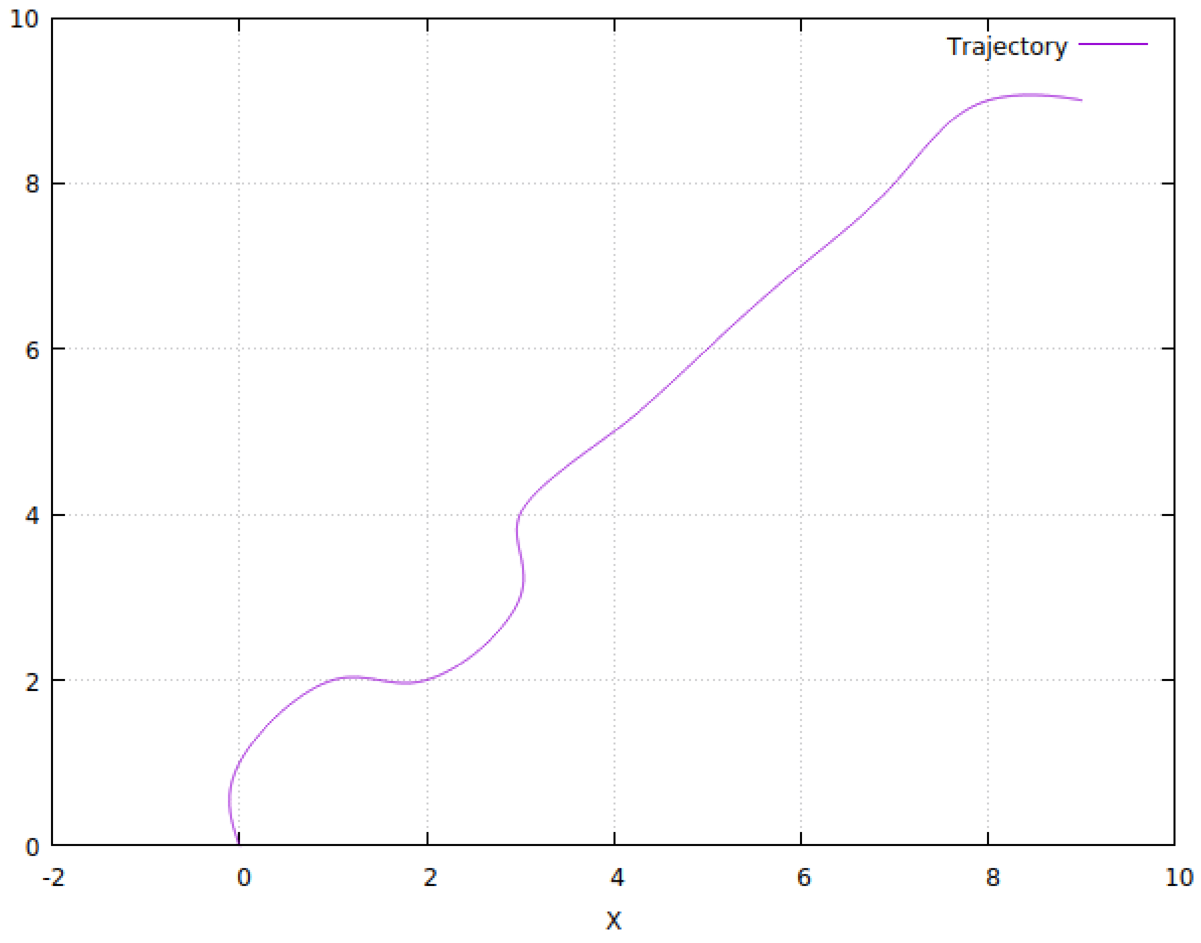

Figure 29 Trajectory found by full_stack_planning.cc

This path was then inputted into the MATLAB code developed for the previous laboratory. The resulting path flown by the quadcopter is plotted in figure 30. As can be seen, the two paths are very similar to one another, which indicates that the settings used to generate the optimal path were strict enough to output a sufficiently smooth desired trajectory for the drone to follow.

