ROBOTICS SCIENCE AND SYSTEMS(CS5335)

PROJECT REPORT

ON

Puma 560 Path Planning using PRM and RRT

Submitted By:

NIRUPAMA SHARMA (001859308)

RUCHITHA MIDIGARAHALLI SHANMUGHA SUNDAR(001838207)

Simulation Youtube Link: https://www.youtube.com/watch?v=VtcB4kd3sW0&feature=youtu.be

GitHub Link To Source Code: https://github.com/Nirucool/Codefactory/tree/master/Robotics_Motion_Planning

Puma 560 Path Planning using PRM and RRT

Problem Statement: Use Corke's Matlab robotics toolbox to implement PRM for the 6DOF Puma 560 robot. Also, test your code using a more complex obstacle environment and include at least three obstacles in the workspace of the robot.

Initial Configurations:
Obstacles: 5 spheres of radius 0.125 each
Start and Goal configurations :

```
qStart = [0.78 0.78 -0.78 -0.78 0 0.78];
xGoal = [-0.75,0.0,-0.5];
```

Multi Query and Single Query methods for Path Planning

Multi Query method: PRM (Probabilistic Roadmap) is a multi-query path planning algorithm. In multi query method there are two phases, generating a roadmap and finding the shortest path. This algorithm entails pre-processing overhead as it involves graph generation. Also, the algorithm is not very efficient while responding to environmental changes.

Single Query method: RRT (Rapidly Exploring Random Tree) is a single query path planning algorithm. In single query path planning generation of roadmap and determination of path, both takes place in a single phase. Therefore, there is no pre-processing overhead.

Probabilistic Roadmap Algorithm (PRM)

1. Generate random nodes from -pi to pi in C-space.
2. Check if the nodes are in collision, collision free nodes are added to the node matrix.
3. Find k nearest neighbours for all nodes in node matrix.
4. Generate roadmap by checking whether edges are in collision and add collision free edges to undirected graph G(V,E) with weights(Euclidean distance in C-space).

RoadMap Construction Algorithm

Input:
n : number of nodes to put in the roadmap
k : number of closest neighbors to examine for each configuration
Output:
A roadmap $G = (V, E)$

$V \leftarrow \emptyset$
$E \leftarrow \emptyset$
while $|V| < n$ **do**
  repeat
    $q \leftarrow$ a random configuration in $\mathcal{Q}$
  until q is collision-free
  $V \leftarrow V \cup \{q\}$
end while
for all $q \in V$ **do**
  $N_q \leftarrow$ the k closest neighbors of q chosen from V according to *dist*
  for all $q' \in N_q$ **do**
    if $(q, q') \notin E$ **and** $\Delta(q, q') \neq$ NIL **then**
      $E \leftarrow E \cup \{(q, q')\}$
    end if
  end for
end for

5. Given the start (qStart)and goal (qGoal) configurations find the shortest path from qStart to qGoal using Dijkstra's shortest path algorithm.

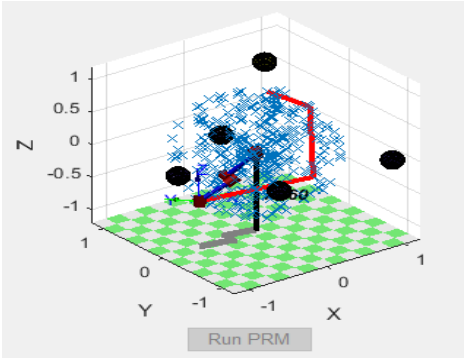

Figure 1 PRM Path Planning for 6 DOF Puma 560

Steps in PRM with elapsed time

1. Find 400 collision free nodes. Elapsed Time=~2.07sec

```
while size(nodes)+1 <400
    %generate a random node
    qRand = (rand(1,dof) * 2 * pi) - pi;
    if robotCollision(rob,qRand,sphereCenter1,sphereCenter2,sphereCenter3,.
            sphereCenter4,sphereCenter5,sphereRadius)
        continue;
    end
    nodes = [nodes; qRand;];
end
```

2. Find 10 nearest neighbours of each of 400 nodes. Elapsed Time=~239.20 sec

```
while (size(firstKNodes,1) <= 10)
    if count >= size(eDist,1)
        break;
    end
    if val(count,:) == 0
        count = count +1;
        continue;
    end
    q1 =  nodes(indx(count,:),:);
    collision = edgeCollision(rob,node,q1,sphereCenter1,sphereCenter2,sphereCenter3,.
        sphereCenter4,sphereCenter5,sphereRadius);
    if collision
        count = count +1;
        continue;
    end

    firstKNodes= [firstKNodes; indx(count,:) val(count,:);];
    count = count +1;
end
```

