SC627-Assignment1
Planning for High-DOF Planar Arm

Krisha Shah, 21d070039

February 2024

1 Compiling Code

Command to compile

```
g++ planner.cpp -o planner.out
```

2 Result

I used :

- A search radius of $\pi/10$ in PRM and $\pi/60$ for RRT* for the nearest neighbor search. Only nodes within that radius will be considered neighbors.
- A neighbor limit of 10 for PRM, to limit the branching factor of all nodes in the graph.
- Uniform Distribution from [0,2π] for sampling the new nodes.
- A goal bias probability of 0.05, to periodically bias sampling directly towards the goal to speed up performance.
- A step size of $\pi/90$ for interpolating and checking collisions when joining nodes. Reducing the step size increased the time for collision checking.
- A goal threshold of $\pi/6$, so the algorithm tries to connect a node to its target whenever it's sampled within that boundary. This is needed because the probability of sampling a single target node is 0, so we have to specify an area.
- Epsilon=$\pi/10$ which is the maximum angle change in a single step. It is used for limiting step size. A larger step size reduced the planning time but increased the cost of the path.

planner	mapName	problemIndex	numSteps	cost	timespent	success	numNodes	Sucess under 5sec
0	./map2.txt	0	15	11.12137	0.0029952	TRUE	36	1
0	./map2.txt	1	34	19.023724	0.224421591	TRUE	267	1
0	./map2.txt	2	60	28.48765	0.134369562	TRUE	448	1
0	./map2.txt	3	19	14.874624	0.00457014	TRUE	75	1
0	./map2.txt	4	12	8.88654	0.003042161	TRUE	17	1
0	./map2.txt	5	32	19.258108	11.83540641	TRUE	5544	0
0	./map2.txt	6	31	20.422772	0.015578889	TRUE	190	1
0	./map2.txt	7	35	20.051894	1.062373438	TRUE	1651	1
0	./map2.txt	8	29	17.37947	0.544024033	TRUE	1247	1
0	./map2.txt	9	10	6.695178	0.003087495	TRUE	24	1
0	./map2.txt	10	30	19.134396	0.22648303	TRUE	841	1
0	./map2.txt	11	13	11.32512	0.004818502	TRUE	61	1
0	./map2.txt	12	58	32.8404194	0.135698415	TRUE	336	1
0	./map2.txt	13	47	29.94716	0.010917435	TRUE	155	1
0	./map2.txt	14	26	18.78373	0.11481189	TRUE	588	1
0	./map2.txt	15	40	27.039342	0.08972468	TRUE	470	1
0	./map2.txt	16	89	48.0615836	4.28799071	TRUE	1999	1
0	./map2.txt	17	33	22.6490162	0.351632437	TRUE	954	1
0	./map2.txt	18	15	10.810928	0.007713008	TRUE	116	1
0	./map2.txt	19	18	13.110932	0.003482095	TRUE	46	1

Figure 1: Stats for RRT

planner	mapName	problemIndex	numSteps	cost	timespent	success	numNodes	Sucess under 5sec
1	./map2.txt	0	14	11.712822	0.007392103	TRUE	116	1
1	./map2.txt	1	31	19.019962	0.064328163	TRUE	170	1
1	./map2.txt	2	32	18.992112	0.085290107	TRUE	309	1
1	./map2.txt	3	19	15.420692	0.073508706	TRUE	436	1
1	./map2.txt	4	16	12.79102	0.005488267	TRUE	72	1
1	./map2.txt	5	34	21.808168	0.389400252	TRUE	972	1
1	./map2.txt	6	22	15.896146	1.048821532	TRUE	1631	1
1	./map2.txt	7	22	13.329052	2.47735978	TRUE	2369	1
1	./map2.txt	8	48	26.573518	5.844407493	TRUE	3459	0
1	./map2.txt	9	5	4.403634	0.002611629	TRUE	5	1
1	./map2.txt	10	28	17.215278	3.100009192	TRUE	2654	1
1	./map2.txt	11	14	10.1817	0.003698439	TRUE	47	1
1	./map2.txt	12	35	22.329604	0.043447087	TRUE	307	1
1	./map2.txt	13	40	23.688512	0.012642669	TRUE	140	1
1	./map2.txt	14	14	11.23352	0.005075499	TRUE	60	1
1	./map2.txt	15	11	6.85514	0.007381738	TRUE	108	1
1	./map2.txt	16	38	17.9707084	9.538848975	TRUE	3030	0
1	./map2.txt	17	34	24.782334	1.529439361	TRUE	1694	1
1	./map2.txt	18	20	15.552478	0.00448263	TRUE	72	1
1	./map2.txt	19	15	7.55818	0.096475491	TRUE	465	1

Figure 2: Stats for RRTstar

The tables below show the stats for RRT, RRT* and PRM.