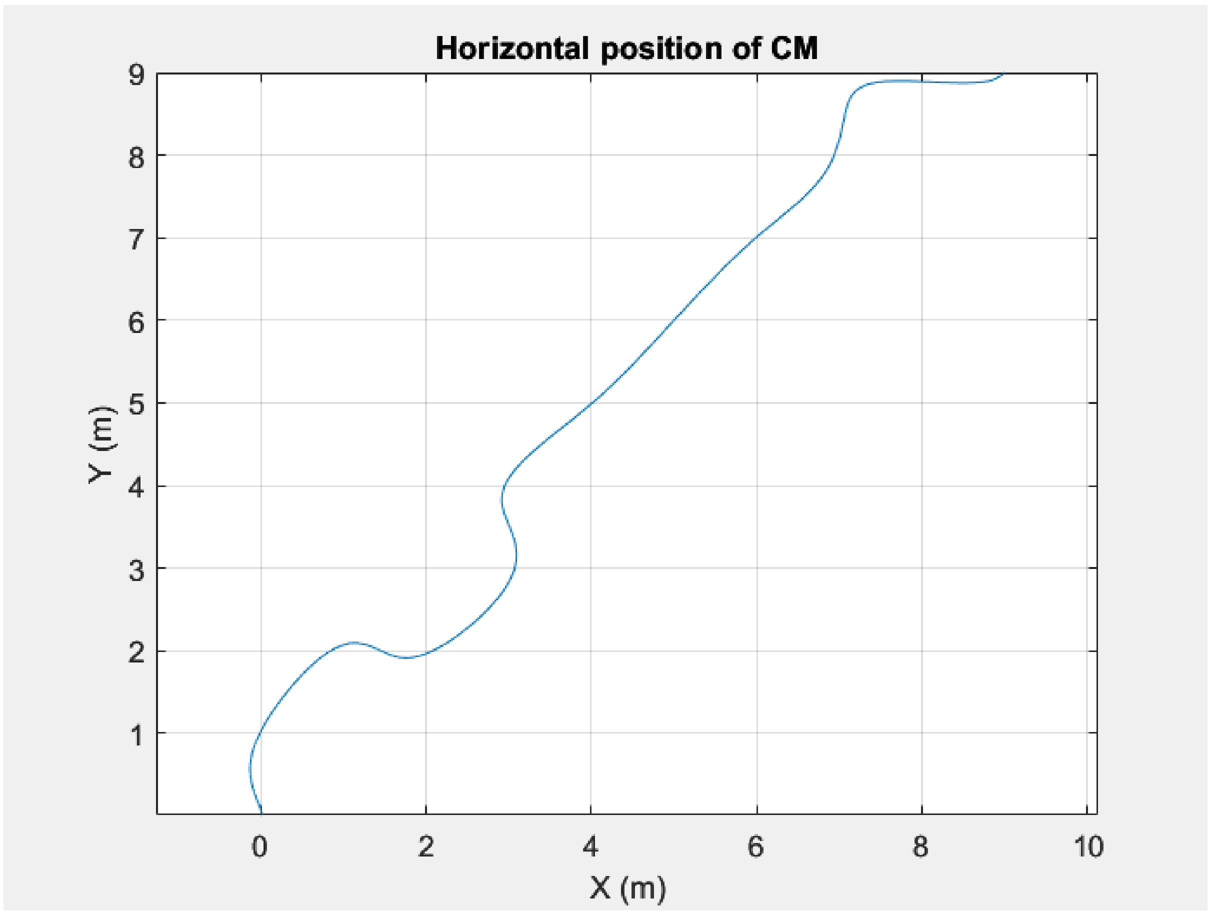

Figure 30 Path flown by quadcopter in matlab simulation, given path found by full_stack_planning.cc

3. Conclusion

In conclusion, this report analysed the difference in efficiency for three pathfinding algorithms: Depth-First Search, Dijkstra's method, and A*. Then, it analysed what settings for Polynomial Smoothing could be changed to generate a sufficiently smooth path. This is important so that the quad can use this path and track it as closely as possible. Finally, the most efficient pathfinding algorithm of the three, A*, was used in conjunction with the settings that were optimal for the Polynomial Smoothing code to generate a path through a predefined grid. This path was then tested using the MATLAB quad simulation to validate that the path is sufficiently smooth to be flown.