Autonomous Robotics
RRT Algorithm
Lab 2 Report

Emre Ozan Alkan
{emreozanalkan@gmail.com}
MSCV-5

February 24, 2014

1 Introduction

Path planning also known as motion planning is an important task for autonomous robotics. It's aim is to find shortest or at least optimal path between start and goal positions. It is hard and highly demanded task today. In this lab, we implemented RRT (Rapidly-Exploring Random Trees) to solve simple path planning problem in Matlab.

1.1 Environment

In this lab, we had two 2D environment. In these environments; obstacles and empty spaces are marked as 1 and 0 respectively. We had to find optimal path between given start position and the goal position.

(a) 683x803 Map Environment (b) 687x802 Maze Environment

Figure 1: Sample Environments

1.2 RRT Algorithm

RRT algorithm is designed to search high dimensional spaces within configuration space by randomly building tree. It compose of creating randomly points or with some probabilty selecting goal, creating vertex and edge on that direction, and so on till it reaches the goal.

So in this lab, we implemented RRT Algorithm that creating random points, searhing nearest vertices to that point, creating new point to that direction with given distance and checking if it and its edge belongs the free space, and so on till it reaches the goal.

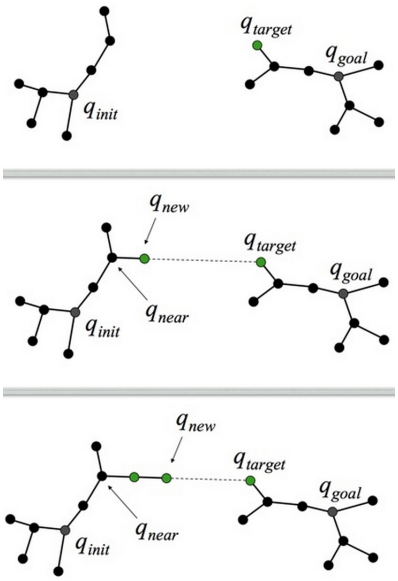

Figure 2: Graphical Demonstration of RRT Algorithm *[3]

1.3 Path Smoothing

Since RRT algorithm creating random points, tend to build tree towards random points, mostly solution path is not optimal and very noisy. Hence, we need some kind of noise reduction method for our path. Therefore, we asked to implement smoothing method on path to get shorter and less noisy path. We used 'greedy approach' where each time we trying our connect our latest point with the start position and so on.

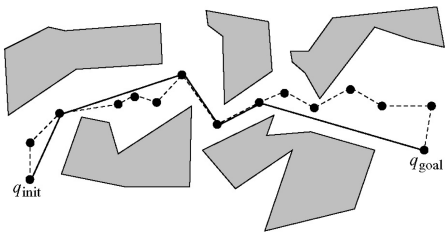

Figure 3: Path Smoothing

2 Implementation

Our RRT Algorithm implementation has two parts, RRT and smoothing for avoiding noises in RRT path. RRT find path but not shortest nor optimal. Smoothing method at least pruning the path, trying to connect points from goal to most reachable one, so path becomes more optimal.

2.1 Problems and Difficulties

In this lab I had difficulties on continuous coordinates and using these coordinates in right dimension of map. I had long debugging session to find a bug due to order of x and y coordinates where you already mentioned it in "important" title, but I forgot it.

2.1.1 Equation

One of the other difficulty was finding new point through randomly created point direction with given distance. In this part, I needed to read and search a lot. I decided to find vector and normalize it to find point with given distance.

Lets $\mathbf{q}\_near = (x_1, y_1)$ and $\mathbf{q}\_rand = (x_0, y_0)$.

So $\mathbf{v} = q\_rand - q\_near$. We normalize this to $\mathbf{u} = \frac{\mathbf{v}}{||\mathbf{v}||}$ to get unit vector to use to find points with given distance.

Then finding point with distance d becomes easy: $qNew(x_3, y_3) = q\_near + d*u$

After using the approach*[2] below, things became more clear.

2.1.2 Edge Free Space

Checking if edge belongs to free space was also challenging. Thanks to the approach above, and using 10 intermediate points as suggested, I was able to achieve it. This time, instead of checking with given distance, I divided my total distance between two points to number of intermediate point count, say delta_q, and each iteration I added delta_q to get new coordinate and checked it if it belongs to free space.

