PRM vs RRT- Path Planning Comparison

Abishek Sampath

CS 4610/5335, Spring 2019: Directed Project

ABSTRACT

This Project aims to perform comparative experiments of two major sampling-based path planning algorithms, namely *Probabilistic Roadmaps (PRM)* and *Rapidly exploring Random Trees (RRT)*. The two algorithms are illustrated using a *Puma 560 Robot* model in *Matlab* environment. I will present with a comparative analysis of both algorithms in the ability to find a collision free path from start to goal in the workspace, based on various input configurations and obstacles.

INTRODUCTION

Path planning algorithms based on configuration-space costs have proven to be effective solutions for problems not only in the field of Robotics, but also in Automation, Manufacturing, Animation and Computational Biology. Many of the path planning algorithms are typically suited to certain problems which they perform better compared to other path planning algorithms. Some early grid-based methods such as A* or D* can be used to compute resolution-optimal paths over a cost-map. But these options only work best with problems involving low-dimensional spaces, whereas sampling-based algorithms like *RRT* give considerably better results with higher dimensional problems.

Most of the sampling algorithms work by generating possible paths by randomly adding workspace configurations that translate to real-world coordinates, either some viable solution is reached, or time runs out. Hence, these algorithms are probabilistically complete, since there is a definite chance that the algorithms would find a solution if allowed to run without a time limit.

In this Project work, two major sampling-based path planning algorithms will be the focus and their working on various scenarios and configurations will be compared. One of the key aspects that is considered is that the robot must not collide with any of the obstacles, with the assumption that all the obstacles are stationary, and their shape doesn't change during the time that the path planning algorithm runs. Thus, the algorithm generates a discrete motion path for the robot between two locations in the workspace based on the current static state of workspace. In the experiments run in this project only *spherical* obstacles are considered.

ALGORITHMS

1) PROBABILISTIC ROADMAP (PRM):

Probabilistic Roadmap (PRM) path planning algorithm is quite basic and easy to implement compared to other sampling-based path planning algorithms, and one of the widely used ones in motion planning. This cannot be considered as the most efficient path planners out there, but it is one of those algorithms with a constant speed and can be used generically with many robots.

PRM algorithm generates motion plan in two phases. The first phase is a preprocessing phase to construct a graph with up to N connected configurations in the free space. The graph generated in the first phase is an undirected graph $G = (V, E)$, where the nodes V correspond to the configurations of the robot in free space C_{free}. The edges E correspond to straight line paths between two configurations with any collisions. The second phase works on the graph generated to find the shortest path possible between a given initial and end configurations.

PHASE 1:

```
$intialize\ (maxNCount)$
$intialize\ \big(G = (V, E)\big)\ as\ empty$
$V \leftarrow qStart$
$n = 0$
$while\ n < N$
        $qRand = sample\ configuration$; not in G;
                                last sample config should be $qGoal$
        $if\ qRand \in C_{free}$
                $n++$
```

```
            $V \leftarrow qRand$
            $U \leftarrow Near(G = (V, E), v, r)$
            $for\ each\ u \in U, do$
                $if\ collisionsFree(u), then\ E \leftarrow E \cup \{(v, u), (u, v)\}$
            $end\ for$
    $end\ while$
    $return\ G = (V, E)$
```

PHASE 2:

Use a shortest path algorithm like Dijkstra's algorithm on $G = (V, E)$ to find the shortest path possible.

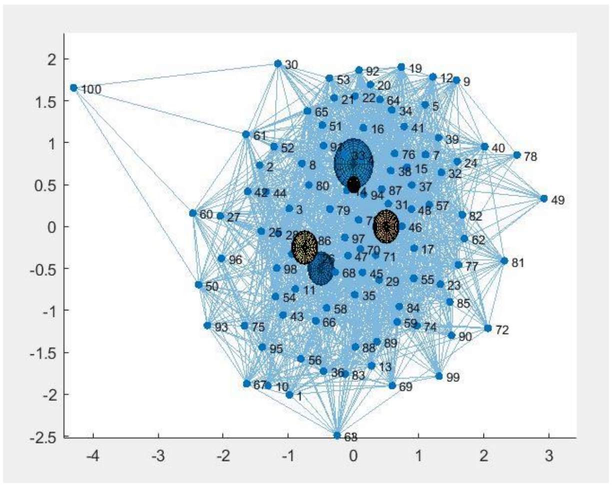
Figure 1: Shortest path applied from the graph

