

ME145 Robotic Planning and Kinematics: Lab 6

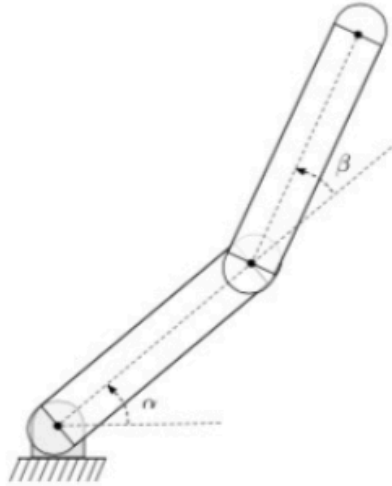
Free Configuration Space of a Two-Link Robot

Eric Perez

Spring 2024

June 2, 2024

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



#### Assumptions:

- $L_1$  and  $L_2$  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  $(x_o, y_o)$ , and radius  $r$ .
- The angles  $\alpha$  and  $\beta$  are expressed in radians.
- Each link has a width of  $2W$ .

## plotEnvironment

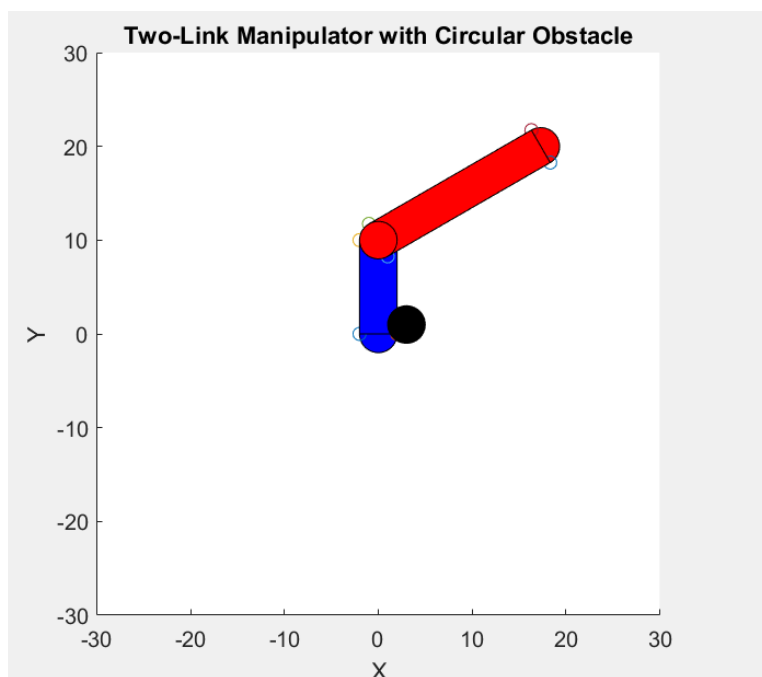
```
1 % Eric Perez
2 % lab 6
3 function plotEnvironment(L1, L2, W, alpha, beta, xo, yo, r)
4     % Calculate the positions of the links
5     x1 = L1 * cos(alpha);
6     y1 = L1 * sin(alpha);
7     x2 = x1 + L2 * cos(alpha + beta);
8     y2 = y1 + L2 * sin(alpha + beta);
9
10    hold on;
11    %plot first semi circle on link1
12    theta = linspace(0, 2*pi, 100);
13    semi0x = 0 + W*cos(theta);
14    semi0y = 0 + W * sin(theta);
15    fill(semi0x, semi0y, 'b')
16    %plot third semi circle on links
17    semi2x = x2 + W*cos(theta);
18    semi2y = y2 + W * sin(theta);
19    fill(semi2x, semi2y, 'r')
20
21    xdiff1 = x1;
22    ydiff1 = y1;
23
24    direct1 = [-ydiff1, xdiff1];
25    unit1 = direct1 / norm(direct1);
26
27    TP1x = 0 + W*unit1(1);
28    TP1y = 0 + W*unit1(2);
29    BP1x = 0 - W*unit1(1);
30    BP1y = 0 - W*unit1(2);
31
32    % plot corners of link 1
33    plot(TP1x,TP1y,'o');
34    plot(BP1x,BP1y,'o');
35
36    TP2x = x1 + W*unit1(1);
37    TP2y = y1 + W*unit1(2);
38    BP2x = x1 - W*unit1(1);
39    BP2y = y1 - W*unit1(2);
40
41    % plot corners of link 1
42    plot(TP2x,TP2y,'o');
