SC 627
Assignment 3: Collision Avoidance

Hitesh Kandala | 180070023

Implementation Details

In this assignment our aim is to reach the goal point from the origin while avoiding any obstacles that the bot may face in its path. To implement this, we use a heuristic based planning approach.

I have implemented a goal condition which checks whether the robot is within a threshold radius to the goal and this threshold radius is a hyper-parameter which can be tuned. If this condition turned true, we terminate the planning algorithm. Since the algorithm is heuristic based, it was hard to get the bot very close to the goal location because of the constraints on velocity and heading direction. So the threshold which we chose is a little big and it does help in avoiding all the obstacles in between.

Iterative code begins with generating feasible samples. We have a maximum velocity constraint and a maximum orientation change constraint. With these two descriptions combined, the reachable velocity area is a sector of a circle with a radius equal to the maximum allowed velocity, an angle equal to twice the deviation allowed from the current orientation, and the center line of the sector along the current velocity direction. Now, in order to sample points, we linearly discretize along the radial and angular directions, and store these in the form of tuples.

The next step is to filter out the points which lie in any of the collision cones. Each point is checked for collision with each obstacle's collision cone by iterating over all the points for all the obstacles. As we know the obstacle radius and the distance from the current position, we can compute the cone angles by using simple geometry. A test is conducted to determine if both sample points are less than half of the cone's center line when derived from the current pose point. If this is the case, then the point lies within the collision cone and we discard it. If not, we append the point to a list of feasible points.

Now, once we have these feasible points, we need to find the optimum velocity point to stick to. Our technique is a hybrid version of the heuristics described in the paper. We try to take the point which is closest to the goal direction. For this, we find the feasible point which has minimum deviation from the current position to goal line and choose that as our next velocity target. This is done so that we will be moving even though the goal direction is blocked and still we will have some notion of moving in the direction of the goal. Although the robot cannot reach the goal completely, it is able to move to a point that is away from obstacles and fairly close to the goal.

After publishing the optimum velocity point, we take a final step which is based on the calculated value. To work with the publisher, we convert the cartesian coordinate velocity to linear and angular components before publishing. The velocity convert function is used as it is. On every iteration, the subscribers call the callback functions for odom and obstacles. On every iteration, the data is stored in appropriate data structures and updated accordingly. The odom callback also logs time and the robot trajectory so as to be able to plot them later.

To run the code, we just instantiate the class and call the main algorithm function and generate the X vs T and the path plots.

Simulation results

We get the following results upon simulation:

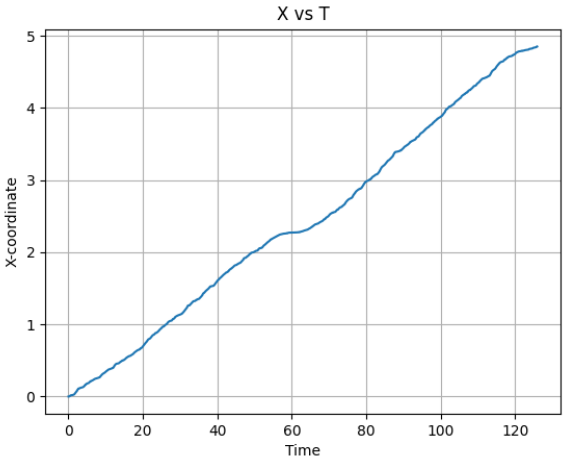

Figure 1: Robot 2: X vs Time

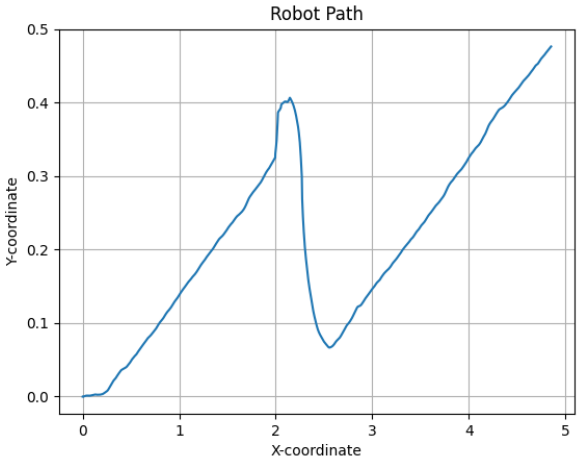

Figure 2: Robot 2: Path Plot