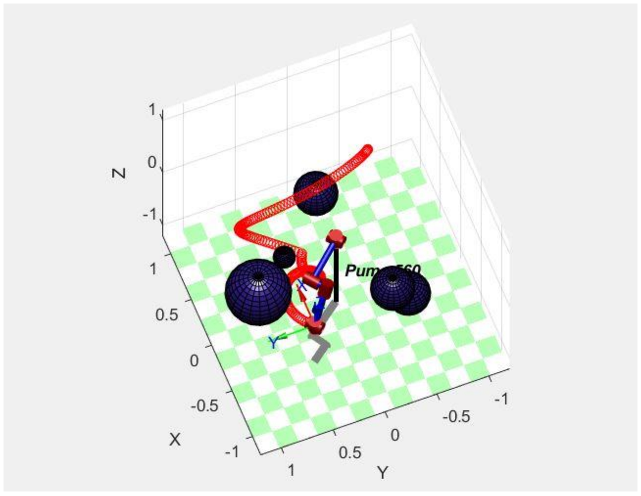

Figure 2: Graph generated for 100 free configurations

2) RAPIDLY EXPLORING RANDOM TREES (RRT):

Rapidly exploring Random Tree (RRT) is an algorithm to efficiently search higher dimensional spaces by randomly building a space-filling tree. Generating a random configuration in the free space C_{free} is done like PRM, but it builds a tree different from the graph in PRM.

As each sample configuration is generated, instead of finding other configurations within a specified radius that is done in PRM, RRT attempts a connection between the sampled configuration and the nearest configuration in the current tree state. If the connection edge is collision free, the sampled configuration is added with that edge to the current tree state. This way, with uniform sampling in the workspace, the tree grows proportional to its *Voronoi region.* Once every n turns the above process is attempted with the goal configuration instead of random sample. The entire process is repeated from the initial configuration till the goal configuration is added to the tree state. And then the tree path from initial state to goal config gives the collision free path to be taken by the robot in the workspace

```
$initialize\_empty\_tree(T = (V, E))$
$V \leftarrow qStart$
$while\ qGoal \notin V$
        $qRand \leftarrow sample\ configuration\ in\ free\ space =$
                    $\{if\ n^{th} loop\ then\ qGoal, else\ random\}$
        $qNear \leftarrow nearestVertex(qRand, V)$
        $if\ no\ collisions\ in\ edge(qNear, qRand)$
                $V \leftarrow qRand$
                $E \leftarrow edge(qNear, qRand)$
        $end\ if$
$end\ while$
$return\ T$
```

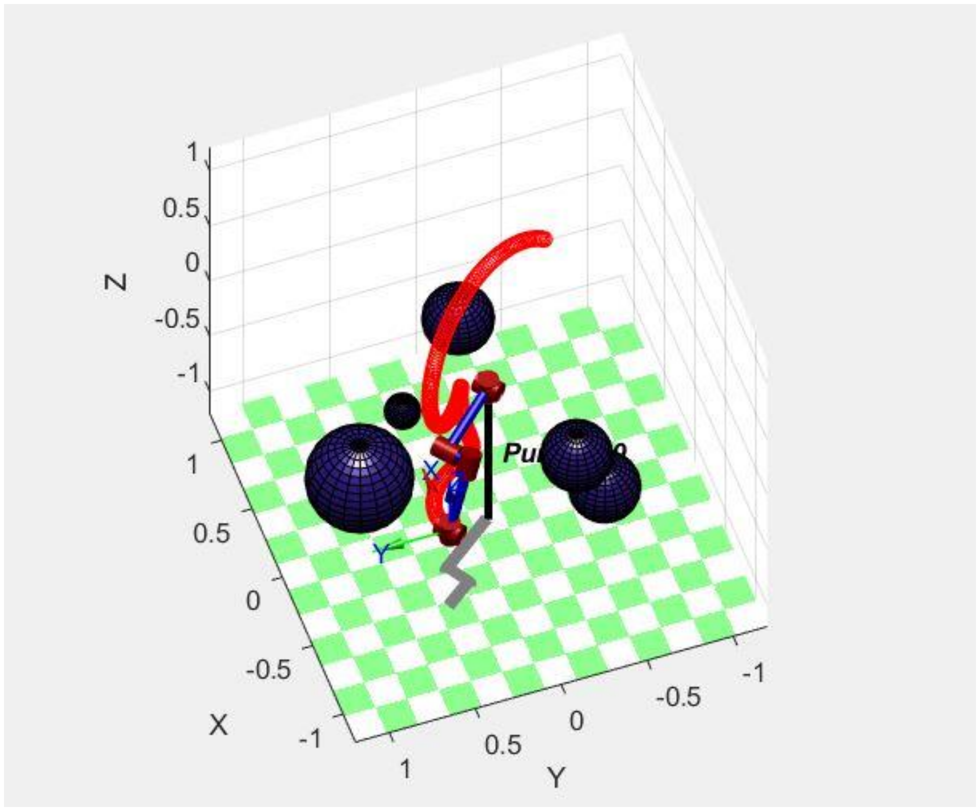

Figure 3: Shortest path found by RRT

EXPERIMENTS AND COMPARISONS

1) EXPERIMENT 1:

The first experiment was to compare the path length generated from running both RRT and PRM with similar configurations. The max configuration count for PRM and count to attempt qGoal for RRT was set to 50. For PRM the connection radius was set at 6 units. Number of obstacles was 5, and all obstacles had the same location, shape and size. The start and goal coordinates were also same. Each algorithm was run for 20 time and path length each iteration was noted.

