PLANNING HW2

NINAAD DAMIS
ndamis@andrew.cmu.edu

INDEX

1. COMPILING AND RUNNING CODE

I have tested my code using MATLAB in Windows 10. The code has been compiled and tested using Microsoft Visual C++ 2019.

To run the code -

- Change to the code directory in MATLAB.
- Run - mex planner.cpp
- Run - runtest('map.txt' , start,goal,0)

Running PRM -

The first time PRM is run on a map, the planner takes some time to return the output as it has to run the preprocessing phase.
For subsequent start and goal configurations, the solution is returned very quickly as it uses the stored roadmap from the first run.

In my implementation, when we want to run PRM on a new map, we need to recompile the code by running mex planner.cpp. After doing this, we can proceed as above.

2. IMPLEMENTATION DETAILS

PRM

The points in Configuration space are stored as a vector<float>, and thus can extend to an arbitrary number of dimensions. These are typedef as CSpaceVertex.

Graph structure - I have used an adjacency list representation for storing the graph constructed during the preprocessing phase. The graph is stored as a std::map, whose key is the CSpaceVertex and whose value is the associated adjacency list.
The adjacency list is represented as a vector of pairs (vector<std::pair<float,CSpaceVertex>>). The second element of the pair is a vertex, and the first element is the edge cost between the two vertices. This way, each element in the graph stores its neighbouring edges and edgecosts.

Sampling - Sampling of the vertex is done uniformly in the configuration space using the sampleRandomVertex() function, which checks the validity of the sample generated before returning the output.

Distance function - The VertexDistance() function is used to calculate the distance between two vertices. Given two configurations q1 and q2, distance is calculated as -
$\sqrt{\Sigma\,(q_1[i] \;-\; q_2[i])^2}$ for i = 1 to numOfDofs.

Local Planner - The LocalPlanner() function takes as input two vertices v1 and v2, and returns true if there exists an obstacle free path between the two vertices, and false otherwise.
This is achieved by linearly interpolating from v1 to v2, and checking if each of the intermediate configurations is valid. The local takes a parameter num_steps as input to decide the number of steps for interpolation. This parameter can be increased to have a finer check (higher resolution) for checking the presence of an obstacle free path.

Pre-Processing Phase - When the PRM planner is called, the ConstructRoadmapPRM() function is called, which takes as input parameters like the Maximum number of vertices to be added in the graph, the number of nearest neighbours k to consider, and the maximum distance value to consider for the nearest neighbours search. The graph is returned after the preprocessing step.

Query Phase - In the query phase we first try to connect the start and goal to the roadmap and obtain start_init and goal_init in the roadmap.Then, we check if start_init and goal_init are connected. If they are, we run the A* search to obtain a path between the start and goal pose.

RRT VARIANTS

The following is common for all the RRT variants -

Tree - The tree is represented as a vector of pointers. Each pointer stores the information of one vertex in the tree. Each vertex is represented by a structure called RRTNode. The RRTNode structure contains the following information - the CSpace vertex, the index in the tree, the index of the parent, and the cost of the path to the vertex.

Sampling - Sampling of the vertex is done uniformly in the configuration space using the sampleRandomVertex() function, which checks the validity of the sample generated before returning the output.

Nearest neighbor search - The nearest neighbor search is done by the RRTNearestNeighbour() function. This conducts a linear (O(n)) search on the tree, and outputs the nearest vertex.

Distance function - The VertexDistance() function is used to calculate the distance between two vertices. Given two configurations q1 and q2, distance is calculated as -

$$D(q_1, q_2) = \sqrt{\Sigma\,(q_1[i] - q_2[i])^2}$$ for i = 1 to numOfDofs.

Local Planner - The LocalPlanner() function takes as input two vertices v1 and v2, and returns true if there exists an obstacle free path between the two vertices, and false otherwise.
This is achieved by linearly interpolating from v1 to v2, and checking if each of the intermediate configurations is valid. The local takes a parameter num_steps as input to decide the number of steps for interpolation. This parameter can be increased to have a finer check (higher resolution) for checking the presence of an obstacle free path.

Extend Function - The functionality of the extend function is provided by the AltRRTGenerateNewVertex() function. The function takes as input the tree, the random vertex generated, the nearest neighbour and the epsilon parameter (max_dist) and generates the new vertex, if feasible.

Goal Bias - As mentioned in class lecture, I have used a goal bias. Thus, for my implementation of RRT and RRT*, with a probability p, we select the goal node as the random node during the sampling process. I have used the p = 0.05.