3. Finding shortest path from qStart to qGoal using Dijkstra's algorithm. Elapsed Time=~11.3 sec

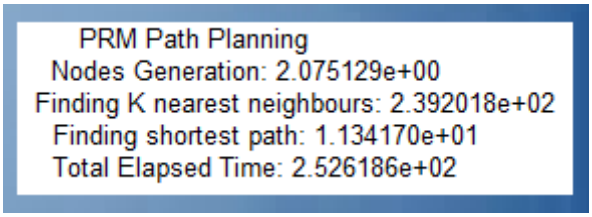

Figure 2 PRM Elapsed Time (in seconds) for each of the above steps (Matlab result for 6DOF Puma 560)

Pre-Processing Step

Finding K nearest neighbours of the collision free nodes: On one hand, since each collision check for a connection is expensive these should be avoided as much as possible. On the other hand, too few connections will give a disconnected graph. Connected components in the graph play a crucial role here. Such components should be connected into larger components if possible. Also, new components can be created in unexplored parts of the configuration space.

Two observations to be considered are as follows:

- It is not useful to make connections to nodes that are already in the same connected component

from a complexity point of view.

- It is not useful to connect to nodes that lie too far away.

Given these two observations, the following techniques can be derived:

1. Nearest K: Instead of looking at all the neighbours, the sample node n should be connected to the nearest K nodes in the graph. The rationale is that nearby nodes lead to short connections that can be checked efficiently. The magic number for K is 10, which works for most situations.
2. Component K (up to K nearest neighbours from each connected component): Sample node n should be connected to at most K nodes in each connected component. For example, if K=2, the sample node n can connect to at most 2 nodes in each connected component, shown in Figure 1. Component K is a method that is trying to bring separate pieces of graphs together.
3. Euclidean distance (The distance of neighbour): Here the distance of neighbour is changed to adjust the graph.

A merge method of 1 and 3, gives all points within some radius up to K. For example, if 400 nodes are within the radius, the algorithm only picks the top 10. That restricts the growth of undirected Graph G(V,E) , reducing the pre-processing overhead for PRM.

Performance Improvement in PRM using Lazy PRM

Firstly, assume the C-space is clearly open, and then create a dense PRM without ANY collision checking. When given qStart and qGoal, the algorithm will find the path from the start point through the graph to the goal. This process will be very fast. And then, take the path just found and check collisions. Different from normal PRM, now the program only needs to check the collision just along the path, instead of to check collisions of the whole graph. If any collision is found (the path is broken), then remove the edge and re-plan the path. The algorithm will re-route around the broken edge and repeat the whole process again until it gets a path to the goal.

Worst case complexity of Lazy PRM is same as normal PRM but average performance is better.

RRT (Rapidly Exploring Random Tree)

Single Query RRT Algorithm

1. Generate random nodes from -pi to pi.
2. Find the nearest neighbour using Euclidean distance in C-space and extend the tree until qGoal is reached.

```
BUILD_RRT(q_init)
 1  T.init(q_init);
 2  for k = 1 to K do
 3      q_rand ← RANDOM_CONFIG();
 4      EXTEND(T, q_rand);
 5  Return T
```

```
EXTEND(T, q)
 1  q_near ← NEAREST_NEIGHBOR(q, T);
 2  if NEW_CONFIG(q, q_near, q_new) then
 3      T.add_vertex(q_new);
 4      T.add_edge(q_near, q_new);
 5      if q_new = q then
 6          Return Reached;
 7      else
 8          Return Advanced;
 9  Return Trapped;
```

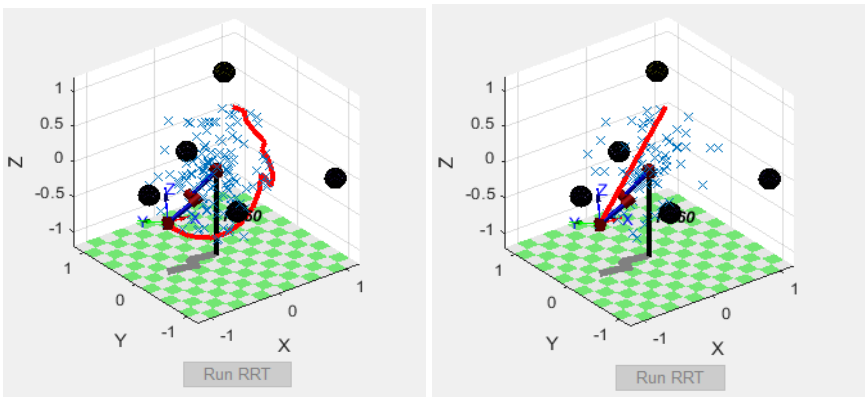

Figure 3 (i)RRT Path Planning for 6 DOF Puma 560
(ii) RRT with Smoothening

Elapsed Time: Collision free nodes are added to tree and the search for qGoal occurs both at the same time, hence there is no pre-processing step unlike PRM.

