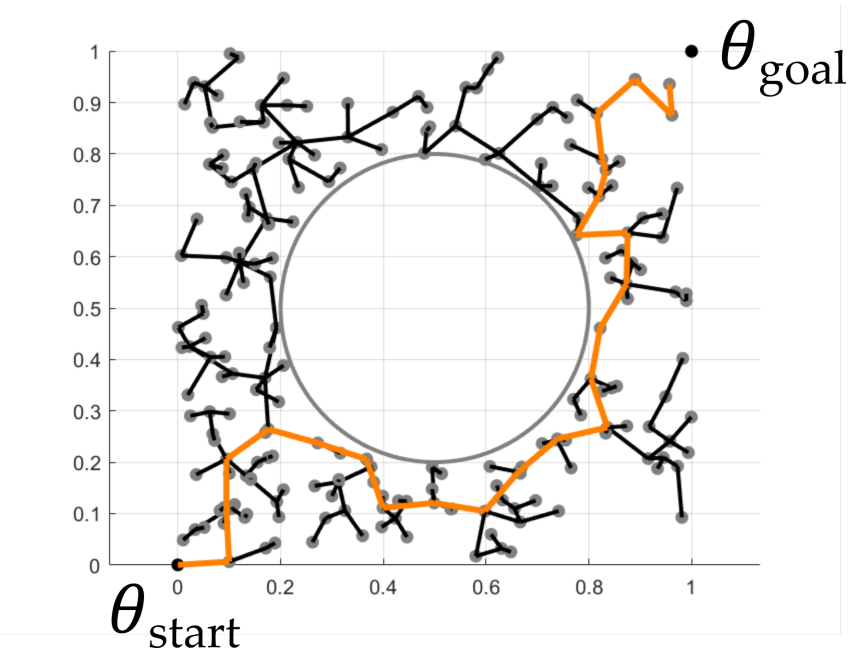
Practice Set 31

Robotics & Automation
Dylan Losey, Virginia Tech

Using your textbook and what we covered in lecture, try solving the following problems. For some problems you may find it convenient to use Matlab (or another programming language of your choice). The solutions are at the end of the document. **Download the starter code provided with this practice set.**



Let's get a working implementation of the Rapidly-exploring Random Tree (RRT) algorithm. Consider the 2-DoF environment shown above where θ is the (x, y) coordinates of the drone. You want the drone to plan a trajectory from θ_{start} to θ_{goal} while avoiding the obstacle. You are given the following information about the environment:

- The start position is $\theta_{start} = [0, 0]^T$ and the goal is $\theta_{goal} = [1, 1]^T$
- The obstacle center is $[0.5, 0.5]^T$ and the obstacle radius is 0.3
- The bounds of the environment are $x \in [0, 1]$ and $y \in [0, 1]$.

Problem 1

Modify the starter code to implement RRT and find a trajectory from start to goal.

Problem 2

Within our RRT code we use Δ to set the maximum distance between θ_{near} and θ_{new}. Try increasing and decreasing Δ. What is one pro of having a higher Δ? What is one con?

Problem 3

By default RRT samples θ_{rand} uniformly at random (i.e., every joint position is equally likely to be sampled). How could you *bias* this sampling to reach the goal position more quickly?

Problem 1

Modify the starter code to find a trajectory from start to goal.

A completed implementation of the RRT code is attached at the end of this document.

Problem 2

Within our RRT code we use Δ to set the maximum distance between θ_{near} and θ_{new}. Try increasing and decreasing Δ. What is one pro of having a higher Δ? What is one con?

We only perform collision checking at the point θ_{new}. Hence, if θ_{new} is far from θ_{near}, there may be a collision along the edge between θ_{new} and θ_{near} that we have failed to check for[a]. Accordingly, one **con** of increasing Δ is that we are more likely to end up with collisions along our final trajectory. One **pro** is that — because we can explore the environment in fewer samples (and jump over obstacles) — we will reach the goal more quickly in expectation.

[a] **Note:** This can be fixed by checking for collisions along the edge between θ_{new} and θ_{near}. Consider discretizing this edge into k points, and then checking each point for a collision.

Problem 3

By default RRT samples θ_{rand} uniformly at random (i.e., every joint position is equally likely to be sampled). How could you *bias* this sampling to reach the goal position more quickly?

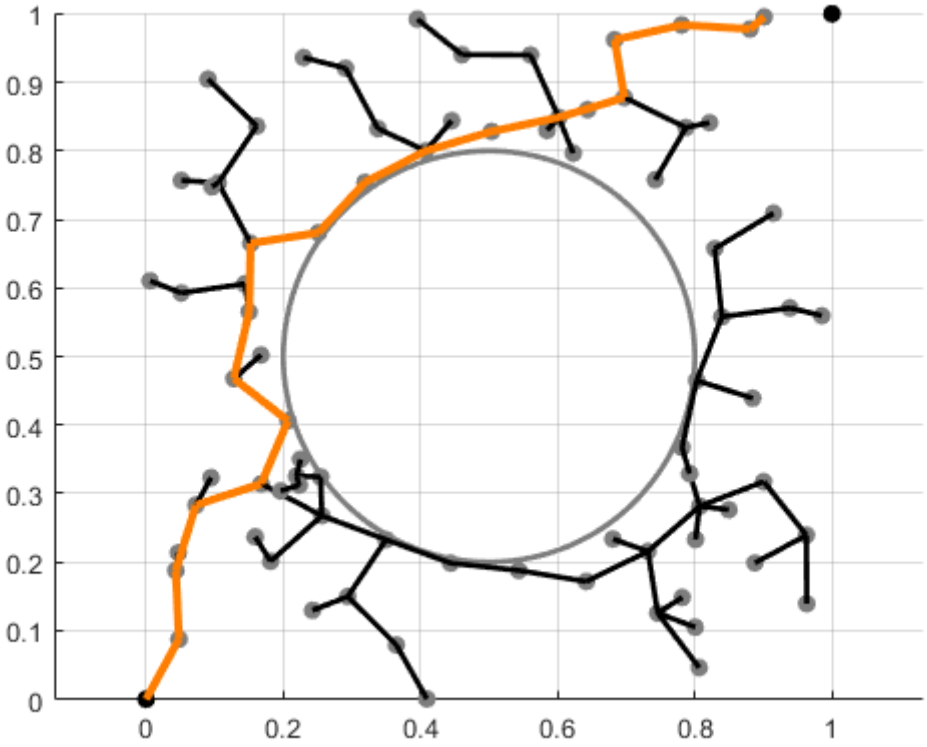
With probability $p < 1$, set θ_{rand} as θ_{goal}. Otherwise sample randomly as normal. Consider the following lines:

```
if rand < p

  theta_rand = theta_goal;

else

  theta_rand = rand(2,1);

end
```

Note: This will accelerate RRT in some environments, but slow it down in others. For example, consider a situation where the robot must move *away* from the goal to get around the obstacles. Here biasing the robot's search towards the goal position will increase the number of samples in expectation.

```
clear
close all

% environment
theta_start.coord = [0; 0];
theta_goal = [1; 1];
center = [0.5; 0.5];
radius = 0.3;

% parameters
epsilon = 0.1;
delta = 0.1;
N = 1000;

% visualize environment
figure
grid on
hold on
axis([0, 1, 0, 1])
axis equal
viscircles(center', radius, 'Color', [0.5, 0.5, 0.5]);
plot(0, 0, 'ko', 'MarkerFaceColor', 'k')
plot(1, 1, 'ko', 'MarkerFaceColor', 'k')

% initialize tree
theta_start.parent = 0;
G(1) = theta_start;

for idx = 1:N

    % stop if the last node in G is close to theta_goal
    if norm(G(end).coord - theta_goal) < epsilon
        break
    end

    % sample random joint position
    theta_rand = rand(2,1);

    % find node in G nearest to theta_rand
    dist = zeros(length(G), 1);
    for jdx = 1:1:length(G)
        dist(jdx) = norm(G(jdx).coord - theta_rand);
    end
    [~, theta_near_index] = min(dist);
    theta_near = G(theta_near_index);

    % take a step from theta_near towards theta_rand
    vec_to_rand = theta_rand - theta_near.coord;
    dist_to_rand = norm(vec_to_rand);
    if dist_to_rand < delta
        theta_new.coord = theta_rand;
    else
```

```
            theta_new.coord = theta_near.coord + delta * ...
                vec_to_rand/dist_to_rand;
        end

        % check if theta_new is collision free
        if norm(theta_new.coord - center) < radius
           continue
        end

        % if collision free, add theta_new to tree with parent theta_near
        theta_new.parent = theta_near_index;
        G = [G, theta_new];

        % plot node and edge
        plot(theta_new.coord(1), theta_new.coord(2), 'o', 'Color', ...
            [0.5, 0.5, 0.5], 'MarkerFaceColor', [0.5, 0.5, 0.5])
        line([theta_near.coord(1), theta_new.coord(1)],
     [theta_near.coord(2), ...
            theta_new.coord(2)], 'Color', 'k', 'LineWidth', 2);
        drawnow

    end

    % work backwards from the final node to the root of the tree
    child_theta = G(end);
    while child_theta.parent ~= 0
        parent_theta_index = child_theta.parent;
        parent_theta = G(parent_theta_index);
        line([child_theta.coord(1), parent_theta.coord(1)], ...
            [child_theta.coord(2), parent_theta.coord(2)], ...
            'Color', [1, 0.5, 0], 'LineWidth', 3);
        child_theta = parent_theta;
    end
```



Published with MATLAB® R2020b