Homework 2 16782: Planning and Decision Making in Robotics

Harsh Dhruva
AndrewID: hdhruva
Planning for a high-DOF planar arm

October 26th, 2021

1. Implementation

The planning ID's are assigned as follows
0 – RRT
1 – RRT Connect
2 – RRT *
3 – PRM

Run the code with the same instructions as shown in the writeup. This generates the following results for map 1.

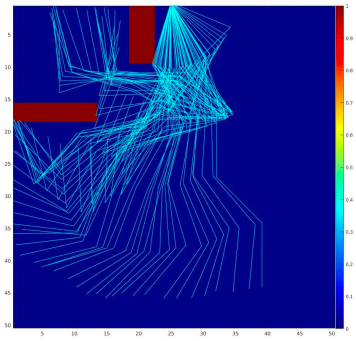

Figure 1: RRT - map 1

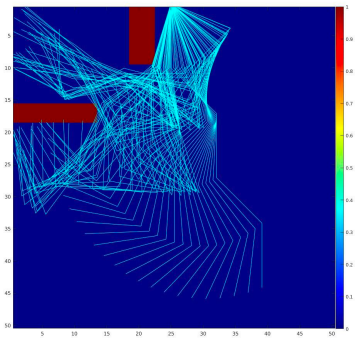

Figure 2: RRT-Connect map 1

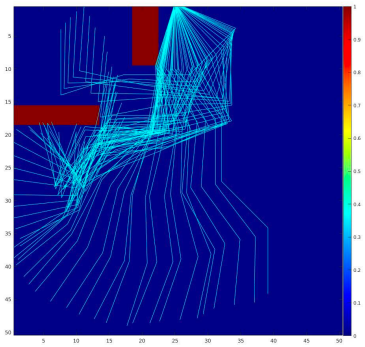

Figure 3: RRT map 1*

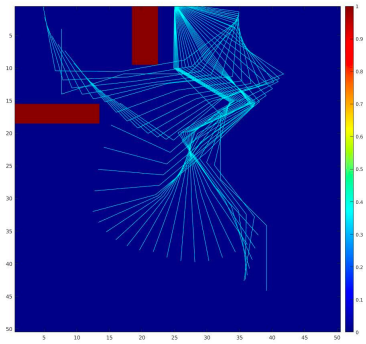

Figure 4: PRM map 1

RRT, RRT-Connect, RRT*

These three algorithms were implemented within the RRTs Class, to reuse functions between them. The BuildRRT, BuildRRTConnect, and BuildRRTStar functions build the respective trees and calculate the path. Random samples are generated 85% of the time, and 15% of the time the samples are generated within a goal region.

PRM

This algorithm has been implemented within the ProbabilisticRoadmap Class. Here the graph is built in the RoadMap function, after which the ConnectStartandGoal function is used to connect the Start and Goal to the graph. Then the Query function is used to run A* to obtain a path.

I have commented other implementation details in the code itself. I followed the pseudocode in the Slides very closely while implementing the algorithms, and several function names correspond to the ones in the pseudo code. Vectors were highly used to store states, nodes in the graph, edges, and to iterate over to find neighbors, etc.

2. Results

The metrics that have been compiled are the following:

1. Average Planning Time – Time for each planner to execute. In the PRM, however I generate a Roadmap every time it runs, and this time accounts for that. Ideally, a graph will be generated, which will be the one used to connect different start and goal configurations, however I have not accounted for that in my implementation.

2. Success Rate for generating solutions in 5 seconds

3. Average number of vertices generated – In a tree for the RRTs, and a graph for the PRM

4. Average Path Qualities – This is calculated as the total distance travelled during the path, which is the sum of RMS distances between adjacent robot configurations in the path. The lower the quality the better, since it indicates a lower distance travelled.

5. Path Length – The number of configurations of the robot in the path.

6. Std Dev. in Path Quality – The deviation in the path quality for the same start and end goal configurations. This gives an idea about the consistency of quality in the solutions for multiple probabilistic runs.