Average Stats for the three algorithms

PRM had the highest average planning time, but this was only because the PRM graph is being generated during each run. Because of this, it was not able to plan within 5 seconds and had a low success rate. Ideally, we can generate the

planner	mapName	problemIndex	numSteps	cost	timespent	success	numNodes	Sucess under 5sec
2	./map2.txt	0	11	11.8685238	5.006054317	TRUE	8002	0
2	./map2.txt	1	13	14.369082	5.005939386	TRUE	8002	0
2	./map2.txt	2	20	22.9642234	4.812751096	TRUE	8002	1
2	./map2.txt	3	15	16.382928	4.817041004	TRUE	8002	1
2	./map2.txt	4	12	12.529248	4.819869101	TRUE	8002	1
2	./map2.txt	5	13	14.42592	4.812099117	TRUE	8002	1
2	./map2.txt	6	15	15.5463946	4.81582728	TRUE	8002	1
2	./map2.txt	7	13	13.23295358	4.813063806	TRUE	8002	1
2	./map2.txt	8	18	21.76183	4.82685664	TRUE	8002	1
2	./map2.txt	9	8	7.5986284	4.948421792	TRUE	8002	1
2	./map2.txt	10	12	13.228036	5.031389389	TRUE	8002	0
2	./map2.txt	11	13	13.478796	5.045206313	TRUE	8002	0
2	./map2.txt	12	12	13.667684	5.029505325	TRUE	8002	0
2	./map2.txt	13	19	23.02815	5.038559483	TRUE	8002	0
2	./map2.txt	14	13	13.69964	4.780615473	TRUE	8002	1
2	./map2.txt	15	12	11.841642	5.018205789	TRUE	8002	0
2	./map2.txt	16	13	13.2818601	4.810520891	TRUE	8002	1
2	./map2.txt	17	18	20.099354	4.82580912	TRUE	8002	1
2	./map2.txt	18	13	14.67847	4.834035192	TRUE	8002	1
2	./map2.txt	19	10	10.3160974	5.055167328	TRUE	8002	0

Figure 3: Stats for PRM

Table 1: Average Stats

Planner	No. of Steps	Cost	Time Spent	No. of Nodes	Success rates under 5 secs
PRM	13.65	14.89997306	4.907346892	8002	60%
RRT	32.3	19.99519786	0.9531570562	753.25	95%
RRTstar	24.6	15.86572902	1.217005456	905.8	90%

graph once, and then re-use it for multiple queries, which will be much faster. RRT* had a lower average cost than RRT, which was expected because of re-wiring in RRT*, which increased its average planning time. PRM had the lowest average cost, because of the large number of nodes sampled. It was always able to find a lower cost path since more nodes were sampled and added to the graph.

Conclusion

- Assuming we have multiple query problem and can pre-build and save the PRM graph, a PRM planner seems to be the most appropriate for this context based on the results above. Because the generation of the PRM graph and sampling both take place throughout each run, the average time for the PRM runs is large. Alternatively, we can store and utilise the graph after it has been constructed. It won't take long to look for a path through the PRM graph. The PRM runs also have the lowest average cost. Since we may need to solve for several start-goal configurations in this static environment, it will be possible to save a PRM graph and later reuse it.

  If we have single query problems, then RRT will be more suitable for this environment. RRT* will take a bit more time, but will provide a lower cost solution. So if time is variable to be optimised, then RRT is suitable and if the cost is to be optimised, then RRT* is most suitable.

- One of the problems with the PRM planner is that, unlike RRT and RRT*, once a graph is formed, it cannot be altered. There's no wiring change occurring. It does not ensure the best possible outcome. If it fails to find a path, we cannot tell if it was due to an obstacle in the environment, or just because the number of sampled nodes is less.

  This planner can be made better by carrying out better pre-processing while creating the PRM graph, such as employing a connected-components technique in place of the condition that limits the maximum number of neighbors for a node, or by implementing the neighborhood function more effectively. Instead of sampling randomly, deterministic sampling can be used. We can also perform post-processing, like path shortening, to obtain better paths.

- The problem with RRT is that the path is fluctuating and not very smooth.

- Additional complexity of RRT*, particularly in rewiring nearby nodes, slows it down compared to the more streamlined RRT.

- Thus there is a trade-off between cheap cost and time spent between RRT* and RRT.