CSE276C Math for Robotics HW 5

Ben Wang – PID A59025305

October 29, 2024

1 Question 1

To generate the configuration space (c-space) for the robot with a grid size of 2x2 and 5 deg in angular resolution, we can first simplify the space by changing the resolution of the map to make it easier to visualize. I made the original grid size of 2x2 world a unit (1x1) grid size world. Therefore, everything in the world should shrink accordingly, the map is now 400x150 and robot is 25x25. To compute the c-space for the robot, an easy way is to calculate the Minkowski sum of the robot and the wall & obstacles.For implementation, I simply compute the Minkowski sum of the robot and an object and the get the convex hull of that sum to get a new polygon. With the vertices of polygons, then I can fill those polygons to get the c-space for the robot. Furthermore, normally, c-space with 5 deg resolution should have $\frac{180}{5} = 36$ c-space with different orientation. To simplify this, I assume the robot to be circular with radius of $12.5\sqrt{2}$ and round it up to 18 to get a conservative simplification(shown as Fig.1.). For orientations 0, 45 and 90(0 and 90 should look the same), I just rotate the rectangular robot and compute the c-space as the method mentioned above.(shown as Fig.2. and Fig.3.)

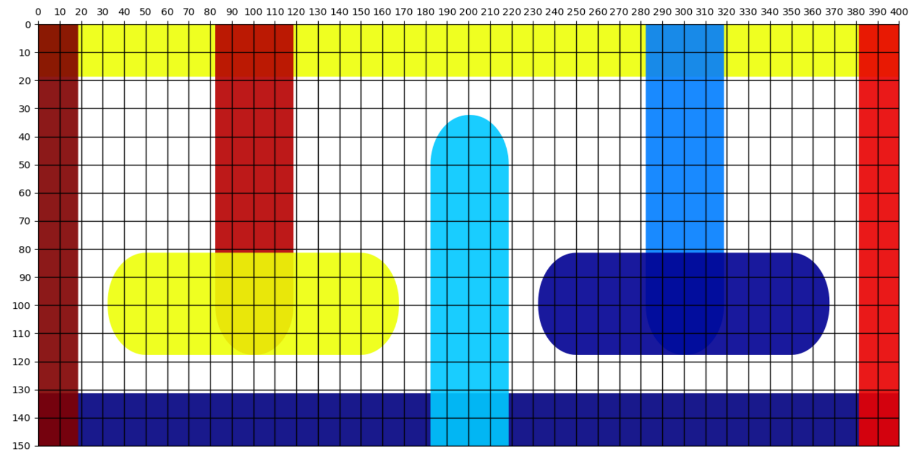

Figure 1: c-space

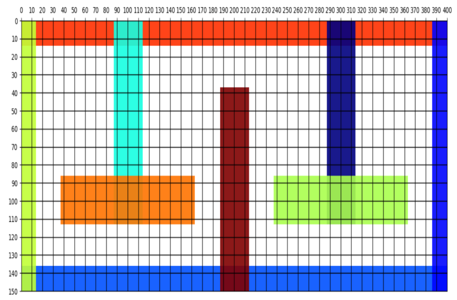
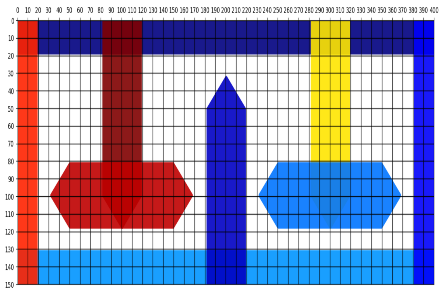