The top five results for both algorithms

RRT	**PRM**
4	0
4	3
4	3
4	3
5	4

The best and worst path lengths

RRT	**PRM**
4	3
8	0 (*because no path found); 6 (the longest path found)*

Average path length for RRT from 20 runs = 5.65

Average path length for PRM from 20 runs = 3.85

Point to note here, PRM did not return any path in one of the runs. This is because the graph generated in phase 1 of PRM was a disconnected graph with start and goal configurations in separate components.

2) EXPERIMENT 2:

This experiment compared the time took by both PRM and RRT with same configurations. The max configuration count for PRM and count to attempt qGoal for RRT was set to 50. For PRM the connection radius was set at 6 units. Number of obstacles was 5, and all obstacles had the same location, shape and size. The start and goal coordinates were also same. Each algorithm was run for 20 time and path length each iteration was noted

The results are as follows

	RRT	**PRM**
Best time	0.67 s	5.07 s
Worst time	17.31 s	6.63 s
Average time	5.11 s	5.72 s

Point to note here, RRT returned a valid path in all 20 runs, whereas PRM returned a valid path for only 14 runs.

Running the same experiment only for RRT, but with the qGoal attempted at every 10th C_{free} config resulted in the following.

	RRT
Best time	0.13 s
Worst time	0.54 s
Average time	0.27 s

3) EXPERIMENT 3:

This experiment compared the ratio of number of samples the were rejected and accepted into C_{free} in both PRM and RRT algorithm runs. The max configuration count for PRM and count to attempt qGoal for RRT was set to 50. For PRM the connection radius was set at 6 units. Number of obstacles was 5, and all obstacles had the same location, shape and size. The start and goal coordinates were also same. Each algorithm was run for 20 time and path length each iteration was noted

The results are as follows

	RRT – samples rejected	RRT – samples accepted	PRM – samples rejected	PRM – samples accepted
Best	6	50	2	50
Worst	72	400	8	50
Average	26	130	5.6	50

4) EXPERIMENT ON PRM ONLY:

The following experiments were run on PRM alone

- When everything else was kept constant and only the max C_{free} count was changed the following results were obtained with respect to obtaining a valid path from the algorithm run for 20 times each.

Max C_{free} count	No valid paths returned count
10	16
25	9
50	3
100	0
150	0

- The number of samples rejected remained in the same range in spite of modifying the other configurations

CONCLUSION

This Project presents a comprehensive analysis between applications or PRM and RRT with a *Puma 560 Robot* in the project workspace. The comparison Report above looked to answer some typical questions in path planning like path length, running time, sampling ratio, generation of valid paths and choosing the right configurations.

From the above analysis the following can be concluded

- PRM does not generate valid path in most cases on a low sampling space, whereas RRT generates a valid path every time.
- The number of samples rejected in PRM is usually low and remains within a constant range. This can be attributed to sampling neighbors based on a threshold radius, which mostly won't have a collision as the neighbors are added to main graph since they are collision free. On the other hand, RRT has a high sample rejection rate.
- Time taken to run PRM is almost constant and depends on the number of C_{free} configurations to be added. This in turn adds an overhead, since PRM only returns valid configurations for higher cap of C_{free} configurations. RRT on the other hand runs very fast for lower sampling count for $qGoal$ and becomes slower as that cap increases
- From an algorithm perspective, PRM spends considerable time in Pre-Processing, with neighbor selection being the most important tradeoff.

On a high level, it can be concluded that, with a lenient constraint on computational time, PRM outperforms RRT in finding a path for the robot from start to goal. However, if time is of utmost importance, RRT is the better way to go.

CODE BASE

The code base can be found in the GitHub repository here - https://github.com/abishek-sampath/Robotics-PRM-vs-RRT-path-planning. Please follow the following steps to run the programs.

- Clone/Download the repo to your local environment.
- Matlab and Peter Corke's Robotics toolbox should be installed.
- Open Matlab with the project folder.
- Run $startup\_rvc$ in the command window to startup the Robotics Toolbox

- Run the program with the following command syntax:

start_path_planning(algorithmType,startX, startY, startZ, endX, endY, endZ)
Required:
    algorithmType
        ('PRM' or 'RRT')
Optional:
    startX, startY, startZ, endX, endY, endZ
        (Default coordinates will be used)

RESOURCES

1) https://parasol.tamu.edu/groups/amatogroup/research/motionPlanning.php
2) https://hal.laas.fr/hal-01986202/document
3) https://en.wikipedia.org/
4) http://www.cs.columbia.edu/~allen/F15/NOTES/Probabilisticpath.pdf
5) Northeastern University, CS 5335 - Robotics Science and Systems – Course Material and resources.