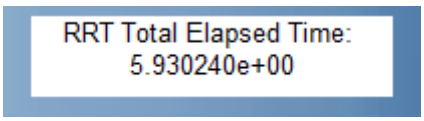

Figure 4 Elapsed Time for RRT

RRT Conclusion:

1. Firstly, the algorithm is sensitive to the metric for evaluating a configuration,
2. Secondly, it searches for nearest neighbours with linear time(O(n)) while some advanced nearest-neighbour searching methods can compute the nearest neighbour in near-logarithmic time (O(log n)).
3. Lastly, the algorithm bases itself on random sampling, which may yield relatively poor performance for a simple problem

Comparative Study of PRM and RRT cases:

PRM Elapsed Time (in seconds)/ k=5	**PRM Elapsed Time (in seconds)/ k=10**	**RRT Elapsed Time (in seconds)**
149.3574	250.9792	6.2855
141.0212	251.9914	6.6053
143.2593	251.2978	6.0788
142.2032	260.4617	8.0389
140.7427	258.8741	4.6686
153.0029	256.2567	4.2543
152.8335	260.5742	7.2107
156.3141	277.3718	5.7083
168.1896	271.0810	3.4702
156.1281	306.3053	5.7932
Average Elapsed Time= 150.3052	Average Elapsed Time=264.51932	Average Elapsed Time=5.811

PRM1: PRM with K=5 nearest neighbours

PRM2: PRM with K=10 nearest neighbours

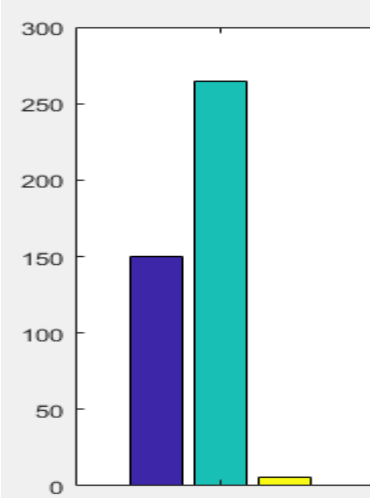

Figure 5 PRM1 PRM2 RRT

Observation 1:The above data shows that pre processing step in PRM is directly proportional to the search for k nearest neighbours for the creation of undirected graph. The more the number of nearest neighbours, the more connected graph will be and poorer will be the performance of PRM.(PRM2>PRM1>RRT).

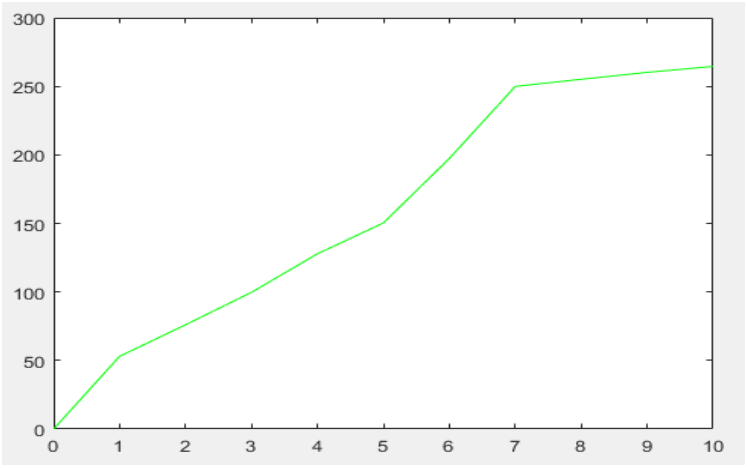

Figure 6 Plot of k(number of nearest neighbours/x axis) vs average elapsed time(in seconds/y axis) for PRM

Elapsed Time for Dijkstra's /for K=10(in seconds)	**Elapsed Time for Dijkstra's /for K=5(in seconds)**
14.8041	9.9374
14.3823	7.8029
14.2696	8.3725
14.4334	7.8463
14.1458	7.8707
14.0023	8.0678
14.1284	9.0852
14.4968	8.8020
14.2286	9.4094
14.2007	8.3738
Average Time=14.3092	Average Time=8.5568

PRM1: PRM with K=5 nearest neighbours

PRM2: PRM with K=10 nearest neighbours

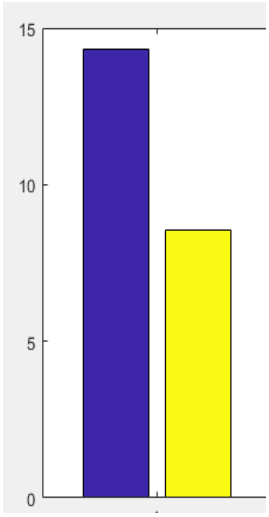

Figure 7 PRM2 PRM1

Observation 2:The above data shows that Dijkstra's algorithm for finding shortest path from qStart to qGoal is directly proportional to the pre-processing step of finding k nearest neighbours in PRM. Hence, it can be concluded that Dijkstra's performance will decrease if the graph is more connected or value of k is more.

