ME145 Robotic Planning and Kinematics: Lab 6

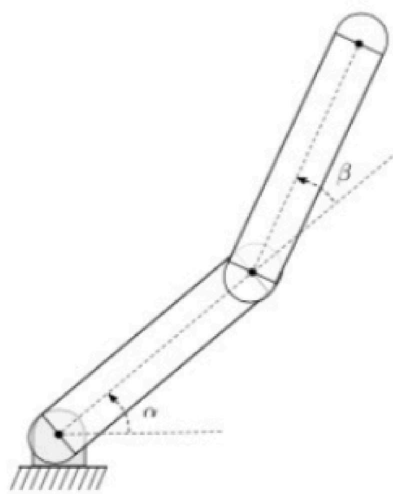
Free Configuration Space of a Two-Link Robot

Eric Perez

Spring 2024

June 2, 2024

E4.3 Sampling the Free Configuration Space for the 2-Link Robot



Assumptions:

- L1 and L2 are the lengths of the first and second links, respectively.
- Each link is shaped like a rectangle with semi-circles at both ends, as in the figure.
- The environment contains 1 circular obstacle, with center (xo, yo), and radius r.
- The angles α and β are expressed in radians.
- Each link has a width of 2W.

plotEnvironment

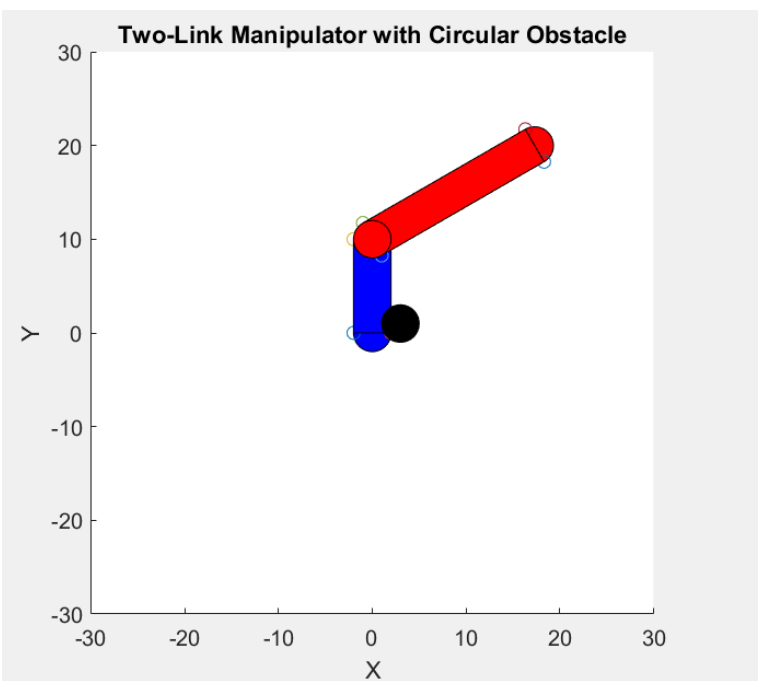
```
% Eric Perez
% lab 6
function plotEnvironment(L1, L2, W, alpha, beta, xo, yo, r)
    % Calculate the positions of the links
    x1 = L1 * cos(alpha);
    y1 = L1 * sin(alpha);
    x2 = x1 + L2 * cos(alpha + beta);
    y2 = y1 + L2 * sin(alpha + beta);

    hold on;
 %plot first semi circle on link1
      theta = linspace(0, 2*pi, 100);
    semi0x = 0 + W*cos(theta);
    semi0y = 0 + W *sin(theta);
   fill(semi0x, semi0y, 'b')
 %plot third semi circle on links
     semi2x = x2 + W*cos(theta);
    semi2y = y2 + W * sin(theta);
   fill(semi2x, semi2y, 'r')

  xdiff1 = x1;
  ydiff1 = y1;

  direct1 = [-ydiff1, xdiff1];
  unit1 = direct1 / norm(direct1);

  TP1x = 0 + W*unit1(1);
  TP1y = 0 + W*unit1(2);
  BP1x = 0 - W*unit1(1);
    BP1y = 0 - W*unit1(2);

  % plot corners of link 1
    plot(TP1x,TP1y,'o');
    plot(BP1x,BP1y,'o');

    TP2x = x1 +  W*unit1(1);
    TP2y = y1 + W*unit1(2);
    BP2x = x1 - W*unit1(1);
    BP2y = y1 - W*unit1(2);

    % plot corners of link 1
    plot(TP2x,TP2y,'o');
    plot(BP2x,BP2y,'o');
    x = [TP1x TP2x  BP2x BP1x];
    y = [TP1y TP2y BP2y BP1y];

   % plot link 1
   fill(x,y,'b')


    xdiff2 = x2-x1;
    ydiff2 = y2-y1;
    direct2 = [-ydiff2, xdiff2];
    unit2 = direct2 / norm(direct2);
    TP3x = x1 +  W*unit2(1);
    TP3y = y1 + W*unit2(2);
    BP3x = x1 - W*unit2(1);
    BP3y = y1 - W*unit2(2);

    % plot corners of link 2
    plot(TP3x,TP3y,'o');
    plot(BP3x,BP3y,'o');

    TP4x = x2 +  W*unit2(1);
    TP4y = y2 + W*unit2(2);
    BP4x = x2 - W*unit2(1);
    BP4y = y2 - W*unit2(2);

    %plot corners of link 2
    plot(TP4x,TP4y,'o');
    plot(BP4x,BP4y,'o');

     % plot link 2
     x = [BP3x BP4x  TP4x TP3x];
     y = [BP3y BP4y  TP4y TP3y];
     fill(x,y,'r')

    %plot second semi circle
     semi1x = x1 + W*cos(theta);
      semi1y = y1 + W * sin(theta);
     fill(semi1x, semi1y, 'r')

        % Plot the obstacle
      x_circle = xo + r*cos(theta);
      y_circle = yo + r * sin(theta);
      fill(x_circle, y_circle, 'k'); % Circular obstacle

      % Plot settings
      axis equal;
      xlim([-L1-L2, L1+L2]);
      ylim([-L1-L2, L1+L2]);
      xlabel('X');
      ylabel('Y');
      title('Two-Link Manipulator with Circular Obstacle');
      hold off;
end
```