3. DATA COLLECTION

SAMPLES USED FOR DATA COLLECTION

SAMPLE NUMBER	START POSE	GOAL POSE
1	[1.13452 0.775402 2.34743 4.02466 5.97538]	[0.862753 0.0873893 3.33991 1.74941 0.079302]
2	[1.05077 0.174019 3.94837 3.22322 1.1896]	[1.32244 2.92097 0.536543 0.922159 5.58804]
3	[0.08550 1.57576 1.61078 3.13507 0.735656]	[1.55874 5.75384 2.59818 0.866437 0.234094]
4	[0.499018 2.92205 1.33736 2.10815 5.62372]	[1.47044 2.68329 3.77077 5.42506 1.36353]
5	[1.68464 1.68475 1.00189 4.99411 0.902222]	[1.52822 6.02173 2.38212 2.82608 5.34999]
6	[0.306276 3.02861 1.66006 1.66847 5.77814]	[0.485078 2.70844 5.84612 3.24169 0.715882]
7	[1.70116 1.46945 0.0668795 2.22213 4.00066]	[1.7627 0.550581 2.01476 4.80366 3.1644]
8	[1.47715 1.86813 0.81348 2.27285 0.280407]	[1.70242 6.11739 2.24533 2.84632 3.98843]
9	[0.943335 2.89366 1.1337 6.20286 1.73184]	[1.51002 0.953774 4.11098 6.13429 2.42943]
10	[1.26679 1.0046 3.26586 3.99377 4.15892]	[0.928946 1.19727 6.25719 2.9084 0.0839472]
11	[0.413208 3.50774 1.76377 0.409426 1.15209]	[0.50103 2.20816 2.06567 3.73925 0.0336251]
12	[1.3525 0.609606 0.986977 5.71787 4.33891]	[1.08588 2.66149 5.94605 4.33251 1.16568]
13	[1.40026 2.32429 2.9699 1.00483 1.81804]	[1.1849 2.60349 5.49354 2.00841 0.183301]
14	[0.384053 2.19838 0.92982 3.67593 3.51442]	[1.25815 2.748 5.95668 1.62569 1.01301]
15	[0.0391452 1.17958 3.26562 2.96905 1.79889]	[0.507979 3.15697 6.27755 1.5587 4.50059]
16	[1.45462 1.04672 1.61121 4.20169 4.46359]	[0.990287 1.07647 0.840748 4.68599 0.623812]
17	[1.8611 0.393748 1.09489 3.36343 4.79103]	[1.82596 3.13197 1.44648 0.589668 4.46153]
18	[1.66906 1.6501 5.85019 5.21679 3.79539]	[1.56555 0.725247 2.49388 2.79798 4.5874]
19	[1.18378 2.15478 6.12457 3.50883 3.08219]	[0.123087 2.03049 3.08741 0.775292 4.16656]
20	[0.41874 2.10517 0.744851 0.391608 4.42777]	[0.874326 1.93315 0.50559 4.47815 4.91142]

PRM

The roadmap generated in the preprocessing phase has 3000 vertices.

SAMPLE NUMBER	Path length	Path Cost	Time(s)
1	19	27.93	120.1
2	26	41	0.04
3	15	17.39	0.02
4	16	20.55	0.02
5	28	39.94	0.06
6	18	26	0.02
7	10	15.61	0.007
8	19	23.31	0.025
9	9	9.9	0.002
10	19	29.27	0.03
11	10	15.26	0.01
12	20	30.69	0.09
13	13	17.56	0.02
14	17	23.693	0.02
15	26	40.67	0.05
16	12	19.05	0.01
17	13	13.9	0.005
18	21	29.28	0.03
19	16	29.70	0.02
20	9	9.62	0.002

RRT

The maximum number of iterations for the main loop is set to 80,000. The value of the epsilon parameter (max_dist) used is 0.2.

Sample Number	Vertices Generated	Path Length	Path Cost	Time (s)
1	58423	108	19.41	469.7
2	1078	114	19.55	0.81
3	19985	91	16.02	53.31
4	22395	84	15.2	70.32
5	18486	131	23.67	47.92
6	1136	67	11.95	0.28
7	229	39	7.20	0.03
8	21686	86	15.04	61.13
9	314	45	8.34	0.054
10	707	49	8.17	0.14
11	177	39	6.74	0.028
12	3165	103	18.46	1.65
13	590	34	5.93	0.108
14	3156	79	14.24	1.56
15	1522	63	11.41	0.49
16	324	41	7.13	0.05
17	242	42	7.28	0.03
18	1019	99	14.29	0.25
19	178	43	7.56	0.03
20	258	46	8.47	0.04

RRT CONNECT

The maximum number of iterations for the main loop is set to 80,000. The value of the epsilon parameter (max_dist) used is 0.2.

