ME145 Robotic Planning and Kinematic: Lab 6

Robotic Planning and kinematics

Elijah Perez

Winter 2025, Mar 7, 2025

Part 1: Plot environment Function:

```matlab
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);


%% Plotting

% Link 2 -----------------------------------------------------------------

% plot semi circle on link2 front
theta = linspace(alpha+beta-pi/2,alpha+beta+pi/2 , 100);
semi0x = x1 - W*cos(theta);
semi0y = y1 - W *sin(theta);
fill(semi0x, semi0y,[0.9290 0.6940 0.1250])
hold on

% Semi circle link 2 end
theta = linspace(beta+alpha-pi/2,beta+alpha+pi/2 , 100);
semi2x = x2 + W*cos(theta);
semi2y = y2 + W * sin(theta);
fill(semi2x, semi2y,[0.9290 0.6940 0.1250])

% plotting square
xx1 = x1 - W*cos(alpha+beta+pi/2);
xx2 = x1 - W*cos(alpha+beta-pi/2);
xx4 = x2 - W*cos(alpha+beta+pi/2);
xx3 = x2 - W*cos(alpha+beta-pi/2);

yy1 = y1 - W *sin(alpha+beta-pi/2);
yy2 = y1 - W *sin(alpha+beta+pi/2);
yy3 = y2 + W *sin(alpha+beta-pi/2);
yy4 = y2 + W *sin(alpha+beta+pi/2);
```

```matlab
% Link 1 -----------------------------------------------------------------


 %plot first semi circle on link1
theta = linspace(alpha-pi/2,alpha+pi/2 , 100);
semi0x = 0 - W*cos(theta);
semi0y = 0 - W *sin(theta);
fill(semi0x, semi0y,[0 0.4470 0.7410])
hold on

%plot semi circle on link1 end
theta = linspace(alpha-pi/2,alpha+pi/2 , 100);
semi0x = x1 + W*cos(theta);
semi0y = y1 + W *sin(theta);
fill(semi0x, semi0y,[0 0.4470 0.7410])
hold on


%plotting square

xx1 = 0 - W*cos(alpha-pi/2);
xx2 = 0 - W*cos(alpha+pi/2);
xx4 = x1 + W*cos(alpha+pi/2);
xx3 = x1 + W*cos(alpha-pi/2);

yy1 = 0 - W *sin(alpha-pi/2);
yy2 = 0 - W *sin(alpha+pi/2);
yy3 = y1 + W *sin(alpha-pi/2);
yy4 = y1 + W *sin(alpha+pi/2);

fill([xx1 xx2 xx3 xx4],[yy1 yy2 yy3 yy4],[0.3010 0.7450 0.9330])
```

```matlab
%% Plotting Obsticle


theta = linspace(0,2*pi, 100);
semi0x = xo + r*cos(theta);
semi0y = yo + r *sin(theta);
fill(semi0x, semi0y,[0.6350 0.0780 0.1840])
hold on


xlim([-L2-L1-1 L2+L1+1])
ylim([-L2-L1-1 L2+L1+1])
axis square
grid on
```

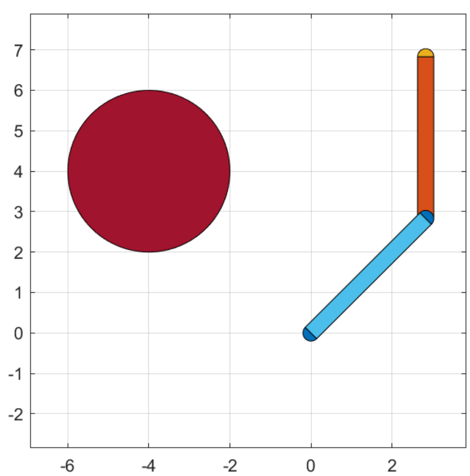
The complete code for plotEnvironment is provided above. It begins by calculating the positions of the two links using their respective kinematic equations. To plot Link 1, the center point's direction was adjusted to [−y,x][-y, x][−y,x], and the unit vector was determined. The x-coordinates for the link's corners were found by adding and subtracting the width multiplied by the unit vector from the x-coordinate. The same approach was used to determine the y-coordinates. Once the positions of the four corners were established, they were used to create a solid rectangle with MATLAB's built-in fill() function. This method was also applied to plot Link 2. The circular obstacle and semi-circles were generated using the fill() function as well. To approximate a circular shape, 100 theta values were evenly spaced between 0 and -pi\pi and substituted into cosine and sine functions, producing 100 corresponding x and y coordinates. These points were then plotted to create a close representation of a circle and semi-circle.

