Robot Motion Planning - Homework III

David Akhihiero

October 8, 2024

1 Introduction

In this assignment, I created a grid-based representation of a 2D environment and implemented the wavefront algorithm over this grid. I also implemented the probabilistic roadmap (PRM), simplified probabilistic roadmap (sPRM), and rapidly-exploring random trees (RRT) algorithms for the same environment. All 3 algorithms plan paths in the roadmaps they generate from a start configuration to a goal configuration. I then compared the preprocessing time, path-finding time, and the costs of the paths from each algorithm for the same start and goal configurations.

2 Bitmap Creation

I wrote a function called $createGridBasedEnvironment()$ that created a matrix grid for the environment and filled each matrix cell with 1 or 0 depending on whether the cell is occupied. To check if a cell is occupied, the $isFree(q)$ function is used to check if any of the cell's four corner points are in the free configuration space. Figure 1 shows the bitmap for both a sparse and a dense environment for different resolutions $(0.1, 0.01, 0.002)$. The bitmap with 0.002 resolution sufficiently captures the environment details.

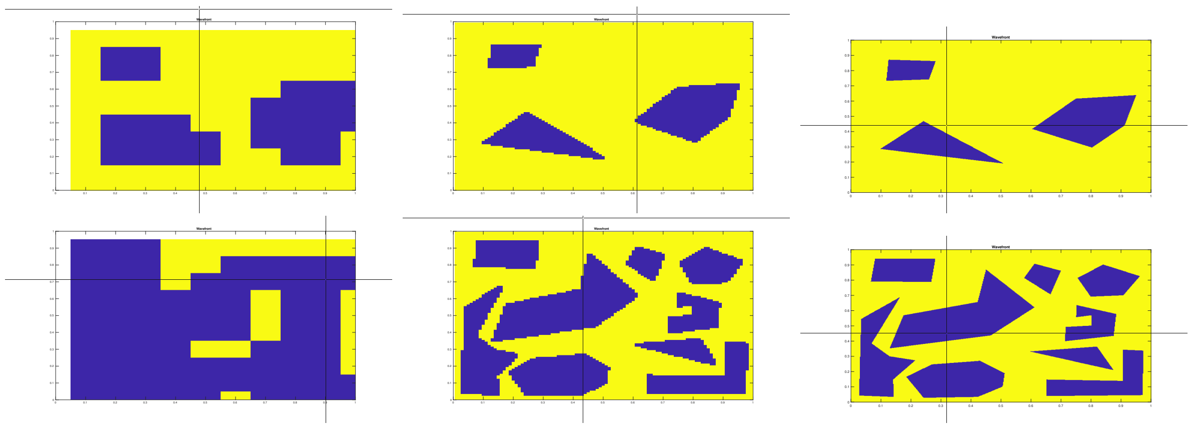

Figure 1: Grid-based representations of sparse and dense environments for different resolutions

3 Discrete Navigation Function (Wavefront)

I wrote a $wavefront()$ MATLAB function that implements the discrete navigation function over a grid environment. To implement the wavefront, my function used the breadth-first search (BFS) algorithm starting from a pre-specified goal ($[0.9, 0.9]$). A path from any start position specified by a user to the goal was planned by following the negative gradient of the navigation function. Example paths for both environments are shown in Figure 2

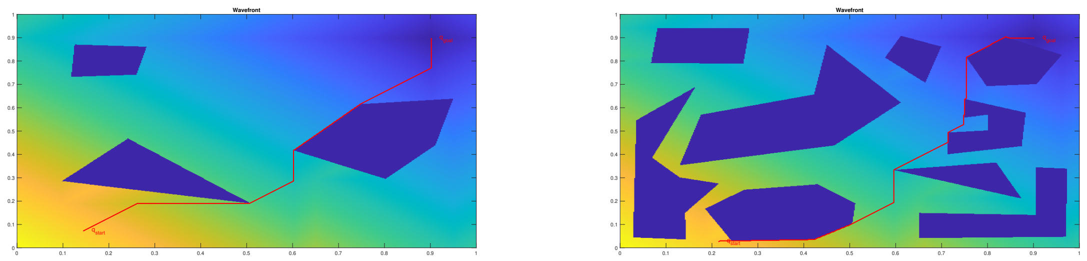

Figure 2: Paths in a wavefront grid from a start configuration (user-specified) to a goal configuration (pre-specified) for the sparse and dense environments for grid resolution 0.002.

