CSE 276A - Homework 4

Chhabra, Rachit (rachhabra@ucsd.edu)
Department of Mechanical and Aerospace Engineering, University of California San Diego

I. Problem Statement

The objective of Homework 4 is to implement versions of two Motion Planning Algorithms. The first is to reach to the goal in shortest time or by traversing minimum distance. The second algorithm is to calculate and traversing through the safest possible path.

II. Setup

The setup of the environment is follows:

1. We have 12 April Tags which we considered as known landmarks during the Path Planning problem. We have 4 different April Tags and 4 repeated April Tags. Fig. 1 shows a sketch of the ground-truth map in the top-down view layout.

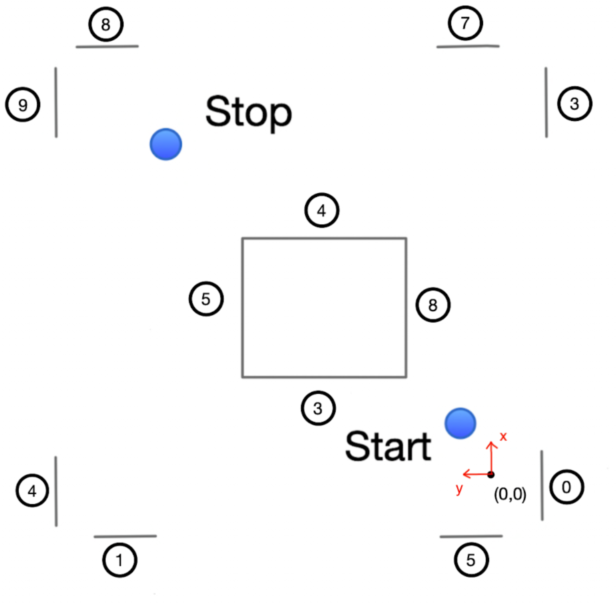

Fig. 1. Placement of Landmarks

2. Here, each landmark does not have a distinct ID (April Tag id) and thus we need to create an algorithm to solve this ambiguity as well while localising the robot.
3. The initial starting position of the robot is $(0.5, 0.5, 0)$ meters and the Stop or Goal position is $(1.95, 1.95)$ meters. This means we can reach the goal at any orientation.
4. The overall width of the area is 8 feet and the size of the obstacle in the center is is 1 foot.

III. Problem Formulation

For the first part, we start with formulating the problem as a Deteministic Shortest Path Algorithm.

We first specify the finite vertex set $\mathcal{V}$. We take into account the robot's position here, and this alone is sufficient to define the robot's state at any time step.

$$v = [x, y] \tag{1}$$

where $v \in \mathcal{V}$ and x, y is the coordinates of the robot in the given 2D map.

The edge set $\mathcal{E} \subseteq \mathcal{V}$ x $\mathcal{V}$, and the edge wieghts are defined as follows:

$$\mathcal{C} := \{c_{ij} \in \mathbb{R} \cup \{\infty\} | (i,j) \in \mathcal{E}\} \tag{2}$$

$$\begin{aligned} c_{ij} &\leftarrow 1 && if\ ||v_j - v_i||_2 = 1 \\ c_{ij} &\leftarrow \sqrt{2} && if\ ||v_j - v_i||_2 = \sqrt{2} \\ c_{ij} &\leftarrow \infty && for\ all\ other\ cases \end{aligned}$$

where c_{ij} denotes the arc length or cost from vertex i to vertex j.

Let's define a path $i_{1:q}$ be defined as a sequence of nodes, $i_1 \in \mathcal{V}$ from i_1 to i_q:

$$i_{1:q} = (i_1, i_1, .., i_q) \tag{3}$$

The goal node here is the given position of the robot from the robot planner, (x_τ, y_τ). We take into consideration all the paths from start $s \in \mathcal{V}$ to target $\tau \in \mathcal{V}$:

$$\mathcal{P}_{s,\tau} := \{i_{1:q} | i_k \in \mathcal{V}, i_1 = s, i_\tau = \tau\} \tag{4}$$

and calculate the path lengths for all of them using the formula:

$$\mathcal{J}^{i_{1:q}} = \sum_{k=1}^{q-1} c_{i_k}, i_{k+1} \tag{5}$$

where $\mathcal{J}^{i_{1:q}}$ is the sum of the edge weights along the path.

Now, after defining all the required variables, we now define the the objective of our program, that is, to find a path that has the minimum length from node s to node τ:.

$$\textbf{dist}(s,\tau) = \min_{i_{1:q}\in\mathcal{P}_{s,\tau}} \mathcal{J}^{i_{1:q}} \tag{6}$$

$$i^*_{1:q} = \underset{i_{1:q}\in P_{s,\tau}}{arg\,min}\, \mathcal{J}^{i_{1:q}} \tag{7}$$

For part two, we formulate the problem as a Voronoi Graph where we calculate the maximum distance to the boundary points and once getting that, we search along the graph for the shortest path. Thus, after creating the Voronoi Graph, we implement the Deterministic Shortest Path.