Example:

Input:

```
% plotEnvironment(L1, L2, W, alpha, beta, xo, yo, r)

figure
L1 = 4;
L2 = 4;
W  = 0.2; %  width
xo = -4;
yo = 4;
r  = 2;

alpha = pi/4;
beta  = pi/4;

plotEnvironment(L1, L2, W, alpha, beta, xo, yo, r)
```

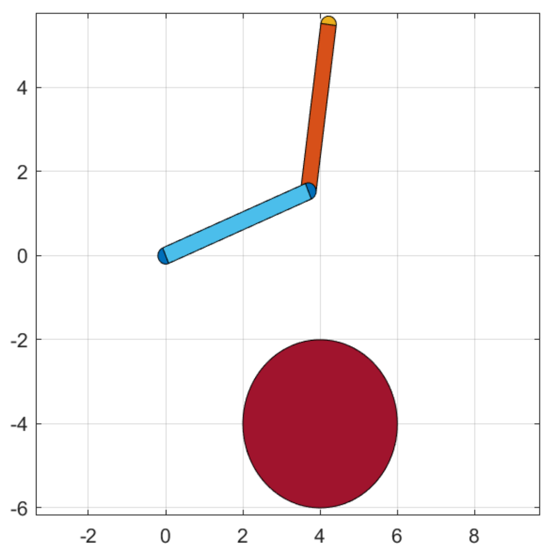
Output:



Input

```
figure
L1 = 4;
L2 = 4;
W  = 0.2; %  width
xo = 4;
yo = -4;
r  = 2;

alpha = pi/8;
beta  = pi/8;

plotEnvironment(L1, L2, W, alpha, beta, xo, yo, r)
```

Output:



Part 2: checkCollisionTwpLink

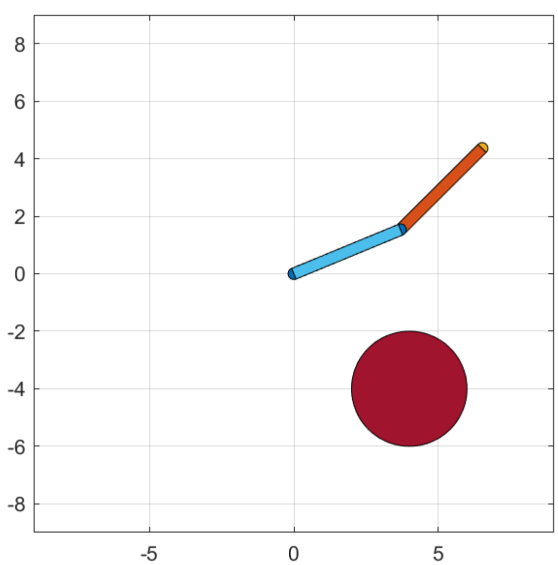
```matlab
function [one two] = checkCollisionTwoLink(L1,L2,W,alpha,beta,xo,yo,r)

%% Position
x1 = L1 * cos(alpha);
y1 = L1 * sin(alpha);

x2 = x1 + L2 * cos(alpha + beta);
y2 = y1 + L2 * sin(alpha + beta);

%% Check rectangle first (easy)

% Define link 1 body

xx1 = 0 - W*cos(alpha-pi/2);
xx2 = 0 - W*cos(alpha+pi/2);
xx4 = x1 + W*cos(alpha+pi/2);
xx3 = x1 + W*cos(alpha-pi/2);

yy1 = 0 - W *sin(alpha-pi/2);
yy2 = 0 - W *sin(alpha+pi/2);
yy3 = y1 + W *sin(alpha-pi/2);
yy4 = y1 + W *sin(alpha+pi/2);

Link1 = [[xx1 xx2 xx3 xx4]' [yy1 yy2 yy3 yy4]'];

% Define Body Link 2

xx1 = x1 - W*cos(alpha+beta+pi/2);
xx2 = x1 - W*cos(alpha+beta-pi/2);
xx4 = x2 - W*cos(alpha+beta+pi/2);
xx3 = x2 - W*cos(alpha+beta-pi/2);

yy1 = y1 - W *sin(alpha+beta-pi/2);
yy2 = y1 - W *sin(alpha+beta+pi/2);
yy3 = y2 + W *sin(alpha+beta-pi/2);
yy4 = y2 + W *sin(alpha+beta+pi/2);

Link2 = [[xx1 xx2 xx3 xx4]' [yy1 yy2 yy3 yy4]'];


%% Define obsticle as a poly with alot of points

theta = linspace(0,2*pi, 100);
semi0x = xo + r*cos(theta);
semi0y = yo + r *sin(theta);

obsticle = [[semi0x'] [semi0y']];

%-----------------------------------------------------------

%Link 1 end
theta = linspace(alpha-pi/2,alpha+pi/2 , 100);
semi0x = x1 + W*cos(theta);
semi0y = y1 + W *sin(theta);
Link1_end = [[semi0x'] [semi0y']];

%Link 2 start
theta = linspace(alpha+beta-pi/2,alpha+beta+pi/2 , 100);
semi0x = x1 - W*cos(theta);
semi0y = y1 - W *sin(theta);
Link2_start = [[semi0x'] [semi0y']];

% Semi circle link 2 end
theta = linspace(beta+alpha-pi/2,beta+alpha+pi/2 , 100);
semi0x = x2 + W*cos(theta);
semi0y = y2 + W * sin(theta);
Link2_end = [[semi0x'] [semi0y']];

%% Check for collisions

TF1 = doTwoConvexPolygonsIntersect(obsticle,Link1);
TF2 = doTwoConvexPolygonsIntersect(obsticle,Link2);
TF3 = doTwoConvexPolygonsIntersect(obsticle,Link1_end);
TF4 = doTwoConvexPolygonsIntersect(obsticle,Link2_start);
TF5 = doTwoConvexPolygonsIntersect(obsticle,Link2_end);


if  (TF1 ==1) || (TF3 ==1)
    one = 1;
else
    one = 0;
end

if  (TF2 ==1) || (TF4 ==1) || (TF5 ==1)
    two = 1;
else
    two = 0;
end

end
```

This code primarily constructs the polygons representing the two-link robot and the circular obstacle. The robot consists of five distinct polygonal shapes. To determine the interactions between the robot and the obstacle, I utilized a function from the previous lab to check whether each segment intersects with the circle. If Link 1 collides with the obstacle, TF1 is set to 1; otherwise, it remains 0. Similarly, if Link 2 makes contact, TF2 is assigned a value of 1; otherwise, it remains 0.

Input: No collision

```
figure
L1 = 4;
L2 = 4;
W  = 0.2; %  width
xo = 4;
yo = -4;
r  = 2;

alpha = pi/8;
beta  = pi/8;

plotEnvironment(L1, L2, W, alpha, beta, xo, yo, r)
[TF1,TF2] = checkCollisionTwoLink(L1,L2,W,alpha,beta,xo,yo,r)
```

