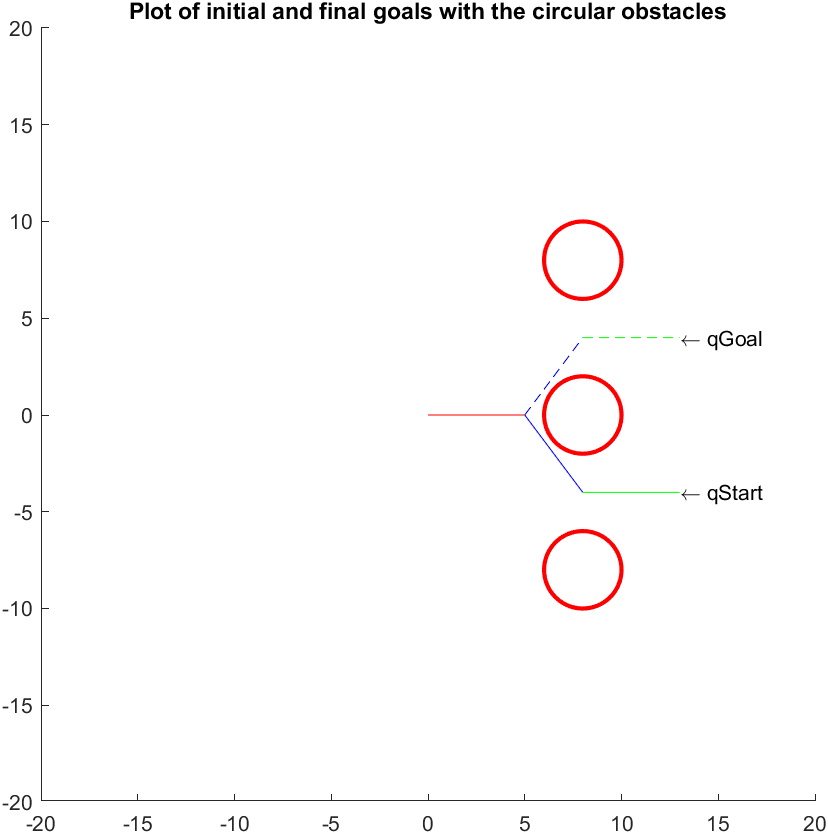
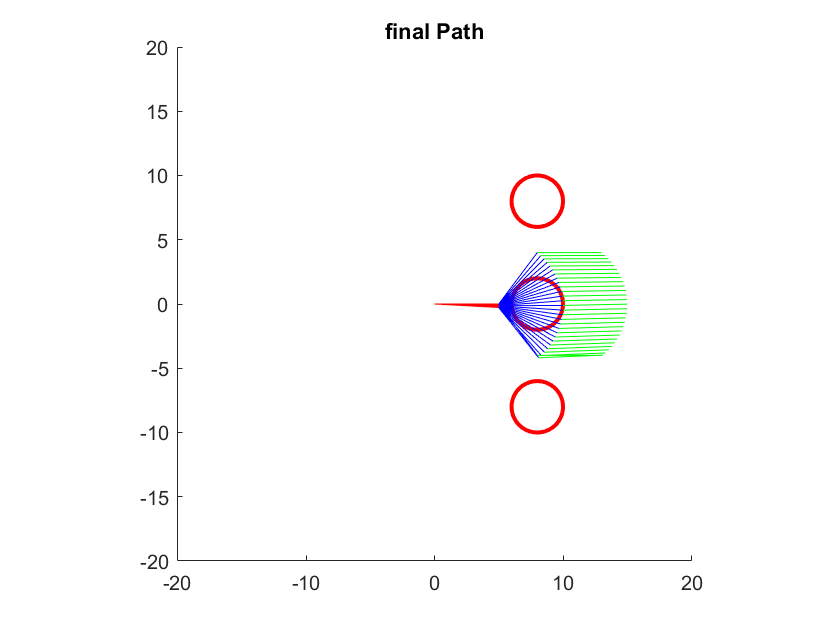
Project Assignment 2

