Homework 2

CMU 16-782 : Planning and Decision Making in Robotics

1 Compilation Details

You can compile the code by :

"mex planner.cpp plannerRRT.cpp treeRRT.cpp plannerPRM.cpp graphPRM.cpp"

Additionally, these are some of important hyper-parameters in "common_header.h". These are constants which are defined at the start of the header file.

1. P_G : Probability with which RRT and RRT-Star planners sample goal configuration.
2. MAX_EPSILON : Maximum number of steps for extend function. Currently 10.
3. MAX_STEP_SIZE : Maximum step size for collision check. Currently at PI/90.
4. NEAR_RADIUS : Near radius is used in both PRM and RRT-Star. In RRT-Star, it is used in the repair function for finding the near by configurations in the graph and reconnecting them in a more optimal way. In PRM, it's used to connect a new point to the graph.
5. MAX_ITERATIONS_PRM : Maximum number of configurations(nodes) added to the PRM graph. Currently at 40000.
6. K : Maximum possible neighbors for PRM. Currently at 5.
7. MAX_TIME : Maximum time for searching for a plan in RRT, RRT-Connect and RRT-Star. Currently at 20 seconds. Please increase it to 40 seconds when testing edge cases.

2 Discussions

I think the RRT-Connect was able to show the best performance out of all the planners. For different test cases which I tried, RRT-Connect was almost always able to find a feasible path within a second. Peformance of RRT and RRT-Star were also satisfactory. My implementations of RRT, RRT Connect and RRT star are pretty fast but they can be improved by using a KD tree for storing the configurations in the tree. In my opinion, RRT connect works the best for these kinds of maps as it almost always produces a solution within a second. There are some cases where I found that RRT Connect actually performed worse than RRT or RRT-Star. These are the cases where start position might be surrounded my obstacles. IN such a case, RRT and RRT-Star always keep exploring near the start position and are therefore able to find a path earlier.
My implementation of PRM performed the worst. There were many cases where my implementation of PRM was not able to produce a path. It is also the slowest performing planner

out of all the planners. I tried changing a lot of hyper-parameters for PRM but was not able to obtain a 100 percent success rate of finding a path. One drawback of the way in which I have implemented PRM is that I can only have a certain number of points in the graph. I think implementing it in a way such that it the solution is bounded by some maximum time rather than being bounded by number of nodes in the graph would have been more useful. Here is a table comparing average results of different planners:

	RRT	**RRT-Connect**	**RRT-Star**	**PRM**
Planing Time Average	1.36	1.21	1.31	12.36
Path Cost Average	8.24	14.203	6.89	20.07
Size of tree/graph	622	846	536	10000

The start and goal configurations which I used for testing my planners can be found in the table below :

Qstart	**Qgoal**
1.56566 6.03025 2.13871 3.67735 1.40625	0.954219 2.39178 5.15862 1.07671 2.0733
1.09951 1.23524 1.57761 3.87072 2.97376	0.954219 2.39178 5.15862 1.07671 2.0733
1.53703 2.12195 5.65521 2.32005 0.698708	0.954219 2.39178 5.15862 1.07671 2.0733
1.71848 1.86184 4.67904 1.18724 4.31514	0.954219 2.39178 5.15862 1.07671 2.0733
0.652642 1.89279 2.9589 1.4482 5.30495	0.954219 2.39178 5.15862 1.07671 2.0733
0.977357 5.80177 2.70307 1.16124 5.68553	0.954219 2.39178 5.15862 1.07671 2.0733
1.86892 1.64753 3.78777 4.4687 1.39328	0.954219 2.39178 5.15862 1.07671 2.0733
0.368878 1.86407 2.00294 2.66512 3.19097	0.954219 2.39178 5.15862 1.07671 2.0733
0.822693 2.10711 4.27086 0.857989 4.53161	0.954219 2.39178 5.15862 1.07671 2.0733
0.333688 2.33992 1.24481 3.0768 2.1331	0.954219 2.39178 5.15862 1.07671 2.0733
1.56566 6.03025 2.13871 3.67735 1.40625	1.34002 2.40607 0.186408 2.96768 2.09464
1.09951 1.23524 1.57761 3.87072 2.97376	1.34002 2.40607 0.186408 2.96768 2.09464
1.53703 2.12195 5.65521 2.32005 0.698708	1.34002 2.40607 0.186408 2.96768 2.09464
1.71848 1.86184 4.67904 1.18724 4.31514	1.34002 2.40607 0.186408 2.96768 2.09464
0.652642 1.89279 2.9589 1.4482 5.30495	1.34002 2.40607 0.186408 2.96768 2.09464
0.977357 5.80177 2.70307 1.16124 5.68553	1.34002 2.40607 0.186408 2.96768 2.09464
1.86892 1.64753 3.78777 4.4687 1.39328	1.34002 2.40607 0.186408 2.96768 2.09464
0.368878 1.86407 2.00294 2.66512 3.19097	1.34002 2.40607 0.186408 2.96768 2.09464
0.822693 2.10711 4.27086 0.857989 4.53161	1.34002 2.40607 0.186408 2.96768 2.09464

3 RRT

I tried 20 random start and goal positions. These random start and goal positions were same for all the planners.I have mentioned these configurations at the last of this document.