Table 1: Average Statistics for Map 2

	Avg. Time Taken (s)	**Success Rate 5 secs (%)**	**Avg. No. of Vertices**	**Avg. Path Quality**	**Avg. Path Length**	**Std Dev. in Path Quality**
RRT	1.811987	89.47	2918	9.8888015	61	3.4051536
RRT-Connect	0.720067	90	2275	11.242990	67	5.3470965
RRT*	2.250503	90	2855	9.4038910	79	3.2622628
PRM	0.921385	100	9780	8.6525680	9	2.1169340

These results were generated for parameters that were held the same for the RRTs, in terms of number of samples, epsilon, and neighborhood radius. We can see that RRT, and RRT* take the maximum amount of time to run, and that RRT-Connect is the fastest amongst the RRTs. RRT* builds completely on RRT, with additional rewiring, and therefore it makes sense why it takes longer to run on average. Here PRM is running fast, however, the graph is being re-generated each run, and if were to only use a precomputed graph for all the runs, PRM would be the fastest planner. The number of samples to draw from in each algorithm were selected to 150000, and we have the number of vertices to be the highest in PRM. The number of vertices generated were lowest in RRT-Connect, but just as RRT, it does not give an optimal solution. RRT-Connect is significantly faster than RRT and RRT*, however the speed compromises the path quality.

The path quality amongst the RRT's is lowest for RRT*, which is expected due to the re-wiring phase, as it produces a more optimal path. Here PRM produces the best path quality, however the path length is small. For map 2, I had to loosen some of the parameters in the PRM, especially in the Connect function to find a solution. The standard deviation in the path quality indicates the consistency of the solution. Here RRT-Connect is the worse, since with speed, we sacrifice consistency and optimality. PRM does consistently well in terms of path quality, and RRT and RRT* are more consistent than RRT-Connect.

The results for the 20 runs from which the statistics above were calculated are in the Appendix. The avg. path length for the PRM does not consider the interpolated states between.

3. Discussion

RRT:

This planner as seen from the results above generates a solution within 5 seconds 89% of the time, and is therefore slow, with a higher number of vertices in the tree even. The solution obtained is not optimal, however, the average length of the path is the shortest.

- For Map 2, with several obstacles, it seems quite suitable to use RRT over PRM.
- Issues include suboptimality of the solution, and no guarantee that the solution will reach the goal state exactly, but just within a goal neighborhood. The growth is always biased by the largest unexplored region.
- One tweak I made was to generate 15% of the random samples, within a region near the goal to drive the search in that direction. Other improvements to RRT in a sense are it's further variations. Implementation improvements can definitely be made in terms of calculating neighborhood, and data structures for finding nearest states in the tree, etc. Post processing should be done to improve results.

RRT-Connect:

This is the fastest of the RRTs, but gives a suboptimal path, with a higher cost, and lesser consistency. The number of vertices is the smallest. This algorithm is probabilistically complete and can find a solution, that is not necessarily within a goal region, but the goal itself, since trees are being extended from start and goal.

- This can be used in an environment where we value speed over the path optimality. This could really depend on several factors and the planning problem we are trying to solve. In Map 1, since there were not as many obstacles, I would choose to use RRT-Connect, however in Map 2, the paths were suboptimal.
- Issues include the suboptimality of the solution, and the compromise on path quality.
- Here goal-biasing is not as useful, and other sampling tricks could be used to influence the performance. Other implementation strategies can be used to improve the planner. Post processing of the tree will yield in more optimal paths.

RRT*

This is the slowest of the algorithms, since it involves a re-wiring step.

- For Map 2, this is the best one to use since it will calculate optimal paths. We would use this planner if we are not worried about the extra time it takes to execute, and need more optimal paths.
- The issue is with the extra time it takes to execute. RRT* is much more expensive, and takes longer to get to the first solution.
- Post processing, and other improvements as mentioned in RRT.

PRM

This algorithm was difficult to implement, and it does not work quite well with my implementation. For obtaining results in Map 2, I had to relax parameters.