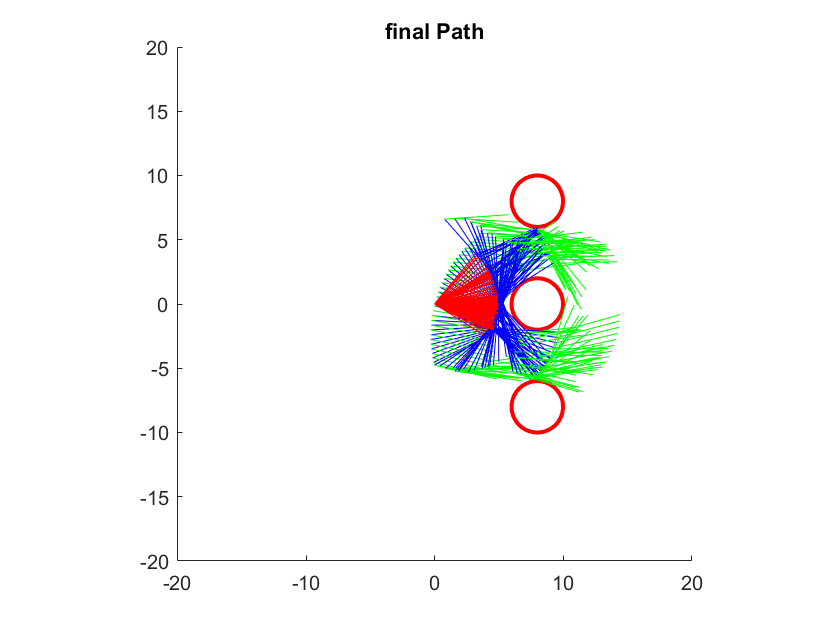
1. The robot utilized is 3 degrees-of-freedom planar robotic manipulator, with three rotational joints
2. The initial and goal positions are shown as below.



1. If we do not account for obstacles

It takes very few nodes to find the path without obstacles



1. If we do account for the obstacles
2. It takes n = 1000, nodes to find the path with obstacles

I have use a third party library for building and accessing trees, the code of which is attached in the submission.

The coding involved going through 7 stages of development, which are described by the code in the appendix.

1. Code – part 1 – understanding the problem.
2. Code – part 2– developing the local planner.
3. Code – part 3– developing the collision checker.
4. Code – part 4- developing single\_multiple function.
5. Code – part 5– developing extend\_multiple function.
6. Code – part 6– developing rrt algorithm function.
7. Code – part 7–plotting the result.

Code – part 1 – understanding the problem:

% understanding the work space.

% There are three circular obstacles placed in the workspace of the robotic manipulator.

% The obstacles are centered at coordinates (8,0), (8,8), and (8,-8), and they all have radii of 2 units.

% plotting qgoal and qstart

figure('Name','Obstacles')

title('Plot of initial and final goals with the circular obstacles')

% Fix the axis limits.

xlim([-20 20])

ylim([-20 20])

% Set the axis aspect ratio to 1:1.

axis square

viscircles([8,0],2);

hold on;

% Clear the axes.

viscircles([8,8],2);

hold on;

viscircles([8,-8],2);

hold on;

q\_start =[0 -0.9273 0.9273];

q\_goal = [0 0.9273 -0.9273];

a = 5;

x\_initial = 0;

y\_initial = 0;

theta1 = q\_start(1,1);

theta2 = q\_start(1,2);

theta3 = q\_start(1,3);

x\_final1 = a\*cos(theta1);

y\_final1 = a\*sin(theta1);

x\_final2 = a\*cos(theta1) + a\*cos(theta1 +theta2) ;

y\_final2 = a\*sin(theta1) + a\*sin(theta1 +theta2) ;

x\_final3 = a\*cos(theta1) + a\*cos(theta1 +theta2) + a\*cos(theta1 +theta2 + theta3);

y\_final3 = a\*sin(theta1) + a\*sin(theta1 +theta2) + a\*sin(theta1 +theta2 + theta3);

line([x\_initial,x\_final1],[y\_initial,y\_final1],'Color','red');

hold on ;

line([x\_final1,x\_final2],[y\_final1,y\_final2],'Color','blue');

hold on ;

line([x\_final2,x\_final3],[y\_final2,y\_final3],'Color','green');

hold on ;

% line([x\_initial,x\_final3],[x\_initial,y\_final3],'Color','black');

hold on;

theta1 = q\_goal(1,1);

theta2 = q\_goal(1,2);

theta3 = q\_goal(1,3);

x\_final1g = a\*cos(theta1);

y\_final1g = a\*sin(theta1);

x\_final2g = a\*cos(theta1) + a\*cos(theta1 +theta2) ;

y\_final2g = a\*sin(theta1) + a\*sin(theta1 +theta2) ;

x\_final3g = a\*cos(theta1) + a\*cos(theta1 +theta2) + a\*cos(theta1 +theta2 + theta3);

y\_final3g = a\*sin(theta1) + a\*sin(theta1 +theta2) + a\*sin(theta1 +theta2 + theta3);

line([x\_initial,x\_final1g],[y\_initial,y\_final1g],'Color','red','LineStyle','--');

hold on ;

line([x\_final1g,x\_final2g],[y\_final1g,y\_final2g],'Color','blue','LineStyle','--');

hold on ;

line([x\_final2g,x\_final3g],[y\_final2g,y\_final3g],'Color','green','LineStyle','--');

hold on ;

text(x\_final3,y\_final3,'\leftarrow qStart ')

text(x\_final3g,y\_final3g,'\leftarrow qGoal ')