IV. TECHNICAL APPROACH

Part I

Here we start with generating a map of the environment to run our Deterministic Shortest Path algorithm. This is done using the file $generate\_map$. We discretize the area into 100 x 100 cells, each of side 0.0245m. Now we create a binary occupancy map where, the zeros depict the free space whereas the ones are the areas which are considered as obstacles. Following is the map generated.

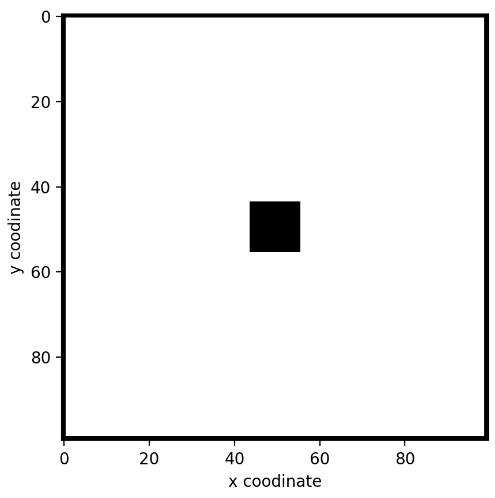

Fig. 2. Binary Occupancy Map

But, the map in Fig. 2 is created using the assumption that the robot is a point. In reality, the robot has some dimensions and thus we need to increase the size of our obstacles. This is done by using the Minkowski Sum. As the robot's orientation can be any orientation between $-\pi\ to\ \pi$, we add 0.12 meters to all the obstacles at every direction inside the map to account for the size of the robot. Fig. 3 shows the map after taking the Minkowski sum, where the white cells represent the free cells and where the robot can move.

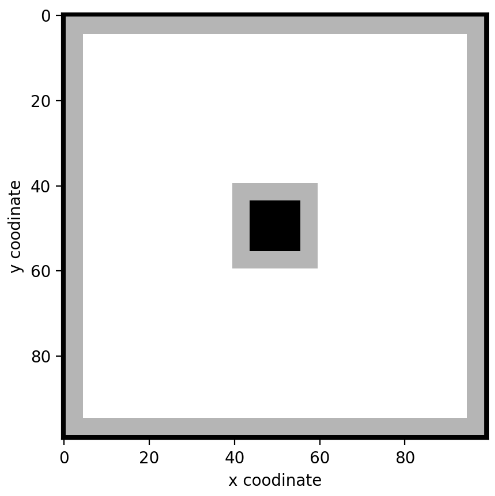

Fig. 3. Minkowski Sum Map

Now we go onto the Path Planning Algorithm. We use A star to find the shortest path between the start and the goal. The algorithm is written in the $astar$ and $robotplanner$ files. Here, we consider that the motion model of the robot is such that the robot can move in 8 directions, that is forward, backward, right, left and four diagonals. The cost is as mentioned in the previous section, and the heuristic considered is the Euclidean Distance to the goal.

The issue with this is in reality the robot can move in every direction, instead of only 8 directions. So another algorithm is applied which uses the waypoints calculated by the A star algorithm and finds the smallest path from start to finish by trying to look for the shortest distance between each waypoint. Here, $bresenham$ function is used to see if any obstacle is there between two waypoints.

Finally, as we are working on the carpet instead of a wooden or cemented floor, there is always some manueverability issues for the robot. Thus, certain waypoints are added to the path, which allows the robot to rotate and align towards the next waypoint which is followed by the forward motion towards the next waypoint. This alignment is pure rotation and thus does not add any more distance/cost to our overall path.

Part II

Next, is using an algorithm to find the safest path from start to finish. Safety is defined as keeping the maximum possible distance from the obstacle (center obstacle and side walls) at any time. For this we use the Voronoi graph function from $scipy.spatial$. It creates a graph an divides the area into different sections, where the edges are lines which corresponds to the safest paths. Following the the result of using the above on our given map.