2.2 Functions

Here is the the brief preview to functions I implemented to solve problem.

2.3 RRT

My RRT implementation consist of 6 main functions; 'rrt' - main function for algorithm, 'findQNear' - finding nearest vertices to created random point, 'findQNew' - finds point towards created random point with given delta distance, 'isEdgeQNearQNewBelongsFreeSpace' - checking if edge between q_near and q_new is belongs to free space, 'isQGoalOnQNearQNewEdge' - checking of goal is on the edge between q_near and q_new, 'fillSolutionPath' - creates solution path from given edges and vertices.

Here are short preview to my rrt.m file with functions:

Listing 1: rrt.m

```
1
2  function [vertices, edges, path] = rrt(map, q_start, q_goal, k, delta_q, p)
3  tic;
4
5  clc;
6
7  if nargin < 3
8      error('First 3 parameter is required: 2D map matrix, start and goal coordinates.');
9  elseif nargin < 4
10     k = 10000;
11     delta_q = 50;
12     p = 0.3;
13 elseif nargin < 5
14     delta_q = 50;
15     p = 0.3;
16 elseif nargin < 6
17     p = 0.3;
18 end
19
20 checkParameters(map, q_start, q_goal, k, delta_q, p);
21
22 % map: <683x803> means: height: 683, width: 803
23 [mapHeight, mapWidth] = size(map); % columns as the x axis and the rows as the y axis.
24
25 q_start = int32(q_start); % q_start = [80, 70] means: q_start(1): y, q_start(2): x
26 q_goal = int32(q_goal); % q_goal = [707, 615] means: q_goal(1): y, q_goal(2): x
27 vertices = q_start; % Initialize the vertices variable with q_start
28
29 edges = int32.empty(0, 2); % Initialize the edges variable as empty
30
31 q_rand = int32.empty(0, 2);
32
33 q_near = int32.empty(0, 2);
34
35 q_new = int32.empty(0, 2);
36
37 for ii = 1 : k % For k samples repeat
38
39     if rand() < p % With p probability use q_rand = q_goal
40         q_rand = q_goal;
41     else % Otherwise generate q_rand in the dimensions of the map.
42         q_rand = int32([randi(mapWidth) randi(mapHeight)]); % columns as the x axis and
               the rows as the y axis.
43     end
44
45     [q_near, qNearIndex] = findQNear(q_rand, vertices); % Find q_near from q_rand in
           vertices
46
47     q_new = findQNew(q_near, q_rand, delta_q); % Generate q_new at delta_q distance from
           q_near in the direction to q_rand
48
49     if q_new(1) < 1 || q_new(2) < 1 || q_new(1) > mapWidth || q_new(2) > mapHeight
50         continue;
51     end
52
53     if map(q_new(2), q_new(1)) == 0 % If q_new belongs to free space
54
55         % If the edge between q_near and q_new belongs to free space
56         if isEdgeQNearQNewBelongsFreeSpace(map, q_near, q_new)
57
58             % Add q_new in vertices
59             vertices = [vertices; q_new];
60             % Add [index(q_new) index(q_near)] in edges
61             [qNewIndex, ~] = size(vertices);
62             edges = [edges; [int32(qNewIndex), int32(qNearIndex)]];
63
64             % If q_new == q_goal or q_goal is on the edge
65             if isequal(q_new, q_goal) || isQGoalOnQNearQNewEdge(q_near, q_new, q_goal)
```

```
66
67                  if ~isequal(q_new, q_goal) % if goal is not q_new but its on edge
68                      % goal is on the last edge, remove last vertex and add it as goal
69                      vertices = vertices(1 : (end - 1), :);
70                      vertices = [vertices; q_goal];
71                  end
72
73                  % Fill path and stop break RRT function
74                  path = fillSolutionPath(edges, vertices);
75
76                  % rrtDraw(map, q_start, q_goal, vertices, edges, path);
77
78                  toc;
79
80                  return;
81              end
82          end
83      end
84  end
85
86      path = int32.empty(0, 2);
87      % rrtDraw(map, q_start, q_goal, vertices, edges, path);
88      toc;
89
90      error('RRT: solution not found :(');
91  end
92
93  function checkParameters(map, q_start, q_goal, k, delta_q, p)
94  ...
95  end
96
97  % Find q_near from q_rand in vertices.
98  function [q_near, qNearIndex] = findQNear(q_rand, vertices)
99  ...
100 end
101
102 % Generate q_new at delta_q distance from q_near in the direction to q_rand
103 function [q_new] = findQNew(q_near, q_rand, delta_q)
104 ...
105 end
106
107 % If the edge between q_near and q_new belongs to free space
108 function [isBelongsFreeSpace] = isEdgeQNearQNewBelongsFreeSpace(map, q_near, q_new)
109 ...
110 end
111
112 % q_near <--> q_goal <--> q_new
113 function [isQGoalOnEdge] = isQGoalOnQNearQNewEdge(q_near, q_new, q_goal)
114 ...
115 end
116
117 % Fill path and stop break RRT function
118 function [path] = fillSolutionPath(edges, vertices)
119 ...
120 end
121
122 % Plots the RRT result
123 function rrtDraw(map, q_start, q_goal, vertices, edges, path)
124 ...
125 end
```

2.4 Path Smoothing

My RRT Path Smoothing implementation consist of 2 main functions; 'smooth' - main function for smoothing path, 'isEdgeBelongsFreeSpace' - incrementally checking if edge between two point is belongs

to free space with given delta distance.

Here are short preview to my smooth.m file with functions:

Listing 2: smooth.m

```
1
2  function [path_smooth] = smooth(map, path, vertices, delta)
3
4  path_smooth = path(1); % initing with goal
5  currentIndex = 1; % path array iterator
6  currentSmoothIndex = numel(path); % path reverse array iterator
7
8  while currentIndex < numel(path)
9
10     while currentIndex < currentSmoothIndex
11
12         if isEdgeBelongsFreeSpace(map, vertices(path(currentSmoothIndex), :), vertices(
               path(currentIndex), :), delta)
13             path_smooth = [path_smooth, path(currentSmoothIndex)];
14             currentIndex = currentSmoothIndex;
15             break;
16         else
17             currentSmoothIndex = currentSmoothIndex - 1;
18         end
19
20     end
21
22     currentSmoothIndex = numel(path);
23
24 end
25
26 % rrtSmoothDraw(map, path_smooth, vertices);
27
28 end
29
30 function [isBelongsFreeSpace] = isEdgeBelongsFreeSpace(map, startPoint, endPoint, delta)
31 ...
32 end
33
34 function rrtSmoothDraw(map, path_smooth, vertices)
35 ...
36 end
```

3 Results

Here is the results I obtained with RRT algorithm, and after path smoothing. Parameters k, delta_q, p and delta for smoothing is used as 10000, 50, 0.3 and 5 respectively.