4 Probabilistic Roadmap (PRM)

I wrote a $PRMLearning()$ MATLAB function that implements the pre-processing phase of the PRM algorithm (graph generation) following the algorithm shown in class. My function used the depth-first search (DFS) algorithm to check if a new node was in a connected component with each of the nodes closest to it. The graph was stored in a $PGraph$ object, the graph had 200 nodes for both environments, the radius used in the $Near()$ function was 0.2 and the seed for the random number generator was 7. I also wrote a $queryRoadmap()$ function to search a roadmap for a path from any start position to any goal position specified by a user. To add both q_{init} and q_{goal} to the graph, the function used the 5 nearest nodes ($K = 5$). A path from q_{init} to q_{goal} was computed using the A* algorithm implemented in the $MyAstar()$ function. Example paths for both environments are shown in Figure 3

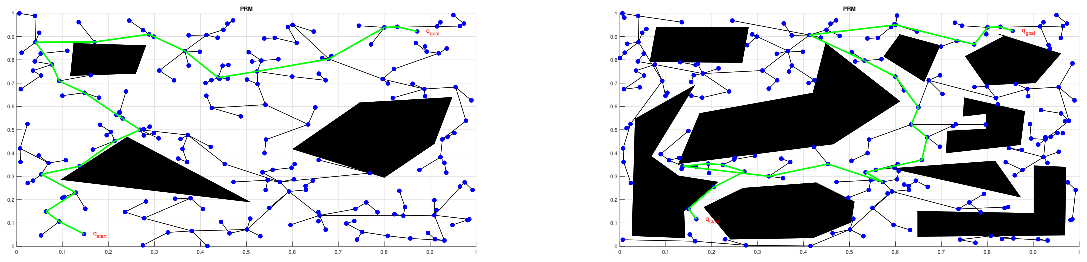

Figure 3: Paths in a PRM graph from a start configuration (user-specified) to a goal configuration (user-specified) for the sparse and dense environments.

5 Simplified Probabilistic Roadmap (sPRM)

I also wrote an $sPRMLearning()$ MATLAB function that implements the pre-processing phase of the sPRM algorithm (graph generation) following the algorithm shown in class. The graph was stored in a $PGraph$ object, the graph also had 200 nodes for both environments, the radius used in the $Near()$ function was 0.2 and the seed for the random number generator was 7. I used the $queryRoadmap()$ function to search a roadmap for a path from any start position to any goal position specified by a user. To add both q_{init} and q_{goal} to the graph, the function used the 5 nearest nodes ($K = 5$). A path from q_{init} to q_{goal} was computed using the A* algorithm implemented in the $MyAstar()$ function. Example paths for both environments are shown in Figure 4

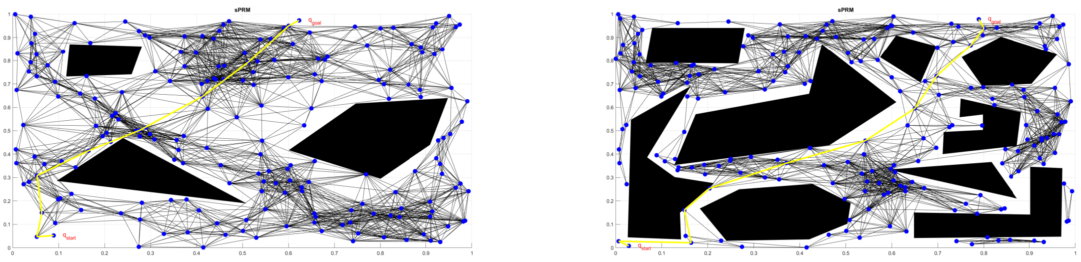

Figure 4: Paths in an sPRM graph from a start configuration (user-specified) to a goal configuration (user-specified) for the sparse and dense environments.

