Homework 2: Planning for a high-DOF planar arm

Rohan Tiwari

Andrew Id: rohantiw

Results

For generating the results I randomly sampled 25 random start and goal configurations for map 2. I run all the 4 planners (RRT, RRTConnect, RRT* , PRM) on these configurations and recorded the planning time, number of vertices, success rate of finishing under 5 sec, and path quality/cost. Currently all planner have threshold of 50000 iterations (excpet PRM, 100000 iterations) and return $No\ path\ found$ if the thresold is exceeded. Thus for computing the average for each of the planners I took the 20 configurations which returned the path for all the 4 planners.

Parameters

Parameters for PRM:

1. Neighbour search radius : 0.5
2. Number of Iterations : 100000

Parameters for RRT:

1. Epsilon : 1
2. Max Number of Iterations : 50000

Planner	Planning Time	No. of Vertices	Success Rate	Path Quality
RRT	0.4445	238	95%	9.824
RRT Connect	0.293	56	100%	10.763
RRT*	1.829	356	91%	9.049
PRM	13.304	5590	0%	34.434

Results

RRT

RRT has higher planning time compared to RRT Connect but almost comparable cost to RRT*. I did not observe a clear pattern in terms of lower cost of RRT* over RRT and over multiple trials showed similar costs. The number of vertices added in RRT were less compared to RRT* but less than RRT Connect. RRT did not get a solution 5% of the times in under 50000 iterations.

RRT Connect

RRT Connect has was the fastest with the least number of nodes added. It was significantly faster than RRT*. It is great for scenarios with less number obstacles and would lead to getting a faster solution. For the random start and goal configurations I generated RRT Connect gave the least cost path on average.

RRT*

RRT* has higher planning time compared to RRT and RRT Connect. RRT* has a slightly lower cost than RRT but it is not significant enough. RRT* did not get a solution 9% of the times in under 50000 iterations. On average RRT* has higher number of nodes added to the tree.

PRM

PRM has the highest planning time if we take into account the time taken to build the graph as part of the planning time.I ran PRM for 100000 iterations and thus had a almost consistent planning time. PRM also has the maximum number of vertices added to the graph. PRM also leads to the highest cost path among all 4 planners. If we just consider finding the path from start to goal given the graph, PRM is the fastest as it simply involves using Dijkstra to get get shortest path.

Conclusion

Based on the results from all 4 planner, RRT connect is great for scenarios with less number obstacles and would lead to getting a faster solution,this was clearly observable in map1. RRT connect doesn't have a significantly higher cost when compared to RRT and RRT*. RRT* should perform better in maps which have more obstacles and require samping more nodes to find a solution, as in those situations rewiring would help reduce the overall cost. PRM was the most reliable planner in terms of finding a path. It takes significantly longer to build the graph, but this only has to be done once for an environment and then can be reused again for different start and goal configurations. Thus it is a good approach for static environments and further refinement/post-processing can help reduce the overall cost of the path.

Implementation Details

I have a base class called Sampling Planner and RRT and PRM inherit from that class. RRT* and RRT connect inherit from RRT. This structure allows reuse of methods accross planners. Sampling Planner has the collision checking, random sampling etc written which is required by all 4 planners. RRT class has extend and interpolate methods which are used for both RRT Connect and RRT Star.
For PRM I used a list of hash maps to store the nodes belonging to a particular component. For RRT I used a hash map with key and value pairs as vectors for O(1) access time.

How to Run

Run the command following commands:
mex planner.cpp prm.cpp rrt.cpp rrtstar.cpp rrtconnect.cpp sampling_planner.cpp -I/usr/local/include
runtest('map2.txt',startQ, goalQ, planner_id);