Figure 2: c-space 0deg and 45deg

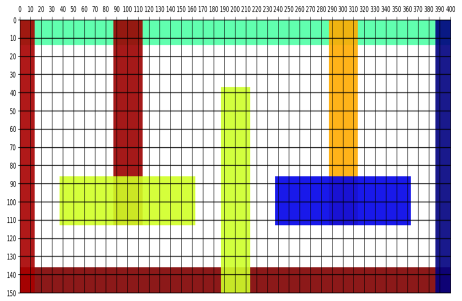

Figure 3: c-space 90deg

2 Question 2

First, as the reason mentioned before, the map and the robot are now 400x150 and 25x25 with grid size of 1x1. Thus, start-point (50,50) and end-point (750,50) are now (25,25) and (375,25). I use A* grid planning algorithm as the greedy search method and choose euclidean distance as my heuristic metrics. A* algorithm selects the path that minimizes $f(n) = g(n) + h(n)$ where n is the next node on the path, g(n) is the cost (distance) of the path from the start node to n, and h(n) is a heuristic function that estimates the cost of the cheapest path from n to the goal. Heuristic function h(n) in our case is the euclidean straight-line distance (i.e. compute norm of two points) to the goal. For implementation, I revised mostly from the A* example in pythonrobotics repo.The example covers most of the algorithm and some visualization (i.e. animation). I simply change the environment to ours and compute length of the path(in euclidean distance) and some metrics. Since A* algorithm is just a variation of Dijkstra algorithm, I also tried Dijkstra for planning(as shown in Fig.4.). Besides computing the euclidean distance of the path, I also calculate the orientation changing of the robot when following the path. The orientation changing is the angle changes from a vector compute from grid 1 and grid 2 and a vector compute from grid 2 and grid 3. Then divide the angle by 5 and count any angular changes in 5 deg as a step, meaning that if a robot turns 45 deg, the robot have to take 9 steps to do that. For the result, the path of A* algorithm transverses 639.563 grids (in euclidean distance) and changed orientation for 576 steps so the total cost for the path should be 1215.563 steps(grids) and it took 27 min to find the path. For Dijkstra algorithm, the path transverses 639.563 grids (in euclidean distance) and changed orientation for 369 steps and it took 35 min to find the path. Since the path from the starting points and the goal points are blocked by many obstacles, A* algorithm visit roughly as many as the exhaustive Dijkstra method so the transverse distance are the same, but Dijkstra did take few orientation changes to reach the goal.

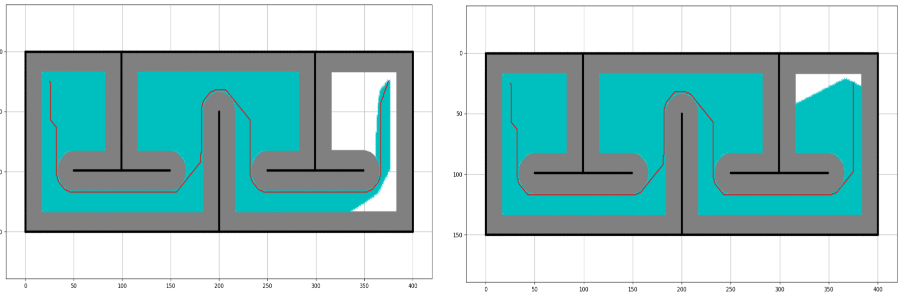

Figure 4: A* and Dijkstra planning algorithm

3 Question 3

To Compute the safest path from start to finish, I use Voronoi diagram to generate the safest road-map and use Dijkstra algorithm to find a path from the start-point (25,25) to end-point (375,25) on that road-map. For implementation, I also revised mostly from the VoronoiRoadMap example in pythonrobotics repo.The example covers most of the algorithm and the visualization. I changed the environment for our usage and compute length of the path(in euclidean distance) and some metrics. For any point in the Voronoi roadmap there are two or more closest points to the obstacle region. In other words, the Voronoi roadmap yields solutions for which a robot has maximum clearance. For the result, Voronoi roadmap planning algorithm transverses 770.6 grids (in euclidean distance) and changed orientation for 183.2 steps so the total cost for the path should be 953.8 steps(grids) and it took 67.78 s to find the path. The vertices of Voronoi diagram should stop at the intersection of gray area but resultant road map will be the same due to symmetry and will follow the path in the obstacle. However, if we consider the gray area to be hollow, the Voronoi diagram will be very complex and Dijkstra algorithm might not be able to find a path.

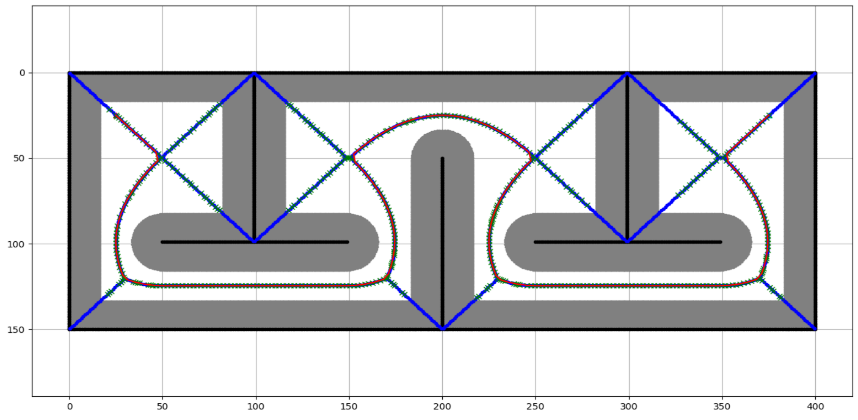

Figure 5: Voronoi road-map planning algorithm

4 Question 4

To Compute the path between start and end-points with probabilistic roadmaps (PRM), I also referenced the ProbabilisticRoadMap example in python-robotics repo. The example covers most of the algorithm and the visualization. I changed the environment for our usage, compute length of the path(in euclidean distance) and some metrics and tried different sample points. PRM first construct a roadmap, approximating the motions that can be made in the environment (generate sampling points in the free space). Then, it is connected to k nearest neighbors and connected start and goal configurations to the graph. Finally the path is obtained by a Dijkstra's shortest path algorithm. For the result, PRM planning algorithm (at sample points = 50) transverses 713.6 grids (in euclidean distance) and changed orientation for 106 steps, at sample points = 100 transverses 677 grids (in euclidean distance) and changed orientation for 111 steps, at sample points = 500 transverses 630.6 grids (in euclidean distance) and changed orientation for 138 steps. The results shows that the path length gets larger when sampled less points which is what we expected.

