```
% Define the workspace and obstacles
x_min = -10; x_max = 10; y_min = -10; y_max = 10;
obstacle_1 = [-5, 8, 8, -5; 4, 4, 6, 6]; % Rectangle obstacle 1
obstacle_2 = [-8, 3, 3, -8; -3, -3, -1, -1]; % Rectangle obstacle 2
obstacle_3 = [6, 8, 8, 6; -6, -6, 2, 2]; % Rectangle obstacle 3

% Define the start and goal points
start_point = [-9, -9];
goal_point = [9, 5];

% Define the step size and threshold distance
delta = 0.005;
threshold = 0.8;

% Define the attractive and repulsive constants
alpha = 0.5;
beta = 50;

% Define the maximum number of iterations and the error tolerance
max_iter = 5000;
tolerance = 0.01;

% Initialize the path and the current point
path = start_point;
current_point = start_point;

% Loop until the goal is reached or the maximum number of iterations is reached
for i = 1:max_iter
    % Calculate the attractive force towards the goal
    attractive_force = alpha * (goal_point - current_point);

    % Calculate the repulsive force from the obstacles
    repulsive_force = zeros(1, 2);
    for j = 1:size(obstacle_1, 2)
        [distance,nearest_point]= min_distance(current_point, obstacle_1);
        if distance <= threshold
            repulsive_force = repulsive_force + beta * (1/distance - 1/threshold)^2 * ↙
(current_point - [obstacle_1(1, j), obstacle_1(2, j)]) / distance^3;
        end
    end
    for j = 1:size(obstacle_2, 2)
        [distance,nearest_point]= min_distance(current_point, obstacle_2);
        if distance <= threshold
            repulsive_force = repulsive_force + beta * (1/distance - 1/threshold)^2 * ↙
(current_point - [obstacle_2(1, j), obstacle_2(2, j)]) / distance^3;
        end
    end
    for j = 1:size(obstacle_3, 2)
        [distance,nearest_point]= min_distance(current_point, obstacle_3);
```

```
        if distance <= threshold
            repulsive_force = repulsive_force + beta * (1/distance - 1/threshold)^2 * ↙
(current_point - [obstacle_3(1, j), obstacle_3(2, j)]) / distance^3;
        end
    end



    % Calculate the total force on the robot
    total_force = attractive_force + repulsive_force;

%     Update the current point
    next_point = current_point + delta * total_force;
    current_point = next_point;

%         Check if the next point is inside the workspace
    if next_point(1) < x_min || next_point(1) > x_max || next_point(2) < y_min || ↙
next_point(2) > y_max
        disp('Error: Next point is outside the workspace.');
        break;
    end

%     Check if the next point collides with any obstacle
    if inpolygon(next_point(1), next_point(2), obstacle_1(1, :), obstacle_1(2, :)) || ↙
...
            inpolygon(next_point(1), next_point(2), obstacle_2(1, :), obstacle_2(2, ↙
:)) || ...
            inpolygon(next_point(1), next_point(2), obstacle_3(1, :), obstacle_3(2, ↙
:))
        disp('Error: Next point collides with an obstacle.');
        break;
    end


%     Add the next point to the path
%     getframe();
%     scatter(next_point(1),next_point(2), 'bo');
%     hold on;
    path = [path; next_point];

    % Check if the goal is reached
    if norm(next_point - goal_point) < tolerance
        disp('Goal reached!');
        break;
    end
end

% Plot the workspace, obstacles, start and goal points, and the path
figure;
```

```
hold on;
fill(obstacle_1(1, :), obstacle_1(2, :),'');
fill(obstacle_2(1, :), obstacle_2(2, :), '');
fill(obstacle_3(1, :), obstacle_3(2, :), '');
plot(start_point(1), start_point(2), 'bo', 'MarkerSize', 5, 'LineWidth', 2);
plot(goal_point(1), goal_point(2), 'ro', 'MarkerSize', 5, 'LineWidth', 2);
plot(path(:, 1), path(:, 2), 'g-', 'LineWidth', 2);
xlabel('X-Coordinate');
ylabel('Y-Coordinate');
axis equal;
xlim([x_min, x_max]);
ylim([y_min, y_max]);
title('Potential Field Path Planning');
legend('Obstacle 1', 'Obstacle 2', 'Obstacle 3','Start','Goal','Path');


function [min_dist,nearest_point]= min_distance(point, obstacle)

    point=[point(1);point(2)];
    % Find the nearest point on the perimeter of the rectangle
    min_dist = Inf;
    nearest_point = NaN(2,1);

    % Iterate over each line segment of the perimeter
    for i = 1:size(obstacle,2)
        % Define the endpoints of the line segment
        p1 = obstacle(:,i);
        if i == size(obstacle,2)
            p2 = obstacle(:,1);
        else
            p2 = obstacle(:,i+1);
        end

        % Find the nearest point on the line segment
        [dist, pt] = point_to_line_segment_distance(point, p1, p2);

        % Update the nearest point if the current distance is smaller
        if dist < min_dist
            min_dist = dist;
            nearest_point = pt;
        end
    end
end



function [dist, nearest_pt] = point_to_line_segment_distance(point, p1, p2)
% Calculates the distance between a point and a line segment defined by two
% endpoints, and returns the nearest point on the line segment.
```

```
% Calculate the vector from p1 to p2
v = p2 - p1;

% Calculate the vector from p1 to the point
w = point - p1;

% Calculate the squared length of the line segment
length_sq = sum(v.^2);

% Calculate the projection of w onto v
proj = dot(w,v)/length_sq;

% If the projection is outside the line segment, return the nearest endpoint
if proj < 0
    dist = norm(point - p1);
    nearest_pt = p1;
elseif proj > 1
    dist = norm(point - p2);
    nearest_pt = p2;
% Otherwise, return the nearest point on the line segment
else
    nearest_pt = p1 + proj*v;
    dist = norm(point - nearest_pt);
end
end
```