6 Rapidly-Exploring Random Trees (RRT)

I wrote an $RRT()$ MATLAB function that implements the RRT algorithm shown in class. The graph was stored in a $PGraph$ object, the function generated 800 nodes for both environments, the radius used for the goal region was 0.05, the $steer()$ function step size was 0.05 and the seed for the random number generator was 7. The function started the root of the tree from a q_{init} specified by a user. It also stored the parent and the cost of each node in the graph using the $PGraph$ $setvdata()$ function and this data was used to recover the path from the start node to the node in the goal region with the smallest cost from q_{init} to q_{goal} (also user-specified). Example paths for both environments are shown in Figure 5

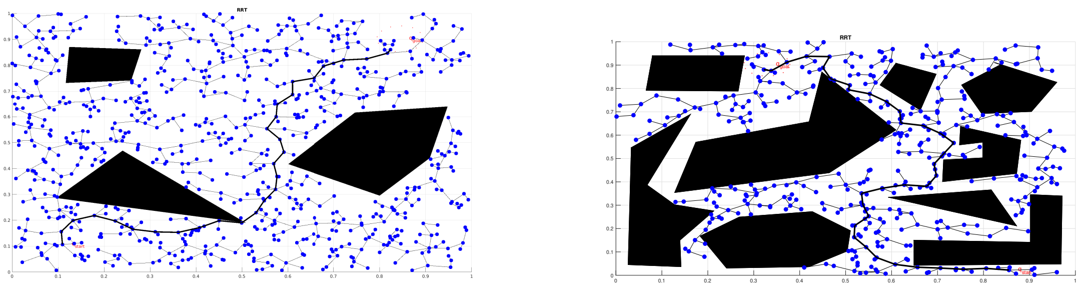

Figure 5: Paths in an RRT graph from a start configuration (user-specified) to a goal configuration (user-specified) for the sparse and dense environments.

7 Comparison between Algorithms

To compare these algorithms, I used each algorithm to plan a path from a start configuration ($[0.05, 0.03]$) to a goal configuration ($[0.9, 0.9]$) on a sparse environment and a dense environment and measured the time each algorithm took both for pre-processing and for path-finding and the length of the path returned. To make the comparison fair, I used the same seed for the probabilistic algorithms (7) and used a high resolution ($grid\ cell\ size = 0.002$ or 500×500) for wavefront. The number of nodes in both the PRM and sPRM graphs was 800 and 800 samples were generated for the RRT graph. The results for the sparse and dense environments are shown in Tables 1 and 2 and the paths with these algorithms are shown in Figures 6 and 7.

Table 1: Path Length and Computation Time in Sparse Environment

Algorithm	Path Length	Pre-processing Time (s)	Path Finding Time (s)
Wavefront	1.166000	20.172243	2.093563
PRM	2.291602	29.195734	0.029357
sPRM	1.294723	21.705937	11.848742
RRT	1.704615	0	0.198561