43    plot(BP2x,BP2y,'o');
44    x = [TP1x TP2x BP2x BP1x];
45    y = [TP1y TP2y BP2y BP1y];
46
47    % plot link 1
48    fill(x,y,'b')
49
50
51    xdiff2 = x2-x1;
52    ydiff2 = y2-y1;
53    direct2 = [-ydiff2, xdiff2];
54    unit2 = direct2 / norm(direct2);
55    TP3x = x1 + W*unit2(1);
56    TP3y = y1 + W*unit2(2);
57    BP3x = x1 - W*unit2(1);
58    BP3y = y1 - W*unit2(2);
59
60    % plot corners of link 2
61    plot(TP3x,TP3y,'o');
62    plot(BP3x,BP3y,'o');
63
64    TP4x = x2 + W*unit2(1);
65    TP4y = y2 + W*unit2(2);
66    BP4x = x2 - W*unit2(1);
67    BP4y = y2 - W*unit2(2);
68
69    %plot corners of link 2
70    plot(TP4x,TP4y,'o');
71    plot(BP4x,BP4y,'o');
72
73    % plot link 2
74    x = [BP3x BP4x TP4x TP3x];
75    y = [BP3y BP4y TP4y TP3y];
76    fill(x,y,'r')
77
78    %plot second semi circle
79    semi1x = x1 + W*cos(theta);
80    semi1y = y1 + W * sin(theta);
81    fill(semi1x, semi1y, 'r')
82
83    % Plot the obstacle
84    x_circle = xo + r*cos(theta);
85    y_circle = yo + r * sin(theta);
86    fill(x_circle, y_circle, 'k'); % Circular obstacle
87
88    % Plot settings
89    axis equal;
90    xlim([-L1-L2, L1+L2]);
91    ylim([-L1-L2, L1+L2]);
92    xlabel('X');
93    ylabel('Y');
94    title('Two-Link Manipulator with Circular Obstacle');
95    hold off;
96    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  $2\pi$  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:

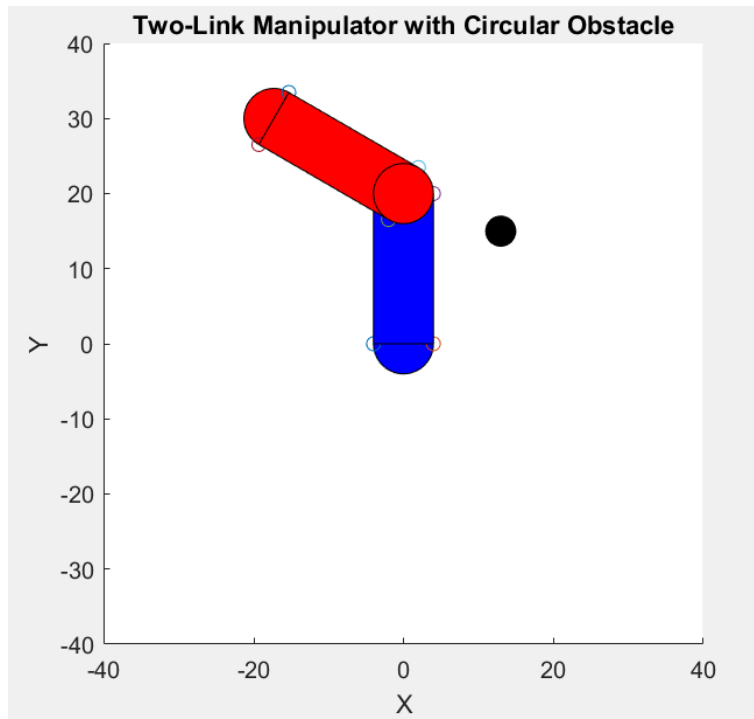
Input: plotEnvironment(10,20,2,pi/2,-pi/3,3,1,2)

Output:



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

Output:



## checkCollisionTwoLink