Above is the complete code for plotEnvironment. This code began by calculating the positions of the two links through their respective kinematic equations. To plot link 1, the direction of the center point was changed to [-y,x] and the unit vector was found. The x coordinate plus and minus the width multiplied by the unit vector gives us the x coordinates for the corners of the link. This was also done to find the y coordinates. After the positions of the four corners were found, these points were used to create a solid rectangle using the built-in Matlab function fill(). The same method was used for link 2. The circular obstacle and semi-circles were also created using the fill() function. To create a circular shape, 100 thetas between 0 and 2pi were generated and plugged into the above cosine and sine formulas to produce 100 x and y coordinates. These coordinates were then plotted to give a close representation of a circle and semi-circle.

Examples:

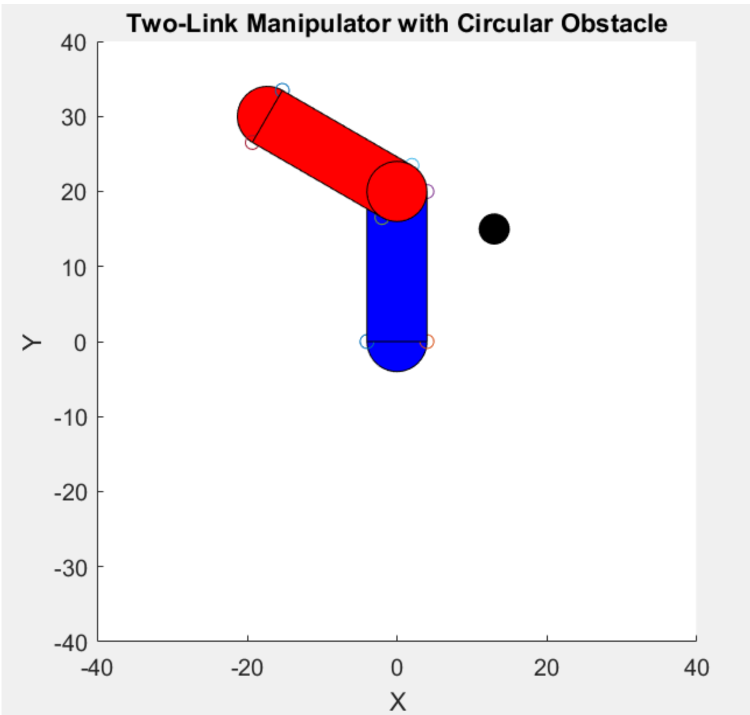
Input: plotEnvironment(10,20,2,pi/2,-pi/3,3,1,2)