- This is best used for one shot planning, and if used with uniform sampling does not work as well on Map 2, compared to Map 1, due to the number of obstacles.
- There were several issues with my implementation. The uniform sampling can lead to some passages being missed when too many obstacles are present, which leads to loss of connectivity. I also implemented the neighbors as the nearest neighbors within a radius, which can definitely be improved.
- In terms of improving the planner, I would consider bias of samples near obstacle boundaries, and data structure implementations for k-nearest neighbors, instead of all the neighbors in a radius. Even selecting K vertices from each component could be a strategy to employ.

4. Appendix

	Time Taken (s)			
	RRT	**RRT-Connect**	**RRT***	**PRM**
1	7.44761	0.000697	14.8004	1.86622
2	0.010245	0.004512	0.012971	2.70989
3	0.003682	0.002398	0.027833	2.10829
4	0.0013	0.00099	0.001889	2.10773
5	0.003549	0.006259	0.003655	0.650296
6	0.000937	0.002079	0.056305	0.638424
7	12.6719	6.36045	13.141	0.606895
8	0.000941	0.000272	0.001819	0.564102
9	0.849594	0.983186	0.723164	0.527365
10	0.574768	0.000119	0.005835	0.578476
11	0.000148	5.40E-05	0.000397	0.542255
12	0.25833	0.033868	0.162877	0.63687
13	0.22633	0.050148	2.17701	0.531692
14	13.6911	6.84758	13.573	0.604076
15	0.001384	0.001788	0.00171	0.514949
16	0.080483	0.087628	0.24917	0.576272
17	0.005436	0.01331	0.007433	0.593762
18	0.409926	0.000235	0.058515	0.669456
19	0.001409	0.004361	0.004261	0.851299
20	0.000671	0.001407	0.000817	0.549391

Vertices			
RRT	**RRT-Connect**	**RRT***	**PRM**
11276	114	8419	9633
517	512	322	9791
342	313	929	9683
187	182	234	9683
238	494	224	9791
94	328	1140	9708
15402	16826	15333	9733
81	79	134	9780
1116	2422	1462	9862
3793	49	421	9827
42	20	101	9810
2362	1537	1954	9711
2197	1696	6503	9910
15399	17336	15539	9830
138	183	137	9873
1386	2278	2343	9862
381	678	276	9874
3216	71	1305	9854
88	192	223	9683
109	196	111	9708

Path Quality			
RRT	**RRT-Connect**	**RRT***	**PRM**
12.6281	5.91527	12.83	5.3928
12.8455	13.1578	11.6602	8.53843
14.6014	12.7461	14.0639	10.7058
10.6819	9.06954	9.44242	8.2238
10.5521	11.0176	10.6201	9.5445
9.40888	15.3362	13.5197	11.605
14.0656	20.9668	15.4906	9.81311
6.17345	4.15934	5.75081	7.46341
7.78325	18.1103	7.24807	7.76861
9.43785	5.04989	6.32242	8.11478
4.47537	3.26471	4.55562	8.74498
11.938	17.3382	10.967	9.10787
8.28172	9.52885	8.57125	6.70628
4.62516	14.1229	4.11766	6.28593
9.73568	10.0756	8.40447	9.14992
16.6349	21.0099	12.3891	13.8203
12.3533	10.5379	10.3739	9.23345
5.7739	6.08262	6.37516	4.40278
6.41632	6.66659	7.37	9.98165
9.36365	10.7037	8.00544	8.44796

Path Length			
RRT	**RRT-Connect**	**RRT***	**PRM**
80	37	97	6
81	84	76	8
92	80	90	11
67	57	78	9
66	69	73	9
59	97	106	11
78	80	87	9
39	26	45	8
38	100	133	8
59	31	41	8
28	20	30	9
75	110	122	9
52	60	153	7
28	74	26	8
61	63	54	10
105	134	143	13
78	66	66	9
35	38	40	6
40	43	68	10
59	67	53	9
61	66.8	79.05	8.85