16-782: Planning and Decision-making in Robotics

Name: Kshitij Kabeer, Andrew ID: kkabeer

Introduction

Test Cases

To test the planning algorithms properly, I calculated the joint angles for 5 end effector points. I chose these points so that each region of the map that is difficult to reach is covered. Hence, even though the points are random, they are in specific pockets of the map, as is shown below.

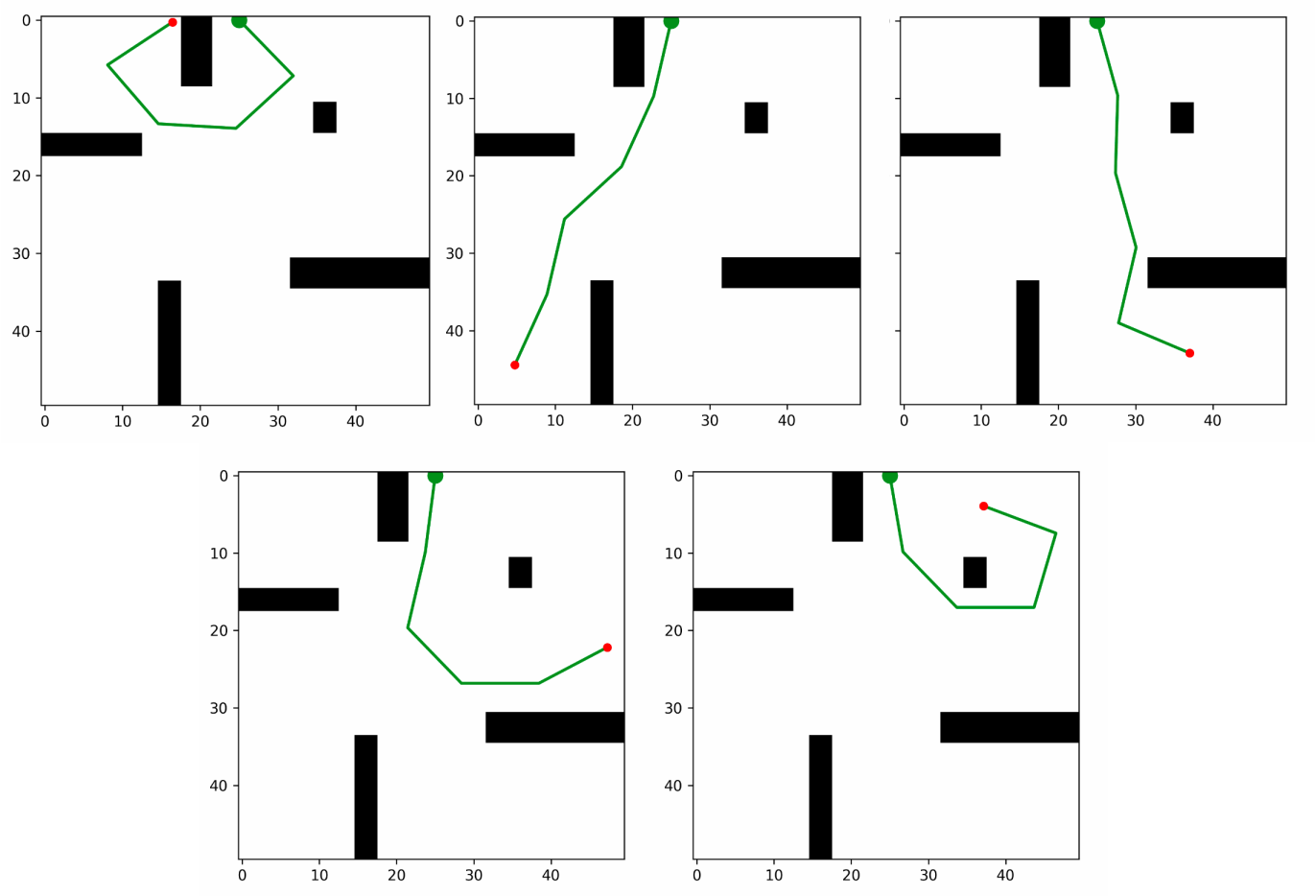

Figure 1: The 5 chosen end effector locations along with the joint angles visualization

As can be seen from the figure, all these points are for 5DOF arms. Here are the corresponding joint angles used, in radians (from left to right, top to bottom in the figure):

1. 0.8,2.4,3.2,4.0,5.7
2. 1.8,2.0,2.4,1.8,2.0
3. 1.3,1.6,1.3,1.8,0.4
4. 1.7,1.8,0.8,0.0,5.8
5. 1.4,0.8,0.0,5.0,3.5

Using these 5 points, I generated all possible permutations for getting the start and goal location - which is exactly 20 permutations, and then I ran the planners on it.