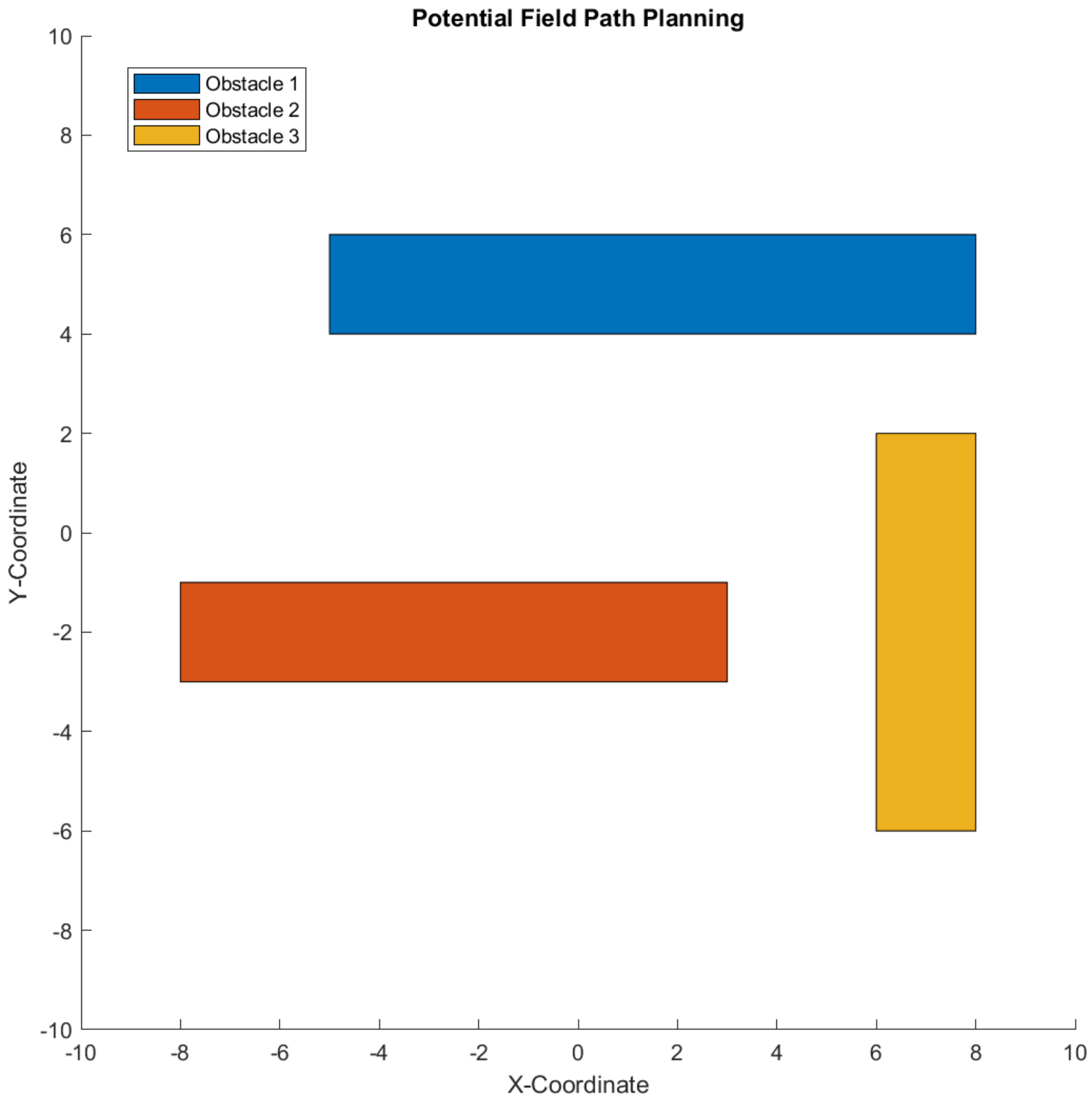

Figure 1: Showing Three Obstacles

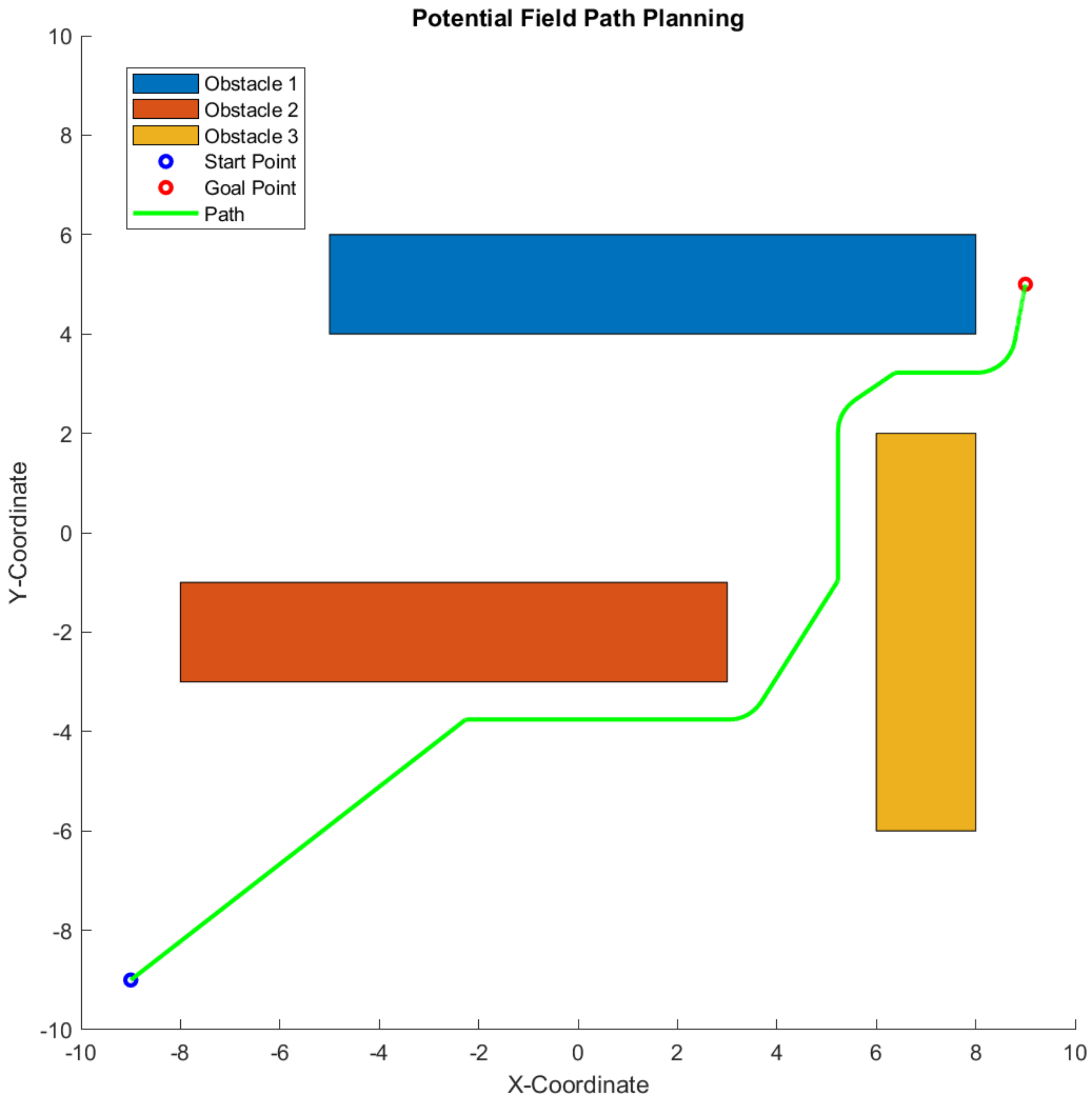