% line([x\_initial,x\_final3],[x\_initial,y\_final3],'Color','black','LineStyle','--');

Code – part 2– developing the local planner:

%% local planner

function [success] = local\_planner(q\_near, q\_int, step\_size)

success = false;

disp('local\_planner called ')

obstacle1 = [8, 0, 2];

obstacle2 = [8, 8, 2];

obstacle3 = [8, -8, 2];

% obstacles = [];

obstacles = [obstacle1; obstacle2 ; obstacle3];

% Fix the axis limits.

xlim([-16 16]);

ylim([-16 16]);

axis square

viscircles([8,0],2);

hold on;

viscircles([8,8],2);

hold on;

viscircles([8,-8],2);

hold on;

q2 = q\_int;

q1 = q\_near;

delta\_q = q2 - q1;

delta\_q = limitAngle(delta\_q);

num\_steps = ceil(norm(delta\_q)/step\_size); % be careful about angle correction

step = delta\_q/ num\_steps;

collision = false;

q = q1;

for i = 1: num\_steps %%4

q = q + step ;

collision = collision || check\_collision( q, obstacles);

if(collision == true)

return;

end

end

success = not(collision);

end

function [doesTouch] = check\_collision(q,obstacles)

disp('check\_collision called ')

doesTouch = false;

a = 5;

x\_initial = 0;

y\_initial = 0;

theta1 = q(1,1);

theta2 = q(1,2);

theta3 = q(1,3);

x\_final1 = a\*cos(theta1);

y\_final1 = a\*sin(theta1);

x\_final2 = a\*cos(theta1) + a\*cos(theta1 +theta2);

y\_final2 = a\*sin(theta1) + a\*sin(theta1 +theta2);

x\_final3 = a\*cos(theta1) + a\*cos(theta1 +theta2) + a\*cos(theta1 +theta2 + theta3);

y\_final3 = a\*sin(theta1) + a\*sin(theta1 +theta2) + a\*sin(theta1 +theta2 + theta3);

line([x\_initial,x\_final1],[y\_initial,y\_final1],'Color','red');

hold on ;

line([x\_final1,x\_final2],[y\_final1,y\_final2],'Color','blue');

hold on ;

line([x\_final2,x\_final3],[y\_final2,y\_final3],'Color','green');

hold on;

% hold on ;

% line([x\_initial,x\_final3],[x\_initial,y\_final3],'Color','black');

for i = 1 : length(obstacles)

obstacles(i,:);

collidesl1 = collisionCheck(obstacles(i,:),[x\_final1,x\_final2],[y\_final1,y\_final2] ); %2nd link

collidesl2 = collisionCheck(obstacles(i,:),[x\_final2,x\_final3],[y\_final2,y\_final3] ); %3rd link

doesTouch = false;

doesTouch = collidesl1 || collidesl2;

if(doesTouch == true)

break;

else

continue;

end

end

end

function [Ta\_mat] = limitAngle(Ta\_mat)

[m, n] = size(Ta\_mat); %m rows, n columns

for i = 1: m

for j = 1: n

if (Ta\_mat(i,j) >= pi )

Ta\_mat(i,j) = Ta\_mat(i,j) - 2\*pi;

elseif (Ta\_mat(i,j) < - pi )

Ta\_mat(i,j) = Ta\_mat(i,j) + 2\*pi ;

end

end

end

end

Code – part 3– developing the collision checker:

function [collides] = collisionCheck(obstacle,X,Y)

disp("collisionCheck called")

collides = false

% %% make a frame

% xlim([0 16]);

% ylim([0 16]);

% axis square

% %% show a circle

% viscircles([4,4],2);

% %% test points

% x =[5,0,6,0,0,2];

% y =[0,4,0,6,10,8];

%

% %% circle points

% c1 = 4;

% c2 = 4;

C = [obstacle(1),obstacle(2)]

r = obstacle(3)

%% ALGORITHM

% E is the starting point of the ray,

% L is the end point of the ray,

% C is the center of sphere you're testing against

% r is the radius of that sphere

% line([0,2],[10,8]);

E1 = [X(1),Y(1)];

L1 = [X(2),Y(2)];

d = L1 - E1 %( Direction vector of ray, from start to end )

f = E1 - C %( Vector from center sphere to ray start )

a = dot( d,d )

b = 2\*dot( f,d ) ;

c = dot( f,f ) - r\*r ;

discriminant = b\*b-(4\*a\*c);

if(discriminant < 0)

