```
clc;
clearvars;
% Define the start and goal points
start = [3, 3];
goal = [10.5, 9.5];

% Define the size of the workspace
xlim = [2, 11];
ylim = [2, 11];

% Define the number of iterations and the step size for the tree expansion
numIterations = 3000;
stepSize = 0.15;

% safe distance from the obstacle
a=0.2;

% Define the obstacles (each obstacle is a polygon defined by its vertices)
obstacle{1} = [4, 4; 4, 9; 5, 9; 5, 4];
obstacle{2} = [7, 3; 7, 8; 8, 8; 8, 3];
obstacle{3} = [7, 9; 7, 10; 10, 10; 10, 9];

box=[xlim(1)-a,ylim(1)-a; xlim(1)-a,ylim(2)+a;xlim(2)+a,ylim(2)+a;xlim(2)+a,ylim(1)-a];

obstacles{1} = [4-a, 4-a; 4-a, 9+a; 5+a, 9+a; 5+a, 4-a];
obstacles{2} = [7-a, 3-a; 7-a, 8+a; 8+a, 8+a; 8+a, 3-a];
obstacles{3} = [7-a, 9-a; 7-a, 10+a; 10+a, 10+a; 10+a, 9-a];

% Initialize the tree with the start point
tree.vertices = start;
tree.edges = [];

% Iterate the tree expansion
for i = 1:numIterations
    % Sample a random point in the workspace
    qRand1 = [randi([xlim(1), xlim(2)]), randi([ylim(1), ylim(2)])];
    qRand2=goal;
    qarray=[qRand1;qRand2];
    index = sum(rand >= cumsum([0, 0.8 0.2]));
    qRand=qarray(index,:);


    % Find the nearest point in the tree to the random point
    [qNearest, qIndex] = findNearest(tree.vertices, qRand);

    % Extend the tree from the nearest point towards the random point
    qNew = extend(qNearest, qRand, stepSize);
```

```
    % Check if the new point collides with any of the obstacles
    if ~checkCollision(qNearest, qNew, obstacles)
        % Add the new point to the tree
        tree.vertices = [tree.vertices; qNew];
        tree.edges = [tree.edges; qIndex, size(tree.vertices, 1)];

        % Check if the new point is close enough to the goal
        if norm(qNew - goal) < stepSize
            % Extend the tree from the new point to the goal
            qFinal = extend(qNew, goal, stepSize);

            % Check if the final point collides with any of the obstacles
            if ~checkCollision(qNew, qFinal, obstacles)
                % Add the final point to the tree
                tree.vertices = [tree.vertices; qFinal];
                tree.edges = [tree.edges; size(tree.vertices, 1) - 1, size(tree.↙
vertices, 1)];

                % Stop the expansion and return the path
                path = getPath(tree.vertices, tree.edges, start, size(tree.vertices,↙
1));
                break;
            end
        end
    end
end



% Plot the final tree and path
figure;
hold on
axis([0 12 0 12]);
xlabel("X-Coordinate")
ylabel("Y-Coordinate")
title("Path Using RRT")
plot(start(1), start(2), 'go', 'MarkerSize', 8, 'LineWidth', 2);
plot(goal(1), goal(2), 'ro', 'MarkerSize', 8, 'LineWidth', 2);

% plot the boundary of box
x=box(:, 1);
y=box(:, 2);
k = boundary(x,y);
hold on;
plot(x(k),y(k),'Color','k','LineWidth',2);

for i = 1:length(obstacle)
    x=obstacles{i}(:, 1);
    y=obstacles{i}(:, 2);
```

```
    k = boundary(x,y);
    hold on;
    plot(x(k),y(k),'Color','#7E2F8E');
    fill(obstacle{i}(:, 1), obstacle{i}(:, 2),'');
end

for i = 1:size(tree.edges, 1)
    getframe(gcf);
    hold on;
    plot(tree.vertices([tree.edges(i, 1), tree.edges(i, 2)], 1), tree.vertices([tree.