Figure 2: Path Find Using Potential Method

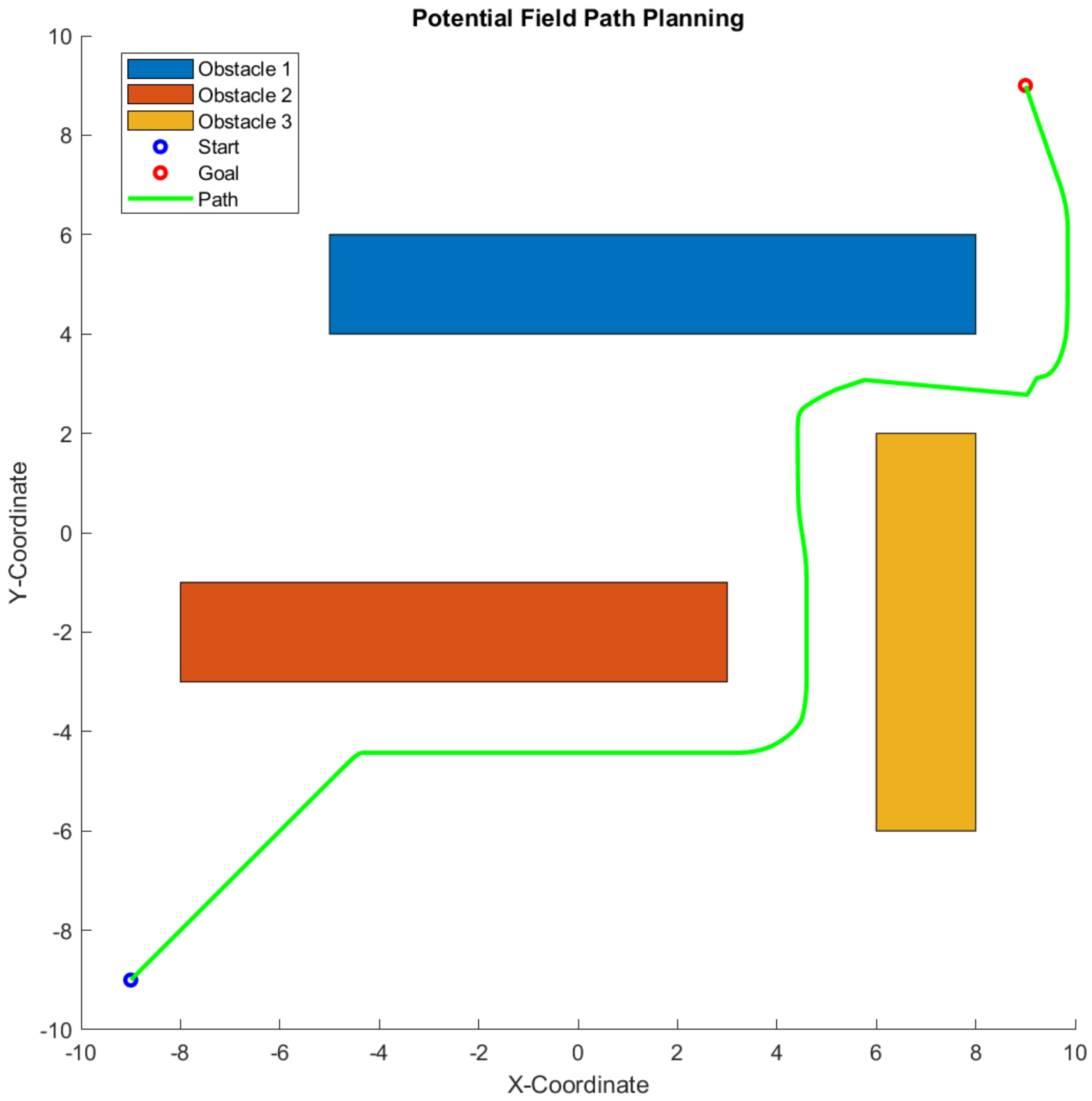

Figure 3: Path find with 2 times repulsive potential with previous case

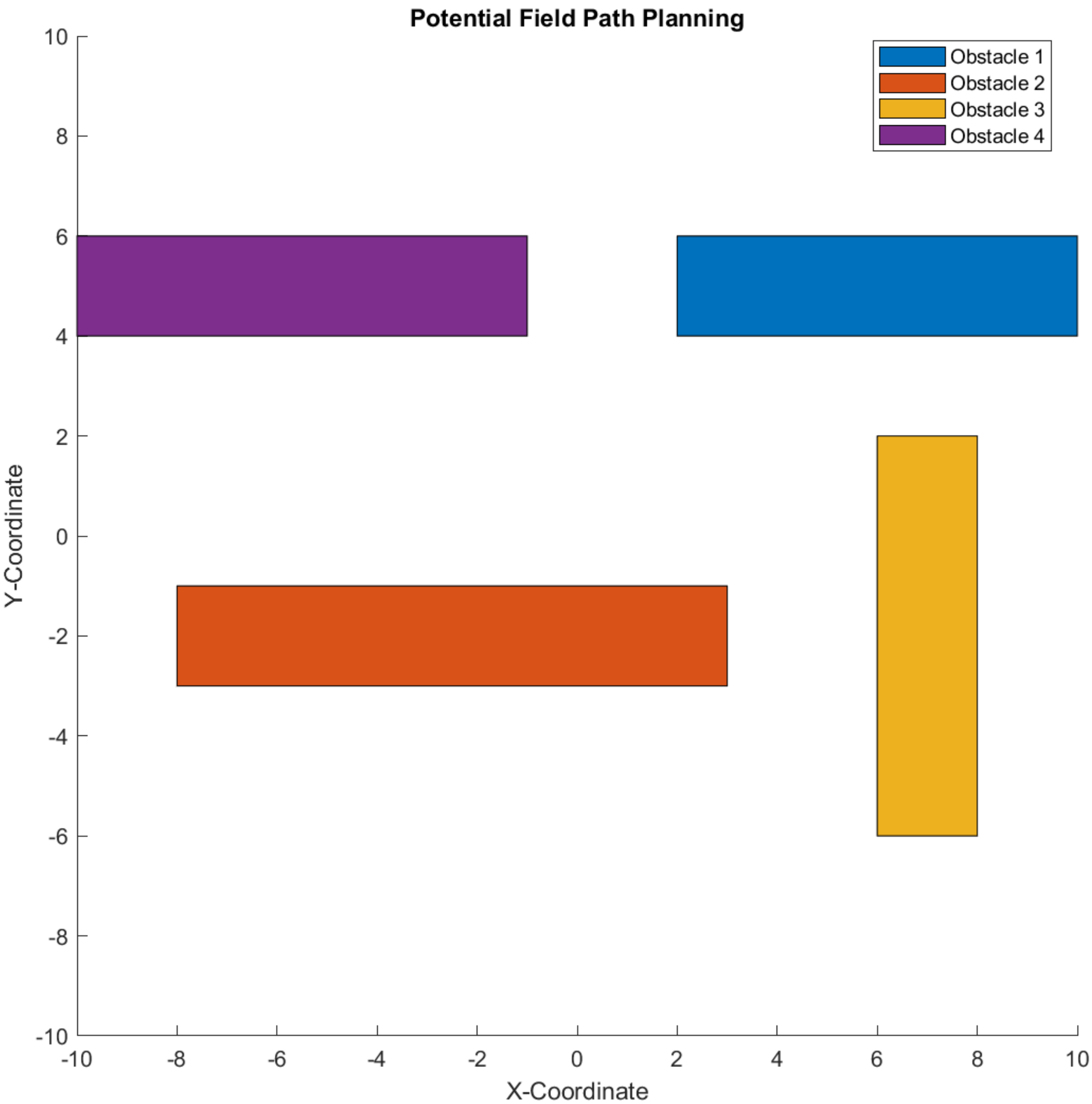

Figure 4: Showing 4 Obstacles