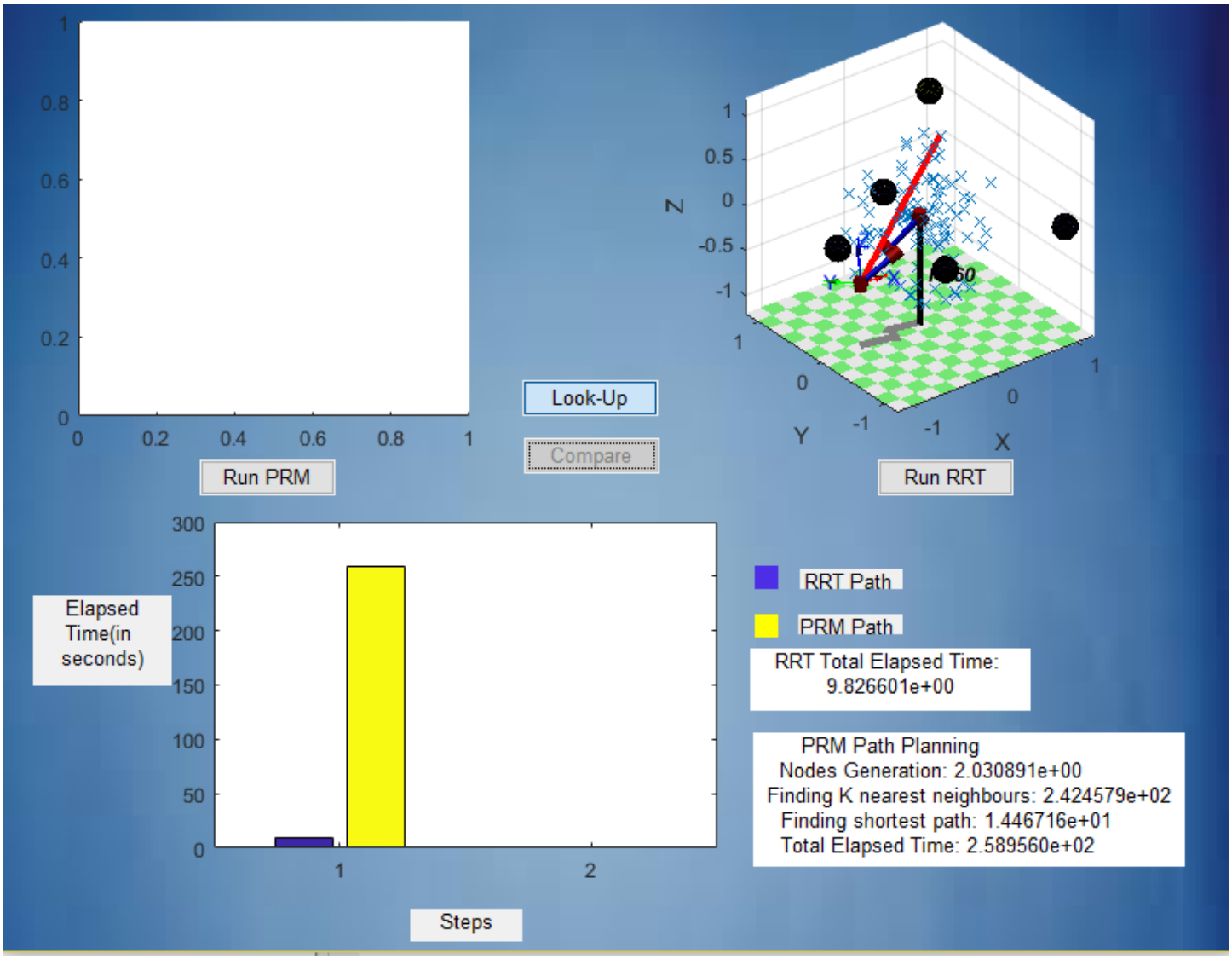

Figure 8 Matlab GUI showing Elapsed time for PRM and RRT path planning for same start and goal configurations for 6DOF Puma 560(Single run results)