1 Summary of Approach

Table 1: Average results of Planning Algorithms

Algorithm	Average Planning Times (ms)	Success Rates	Average Number of Vertices Generated	Average Path Qualities
RRT	1084.1	90%	390.5	5.2536635
RRTConnect	0.15	100%	10.75	4.7141
RRT*	1226.6	90%	390.5	2.4381655
PRM	1440.95	100%	2000	3.8816725

1.1 RRT

In the case of the RRT, this algorithm performs in the mid range with an average planning time of 1 second and a success rate of 90%. However, out of all the planners this has the highest average path qualities as this suggests that the paths are generally very long and suboptimal paths. When the planning time is not exactly a critical factor this would be the best planner as it has an acceptable quality for the path itself. However, while this planner may seem highly efficient in terms of planning time the path quality itself is not the best among the planners, so where paths need a high quality to be efficient this would not be the most efficient. In order to improve this approach, a strategy that incorporates a bias to improve the high path quality could be an area of improvement.

1.2 RRT Connect

The RRT Connect according to the time results appears to be the best planner as it has a very low average planning time with a 100% success rate and a low number of vertices generated. Meaning that it did not take much time or exploration of the space to find the optimal path for the robotic arm to traverse. This is key when the goal here was to successfully plan a path in under 5 seconds. While this planner forms very well, I believe that the results may not be fully accurate as this was one of the implementations I expected to have a higher time than the RRT planner, as in the RRT Connect algorithm the space is being explored from both the goal and start state. Meaning that two trees are being constructed and then eventually connected. To improve this method, I would extend the planner to try and consider connections directly between the start and goal state, depending on the complexity of the environments and the obstacles, this way instead of sampling and continuously extending towards the nodes generated from either side of the tree, a path can be found immediately. A post-processing solution could help with improving this planning performance like smoothing path (potentially with some kind of moving average filter).

1.3 RRT*

The RRT* planner has an average planning time of 1.23 seconds, a very high success rate as well and generates the same number of vertices as RRT (expected as they use the same vertex generating strategy). In terms of path quality it has a better quality than the RRTConnect algorithm and is able to balance between the planning time and path quality as well. This was to be expected with the RRT* algorithm, as essentially it is rewiring the path to find a path that is the least costly, i.e. even once a path is found this path will be guaranteed to be the least cost path

as the RRT* algorithm takes the cost into consideration when constructing the path. Versus its counterparts RRT and RRTConnect do not have this sort of approach for the path planning. The only room for improvement would be in the planning time itself, by iteratively rewiring this could help improve the optimality of the solution as well as experimenting with different parameter tunings.

1.4 PRM

This planner works very efficiently in terms of planning time and successful generation within the correct amount of time, but does so with a rather high number of vertices generated at 2000. This might not be optimal for real-time planning, as a large number of vertices could potentially affect the path quality. By adjusting the roadmap size this could help to improve the performance, but this would depend on the complexity of the given environment. An area of improvement would be to add collision checking in the construction of the roadmap, this could help improve the path quality, and also properly manage the memory, as the PRM will not use all the vertices and "connections" in the original roadmap. The collision checking would ensure only valid paths are in the roadmap.

1.5 Parameter Selection and Tuning

Goal Biasing

The goal biasing was implemented in the RRT planners as this helped to ensure that the goal state would end up in the path eventually. As I sampled the goal state directly at least 10% of the time and was able to connect nodes/find paths closer to it more efficiently.

Epsilon and Search Radius

The epsilon value allowed me to tune the exploration of the configuration space but when I made this value too high (i.e. 0.8) this caused suboptimal paths to be produce

Sample Size

The tuning of the sample size was largely based on how long it took my planners to find a solution. Initially I was running with only 10,000 samples and this did not seem like there were enough samples being drawn, and additionally this was prior to implementing goal biasing so the goal sample would often not be included in the final path. By tuning this value I ensured that the samples taken were efficient for the planners to run.

Roadmap Size

The size of the PRM roadmap impacts the computation time and path quality, as I noticed that a larger roadmap can help with the coverage but also impacts the planning time. To tune this I experimented with a range of different values and observed how this impacted the planning times and path quality.

Evaluate Plan Function (How it Works)

2 Compiling Code

To compile the code simple run the commands provided in the homework prompt:

1. g++ planner.cpp -o planner.out -g
2. ./planner.out map1.txt 5 1.57,0.78,1.57,0.78,1.57 0.392,2.35,3.14,2.82,4.71 2 myOutput.txt
3. python3 visualizer.py myOutput.txt --gifFilepath myGif.gif

To print results uncomment the print and clock/time code in the planner functions.