```
1 % Eric Perez
2 % lab 6
3 function [TF,collision] = checkCollisionTwoLink(L1, L2, W, alpha, beta, xo, yo, r)
4     collision = 0;
5     % Calculate the positions of the links
6     x1 = L1 * cos(alpha);
7     y1 = L1 * sin(alpha);
8     x2 = x1 + L2 * cos(alpha + beta);
9     y2 = y1 + L2 * sin(alpha + beta);
10
11     hold on;
12     theta = linspace(0, 2*pi, 100);
13     semi0x = 0 + W*cos(theta);
14
15     semi0y = 0 + W *sin(theta);
16     fill(semi0x, semi0y, 'b')
17     semi0 = [semi0x;semi0y]';
18
19     semi2x = x2 + W*cos(theta);
20     semi2y = y2 + W * sin(theta);
21     fill(semi2x, semi2y, 'r')
22     semi2 = [semi2x;semi2y]';
23
24     xdiff1 = x1;
25     ydiff1 = y1;
26
27     direct1 = [-ydiff1, xdiff1];
28     unit1 = direct1 / norm(direct1);
29
```

```
29
30     TP1x = 0 + W*unit1(1);
31     TP1y = 0 + W*unit1(2);
32     BP1x = 0 - W*unit1(1);
33     BP1y = 0 - W*unit1(2);
34     plot(TP1x,TP1y,'o');
35     plot(BP1x,BP1y,'o');
36     TP2x = x1 + W*unit1(1);
37     TP2y = y1 + W*unit1(2);
38     BP2x = x1 - W*unit1(1);
39     BP2y = y1 - W*unit1(2);
40     plot(TP2x,TP2y,'o');
41     plot(BP2x,BP2y,'o');
42
43     x = [TP1x TP2x BP2x BP1x];
44     y = [TP1y TP2y BP2y BP1y];
45     Link1 = [x(1) y(1); x(4) y(4);x(3) y(3);x(2) y(2)];
46     fill(x,y,'b');
47
48     xdiff2 = x2-x1;
49     ydiff2 = y2-y1;
50
51     direct2 = [-ydiff2, xdiff2];
52     unit2 = direct2 / norm(direct2);
53     TP3x = x1 + W*unit2(1);
54     TP3y = y1 + W*unit2(2);
55     BP3x = x1 - W*unit2(1);
56     BP3y = y1 - W*unit2(2);
57     plot(TP3x,TP3y,'o');
```

```
58     plot(BP3x,BP3y,'o');
59     TP4x = x2 + W*unit2(1);
60     TP4y = y2 + W*unit2(2);
61     BP4x = x2 - W*unit2(1);
62     BP4y = y2 - W*unit2(2);
63     plot(TP4x,TP4y,'o');
64     plot(BP4x,BP4y,'o');
65
66     semi1x = x1 + W*cos(theta);
67     semi1y = y1 + W * sin(theta);
68     fill(semi1x, semi1y, 'r')
69     semi1 = [semi1x;semi1y]';
70
71     x = [BP3x BP4x TP4x TP3x];
72     y = [BP3y BP4y TP4y TP3y];
73     Link2 = [x(1) y(1); x(4) y(4);x(3) y(3);x(2) y(2)];
74     fill(x,y,'r');
75
76
77     % Plot the obstacle
78     x_circle = xo + r*cos(theta);
79     y_circle = yo + r * sin(theta);
80     obstacle = [x_circle;y_circle]';
81     [n,~] = size(obstacle);
82     fill(x_circle, y_circle, 'k'); % Circular obstacle
83
84     % Plot settings
```

```
84     % Plot settings
85     axis equal;
86     xlim([-L1-L2, L1+L2]);
87     ylim([-L1-L2, L1+L2]);
88     xlabel('X');
89     ylabel('Y');
90     title('Two-Link Manipulator with Circular Obstacle');
91     hold off;
92     TF = false;
93     for i = 1:n %checking if obstacle hits link1
94         q = obstacle(i,:);
95         P = Link1;
96         inside = isPointInConvexPolygon(q, P);
97         if inside == true
98             TF = true;
99             collision = 1;
100             return;
101         end
102     end
103
104     for i = 1:n %checking if obstacle hits link2
105         q = obstacle(i,:);
106         P = Link2;
107         inside = isPointInConvexPolygon(q, P);
108         if inside == true
109             TF = true;
110             collision = 2;
111             return;
112         end
113     end
```

```

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
140
141             inside = isPointInConvexPolygon(q, P);
142             if inside == true
143                 TF = true;
144                 collision = 2;
145                 return;
146             end
147         end
148     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:

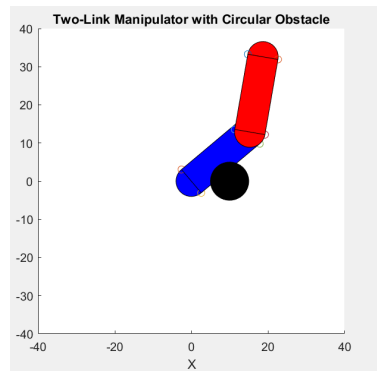
```
interesection =
```

```
logical
```

```
1
```

```
collision =
```

```
1
```