Output:



Input: plotEnvironment(20,20,4,pi/2,pi/3,13,15,2)

Output:



checkCollisionTwoLink

```
% Eric Perez
% lab 6
function [TF,collision] = checkCollisionTwoLink(L1, L2, W, alpha, beta, xo, yo, r)
    collision = 0;
   % Calculate the positions of the links
   x1 = L1 * cos(alpha);
   y1 = L1 * sin(alpha);
   x2 = x1 + L2 * cos(alpha + beta);
   y2 = y1 + L2 * sin(alpha + beta);

   hold on;
   theta = linspace(0, 2*pi, 100);
   semi0x = 0 + W*cos(theta);

   semi0y = 0 + W *sin(theta);
   fill(semi0x, semi0y, 'b')
   semi0 = [semi0x;semi0y]';

   semi2x = x2 + W*cos(theta);
   semi2y = y2 + W * sin(theta);
   fill(semi2x, semi2y, 'r')
   semi2 = [semi2x;semi2y]';

   xdiff1 = x1;
   ydiff1 = y1;

   direct1 = [-ydiff1, xdiff1];
   unit1 = direct1 / norm(direct1);


   TP1x = 0 + W*unit1(1);
   TP1y = 0 + W*unit1(2);
   BP1x = 0 - W*unit1(1);
   BP1y = 0 - W*unit1(2);
   plot(TP1x,TP1y,'o');
   plot(BP1x,BP1y,'o');
   TP2x = x1 +  W*unit1(1);
   TP2y = y1 + W*unit1(2);
   BP2x = x1 - W*unit1(1);
   BP2y = y1 - W*unit1(2);
   plot(TP2x,TP2y,'o');
   plot(BP2x,BP2y,'o');

   x = [TP1x TP2x  BP2x BP1x];
   y = [TP1y TP2y BP2y BP1y];
   Link1 = [x(1) y(1); x(4) y(4);x(3) y(3);x(2) y(2)];
   fill(x,y,'b');

   xdiff2 = x2-x1;
   ydiff2 = y2-y1;

   direct2 = [-ydiff2, xdiff2];
   unit2 = direct2 / norm(direct2);
   TP3x = x1 +  W*unit2(1);
   TP3y = y1 + W*unit2(2);
   BP3x = x1 - W*unit2(1);
   BP3y = y1 - W*unit2(2);
   plot(TP3x,TP3y,'o');
   plot(BP3x,BP3y,'o');
   TP4x = x2 +  W*unit2(1);
   TP4y = y2 + W*unit2(2);
   BP4x = x2 - W*unit2(1);
   BP4y = y2 - W*unit2(2);
   plot(TP4x,TP4y,'o');
   plot(BP4x,BP4y,'o');

    semi1x = x1 + W*cos(theta);
     semi1y = y1 + W * sin(theta);
    fill(semi1x, semi1y, 'r')
    semi1 = [semi1x;semi1y]';

    x = [BP3x BP4x  TP4x TP3x];
    y = [BP3y BP4y  TP4y TP3y];
  Link2 = [x(1) y(1); x(4) y(4);x(3) y(3);x(2) y(2)];
    fill(x,y,'r');


      % Plot the obstacle
     x_circle = xo + r*cos(theta);
     y_circle = yo + r * sin(theta);
     obstacle = [x_circle;y_circle]';
     [n,~] = size(obstacle);
     fill(x_circle, y_circle, 'k'); % Circular obstacle

     % Plot settings
     axis equal;
     xlim([-L1-L2, L1+L2]);
     ylim([-L1-L2, L1+L2]);
     xlabel('X');
     ylabel('Y');
     title('Two-Link Manipulator with Circular Obstacle');
     hold off;
     TF = false;
     for i = 1:n                        %checking if obstacle hits link1
     q = obstacle(i,:);
     P = Link1;
     inside = isPointInConvexPolygon(q, P);
         if inside == true
         TF = true;
         collision = 1;
         return;
         end
     end

     for i = 1:n                        %checking if obstacle hits link2
     q = obstacle(i,:);
     P = Link2;
     inside = isPointInConvexPolygon(q, P);
         if inside == true
         TF = true;
         collision = 2;
          return;
```

```
113         end
114 
115         for i = 1:n                     %checking if obstacle hits semi0
116         q = obstacle(i,:);
117         P = semi0;
118          inside = isPointInConvexPolygon(q, P);
119            if inside == true
120             TF = true;
121             collision = 1;
122              return;
123             end
124         end
125          for i = 1:n                     %checking if obstacle hits semi1
126         q = obstacle(i,:);
127         P = semi1;
128          inside = isPointInConvexPolygon(q, P);
129            if inside == true
130             TF = true;
131             collision = 2;
132              return;
133             end
134          end
135           for i = 1:n                     %checking if obstacle hits semi2
136         q = obstacle(i,:);
137         P = semi2;
138          inside = isPointInConvexPolygon(q, P);
139            if inside == true
```

```
138         inside = isPointInConvexPolygon(q, P);
139           if inside == true
140            TF = true;
141            collision = 2;
142             return;
143            end
144         end
145 
146 
147     end
```