edges(i, 1), tree.edges(i, 2)], 2), 'b', 'LineWidth', 2);
end
if exist('path', 'var')
    getframe(gcf);
    hold on;
    plot(path(:, 1), path(:, 2), 'r', 'LineWidth', 2);
end

% Define the findNearest function
function [qNearest, qIndex] = findNearest(vertices, q)
    % Find the index of the nearest vertex to the random point
    [~, qIndex] = min(sum(bsxfun(@minus, vertices, q).^2, 2));

    % Get the nearest vertex
    qNearest = vertices(qIndex, :);
end

% Define the extend function
function qNew = extend(qNearest, qRand, stepSize)
    % Compute the unit vector towards the random point
    qDirection = (qRand - qNearest) / norm(qRand - qNearest);

    % Extend the tree from the nearest point towards the random point
    qNew = qNearest + stepSize * qDirection;
end

% Define the checkCollision function
function collision = checkCollision(q1, q2, obstacles)
    % Check if the line segment between the two points intersects any of the
obstacles
    collision = false;
    for i = 1:length(obstacles)
        if intersect(q1, q2, obstacles{i})
            collision = true;
            break;
        end
    end
end
```

```
% Define the intersect function
function intersection = intersect(q1, q2, obstacle)
    % Check if the line segment between the two points intersects the obstacle