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

Output:

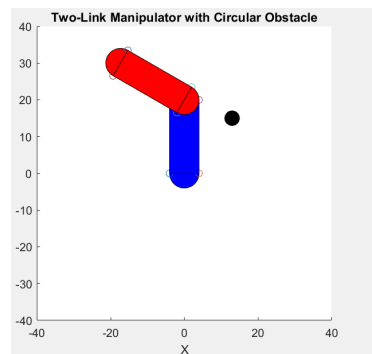
```
interesection =
```

```
logical
```

```
0
```

```
collision =
```

```
0
```



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

Output:

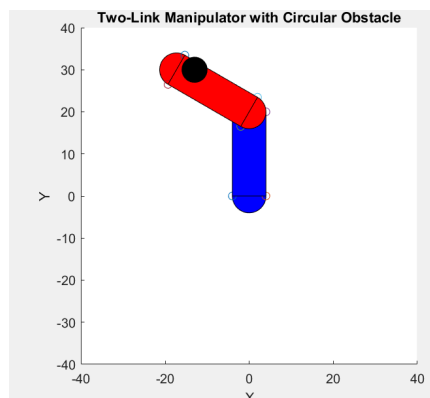
```
interesection =
```

```
logical
```

```
1
```

```
collision =
```

```
2
```





## plotSampleConfigurationSpaceTwoLink

```
1 % Eric Perez
2 % lab 6
3 function plotSampleConfigurationSpaceTwoLink(L1, L2, W, xo, yo, r, sampling_method,n)
4
5     alpha1 = [];
6     alpha2 = [];
7     alpha3 = [];
8     beta1 = [];
9     beta2 = [];
10    beta3 = [];
11
12    if strcmp(sampling_method, 'Sukharev') %Sukharev Graph
13        figure(1);
14        [Grid] = computeGridSukharev(n);
15        suka1 = (Grid(:,1)*(2*pi)) - pi;
16        suka2 = (Grid(:,2)*(2*pi)) - pi;
17        [SukaN,~] = size(Grid);
18        close.figure(1));
19        for i = 1:SukaN
20            alpha = suka1(i);
21            beta = suka2(i);
22            [TF,collision] = checkCollisionTwoLink(L1, L2, W, alpha, beta, xo, yo, r);
23            if (TF == true) && (collision == 1)
24                alpha1(end+1) = alpha;
25                beta1(end+1) = beta;
26            elseif (TF == true) && (collision == 2)
27                alpha2(end+1) = alpha;
28                beta2(end+1) = beta;
29            else (TF == true) && (collision == 0)
30
31                else (TF == true) && (collision == 0)
32                    alpha3(end+1) = alpha;
33                    beta3(end+1) = beta;
34                end
35            end
36        end
37
38        elseif strcmp(sampling_method, 'Random')
39            [Grid] = computeGridRandom(n);
40            Ran1 = (Grid(:,1)*(2*pi)) - pi;
41            Ran2 = (Grid(:,2)*(2*pi)) - pi;
42            [RanN,~] = size(Grid);
43            for i = 1:RanN
44                alpha = Ran1(i);
45                beta = Ran2(i);
46                figure(2)
47                [TF,collision] = checkCollisionTwoLink(L1, L2, W, alpha, beta, xo, yo, r);
48                if (TF == true) && (collision == 1)
49                    alpha1(end+1) = alpha;
50                    beta1(end+1) = beta;
51                elseif (TF == true) && (collision == 2)
52                    alpha2(end+1) = alpha;
53                    beta2(end+1) = beta;
54                else (TF == true) && (collision == 0)
55                    alpha3(end+1) = alpha;
56                    beta3(end+1) = beta;
57                end
58            end
59        end
60    else strcmp(sampling_method, 'Halton') % Grid Halton
```

```

56     else strcmp(sampling_method, 'Halton') % Grid Halton
57         [Grid] = computeGridHalton(n, 2, 3);
58         Hal1 = (Grid(:,1)*(2*pi)) - pi;
59         Hal2 = (Grid(:,2)*(2*pi)) - pi;
60         [HalN,~] = size(Grid);
61         for i = 1:HalN
62             alpha = Hal1(i);
63             beta = Hal2(i);
64             figure(2)
65             [TF,collision] = checkCollisionTwoLink(L1, L2, W, alpha, beta, xo, yo, r);
66             if (TF == true) && (collision == 1)
67                 alpha1(end+1) = alpha;
68                 beta1(end+1) = beta;
69             elseif (TF == true) && (collision == 2)
70                 alpha2(end+1) = alpha;
71                 beta2(end+1) = beta;
72             else (TF == true) && (collision == 0)
73                 alpha3(end+1) = alpha;
74                 beta3(end+1) = beta;
75             end
76         end
77     end
78     close(figure(1));
79     figure;
80     hold on
81     plot(alpha1,beta1, 'ko');
82     plot(alpha2,beta2, 'ro');
83     plot(alpha3,beta3, 'bo');
84     end