Results

The results are summarized in the following table

Data → Planner ↓	**Average Planning Time (seconds)**	**Number of times planning was done in under 5 seconds**	**Average Number of Vertices Generated***	**Average Path Cost**
RRT	0.863	20	1194	16.24495559
RRT Connect	0.059	20	247	16.95831853
RRT*	10.280	0*	7148	14.20479633
PRM	3.791	20	1649	19.608

* This number indicates the actual number of vertices that are inserted into the tree, not the total number of iterations run. The number of iterations run is higher than this, since in many iterations a vertex doesn't get added because it or the path to it is in collision.

As you can see from the table, the RRT Connect algorithm has the least running time, as it needs to generate far fewer vertices to reach the goal. RRT* takes the most time, but also gives the best cost. However, one thing to note here is that planning time for RRT* can be reduced to under 5 seconds if desired, just by decreasing the number of iterations. I have just chosen it arbitrarily to be 10000. PRM has the highest path cost, and execution time is also high. The averages don't tell the whole picture, as for some of the problems the time required was much higher/lower than the average. Please refer to the appendix to see the full results.

The hyper-parameters chosen are the following:

1. Resolution* = 0.02 radians/cells
2. Maximum number of iterations for RRT, RRTConnect and RRT* = 10000
3. Epsilon Distance**: 5 cells
4. Seed: 34234
5. Maximum Neighboorhood distance**: 10 cells
6. Maximum PRM Edge distance**: 50 cells
7. Maximum number of PRM Iterations = 25000

The main hyperparameters I experimented with the most were Number of Iterations, Neighborhood Distance, and Edge Distance. The maximum number of iterations didn't have much effect on RRT and RRT Connect, since those algorithms converged way before the cap was reached. But RRT* and PRM were significantly affected. Increasing the max iterations for RRT* above 10000 led to exponential increase in planning time, since each iteration also starts taking more time as the tree grows. Lowering the number of iterations leads to more suboptimality in paths, but faster convergence. The same goes for the PRM iterations. 10000 and 25000 seemed to be the sweet spot for RRT* and PRM respectively.

The neighborhood distance was another factor that affected the RRT* algorithm significantly. Increasing the neighborhood distance led to a substantial slow down. It didn't affect the execution times when the number of iterations were low, but as RRT* got to the higher number of iterations, it was significantly slowed down if the neighborhood distance was much higher than 10.

For PRM, the Edge distance affected the quality of paths it generated. If the Edge distance was small, a lot of orphaned nodes and disconnected graphs were generated, and hence, it was not able to find a feasible solution in many cases. 50 seemed to be a good value, as it didn't slow down the execution too much and PRM was also able to find the result for all test cases.

* I do planning in discrete space, and then convert to Continuous
** My distance measure is the L∞ norm. The way I extrapolate a point is by extrapolating the joint with the maximum change in angle by epsilon units, and then extrapolating the other joints by the same fraction as the joint with the maximum change in angle.

Suitable Planner for the Problem

I think the type of planner that is best suited for this environment depends on what is desired.

Case1: Single Shot Planning on a low time budget

For this case, the RRT Connect algorithm works best, as it generates the path in far less time than the rest of the planners, and the cost isn't too bad either. RRT Connect works well because the tree grows from both the start and the goal, and hence, the path is found quickly even if the goal is in a very secluded location.

Case2: Single Shot Planning on a high time budget

For this case, RRT* algorithm works best. Even though it takes a long time, it leads to a significant reduction in cost. Also, the number of vertices generated can be varied according to the time budget that planner has, trading off cost for the same.

Case3: Multi Shot Planning with high time budget in the beginning

Even though the PRM method looks bad from the table (it takes a long time and also generates the worst path), for this type of planning problem, the PRM method works best. The average execution time in the results table seems high, however, a large portion of the execution time is taken by the PRM graph generation. Only a small fraction of the time is required for the actual path planning using A*. Once the PRM graph is generated, the actual planning only takes a few milliseconds. Again, the number of vertices in the PRM can be varied according to the time budget available at the beginning, by trading off cost.

Issues with the Planner

- The RRT* planner is very slow, compared to the other methods.
- The PRM method is also slow, and the cost is not good as well.