    intersection = false;
    for i = 1:size(obstacle, 1)
        if i == size(obstacle, 1)
            j = 1;
        else
            j = i + 1;
        end
        if lineSegmentIntersection(q1, q2, obstacle(i, :), obstacle(j, :))
            intersection = true;
            break;
        end
    end
end

% Define the lineSegmentIntersection function
function intersection = lineSegmentIntersection(p1, p2, q1, q2)
    % Check if the line segments between the two pairs of points intersect
    intersection = false;
    v1 = p2 - p1;
    v2 = q2 - q1;
    t = cross([q1 - p1, 0], [v1, 0]) / cross([v1, 0], [v2, 0]);
    u = cross([q1 - p1, 0], [v2, 0]) / cross([v1, 0], [v2, 0]);
    if t >= 0 && t <= 1 && u >= 0 && u <= 1
        intersection = true;
    end
end

% Define the getPath function
function path = getPath(vertices, edges, start, goalIndex)
    % Compute the path from the start point to the goal point
    path = vertices(goalIndex, :);
    while goalIndex ~= 1
        goalIndex = edges(find(edges(:, 2) == goalIndex, 1), 1);
        path = [vertices(goalIndex, :); path];
    end
    path = [start; path];
end
```

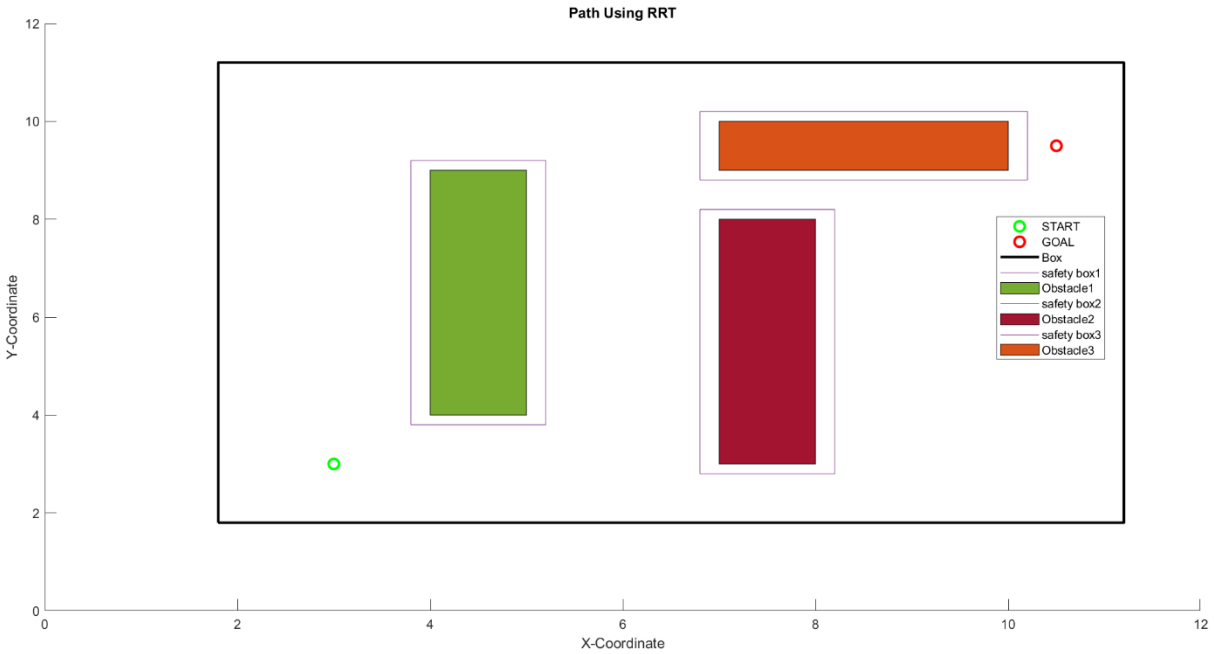


Figure1: Showing 3 Obstacles.

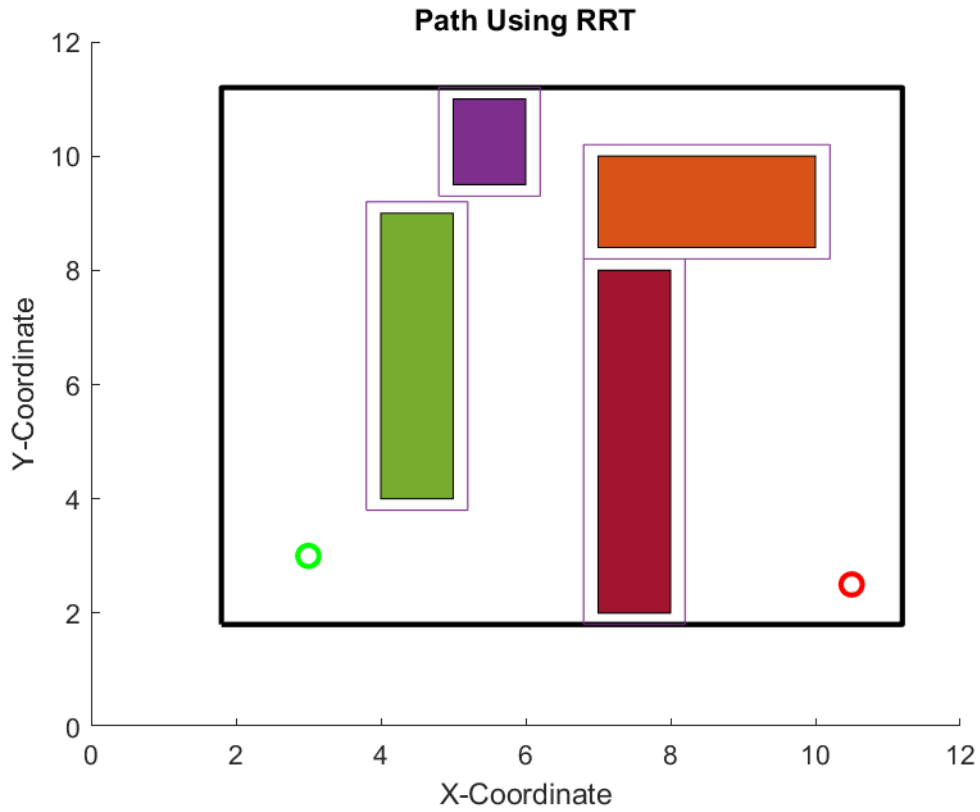


Figure2: Showing 4 Obstacles.