```

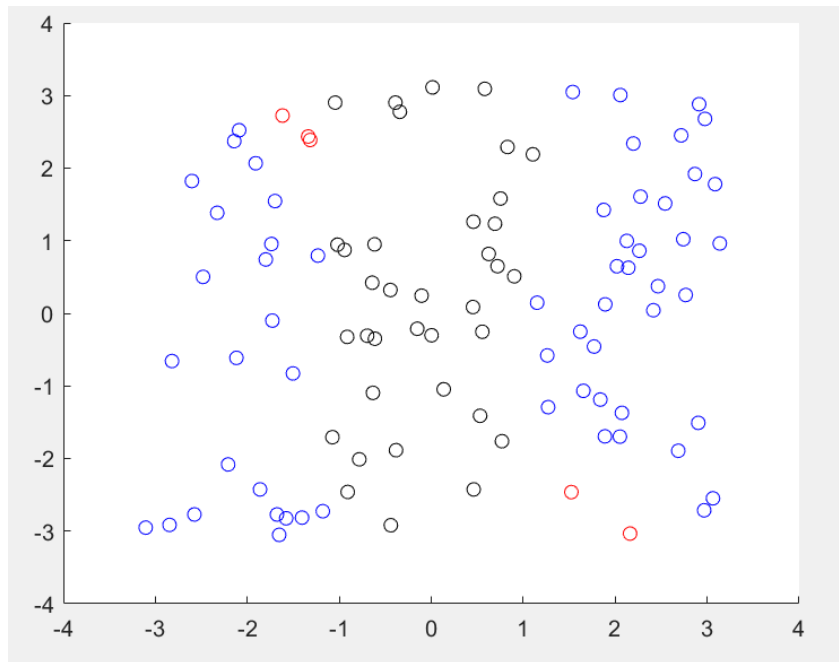
Above is the complete code for the plotSampleConfigurationSpaceTwoLink function.

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) :

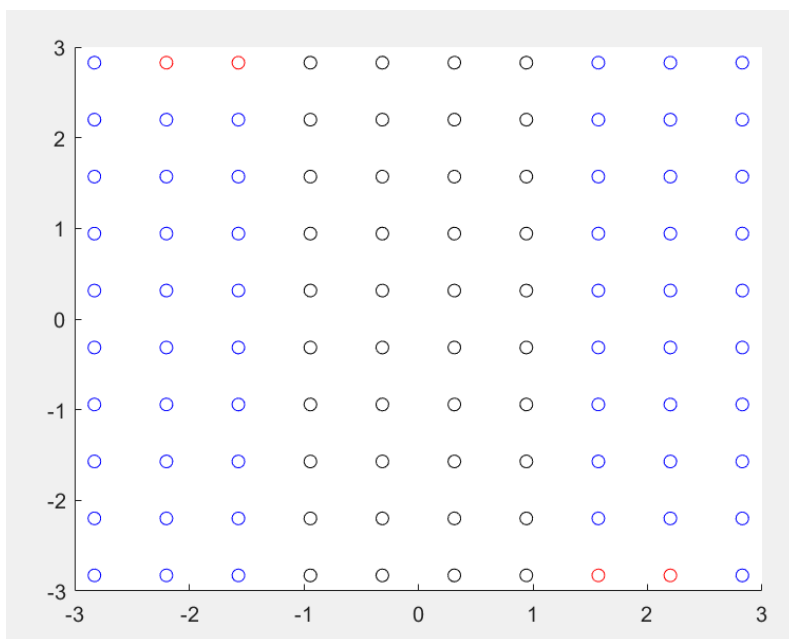
Input : plotSampleConfigurationSpaceTwoLink(20,20,4,10,0,5, 'Random', 100)

Output:



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

Output:



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

Output:

