Assignment 2 Report

Aditya Rathi

Results from 4 runs of 5 randomly generated start goal pairs

To generate these results I randomly generated 5 valid samples in the given map 2 file. All of the samples are for a 5 DOF arm. To clock() was used to randomly seed the generator and thereby ensure stochastic behavior from the planners

The results for each problem are shown in the tables below. These results combine the results from the 4 runs of each problem on the map.

Problem 0				
Planner		numSteps	cost	time
RRT	Mean	8	3.875645	0.009892
	Std Dev	0	0.07337	0.000601
RRT-Connect	Mean	12	3.84396	0.008124
	Std Dev	0	0	0.000452
RRT-Star	Mean	7	3.89418	0.00986
	Std Dev	0	0	0.00051
PRM	Mean	9	7.04316	19.39105
	Std Dev	0	0	0.24375

Problem 1				
Planner		numSteps	cost	time
RRT	Mean	113	39.07575	0.599486
	Std Dev	0	0	0.002145
RRT-Connect	Mean	122	37.10275	0.164325
	Std Dev	0	0	0.000344
RRT-Star	Mean	50.5	22.00743	0.225104
	Std Dev	0.57735	0.443615	0.019754
PRM	Mean	46	48.47632	21.07325
	Std Dev	0	0	0.123593

Problem 2				
Planner		numSteps	cost	time
RRT	Mean	44	15.42463	0.341814
	Std Dev	4	0.840504	0.116707
RRT-Connect	Mean	48.5	16.10934	0.013347
	Std Dev	7	2.700584	0.000719
RRT-Star	Mean	37	17.55538	0.195412
	Std Dev	0	0.188264	0.220641
PRM	Mean	19.75	18.52241	20.53302
	Std Dev	0.5	1.474718	0.172224

Problem 3				
Planner		numSteps	cost	time
RRT	Mean	76.75	24.04488	3.554959
	Std Dev	11.5	0.91576	2.986566
RRT-Connect	Mean	68.75	17.75224	1.819778
	Std Dev	14.5	2.336889	0.958528
RRT-Star	Mean	76	32.3277	2.341054
	Std Dev	5.773503	4.428805	0.316055
PRM	Mean	15.25	13.73014	19.36447
	Std Dev	10.5	10.45043	0.111981

Problem 4				
Planner		numSteps	cost	time
RRT	Mean	70	26.54394	1.801483
	Std Dev	0	0	0.015112
RRT-Connect	Mean	77	22.35197	0.031993
	Std Dev	0	0	0.000828
RRT-Star	Mean	25.5	23.72293	3.593135
	Std Dev	1.732051	0.473844	0.071509
PRM	Mean	9	7.04316	19.39105
	Std Dev	0	0	0.24375

From the results we can see that for Problem 0 all of RRT, RRT-Connect and RRT-Star have almost the same path cost with differences only showing up in the second decimal place. These differences can be accounted towards the probabilistic nature of the planners and the fact that a different seed was used for all of them. These 3 planners also take less than 0.01 seconds to

find the path and have a little standard deviation in all four runs. PRM takes the longest to plan because it first has to create the path and then traverse it to find the path from the start to the goal. This results in PRM expanding a lot more points than the other planners. The path lengths of all the planners have no deviation and are within 4 steps of each other.

For Problem 1, we see that RRT* is the most optimal solver and has a significantly lower cost than the others. This is because the path for this case is long, and RRT* has a higher chance of rewiring the nodes to find a more optimal solution. It takes a longer time compared to RRT-Connect but is lower than RRT. PRM still takes much longer than the other planners as it. has to expand a more significant number of nodes than the rest during the planning phase.

For Problem 2, we see that RRT has the most optimal solution, with the rest all lying pretty close to each other. This is probably a combination of the stochastic factors and the positioning of the start and the goal closer to each other. Interestingly PRM can find a solution that takes a significantly lower number of steps as it is able to take longer strides and has a much more even distribution of samples throughout the space compared to the other planners. Planning times for all RRT variants are pretty similar (under one second), with RRT Connect taking the lowest of the trio.

For Problem 3, we see that PRM provides the most optimal solution, closely followed by RRT-Connect, likely indicating that the start and the goal are aligned well, allowing RRT-Connect to find a straight shot path from the goal to the start. RRT and RRT* have similar paths, while RRT* has a much higher deviation, likely due to random seeding. This problem likely has a more complicated path finding as all planners take significantly longer than the previous problems. RRT* is penalized here due to the rewiring. PRM can find the most optimal solution as it has a much higher population of samples that it calculated before the path finding but also has the highest planning time.

For Problem 4, PRM is again able to find the lowest cost while all variants of RRT have a much higher but similar cost. RRT and RRT-Connect have a much higher number of steps needed to reach a solution while RRT* is much lower due to the rewiring. Surprisingly RRT* takes much longer to plan for this path, indicating a much higher number of rewiring between the nodes however the cost of all of these planners is still much lower than PRM.