Sample Number	Vertices Generated	Path Length	Path Cost	Time (s)
1	1305	154	22.42	2.01
2	672	88	13.04	0.43
3	2209	101	16.79	3.75
4	401	80	14.01	0.11
5	616	95	16.08	0.42
6	175	87	14.35	0.03
7	23	23	3.98	0.002
8	2608	76	11.85	2.38
9	24	24	4.20	0.003
10	87	49	7.97	0.02
11	56	29	4.85	0.005
12	334	114	19.5	0.08
13	78	58	8.56	0.02
14	142	67	11.51	0.02
15	212	80	13.18	0.04
16	25	25	4.51	0.002
17	24	24	4.3	0.002
18	163	81	12.53	0.03
19	66	44	7.29	0.007
20	33	28	5.027	0.004

RRT STAR

The maximum number of iterations for the main loop is set to 80,000. The value of the epsilon parameter (max_dist) used is 0.2.
As most pairs of start and goal are found quickly, and we do not want to run each start-goal configuration for all 80,000 iterations - Once the initial solution is found, I only run 5000 more iterations (for rewiring) before I break from the main loop. We can increase this number of iterations to obtain a more optimal path.

Sample Number	Vertices Generated	Path Length	Path Cost	Time till first solution (s)	Total Time(s)
1	41166	45	9.26	0.19	376.18
2	3792	56	11.21	0.43	4.23
3	21329	56	11.15	97.97	144.96
4	53292	80	17.11	817	964.35
5	23409	101	21.22	100.01	147.54
6	4525	55	10.78	0.14	4.97
7	4222	26	5.09	0.03	4.43
8	29598	74	15.60	167.612	228
9	4392	28	5.38	0.05	4.62
10	6094	39	7.85	1.09	8.65
11	4416	44	8.61	0.09	4.79
12	9251	74	14.84	7.3	19.94
13	4475	43	8.69	0.12	4.99
14	6240	53	10.74	1.66	9.33
15	5034	50	10.51	0.45	6.37
16	4220	30	5.84	0.05	4.39
17	4270	29	6.04	0.06	4.56
18	5247	54	11.25	0.47	6.57
19	4245	35	6.91	0.09	4.45
20	4290	39	7.56	0.04	4.71

4. RESULTS

Planner	Average Number of Vertices	Average Path Length	Average Path Cost	Average Planning Time(s)	Success Rate for Generating Solutions under 5s (%)
PRM	3000	17	24.06	6.03	95%
RRT	7754	71	12.30	35.35	75%
RRT Connect	462.65	67	10.79	0.47	100%
RRT*	12175.35	51	10.282	97.9	50%

Above are the statistics obtained from running the planners for the given 20 sample configuration start and end poses.

The RRT-Connect planner is by far the fastest planner, with an average planning time of less than 1s, and a 100% success rate of generating solutions under 5s.

The RRT Planner is slower compared to RRT-Connect, with an average planning time of 35.35 seconds. It is able to generate a solution under 5s for most of the configurations, but a few difficult configurations drastically increases the average planning time.

The RRT* planner is the slowest of all the RRT variants, with an average planning time of 97.9s. This is because the planner has to spend a lot of computation time to rewire the edges in the tree to ensure vertices are reached through minimum cost paths. We also see that the RRT* planner produces paths with the least average path length (compared to other RRT variants) and average path cost. This is due to the rewiring of the tree. Running the planner for a higher number of iterations will lead to an even lower path cost.
Also, the average planning time till first solution is 59.74s for RRT*, which is larger than the average planning time of RRT (35.35 s). This is understandable, as the steps for rewiring of the tree adds extra computation time to the algorithm.

The PRM planner is the second fastest planner, with an average planning time of 6.03 seconds. The first time the planner is run, the planning time is very high as the preprocessing phase is completed to produce the roadmap. All of the subsequent calls to the planner are very quick, as the previously built roadmap is used, and only the call to the A* planner is required to produce the path during the query phase.

5. CONCLUSION

The most suitable planner for the environment is the RRT-Connect Planner. The RRT-Connect planner is the only planner to consistently produce plans within 5s, with most plans being produced within a second. Due to the greedy connect heuristic, it is able to converge to a solution very quickly even for difficult start and goal configurations, while it takes a long time for RRT and RRT* to produce a valid path for such configurations. Also, compared to PRM, it does not need to have a long preprocessing step for each map, and can produce valid paths very quickly for different map environments.

The RRT-Connect planner still has a few disadvantages compared to the other planners. The path output by the planner is not optimal. Compared to RRT*, RRT-Connect has a higher average path length and a higher average path cost. Thus, even though the planner is able to output a valid path extremely fast, the path itself is not optimal.

We can improve the RRT-Connect Planner by combining it with RRT*. Such a planner will combine the advantages of both the planners - produce a asymptotically optimal path like in RRT* combined with the quickness of the RRT-Connect planner.
We can do this by modifying the RRT-Connect planner such that whenever a new vertex is being added to the tree, we connect it with the neighbour which gives us the least cost path to the new vertex. We also would rewire the tree similar to RRT* so as to only keep lowest cost paths in the tree, and remove non optimal paths.