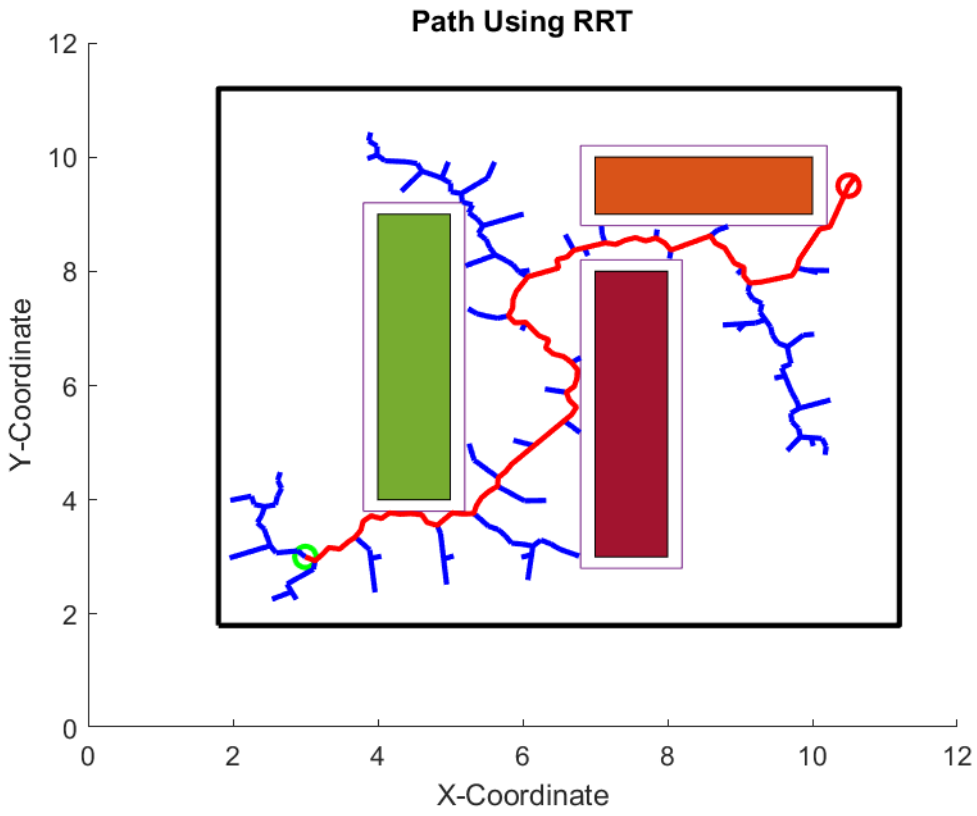
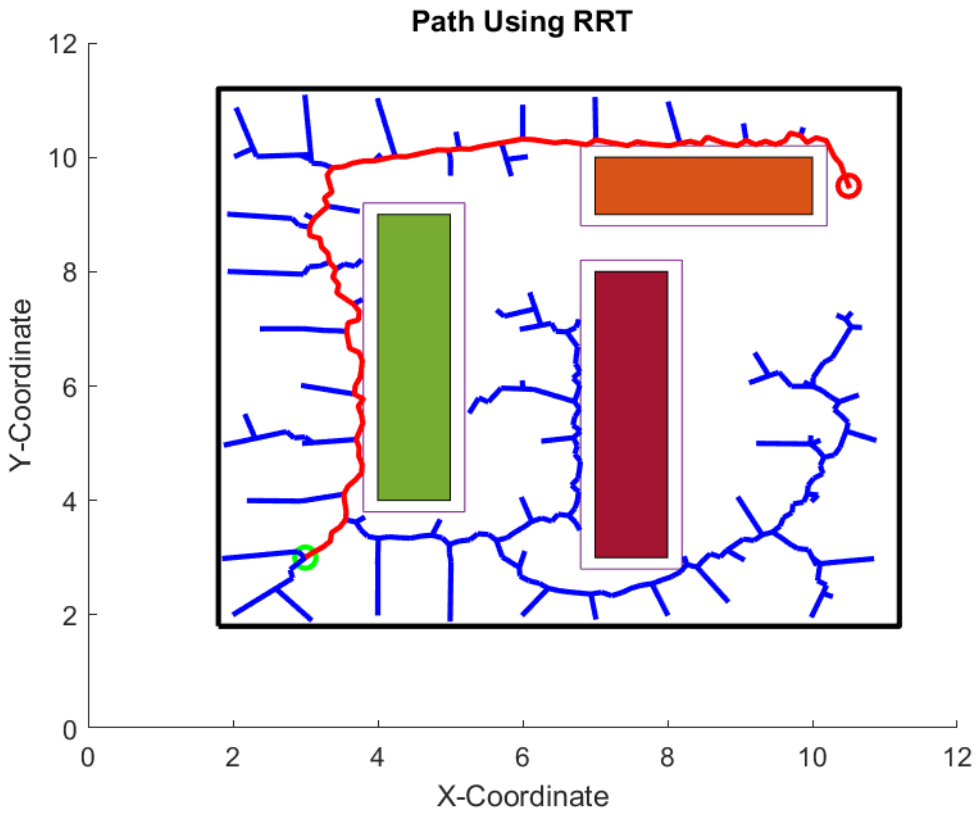


Figure3: Path