3.1 RRT

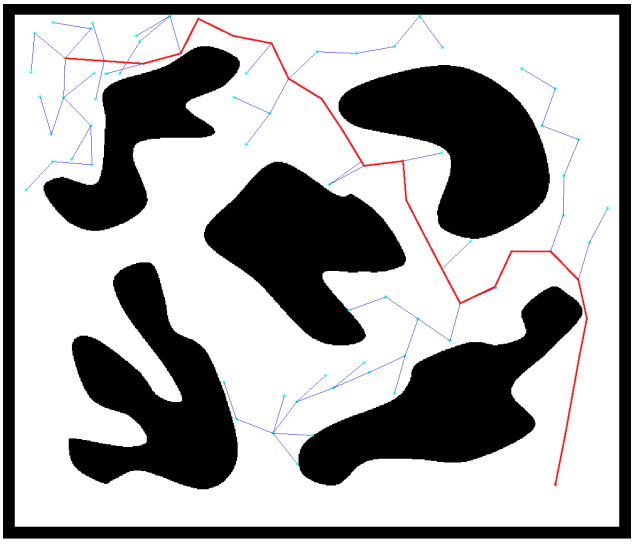

Figure 4: RRT Algorithm on Map

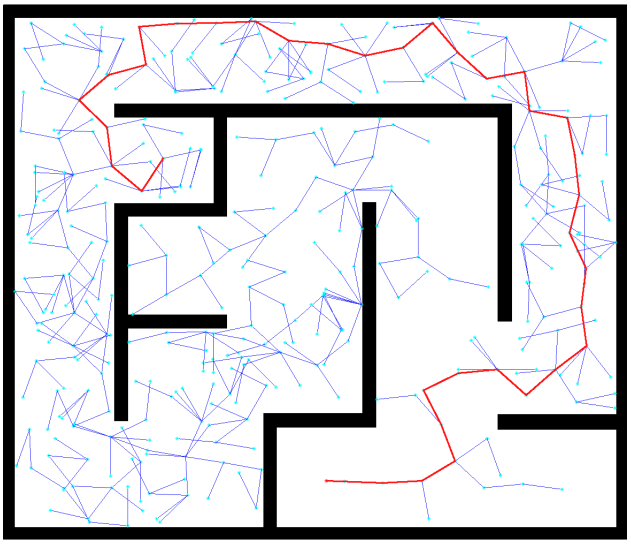

Figure 5: RRT Algorithm on Maze

3.2 Smoothing Path

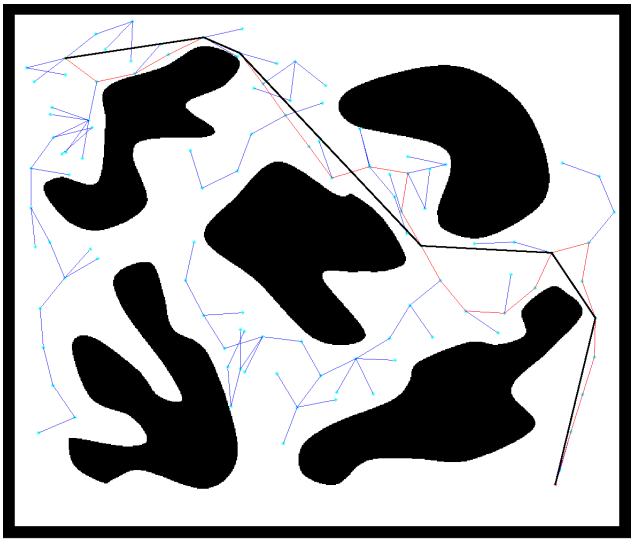

Figure 6: RRT with Smooth Path on Map

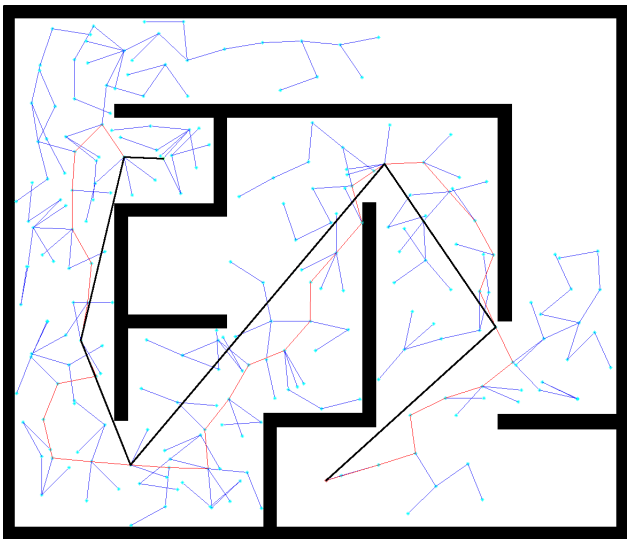

Figure 7: RRT with Smooth Path on Maze

4 References

1. http://math.stackexchange.com/questions/175896/finding-a-point-along-a-line-a-certain-distance-away-from-another-point
2. http://stackoverflow.com/questions/1061276/how-to-normalize-a-vector-in-matlab-efficiently-any-related-built-in-function
3. http://kovan.ceng.metu.edu.tr/ asil/old/_1./hw4.html
4. http://en.wikipedia.org/wiki/Rapidly_exploring_random_tree