Planning & Decision-making in Robotics

Manuj Trehan
Andrew ID: mtrehan

16-782

Homework 2

October 31, 2022

Compiling Code

Command to compile

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

Results and Statistics

The 5 start and goal configurations are,

Start	**Goal**
1.7, 1, 0.3, 5.6, 4.2	1.8, 0.2, 1.5, 2.6, 5.2
1.5, 1.9, 3.1, 1.3, 3.9	1.6, 0.9, 0, 5.2, 1.5
1.3, 0.7, 2.8, 1.2, 3	1.7, 2.3, 4, 4.1, 1.8
1.1, 0.9, 2, 1.9, 1.1	1.7, 6.2, 1.7, 0.4, 0.6
1.5, 1.7, 5.9, 0.1, 4.9	1.8, 2.5, 1.8, 1.4, 4.3

The four tables below show the statistics of the 4 runs for all start-goal configurations for each planner

planner	mapName	problemIndex	numSteps	cost	timespent	success	numNodes
0	./map2.txt	0	33	22.852762	0.133893044	True	287
0	./map2.txt	1	25	18.464392	0.08668601701	True	280
0	./map2.txt	2	39	20.293808	8.321813774	True	2606
0	./map2.txt	3	41	22.407108	31.190708222	True	4815
0	./map2.txt	4	27	17.725862	0.069140336	True	240
1	./map2.txt	0	11	20.404248	0.01008551201	True	69
1	./map2.txt	1	10	19.739908	0.005698466	True	17
1	./map2.txt	2	11	25.899614	0.018861024	True	108
1	./map2.txt	3	42	51.77208	0.17484373301	True	415
1	./map2.txt	4	34	34.636116	3.392749463	True	1717
2	./map2.txt	0	32	20.670544	0.112383193	True	274
2	./map2.txt	1	18	14.358208	0.00865340499	True	64
2	./map2.txt	2	19	15.091508	1.845577431	True	1165
2	./map2.txt	3	48	23.517104	4.844422451	True	1805
2	./map2.txt	4	26	18.3777638	0.047419731	True	117
3	./map2.txt	0	13	14.139118	18.35283677	True	10002
3	./map2.txt	1	13	14.423064	18.364176816	True	10002
3	./map2.txt	2	13	15.044352	18.298336106	True	10002
3	./map2.txt	3	17	19.861148	18.26331252	True	10002
3	./map2.txt	4	14	15.26531	18.32068277	True	10002

Figure 1: Stats for run 1

planner	mapName	problemIndex	numSteps	cost	timespent	success	numNodes
0	./map2.txt	0	17	16.63298	0.050322464	True	114
0	./map2.txt	1	23	17.375866	0.02951797799	True	197
0	./map2.txt	2	38	21.419648	8.233104257	True	2873
0	./map2.txt	3	24	14.602898	6.54259998001	True	2371
0	./map2.txt	4	52	22.86284	1.01274590401	True	394
1	./map2.txt	0	45	47.880734	2.88258259301	True	1657
1	./map2.txt	1	11	19.19138131	0.006947574	True	46
1	./map2.txt	2	33	22.538644	0.02948758101	True	210
1	./map2.txt	3	11	19.79566874	0.006154708	True	40
1	./map2.txt	4	23	28.109756	0.07731675499	True	262
2	./map2.txt	0	25	15.940134	0.171807885	True	338
2	./map2.txt	1	21	15.016914	0.03589102899	True	172
2	./map2.txt	2	38	24.705858	17.055518437	True	3722
2	./map2.txt	3	54	31.688776	11.41296779	True	3108
2	./map2.txt	4	29	16.20048	0.152575809	True	160
3	./map2.txt	0	13	12.742326	18.741066333	True	10002
3	./map2.txt	1	15	16.54315	19.41404806	True	10002
3	./map2.txt	2	12	13.270718	18.908170816	True	10002
3	./map2.txt	3	20	21.460794	19.240365764	True	10002
3	./map2.txt	4	14	16.646568	18.600349023	True	10002

Figure 2: Stats for run 2

planner	mapName	problemIndex	numSteps	cost	timespent	success	numNodes
0	./map2.txt	0	22	18.569502	1.84942163101	True	1404
0	./map2.txt	1	20	26.83138	0.00786997999	True	63
0	./map2.txt	2	36	21.598546	0.457086173	True	719
0	./map2.txt	3	64	36.093134	24.056924635	True	4558
0	./map2.txt	4	32	19.2078946	0.17121946599	True	178
1	./map2.txt	0	13	18.358096	0.011969981	True	88
1	./map2.txt	1	9	19.58428731	0.00493412699	True	22
1	./map2.txt	2	10	18.05793	0.007057326	True	60
1	./map2.txt	3	13	12.9706638	0.13655618501	True	385
1	./map2.txt	4	13	24.786318	0.02322423	True	143
2	./map2.txt	0	56	21.977694	2.085230246	True	1340
2	./map2.txt	1	36	24.793996	2.714080091	True	1467
2	./map2.txt	2	32	20.981932	25.107751552	True	4630
2	./map2.txt	3	39	19.896046	13.258320557	True	3181
2	./map2.txt	4	27	16.470622	0.105111094	True	196
3	./map2.txt	0	11	11.555756	18.561522991	True	10002
3	./map2.txt	1	15	16.910828	18.435397803	True	10002
3	./map2.txt	2	13	15.62465	18.40961242	True	10002
3	./map2.txt	3	16	18.421944	18.426431707	True	10002
3	./map2.txt	4	15	16.411984	18.816875275	True	10002