Find Using RRT from start to goal.



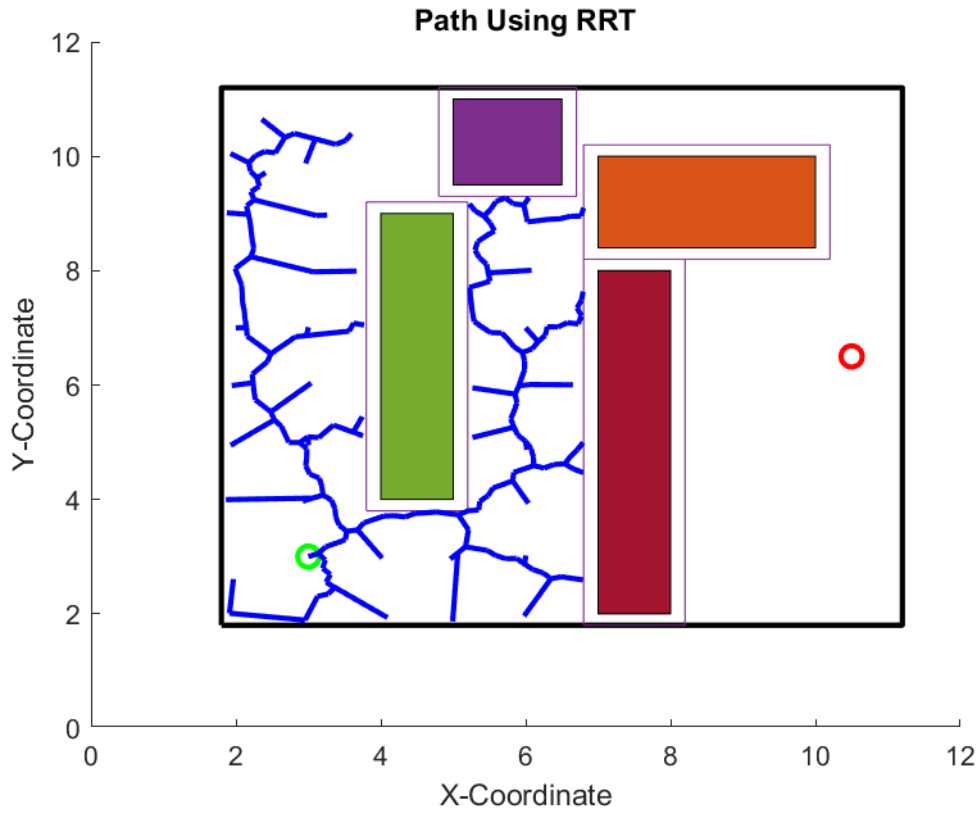


Figure5: Path Not Found