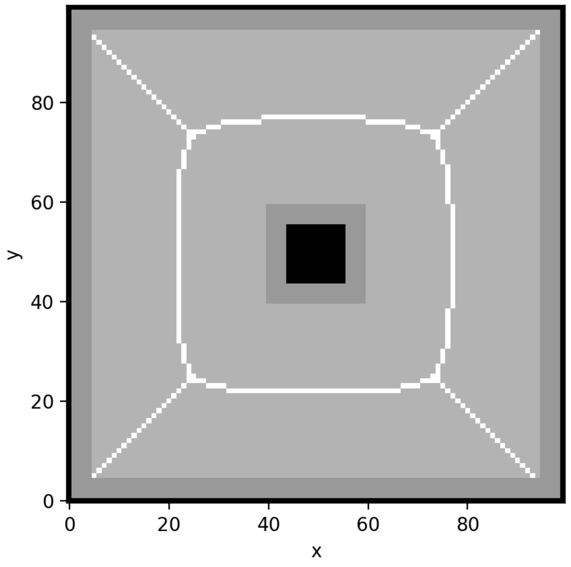

Fig. 4. Minkowski Sum Map

The white cells here Voronoi edges. We use this updated map and find the shortest path from our algorithm in Part I to find the required path.

Ambiguity in Apriltags As seen in the setup, we have 4 Apriltags which are repeated. This means that when when the robot detects a repeated Apriltag, it has two world coordinate values of the Apriltag in it's dictionary. Both the values would provide a very different estimate of the robot's current position in world coordinates. This is solved as below.

Whenever the robots sees the Apriltag which is repeated, it calculates the world coordinate position of the robot using both the world coordinate positions of the April tag. The value which is closer to the previous state of the robot, is taken to be the state of the robot at the current time iteration. $np.linalg.norm$ function is used in the code to find this difference.

V. Results and Analysis

Following is the setup used for the testing of the algorithms on the robot.

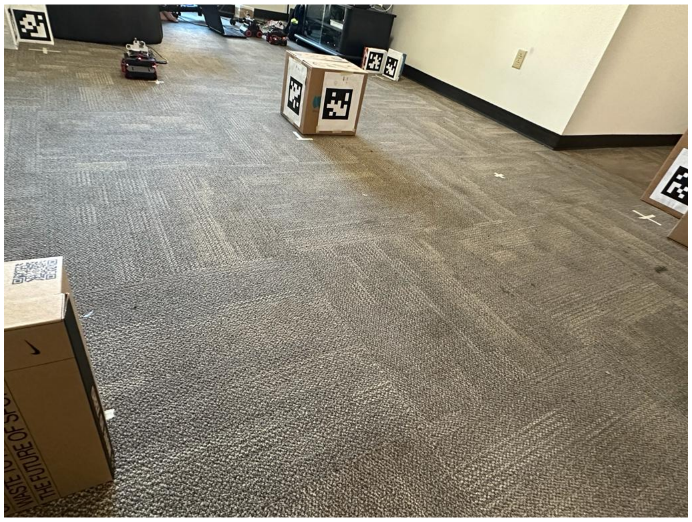

Fig. 5. Test Setup

The result path for A star is shown in Fig. 6 using the blue line.

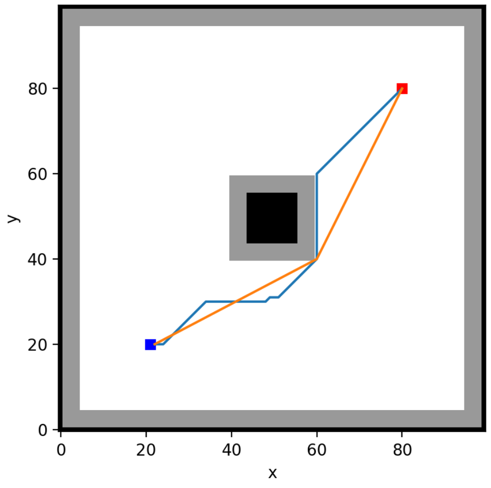

Fig. 6. A star

As it can be seen that it is the shortest path for our A star formulation, but in real robot experiment, we can find a faster route which is shown by the orange line.

Below is the link to the video showing the shortest path algorithm running on the robot.
Shortest Path Video

The safest path using the Voronoi Graph is shown below. It can be seen that the path is equidistant from the obstacles, i.e., it is exactly at the center of any two obstacles.

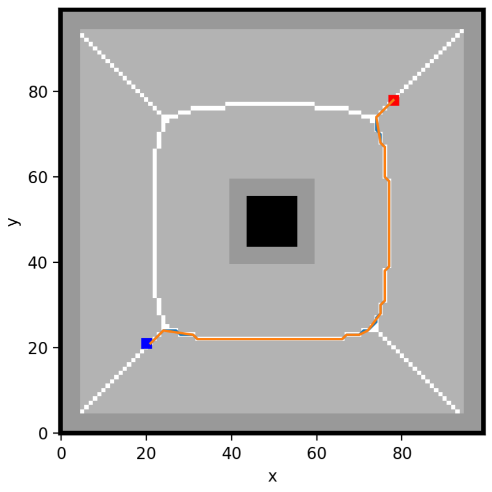

Fig. 7. A star

Below is the link to the video showing the safest path algorithm running on the robot.
Safest Path Video