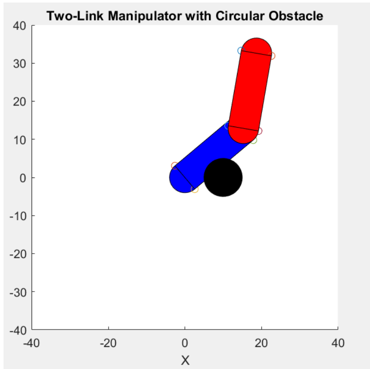
Above is the complete code for checkCollisionTwoLink. The first 74 lines of the code is copied over from the previous code, plotEnvironment, with the addition of arranging the semi-circles into polygons. The code is comprised of multiple if statements and for-loops that check if there is a collision for each link and semi-circle attached. The function “isPointInConvexPolygon” is called within each for loop that takes in a polygon (links and semi-circle) and will check if any of the obstacle’s points are inside the polygon. If a point is inside the polygon, then this means there is a collision and the function will output a “1” for true and will also output “collision = 1 or 2 “ that will identify which link the collision occurs.

Examples:

Input : [interesection, collision] = checkCollisionTwoLink(20,20,4,0.7,0.7,10,0,5)

Output:

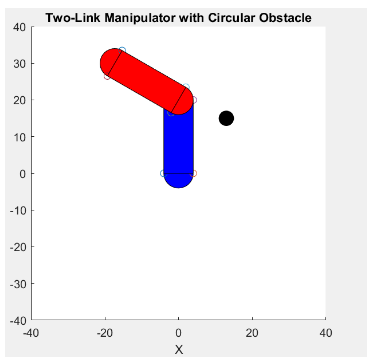
```
interesection =

  logical

   1


collision =

     1
```



Input: [interesection,collision]=checkCollisionTwoLink(20,20,4,pi/2,pi/3,13,15,2)

Output:

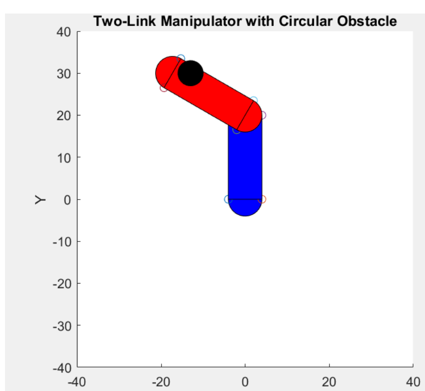
```
interesection =

  logical

   0


collision =

     0
```