3.1 Results

I obtained the following results with RRT :

Path Cost	Size of Tree	Plan Time
17.756	2219	2
7.48414	66	1
3.01882	13	1
3.28066	53	1
16.4977	2715	2
15.0137	228	1
6.62544	361	1
6.62657	172	1
3.80931	38	1
14.7993	3795	3
8.77058	230	1
3.48089	20	1
10.4955	91	1
9.63021	1104	1
5.66426	27	1
12.8462	503	1
6.20473	153	1
2.58107	18	1
2.12041	20	1

3.2 Implementation Details

I have implemented my RRT planner in the file plannerRRT.cpp. This file also has RRT-connect and RRT-star implementations. I have created a tree class in the file treeRRT.cpp for handling the tree generated by my RRT planner. A vector NodesPtrList contains all the nodes in the tree. In this implementation of RRT, I sample the goal configuration 20 percent of the time, and a random configuration otherwise. The RRT planner tries to extend towards the sampled configuration but can only move with a maximum step size of PI/9 radians. This extend method is implemented in "extend" function within the tree object. The planner finds a plan if it samples the goal configuration and it happens to be within PI/9 radians of the nearest point in the tree. If the planner is not able to find a path within 15 seconds, it quits and responds with a "time out" message. Except for some edge cases(not included in the results), RRT is able to generate a plan within 20 seconds.

4 RRT Connect

I tried 20 random start and goal positions. These random start and goal positions were same for all the planners. I have mentioned these configurations at the last of this document.

4.1 Results

The results which I obtained for random start and goal positions for RRT-Connect are in the table below:

Path Cost	Size of Tree	Plan Time
36.3028	2922	2
3.35103	25	1
4.11898	4	1
4.95674	16	1
4.60767	15	1
57.9232	6084	5
9.84366	305	1
4.39823	19	1
9.84366	96	1
7.47001	33	1
34.2085	1820	1
4.39823	4	1
16.7552	143	1
12.2871	119	1
4.46804	6	1
33.3009	4303	3
14.591	160	1
4.39823	4	1
2.58309	4	1

4.2 Implementation Details

My implementation of RRT Connect planner is very similar to my implementation of the RRT planner. In this planner, I maintain two instances of the tree object. The pointers to these trees are "ForwardTreePtr" and "BackwardTreePtr". In one iteration, I extend one of the trees towards a randomly sampled configuration while I connect the other tree directly to the newly added configuration in the extended tree. I swap the pointers in the next iteration and so in the next iteration I extend the backward tree, The "extend" function in tree object also works as "connect" function with just a boolean to set the step size to infinity. Except for some edge cases(not included in the results), RRT Connect is able to generate a plan within 20 seconds.

5 RRT Star

I tried 20 random start and goal positions. These random start and goal positions were same for all the planners. I have mentioned these configurations at the last of this document.

5.1 Results

The results which I obtained for random start and goal positions for RRT-Star are in the table below:

Path Cost	Tree Size	Plan Time
10.8275	325	1
6.89865	104	1
2.07521	14	1
2.55218	12	1
7.38969	865	1
18.4055	754	2
9.39121	1168	2
5.63531	3578	4
2.87779	18	1
7.61687	1593	1
10.5631	401	1
2.26829	19	1
7.34984	79	1
6.08119	67	1
4.75957	30	1
13.6508	923	2
8.6082	196	1
2.49129	18	1
1.46652	24	1

5.2 Implementation Details

Implementation of RRT-Star is similar to RRT and RRT-Connect. Like RRT, in RRT-Star, I have sampled the goal configuration for 20 percent of the time. I have implemented an extra repair function in tree object which repairs the tree and makes the path more optimal. Except for some edge cases(not included in the results), RRT star is able to generate a plan within 20 seconds.

6 PRM

I tried 20 random start and goal positions. These random start and goal positions were same for all the planners. I have mentioned these configurations at the last of this document.

6.1 Results

The results which I obtained for random start and goal positions for RRT-Star are in the table below:

Path Cost	Plan Time	No. of points in graph
26.3735	9	10000
23.3303	8	10000
fail	8	10000
fail	12	10000
12.654	12	10000
18.3915	12	10000
7.2489	13	10000
fail	12	10000
8.67617	12	10000
15.7253	21	10000
25.5912	12	10000
fail	12	10000
22.5397	13	10000
19.3632	12	10000
fail	16	10000
27.205	13	10000
24.7251	12	10000
23.6541	15	10000
26.0943	11	10000

6.2 Implementation Details

In my implementation of PRM, I always a generate a fixed number of points(in collision free space) to generate a graph. I then locally connect my start and goal configuration to this graph and apply A-star to find the most optimum path from start position to goal position. In my current implementation, I sample around 10,000 points and then try to find a path between start and goal.