Output

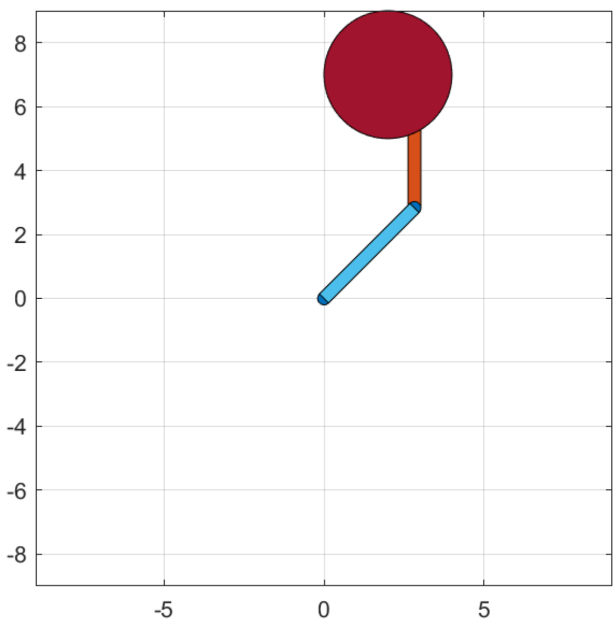



```
TF1 =

     0


TF2 =

     0
```

Input: Link 2 collision

```
figure
L1 = 4;
L2 = 4;
W  = 0.2; %  width
xo = 2;
yo = 7;
r  = 2;

alpha = pi/4;
beta  = pi/4;

plotEnvironment(L1, L2, W, alpha, beta, xo, yo, r)
[TF1,TF2] = checkCollisionTwoLink(L1,L2,W,alpha,beta,xo,yo,r)
```

Output:



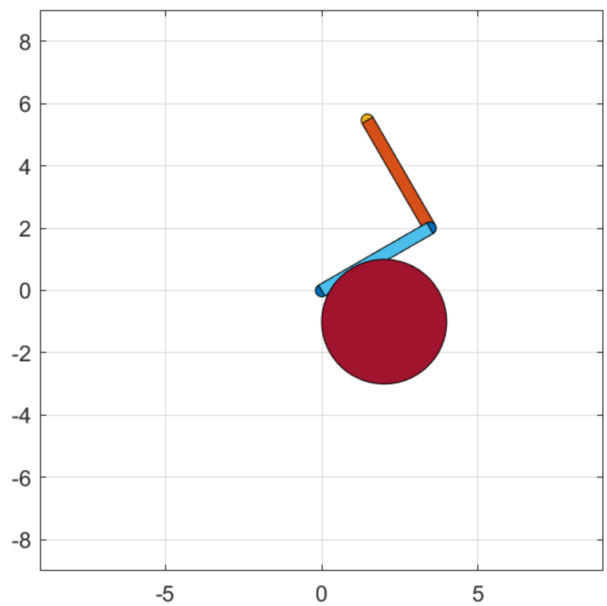

```
TF1 =

     0


TF2 =

     1
```

TF2 = 1 since link two is in collison

Input: Link one collision

```
figure
L1 = 4;
L2 = 4;
W  = 0.2; %  width
xo = 2;
yo = -1;
r  = 2;

alpha = pi/6;
beta  = pi/2;

plotEnvironment(L1, L2, W, alpha, beta, xo, yo, r)
[TF1,TF2] = checkCollisionTwoLink(L1,L2,W,alpha,beta,xo,yo,r)
```

Output:




```
TF1 =

     1


TF2 =

     0
```

TF1 = 1, since link one is in collision.