Input : [interesection,collision]=checkCollisionTwoLink(20,20,4,pi/2,pi/3,-13,30,3)

Output:

```
interesection =

  logical

   1


collision =

     2
```



plotSampleConfigurationSpaceTwoLink

```
% Eric Perez
 % lab 6
 function plotSampleConfigurationSpaceTwoLink(L1, L2, W, xo, yo, r, sampling_method,n)

 alpha1 = [];
 alpha2 = [];
 alpha3 = [];
 beta1  = [];
 beta2  = [];
 beta3 = [];

if strcmp(sampling_method, 'Sukharev') %Sukharev Graph
     figure(1);
      [Grid] = computeGridSukharev(n);
       suka1 = (Grid(:,1)*(2*pi)) - pi;
       suka2 = (Grid(:,2)*(2*pi)) - pi;
     [SukaN,~] = size(Grid);
     close(figure(1));
     for i = 1:SukaN
         alpha = suka1(i);
         beta = suka2(i);
         [TF,collison] = checkCollisionTwoLink(L1, L2, W, alpha, beta, xo, yo, r);
         if (TF == true) && (collison == 1)
             alpha1(end+1) = alpha;
             beta1(end+1) = beta;
         elseif (TF == true) && (collison == 2)
             alpha2(end+1) = alpha;
             beta2(end+1) = beta;
         else (TF == true) && (collison == 0)
             alpha3(end+1) = alpha;
             beta3(end+1) = beta;
         end
     end

     elseif strcmp(sampling_method, 'Random')
        [Grid] = computeGridRandom(n);
        Ran1 = (Grid(:,1)*(2*pi)) - pi;
        Ran2 = (Grid(:,2)*(2*pi)) - pi;
      [RanN,~] = size(Grid);
      for i = 1:RanN
          alpha = Ran1(i);
          beta = Ran2(i);
          figure(2)
          [TF,collison] = checkCollisionTwoLink(L1, L2, W, alpha, beta, xo, yo, r);
          if (TF == true) && (collison == 1)
              alpha1(end+1) = alpha;
              beta1(end+1) = beta;
          elseif (TF == true) && (collison == 2)
              alpha2(end+1) = alpha;
              beta2(end+1) = beta;
          else (TF == true) && (collison == 0)
              alpha3(end+1) = alpha;
              beta3(end+1) = beta;
          end
      end
  else strcmp(sampling_method, 'Halton') % Grid Halton
```

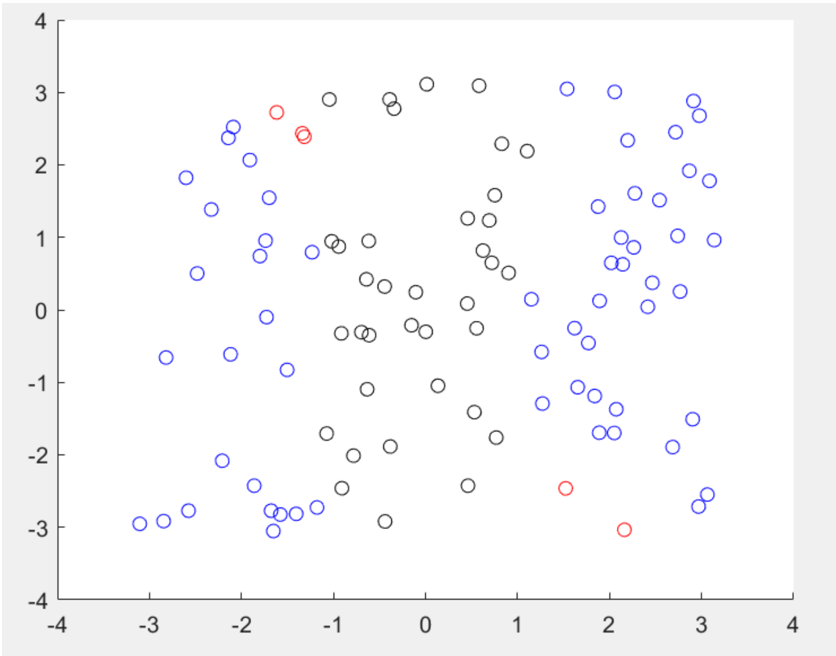
```
else strcmp(sampling_method, 'Halton') % Grid Halton
     [Grid] = computeGridHalton(n, 2, 3);
      Hal1 = (Grid(:,1)*(2*pi)) - pi;
      Hal2 = (Grid(:,2)*(2*pi)) - pi;
    [HalN,~] = size(Grid);
    for i = 1:HalN
        alpha = Hal1(i);
        beta = Hal2(i);
        figure(2)
        [TF,collison] = checkCollisionTwoLink(L1, L2, W, alpha, beta, xo, yo, r);
        if (TF == true) && (collison == 1)
            alpha1(end+1) = alpha;
            beta1(end+1) = beta;
        elseif (TF == true) && (collison == 2)
            alpha2(end+1) = alpha;
            beta2(end+1) = beta;
        else (TF == true) && (collison == 0)
            alpha3(end+1) = alpha;
            beta3(end+1) = beta;
        end
    end
end
close(figure(1));
figure;
 hold on
plot(alpha1,beta1, 'ko');
plot(alpha2,beta2, 'ro');
plot(alpha3,beta3,'bo');
end
```