Overall taking the mean across all values for all the problems, we get:

Overall					
Planner		numSteps	cost	time	Success Rate for Solution under 5 Seconds
RRT	Mean	62.35	21.79297	1.261527	100%
	Std Dev	3.1	0.365927	0.624226	
RRT-Connect	Mean	65.65	19.43205	0.407514	100%
	Std Dev	4.3	1.007495	0.192174	
RRT-Star	Mean	39.2	19.90152	1.272075	100%
	Std Dev	1.616581	1.106906	0.125694	
PRM	Mean	19.8	18.96304	19.95057	0%
	Std Dev	2.2	2.385029	0.17906	

Thus we see that for these problems RRT-Connect and RRT* have very similar costs but are both slightly beat by PRM. This is likely because PRM generates a much higher number of nodes in the graph before finding paths from the start to the goal and is thus able to find more efficient paths. RRT is the worst as it has no directionality to guide its search. In terms of efficiency, RRT-Connect is by far the most efficient as it expands both trees and is able to connect much faster. RRT* is less efficient than vanilla RRT as it does the rewiring of nodes, while PRM is relatively inefficient due to the much larger number of nodes expanded. However, PRM is the most consistent in terms of its planning times as it likely expands a very similar number of nodes each time the algorithm runs. The number of steps made from the start to the goal are much lower in PRM as its step size is larger; next is RRT* with rewiring adding to the efficiency followed by RRT and RRT-connect, both being very close. Currently, the PRM algorithm is set to generate 10000 nodes each time the planner runs, and this is why the planner takes so long to run.

For this environment; I believe that RRT-Connect is the most efficient planner as it runs the fastest and provides paths comparable in efficiency to RRT* and PRM. The bidirectional expansion of trees allows it to converge much faster than the other algorithms, allowing it to be used in these scenarios and likely with even higher DOF setups. Convergence in obstacle-ridden environments is also better as it expands trees from both directions, and they have a higher likelihood of meeting in the middle. While the planner can be inefficient in terms of

the number of steps, the fact that the final cost is still lower indicates that these steps are likely redundant and can be drastically reduced by shortcutting or a similar implementation.
As mentioned, currently, the base implementation of RRT-connect is implemented and there are some shortcomings. No postprocessing has yet been implemented in the planner. All the nodes are currently stored in a vector implementation which is inefficient and the cost grows linearly with the number of nodes. This needs to be improved as well. Furthermore, currently, the samples are generated uniformly with a 15% probability of being in the goal region. This is not ideal especially in connecting regions with higher obstacle density and can be improved upon. Self collisions with the arm is not implemented and this needs to be improved upon for a practical implementation.
There are multiple ways that this implementation can be improved. Firstly, I plan to implement K-D trees to make the search O(log n) instead of O(n). This will drastically speed up the algorithm even beyond its current impressive speed. Postprocessing, such as shortcutting is a dire need. This will reduce the large number of vertices generated in the plan and make the paths more efficient. Lastly better sampling techniques such as local biasing will improve performance in obstacle-heavy regions.

Hyper Parameters

There are some hyperparameters in the planner that greatly influence the behavior.
Using a large epsilon causes the planner to take larger steps making replanning more frequent and samples wasted more common. It also lead to larger jumps between actual steps making it hard to track the movement of the arm. At the same time, a very small epsilon caused the planner not to move enough resulting in an extremely large number of points being added to the tree very close to each other. For this map, I found an epsilon value of 0.2 rad (combined across all dimensions) achieves good performance in terms of step size and accuracy and the speed o solution. At the same time, I also noticed that for smaller dimensions the epsilon value can be decreased as it is distributed over less dimensions making the steps lager for the same effective total epsilon.
For generating samples, I have added a small probability of generating sample from around the goal region. This has two hyperparameters to tune - The probability of choosing from around the goal region and the size of the region. After tuning I found that a value of 15% worked well. Higher values led the planner to make no progress and get stuck on obstacles (akin to a greed approach of weighted A*) and lower values served a limited purpose in guiding the planner. In terms of the goal region I tuned the values in accordance with the obstacles surrounding the goal and found that a value of ±0.06 rad around the goal in each dimension worked well for the problem at hand with the maps that were provided to us. Larger values led to points being chosen so far away that they were misguiding the arm instead of providing appropriate heuristics.
For the radius of the hypersphere, I found that the default value of the sphere works well with gamma = 1. Larger values cause too many nodes to be found for rewiring making RRT* very slow and lower values caused very few values to be found for rewiring.
For PRM we see that the path cost is consistently low due to the larger number of nodes that are sampled from the graph (Currently 10000). The other reason that the cost is low is that it

used the Djikstra algorithm to find the path from the start to the goal thus given a graph it is guaranteed to be optimal.

Compiling

As I worked with the planner on windows I compiled the planner into a ‘.exe’ file

“g++ planner.cpp -o planner.exe”