Ways to Improve the Planner

Some optimizations can be made to make the planner work faster. This will not have a large effect on RRT and RRT* since they are already very fast, but can mean a significant time reduction in RRT*. For example, using an efficient data structure to store the nodes, like a k-d tree, will help in finding neighbors much faster than the currently used data structure (hash map). Also, instead of sampling points from a uniform distribution, if biased sampling (like mentioned in the extra credit 1) or gaussian sampling methods (taught in class) are used, that will also lead to faster execution and better path cost. The same optimizations can be applied to PRM and the graph building time and path cost can be improved by using K-D trees and biased sampling.

Appendix:

Here is a list of all test cases and all results

Test Cases:

Format is (**problemIndex** : **Start** -> **Goal**)

0 : 0.8,2.4,3.2,4.0,5.7 -> 1.8,2.0,2.4,1.8,2.0
1 : 0.8,2.4,3.2,4.0,5.7 -> 1.3,1.6,1.3,1.8,0.4
2 : 0.8,2.4,3.2,4.0,5.7 -> 1.7,1.8,0.8,0.0,5.8
3 : 0.8,2.4,3.2,4.0,5.7 -> 1.4,0.8,0.0,5.0,3.5
4 : 1.8,2.0,2.4,1.8,2.0 -> 1.3,1.6,1.3,1.8,0.4
5 : 1.8,2.0,2.4,1.8,2.0 -> 1.7,1.8,0.8,0.0,5.8
6 : 1.8,2.0,2.4,1.8,2.0 -> 1.4,0.8,0.0,5.0,3.5
7 : 1.3,1.6,1.3,1.8,0.4 -> 1.7,1.8,0.8,0.0,5.8
8 : 1.3,1.6,1.3,1.8,0.4 -> 1.4,0.8,0.0,5.0,3.5
9 : 1.7,1.8,0.8,0.0,5.8 -> 1.4,0.8,0.0,5.0,3.5
10 : 1.8,2.0,2.4,1.8,2.0 -> 0.8,2.4,3.2,4.0,5.7
11 : 1.3,1.6,1.3,1.8,0.4 -> 0.8,2.4,3.2,4.0,5.7
12 : 1.7,1.8,0.8,0.0,5.8 -> 0.8,2.4,3.2,4.0,5.7
13 : 1.4,0.8,0.0,5.0,3.5 -> 0.8,2.4,3.2,4.0,5.7
14 : 1.3,1.6,1.3,1.8,0.4 -> 1.8,2.0,2.4,1.8,2.0
15 : 1.7,1.8,0.8,0.0,5.8 -> 1.8,2.0,2.4,1.8,2.0
16 : 1.4,0.8,0.0,5.0,3.5 -> 1.8,2.0,2.4,1.8,2.0
17 : 1.7,1.8,0.8,0.0,5.8 -> 1.3,1.6,1.3,1.8,0.4
18 : 1.4,0.8,0.0,5.0,3.5 -> 1.3,1.6,1.3,1.8,0.4
19 : 1.4,0.8,0.0,5.0,3.5 -> 1.7,1.8,0.8,0.0,5.8

Results:

Planner 0 is RRT
Planner 1 is RRT Connect
Planner 2 is RRT*
Planner 3 is PRM

planner	mapName	problemIndex	numSteps	numVerticesGenerated	cost	timeSpent	success
0	./map2.txt	0	38	140	14.38	0.021887572	TRUE
0	./map2.txt	1	47	202	18.06	0.041618119	TRUE
0	./map2.txt	2	63	772	22.88	0.39273485	TRUE

0	./map2.txt	3	58	4114	19.72	3.272379045	TRUE
0	./map2.txt	4	25	81	10.16	0.01541836	TRUE
0	./map2.txt	5	24	494	12.54318531	0.26155212	TRUE
0	./map2.txt	6	53	3254	18.18	2.417712558	TRUE
0	./map2.txt	7	61	565	19.3	0.301580795	TRUE
0	./map2.txt	8	60	4445	18.78	3.652502217	TRUE
0	./map2.txt	9	84	5603	26.96	4.99123754	TRUE
0	./map2.txt	10	37	1418	16.22318531	0.627570592	TRUE
0	./map2.txt	11	52	406	18.42	0.139899042	TRUE
0	./map2.txt	12	64	367	21.56	0.112653413	TRUE
0	./map2.txt	13	14	92	11.44318531	0.04029326799	TRUE
0	./map2.txt	14	33	186	10.92	0.031136363	TRUE
0	./map2.txt	15	45	109	15.74	0.029623536	TRUE
0	./map2.txt	16	9	82	9.543185307	0.035081545	TRUE
0	./map2.txt	17	10	22	6.143185307	0.009513331002	TRUE
0	./map2.txt	18	9	82	9.143185307	0.03689195	TRUE
0	./map2.txt	19	72	1446	24.8	0.837904093	TRUE
1	./map2.txt	0	66	330	20.82	0.03520544399	TRUE
1	./map2.txt	1	72	291	24.34	0.036739601	TRUE
1	./map2.txt	2	58	303	20.04	0.036978777	TRUE