Part 3: plotSampleConfiguretionTwoLink

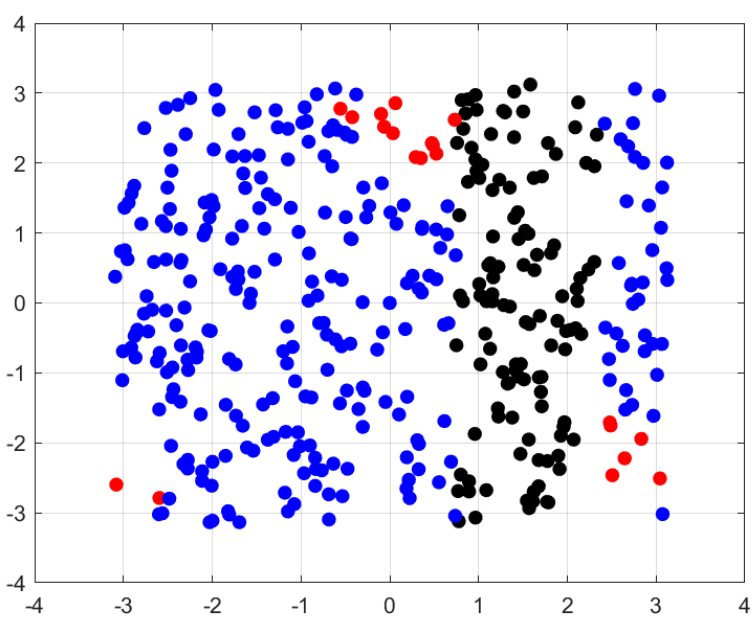
```matlab
function Grid = plotSampleConfigurationSpaceTwoLink(L1,L2,W,xo,yo,r,sampling_method,n)
%% Define method and obtaining grid
if strcmp(sampling_method, 'Sukharev')
    Grid = computeGridSukharev(n);
    Grid = Grid';
elseif strcmp(sampling_method, 'Random')
    Grid = computeGridRandom(n);

elseif strcmp(sampling_method, 'Halton')
    Grid = computeGridHalton(n,2,3);

else
    error("Unknown sampling method")
end

%% Defining possible configuration expansion

alpha_net = (Grid(:,1) * 2 - 1) * pi;
beta_net = (Grid(:,2) * 2 - 1) * pi;


%% Define Geometry
for i =1:length(alpha_net)

alpha = alpha_net(i);
beta = beta_net(i);
% Position
x1 = L1 .* cos(alpha);
y1 = L1 .* sin(alpha);

x2 = x1 + L2 .* cos(alpha + beta);
y2 = y1 + L2 .* sin(alpha + beta);

% Define link 1 body

xx1 = 0 - W.*cos(alpha-pi/2);
xx2 = 0 - W.*cos(alpha+pi/2);
xx4 = x1 + W.*cos(alpha+pi/2);
xx3 = x1 + W.*cos(alpha-pi/2);

yy1 = 0 - W .*sin(alpha-pi/2);
yy2 = 0 - W .*sin(alpha+pi/2);
yy3 = y1 + W .*sin(alpha-pi/2);
yy4 = y1 + W .*sin(alpha+pi/2);

Link1 = [[xx1 xx2 xx3 xx4]' [yy1 yy2 yy3 yy4]'];

% Define Body Link 2

xx1 = x1 - W.*cos(alpha+beta+pi/2);
xx2 = x1 - W.*cos(alpha+beta-pi/2);
xx4 = x2 - W.*cos(alpha+beta+pi/2);
xx3 = x2 - W.*cos(alpha+beta-pi/2);

yy1 = y1 - W .*sin(alpha+beta-pi/2);
yy2 = y1 - W .*sin(alpha+beta+pi/2);
yy3 = y2 + W .*sin(alpha+beta-pi/2);
yy4 = y2 + W .*sin(alpha+beta+pi/2);

Link2 = [[xx1 xx2 xx3 xx4]' [yy1 yy2 yy3 yy4]'];

% Define obsticle as a poly with alot of points

theta = linspace(0,2*pi, 100);
semi0x = xo + r.*cos(theta);
semi0y = yo + r.*sin(theta);

obsticle = [[semi0x'] [semi0y']];

%-----------------------------------------------------------------

%Link 1 end
theta = linspace(alpha-pi/2,alpha+pi/2 , 100);
semi0x = x1 + W.*cos(theta);
semi0y = y1 + W.*sin(theta);
Link1_end = [[semi0x'] [semi0y']];

%Link 2 start
theta = linspace(alpha+beta-pi/2,alpha+beta+pi/2 , 100);
semi0x = x1 - W.*cos(theta);
semi0y = y1 - W.*sin(theta);
Link2_start = [[semi0x'] [semi0y']];

% Semi circle link 2 end
theta = linspace(beta+alpha-pi/2,beta+alpha+pi/2 , 100);
semi0x = x2 + W.*cos(theta);
semi0y = y2 + W.* sin(theta);
Link2_end = [[semi0x'] [semi0y']];

[TF1 TF2] = checkCollisionTwoLink(L1,L2,W,alpha,beta,xo,yo,r);

if TF1 ==1
    black(i,1) = alpha;
    black(i,2) = beta;
    red(i,1) = 0;
    red(i,2) = 0;
    blue(i,2) = 0;
    blue(i,1) = 0;

elseif TF2 ==1
    black(i,1) = 0;
    black(i,2) = 0;
    red(i,1) = alpha;
    red(i,2) = beta;
    blue(i,1) = 0;
    blue(i,2) = 0;

else
    black(i,1) = 0;
    black(i,2) = 0;
    red(i,1) = 0;
    red(i,2) = 0;
    blue(i,1) = alpha;
    blue(i,2) = beta;
end

end

plot(black(:,1), black(:,2), 'ko', 'MarkerFaceColor', 'k') % Black circles
hold on
plot(red(:,1), red(:,2), 'ro', 'MarkerFaceColor', 'r') % Red circles
hold on
plot(blue(:,1), blue(:,2), 'bo', 'MarkerFaceColor', 'b') % Blue circles
hold on
grid on

end
```