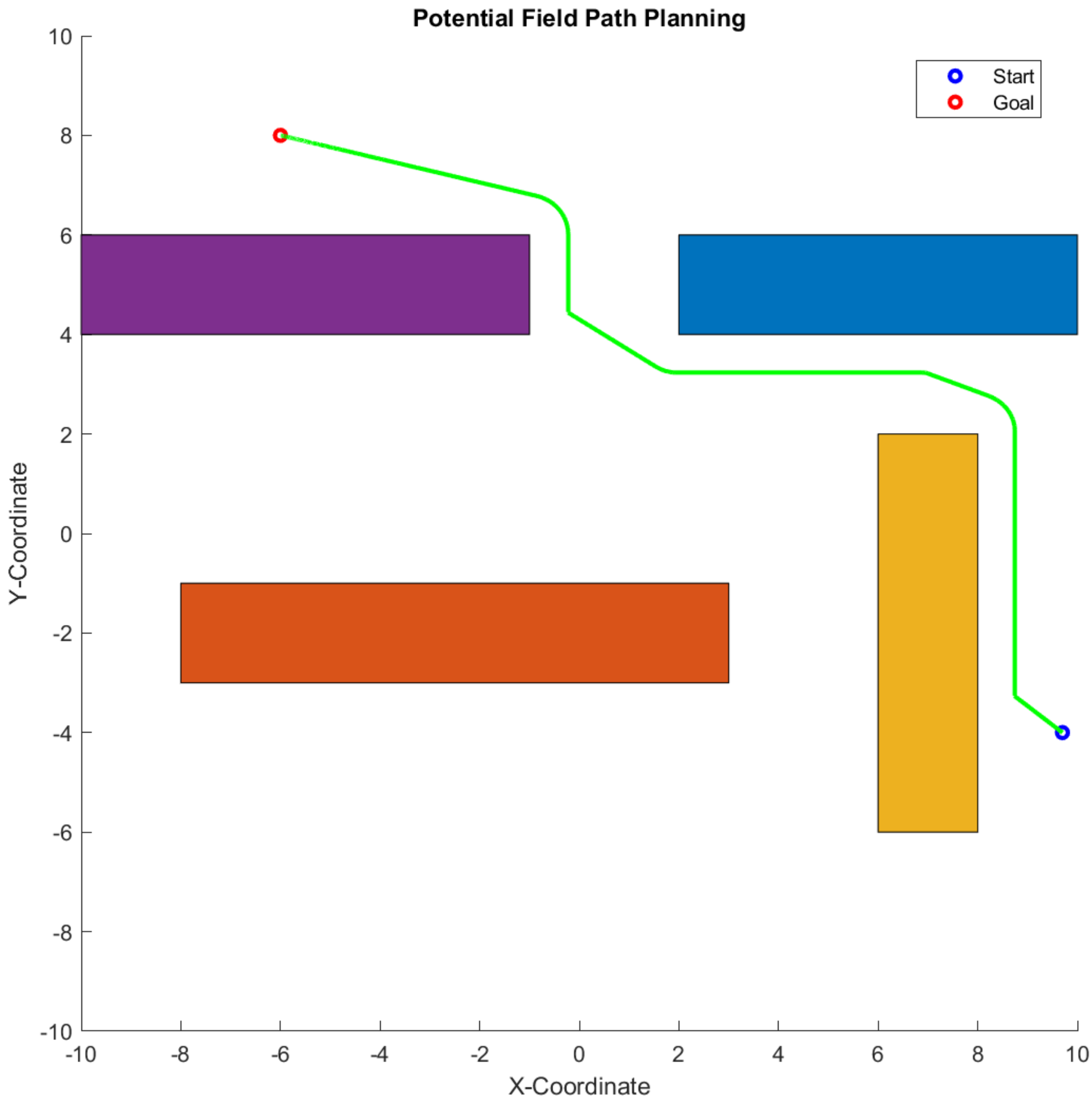

Figure 5: Path Find Using Potential Method

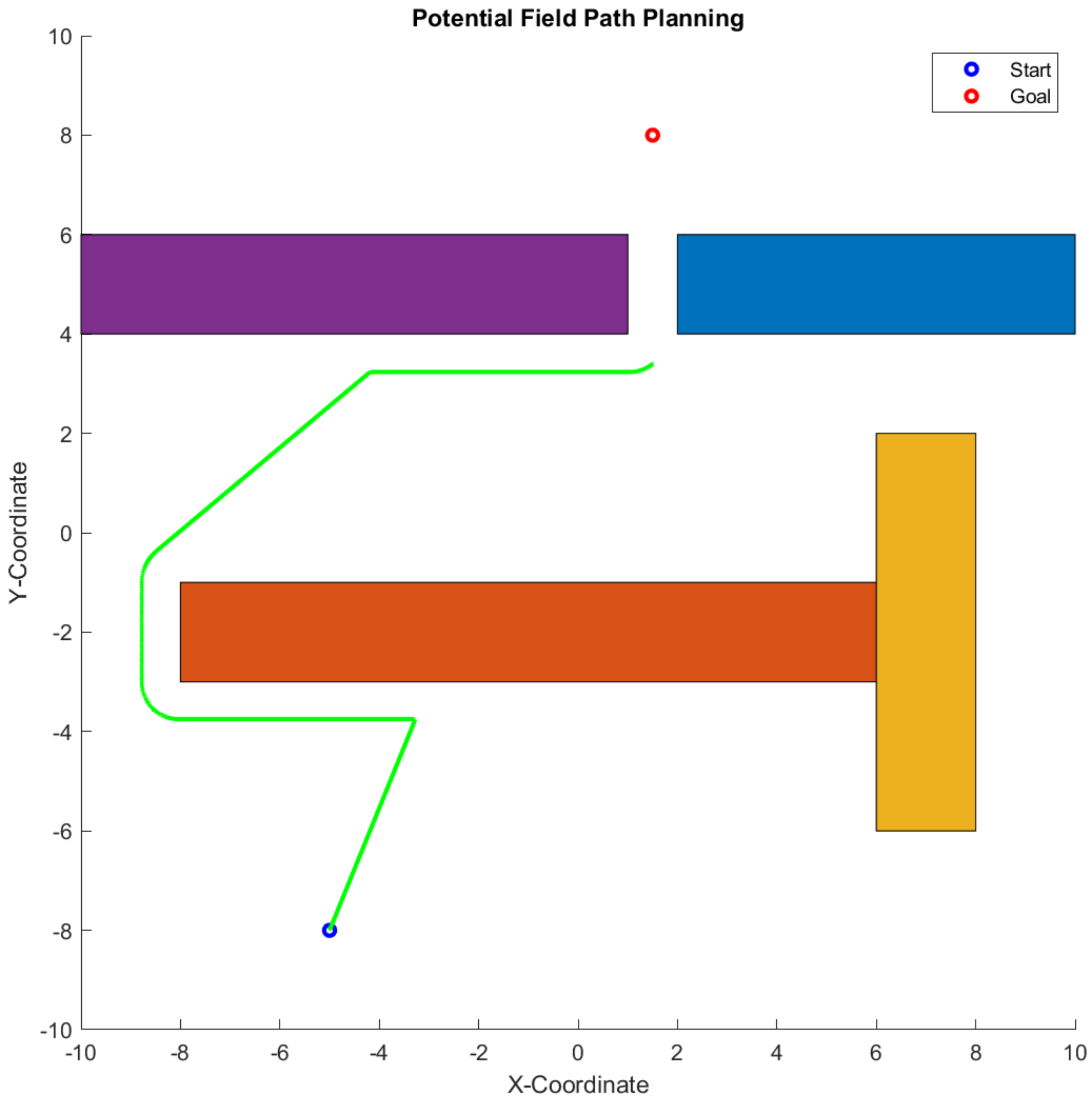

Figure 6: Path not able to find.

In this case robot get caught and not able to find path because of narrow passage. The repulsive potential and attractive potential get equal forming local minimum at that point.