1	./map2.txt	3	40	121	17.2	0.031563685	TRUE
1	./map2.txt	4	48	163	13.36	0.015551845	TRUE
1	./map2.txt	5	31	107	12.1	0.014384326	TRUE
1	./map2.txt	6	26	493	13.46	0.207790673	TRUE
1	./map2.txt	7	43	94	13.18	0.013731821	TRUE
1	./map2.txt	8	30	286	14.1	0.080914611	TRUE
1	./map2.txt	9	47	261	19.04318531	0.080200758	TRUE
1	./map2.txt	10	61	226	18.14	0.02487912	TRUE
1	./map2.txt	11	54	252	19.5	0.031177782	TRUE
1	./map2.txt	12	72	217	22.88	0.026057331	TRUE
1	./map2.txt	13	52	352	20.76	0.14296364	TRUE
1	./map2.txt	14	50	213	13.44	0.021442551	TRUE
1	./map2.txt	15	48	123	16.94	0.014037573	TRUE
1	./map2.txt	16	26	210	13.62	0.04566879	TRUE
1	./map2.txt	17	39	87	12.86	0.010994785	TRUE
1	./map2.txt	18	28	414	14.54	0.162500469	TRUE
1	./map2.txt	19	56	399	18.84318531	0.156084935	TRUE
2	./map2.txt	0	28	7087	13.34	9.471079875	TRUE
2	./map2.txt	1	37	7087	16.94	10.67948863	TRUE
2	./map2.txt	2	43	7087	18.64	10.51895068	TRUE
2	./map2.txt	3	43	7087	17.64	10.17027809	TRUE

2	./map2.txt	4	15	7406	8.68	11.42732768	TRUE
2	./map2.txt	5	21	7406	12.10318531	11.67907512	TRUE
2	./map2.txt	6	43	7406	16.74	10.91078612	TRUE
2	./map2.txt	7	21	7434	10.46318531	11.68998878	TRUE
2	./map2.txt	8	49	7434	17.02	10.92666873	TRUE
2	./map2.txt	9	66	7396	24.84	11.1210553	TRUE
2	./map2.txt	10	34	7406	13.38	10.19920298	TRUE
2	./map2.txt	11	43	7434	17.54	10.18260777	TRUE
2	./map2.txt	12	49	7396	17.2	10.28544366	TRUE
2	./map2.txt	13	11	6416	11.32	8.212983802	TRUE
2	./map2.txt	14	26	7434	10.24	10.4130568	TRUE
2	./map2.txt	15	20	7396	10.74	10.2345917	TRUE
2	./map2.txt	16	7	6416	9.543185307	8.11943988	TRUE
2	./map2.txt	17	11	7396	6.663185307	11.51761332	TRUE
2	./map2.txt	18	7	6416	9.143185307	9.021800725	TRUE
2	./map2.txt	19	52	6416	21.92	8.817143073	TRUE
3	./map2.txt	0	11	1649	22.34	3.654377414	TRUE
3	./map2.txt	1	13	1649	26.86	3.805486134	TRUE
3	./map2.txt	2	13	1649	26.4	3.519590733	TRUE
3	./map2.txt	3	11	1649	24.12	3.394788993	TRUE
3	./map2.txt	4	7	1649	10.52	4.090861917	TRUE

3	./map2.txt	5	9	1649	15.18	3.913892689	TRUE
3	./map2.txt	6	10	1649	16.02	3.733469123	TRUE
3	./map2.txt	7	10	1649	19.1	4.082144208	TRUE
3	./map2.txt	8	10	1649	17.18	3.990214536	TRUE
3	./map2.txt	9	13	1649	22.12	3.623824727	TRUE
3	./map2.txt	10	11	1649	22.46	3.697532663	TRUE
3	./map2.txt	11	13	1649	27.5	3.768912748	TRUE
3	./map2.txt	12	11	1649	25.88	3.57697841	TRUE
3	./map2.txt	13	11	1649	23.16	3.481601665	TRUE
3	./map2.txt	14	7	1649	10.4	4.234554128	TRUE
3	./map2.txt	15	8	1649	13.02	3.930972629	TRUE
3	./map2.txt	16	9	1649	14.7	3.69920103	TRUE
3	./map2.txt	17	9	1649	16.06	4.107871274	TRUE
3	./map2.txt	18	9	1649	16.9	3.982932312	TRUE
3	./map2.txt	19	12	1649	22.24	3.530430796	TRUE