Figure 3: Stats for run 3

planner	mapName	problemIndex	numSteps	cost	timespent	success	numNodes
0	./map2.txt	0	34	22.13005	0.432537559	True	237
0	./map2.txt	1	21	16.123282	0.00755759199	True	61
0	./map2.txt	2	37	21.688262	0.733544802	True	874
0	./map2.txt	3	49	27.8651	12.767764764	True	3553
0	./map2.txt	4	33	18.7907008	0.060606317	True	164
1	./map2.txt	0	12	23.36525	0.004996555	True	23
1	./map2.txt	1	11	22.95762931	0.00548789	True	26
1	./map2.txt	2	9	15.310102	0.00578063101	True	31
1	./map2.txt	3	19	19.859134	0.20049359801	True	487
1	./map2.txt	4	16	21.031646	0.06491227901	True	222
2	./map2.txt	0	45	20.777726	0.569039922	True	615
2	./map2.txt	1	38	19.013976	23.139821211	True	4199
2	./map2.txt	2	37	21.246532	0.199148666	True	455
2	./map2.txt	3	38	19.352514	6.08448895399	True	2236
2	./map2.txt	4	55	17.010972	0.3044029	True	359
3	./map2.txt	0	13	14.753172	19.404907525	True	10002
3	./map2.txt	1	13	15.08018	19.188224848	True	10002
3	./map2.txt	2	14	15.046322	19.912442615	True	10002
3	./map2.txt	3	19	21.4448196	19.723224866	True	10002
3	./map2.txt	4	14	15.1540226	19.417058969	True	10002

Figure 4: Stats for run 4

Average stats for all runs

	RRT	RRT-Connect	RRT*	PRM
Avg Planning Time	4.81075324475023	0.35350701055286	5.4627306176975	18.83995219985
Success Rate	70.00%	100.00%	70.00%	0.00%
Avg Num Nodes	1299.4	301.4	1480.15	10002
Avg Cost	21.17680077	24.3124603230769	19.85446499	15.99001131
Std Dev Time	8.69674422345746	0.95758976970943	8.0853923718623	0.52429113517
Std Dev Success Rate	0.47016234598163	0	0.4701623459816	0
Std Dev Num Nodes	1577.49799899781	494.784312392252	1512.4187764185	0
Std Dev Cost	4.85602452690235	9.89185591456955	4.2195225652285	2.629119280324

Figure 5: Average and Std Dev

RRT-Connect had the lowest average planning time, because of bi-directional growth and relaxation of the epsilon constraint. 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 success rate of 0. Ideally, we can generate the graph once, and then re-use it for multiple queries, which will be much faster. RRT* had a lower average cost than RRT and RRT-Connect, 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.

The hyper-parameters I used include,

- Epsilon for limiting step size. A larger step size reduced the planning time but increased the cost of the path
- A small step size for interpolating and checking collisions when joining nodes. Reducing the step size increased the time for collision checking and the overall planning time
- A goal threshold, so that whenever a node is sampled within that threshold, the algorithm tries and connect the node to the goal. Basically defining a 'goal region'. This is required because if we try and sample a single goal node, the probability for that is 0, which is why we need to define a region
- A goal bias, to periodically bias sampling directly towards the goal to speed up performance
- A search radius for the nearest neighbor search. Only nodes within that radius will be considered neighbors. A higher radius meant connecting to nodes which were further away
- A neighbor limit for PRM, to limit the branching factor of all nodes in the graph

Conclusion

1. Based on the above results, I believe that a PRM planner would be the best suited for this environment, provided we can pre-build and save the PRM graph. The average time for the PRM runs is high because that includes the time to sample and generate the PRM graph as well, which is happening during each run. Instead, once we have generated the graph, we can save it, and then re-use it. Searching through the PRM graph for a path will take a very short time. The average cost for the PRM runs is also the lowest.

   This is a static environment where we might need to solve for multiple start-goal configurations, hence saving a PRM graph and then re-using it will be feasible. If we are not able to do that, and instead have to build the PRM graph during each run, then RRT or RRT-Connect will be more suitable for this environment. RRT* will take a bit more time, but will provide a lower cost solution. So if we need to plan fast, then it might not be suitable.

2. The issues still there in the PRM planner is that once a graph is generated, we do not change it anymore, such as in RRT*. There is no re-wiring taking place. It does not guarantee an optimal solution. If it fails to find a path, we cannot be sure if it was due to an obstacle in the environment, or just because the number of sampled nodes is less.

3. To improve this planner, we can perform better pre-processing while building the PRM graph, such as a more efficient implementation of the neighborhood function, or using a connected-components approach, instead of the condition limiting the maximum number of neighbors of a node. We can also use better sampling strategies while building the graph instead of sampling uniformly. This could include bias sampling towards obstacle boundaries, or use Gaussian distributions around already existing samples. We can also perform post-processing, like path shortening, to obtain better paths.

Extra Credit

The highest variance in terms of consistency of solutions for different runs with the same start and goal configurations was observed in both RRT and RRT*. They showed the maximum deviation in planning time as well as cost. One hyper-parameter that helped in decreasing the planning time was increasing the goal threshold. If the goal threshold is slightly larger, but we perform a collision-check between the second last node and the goal, it helped in reducing the planning time by a lot, since it meant a higher probability of sampling in the 'goal region'. RRT* took the most time on average (if we exclude the time of building the PRM graph), which was mainly because of the re-wiring and collision-checking process. In difficult configurations, the variation in RRT and RRT* planning times ranged from anywhere between 4 seconds and 130 seconds. This variance in planning times was much lower in PRM and RRT-Connect.