The complete code for the plotSampleConfigurationSpaceTwoLink function is provided above. This function allows the user to choose between three sampling methods—Sukharev, random, and Halton—by entering their respective names. To visualize the free configuration space, black points represent collisions with Link 1, red points indicate collisions with Link 2, and blue points denote no collisions. To achieve this, three alpha and three beta arrays were created to store the respective outcomes. The sampling functions for each method were developed in the previous lab and are integrated into this code. Three if statements determine the selected sampling method, and a for loop runs until the specified number of samples is reached. The configuration space is then plotted with both the x-axis and y-axis spanning from -π to π.

Input: N = 20^2 , “random”

```
%% Sampling
L1 = 4;
L2 = 4;
W  = 0.2; %  width
alpha = 0;
beta =  0;
xo = 0;
yo = 3;
r  = 2;
n = 20^2;
% sampling_method = "Sukharev";
% sampling_method = "Halton";
sampling_method = "Random";
grid = plotSampleConfigurationSpaceTwoLink(L1,L2,W,xo,yo,r,sampling_method,n);
```

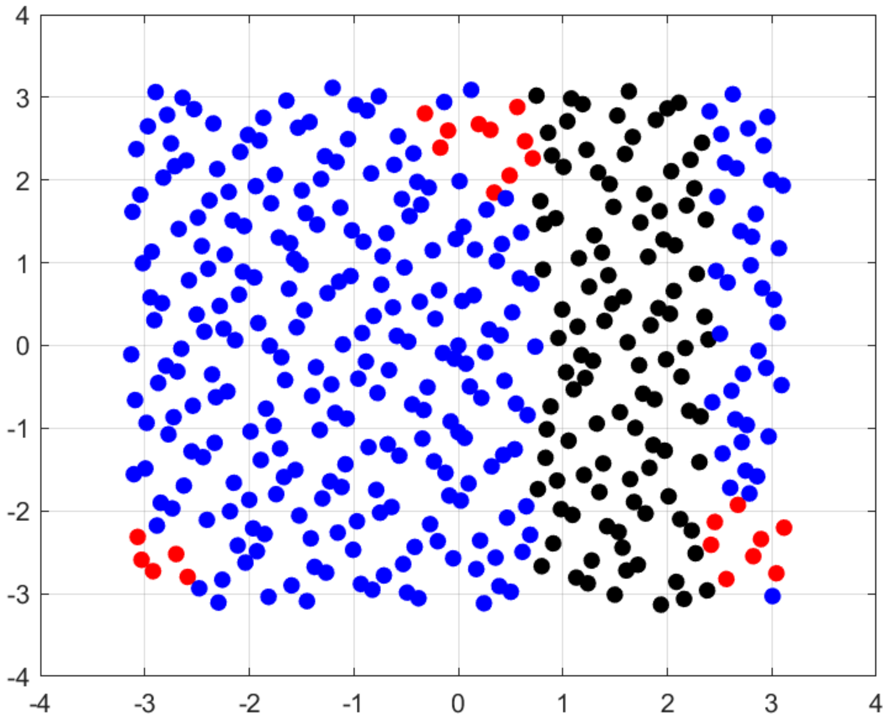
Output:



Input: N = 20^2, Halton

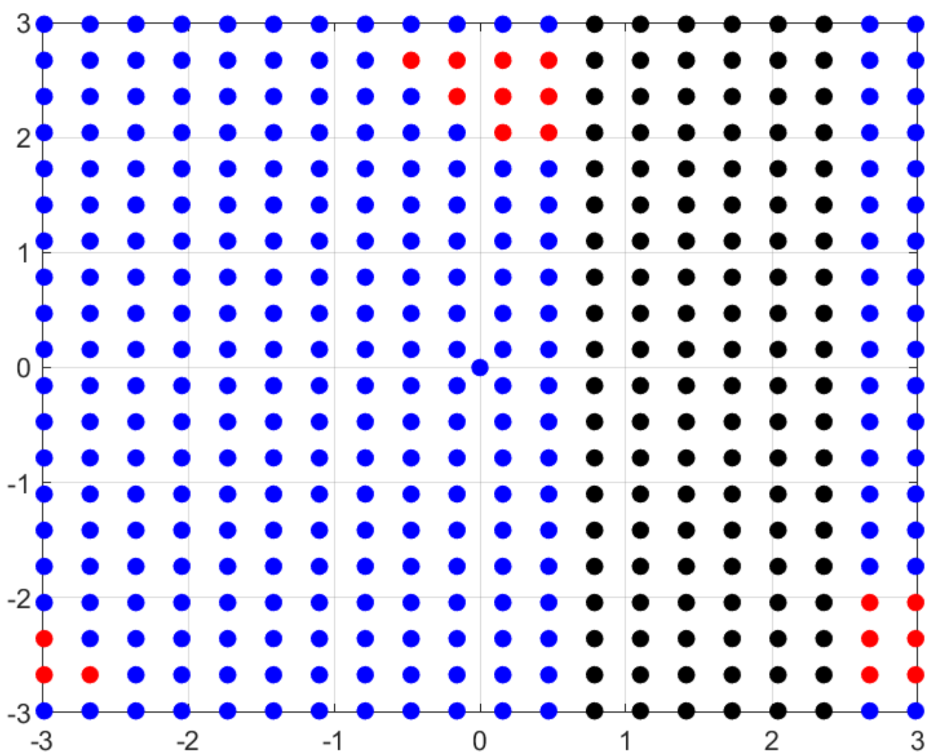
```
L1 = 4;
L2 = 4;
W  = 0.2; %  width
alpha = 0;
beta =  0;
xo = 0;
yo = 3;
r  = 2;
n = 20^2;
% sampling_method = "Sukharev";
sampling_method = "Halton";
% sampling_method = "Random";
grid = plotSampleConfigurationSpaceTwoLink(L1,L2,W,xo,yo,r,sampling_method,n);
```

Output:



Input: N=20^2, “Sukharev”

```
%% Sampling
L1 = 4;
L2 = 4;
W  = 0.2; %  width
alpha = 0;
beta =  0;
xo = 0;
yo = 3;
r  = 2;
n = 20^2;
sampling_method = "Sukharev";
%sampling_method = "Halton";
% sampling_method = "Random";
grid = plotSampleConfigurationSpaceTwoLink(L1,L2,W,xo,yo,r,sampling_method,n);
```



For fun. I made an interactive live plot. You can move your cursor in the configuration space and the plot will live to update and animate the Robot arm.