Above is the complete code for the plotSampleConfigurationSpaceTwoLink function. The Sukharev, random, and Halton sampling methods are the three methods the user can choose by typing in their respective names. Since the free configuration space must display black points for link 1 collision, red points for link 2 collision, and blue points for no collision, three alpha and three beta arrays were created for each outcome. The sampling functions for each one were created in the previous lab and are utilized in this code. Three if statements determine which sampling method is inputted and there is a for loop that will run until the specified number of samples are completed. The configuration space is created with the x-axis spanning from -pi to pi and the y-axis also spanning from -pi to pi.

Examples (notice the same environment is used but with different sampling methods) :

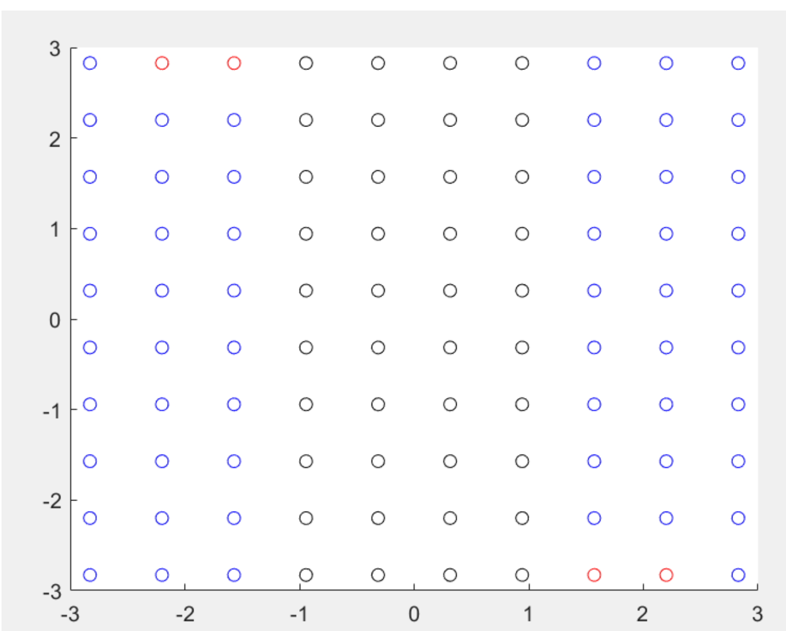
Input : plotSampleConfigurationSpaceTwoLink(20,20,4,10,0,5, 'Random', 100)

Output:



Input : plotSampleConfigurationSpaceTwoLink(20,20,4,10,0,5, 'Sukharev', 100)

Output:



Input: plotSampleConfigurationSpaceTwoLink(20,20,4,10,0,5, 'Halton', 100)]

Output: