RRT

RRT (Part A)

0.0/10.0 points (graded)

In this problem, we will implement a Rapidly Exploring Randomized Tree (RRT). For easy visualization, we will consider a two dimensional example. However, it will be relatively straightforward to extend your code to handle higher dimensional environments.

(a) The most computationally expensive portion of the algorithm is typically the collision checker. Write code that will take a set of points $\{x_i\}$ in two dimensions and determine if a given set of points $\{x_{0,i}\}$ **are in the convex hull** of the points $\{x_i\}$. The output of your code should be a row vector of boolean variables ("collisionFree" in the code below) whose i^{th} element is "true" if and only if $x_{0,i}$ is **not** in the convex hull.

Hint: The Matlab functions convhull and inpolygon may be useful to you here (but of course, you don't have to use them). Also, check your code by making plots a few times before submitting it.

```
%%% Problem Setup (DO NOT MODIFY) %%%
xi = randn(2,10); % Each column is a point (x,y)
x0 = randn(2,10);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%% Your code here %%%%%%%%%%%%%%%%%%


collisionFree = ;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%



```

Unanswered

```
xi = randn(2,10); % Each column is a point (x,y)
x0 = randn(2,10);

% Get convex hull of points
K = convhull(xi(1,:),xi(2,:));
xobs = xi(1,K);
yobs = xi(2,K);

% Check to see if x0 is in polygon
in = inpolygon(x0(1,:),x0(2,:),xobs,yobs);

collisionFree = ~in;
```

Run Code

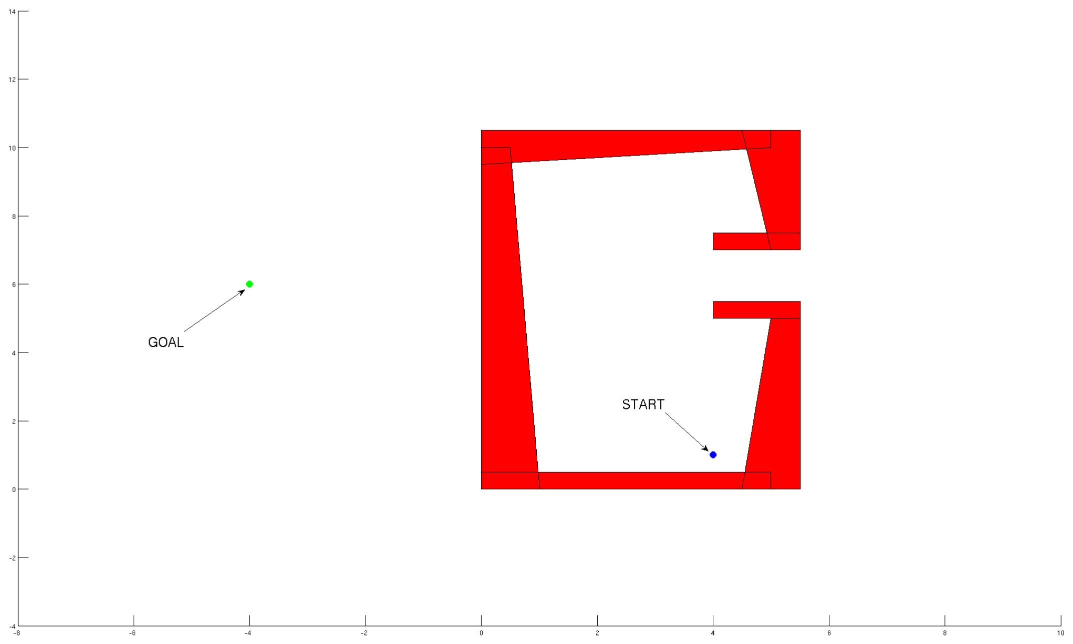
Submit You have used 0 of 3 attempts

Answers are displayed within the problem

RRT (Part B)

0.0/20.0 points (graded)

(b) Next, we will implement the entire RRT algorithm and test it out on a "bug trap" environment (shown in the figure below). This kind of environment is typically quite challenging for motion planning algorithms since it requires discovering and navigating through a narrow passage in order to get out of the "trap". However, hopefully you'll see that the RRT algorithm is able to do this with relative ease.



We have provided some stub code here. Fill in the portions that say "FILL ME IN". Run your code a few times to get intuition for how the RRT grows. Once you are satisfied that the code is running correctly, paste in the RRT your code found below (this is the variable "rrt_verts_grade" in the code). The first row should be the start state and the last row should be (close to) the goal state. You may find the "clipboard" Matlab function useful for copying data in variables.

```
rrt_verts_grade = [];

```

Unanswered

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Function for finding closest vertex in tree
function closest_vert = closestVertex(rrt_verts,xy)

% Find distances from points in rrt_verts to xy
diffs = rrt_verts - repmat(xy,1,size(rrt_verts,2));
dists = sqrt(diffs(1,:).^2 + diffs(2,:).^2);

% Find closest vertex
[~,minind] = min(dists);

closest_vert = rrt_verts(:,minind);

end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Function for collision checking
function collFree = isCollisionFree(Obs,xy)

collFree = zeros(length(Obs),1);
for k = 1:length(Obs)
    % Convex hull
    K = convhull(Obs{k}(1,:),Obs{k}(2,:));
    xobs = Obs{k}(1,K);
    yobs = Obs{k}(2,K);

    % Check if point is inside polygon
    in = inpolygon(xy(1),xy(2),xobs,yobs);

    if ~in
        collFree(k) = 1;
    end


end

collFree = all(collFree);

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% rrt_bugtrap.m

%% Code setup (Do not modify, but please read) %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Obstacles. Each obstacle is a cell in Obs. An obstacle is
% represented as a convex hull of a number of points. These points are
% stored in the cells of Obs.
% First row is x, second is y (position of vertices)
w = 0.5;
Obs{1} = [0 0;5 0;5 w;0 w]';
Obs{2} = [0 0;2*w 0;w 10;0 10]';
Obs{3} = [0 10-w;5 10;5 10+w;0 10+w]';
Obs{4} = [5-w 0;5+w 0;5+w 5;5 5]';
Obs{5} = [5-w 10+w;5+w 10+w;5+w 7;5 7]';
Obs{6} = [4 5;5+w 5;5+w 5+w;4 5+w]';
Obs{7} = [4 7;5+w 7;5+w 7+w;4 7+w]';

% Bounds on world
world_bounds_x = [-8,10];
world_bounds_y = [-4,14];


% Draw obstacles
figure(1); clf; hold on;
axis([world_bounds_x world_bounds_y]);

for k = 1:length(Obs)
    patch(Obs{k}(1,:),Obs{k}(2,:),'r');
end

% Start and goal positions
xy_start = [4;1]; plot(xy_start(1),xy_start(2),'bo','MarkerFaceColor','b','MarkerSize',10);
xy_goal = [-4;6]; plot(xy_goal(1),xy_goal(2),'go','MarkerFaceColor','g','MarkerSize',10);

% Initialize RRT. The RRT will be represented as a 2 x N list of points. So
% each column represents a vertex of the tree.
rrt_verts = zeros(2,1000);
rrt_verts(:,1) = xy_start;
N = 1;
nearGoal = false; % This will be set to true if goal has been reached
minDistGoal = 0.25; % This is the convergence criterion. We will declare
                    % success when the tree reaches within 0.25 in distance
                    % from the goal. DO NOT MODIFY.

% Extension parameter
d = 0.5; % This controls how far the RRT extends in each step. DO NOT MODIFY.

%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% RRT algorithm
while ~nearGoal
    % Sample point
    rnd = rand(1);
    % With probability 0.05, sample the goal. This promotes movement to the
    % goal.
    if rnd < 0.05
        xy = xy_goal;
    else
        % Sample from space with probability 0.95
        xs = (world_bounds_x(2) - world_bounds_x(1))*rand(1) + world_bounds_x(1);
        ys = (world_bounds_y(2) - world_bounds_y(1))*rand(1) + world_bounds_y(1);
        xy = [xs;ys];
    end

    %% FILL ME IN %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Check if sample is collision free
    collFree = isCollisionFree(Obs,xy); % Write this function.
                                        % Your code from part (a) will be useful here.

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    % If it's not collision free, continue with loop
    if ~collFree
        continue;
    end

    %% FILL ME IN %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % If it is collision free, find closest point in tree
    closest_vert = closestVertex(rrt_verts(:,1:N),xy); % Write this function

    % Extend tree towards xy from closest_vert
    new_vert = closest_vert + d*(xy - closest_vert)/norm(xy - closest_vert);

    % Check if new_vert is collision free
    collFree = isCollisionFree(Obs,new_vert); % Same function you wrote before.
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


    % If it is not collision free, continue with loop
     if ~collFree
        continue;
     end

    % Plot extension (Comment the next 3 lines out if you want your code to
    % run a bit quicker. The plotting is useful for debugging though.)
    figure(1)
    plot(new_vert(1),new_vert(2),'bo','MarkerFaceColor','b','MarkerSize',5);
    line([closest_vert(1),new_vert(1)],[closest_vert(2),new_vert(2)]);


    %% DO NOT MODIFY THIS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % If it is collision free, add it to tree
```

```
    N = N+1;
    if N > size(rrt_verts,2)
        rrt_verts = [rrt_verts zeros(size(rrt_verts))];
    end
    rrt_verts(:,N) = new_vert;

    % Check if we have reached goal
    if norm(xy_goal-new_vert) < minDistGoal
        break;
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


end

% Plot vertices in RRT
plot(rrt_verts(1,:),rrt_verts(2,:),'bo','MarkerFaceColor','b','MarkerSize',5);

%% Submit rrt_verts_grade for grading %%%%
format long
rrt_verts_grade = rrt_verts(:,1:N)'
clipboard('copy',rrt_verts_grade);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Run Code

Submit You have used 0 of 3 attempts

Answers are displayed within the problem