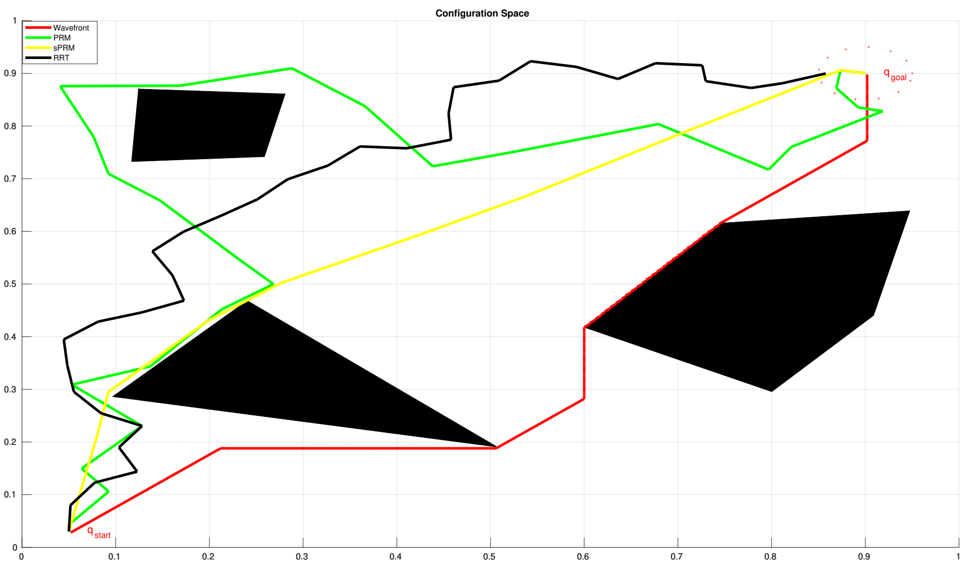

Figure 6: Paths in the configuration space from a start node to a goal for a sparse environment for all algorithms

From the results, the wavefront algorithm finds the most optimal path, the navigation function (pre-processing) takes the shortest time to compute (almost the same as the sPRM though), and path-finding takes the second longest time of all the other algorithms. The sPRM algorithm finds the next best path, with the longest time (pre-processing and path-finding) of all the algorithms, but as the number of nodes increases, I expect that the path found by the sPRM will approach found by the wavefront (since the sPRM is asymptotically optimal). Both the PRM and the RRT find poorer paths but in a much faster time. The RRT finds a better path than the PRM in a much shorter time (it also has no pre-processing phase).

Table 2: Path Length and Computation Time in Dense Environment

Algorithm	Path Length	Pre-processing Time (s)	Path Finding Time (s)
Wavefront	1.042000	43.199040	1.335937
PRM	2.431717	28.170000	0.023476
sPRM	1.319868	44.934151	6.427117
RRT	1.917256	0	0.334017

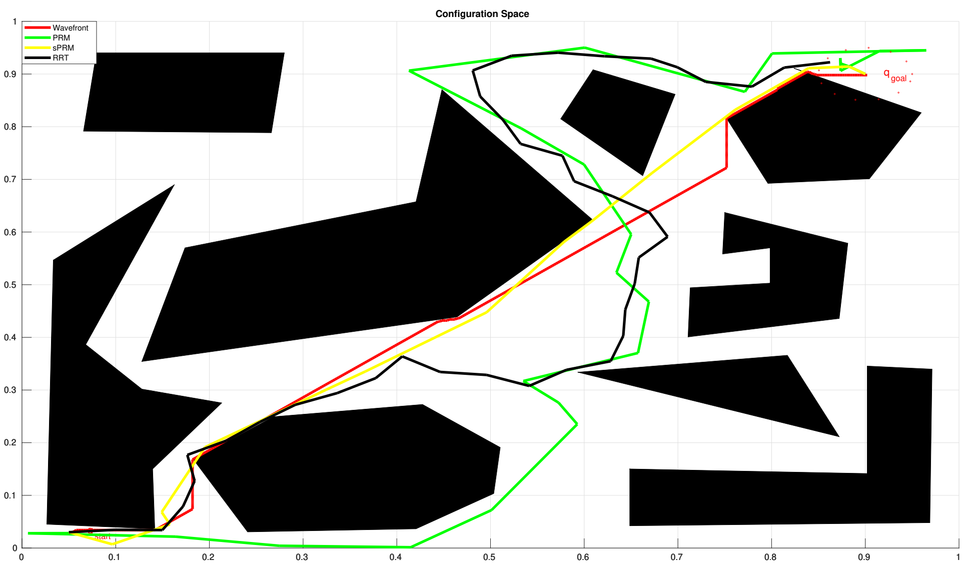

Figure 7: Paths in the configuration space from a start node to a goal for a dense environment for all algorithms

For the dense environment, the results show similar trends as in the sparse environment but this time, PRM has about the same pre-processing time as wavefront and sPRM has a longer pre-processing time than PRM. The path-finding time of wavefront does not change much. This makes sense since following the negative gradient should take roughly the same time irrespective of the environment's density. As expected, the time taken to pre-process increases as the density of the environment increases but for the sPRM, the time to find a path decreased significantly.