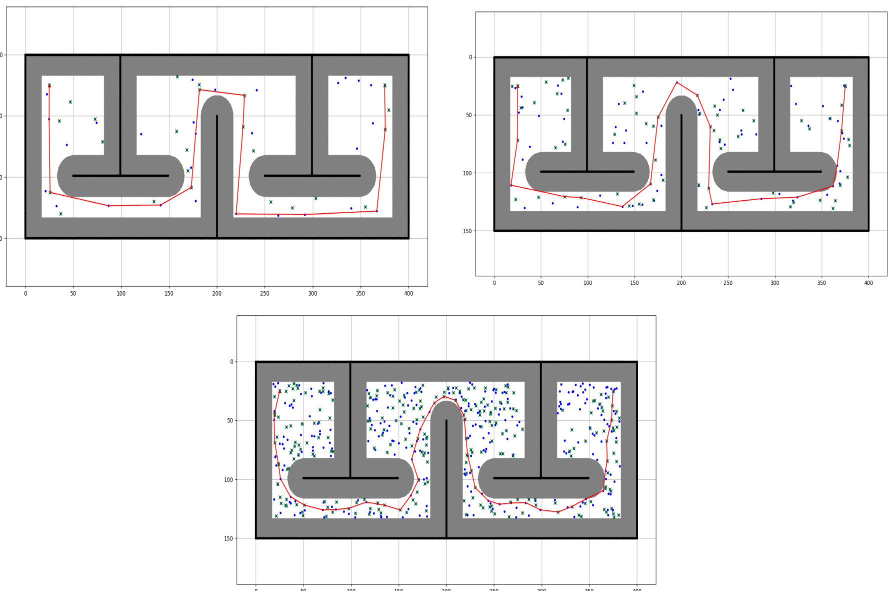

Figure 6: PRM with sampling points 50(top left), 100(top right) and 500(bottom)

5 Question 5

To Compute the path between start and end-points with Rapid exploring random trees (RRT), I also referenced the RRT example in pythonrobotics repo.The example covers most of the algorithm and the visualization. I changed the environment for our usage, compute length of the path(in euclidean distance) and some metrics and tried different maximum edge length. The main difference between RRT and PRM is that RRT is much faster since it is single-query problems but we can tell from the path that PRM is much more robust and usually have less path length. For the result, RRT planning algorithm (maximum edge length = 15) transverses 854 grids (in euclidean distance) and changed orientation for 401 steps, (maximum edge length = 30) transverses 835 grids (in euclidean distance) and changed orientation for 277 steps, (maximum edge length = 55) transverses 887.6 grids (in euclidean distance) and changed orientation for 295 steps.

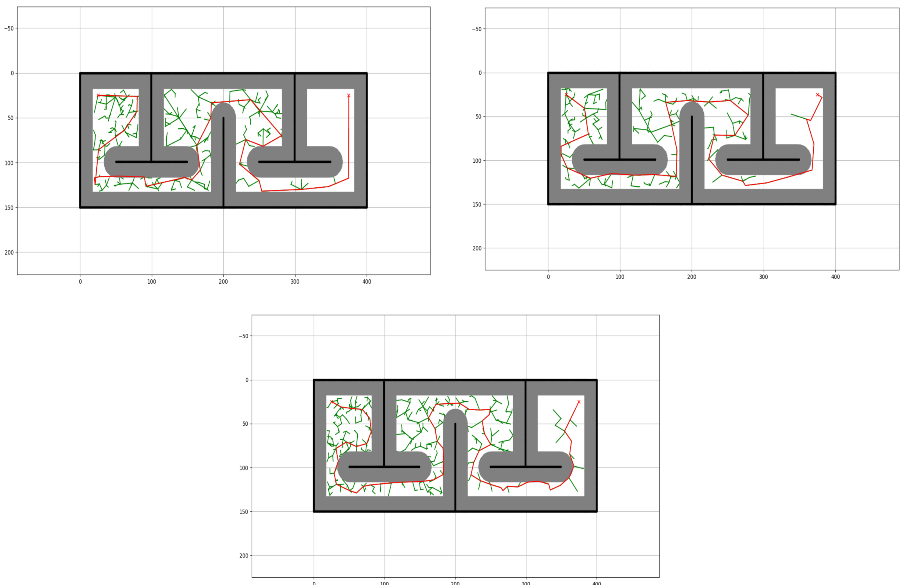

Figure 7: RRT with different maximu edge length 15(top left), 30(top right) and 55(bottom)

Code for ALL Questions