% //no intersection

collides = false;

else

discriminant = sqrt( discriminant );

t1 = (-b - discriminant)/(2\*a)

t2 = (-b + discriminant)/(2\*a)

if( t1 >= 0 && t1 <= 1 )

% // t1 is the intersection, and it's closer than t2

% // (since t1 uses -b - discriminant)

% // Impale, Poke

collides = true;

end

if ( t2 >= 0 && t2 <= 1 )

collides = true;

end

end

end

Code – part 4– developing extend\_single function:

function [result, Ta, q\_target ] = rrt\_extend\_single\_func(Ta, q\_random, step\_length)

step\_size = 0.01;

disp("rrt\_extend\_single\_func called")

q\_target= [];

[q\_near,indexnear] = find\_nearest(Ta, q\_random);

q\_int = limit( q\_random, q\_near, step\_length); % be careful about angle correction

result = local\_planner(q\_near, q\_int, step\_size);

if (result == true)

Ta = Ta.addnode( indexnear , q\_int);

q\_target = q\_int;

end

end

%% find\_nearest

function[q\_near,indexnear] = find\_nearest(Ta, q\_random)

disp("find\_nearest called")

n = nnodes(Ta); % number of nodes in Tree A

Ta\_mat = zeros(n,3);% magnitude matrix for Tree A initialised to zero

diff\_mat = zeros(1,n);

for i = 1 : n

Ta.get(i)

diff\_mat(1,i) = norm(limitAngle(Ta.get(i) - q\_random)) % array of Tree A

end

index = find(diff\_mat == min(diff\_mat));

q\_near = Ta.get(index(1));

indexnear = index(1);

end

%% limit

% The LIMIT function finds an intermediate configuration q\_int from between q\_near and q\_target,

% such that q\_int is at a distance of step\_length from q\_near.

% If the distance between q\_near and q\_target is less than step\_length, it would return q\_target

function [q\_int] = limit( q\_random, q\_near, step\_length)

disp("limit called")

delta\_q = q\_random - q\_near ;

delta\_q = limitAngle(delta\_q);

step = step\_length \* delta\_q / norm(delta\_q) ;

% step\_length = norm(step)

flag = step\_length < norm(delta\_q);

if(flag == true)

q\_int = q\_near + step ;

else

q\_int = q\_random;

end

end

%% random node generator

function[q\_rand] = random\_node\_gen()

a = 0;

b = 2 \* pi;

q\_rand = a + (b-a).\* rand(1,3);

% angleInDegrees = rad2deg(q\_rand);

for i = 1: length(q\_rand)

if (q\_rand(1,i) > pi)

q\_rand(1,i) = -( b - q\_rand(1,i));

end

end

end

%% limitAngle

function [Ta\_mat] = limitAngle(Ta\_mat)

[m, n] = size(Ta\_mat); %m rows, n columns

for i = 1: m

for j = 1: n

if (Ta\_mat(i,j) >= pi )

Ta\_mat(i,j) = Ta\_mat(i,j) - 2\*pi;

elseif (Ta\_mat(i,j) <- pi )

Ta\_mat(i,j) = Ta\_mat(i,j) + 2\*pi ;

end

end

end

end

Code – part 5– developing extend\_multiple function:

function [result, Tb, q\_connect] = rrt\_extend\_multiple\_func(Tb,q\_target,step\_length)

step\_size = 0.1;

disp("rrt\_extend\_multiple\_func called")

q\_connect= [];

result = false;

[q\_near,indexnear]= find\_nearest(Tb, q\_target);

q\_int = limit( q\_target, q\_near, step\_length); % be careful about angle correction

q\_last= q\_near;

num\_steps= ceil(norm(limitAngle(q\_target-q\_near))/step\_length);

for i=1:num\_steps

result = local\_planner(q\_int, q\_last, step\_size);

if (result == true)

% [ t node1 ] = t.addnode(1, 'Node 1'); %% attach to root

[Tb, idx]= Tb.addnode(indexnear , q\_int);

q\_connect = q\_int;

if (i< num\_steps)

q\_last = q\_int;

q\_int = limit( q\_target, q\_int, step\_length);

indexnear = idx; %%% have added qint, its index in that tree

end

else

return;

end

end

end

%% find\_nearest

function[q\_near,indexnear] = find\_nearest(Ta, q\_random)

n = nnodes(Ta); % number of nodes in Tree A

Ta\_mat = zeros(n,3);% magnitude matrix for Tree A initialised to zero

diff\_mat = zeros(1,n);

for i = 1 : n

Ta.get(i)

diff\_mat(1,i) = norm(limitAngle(Ta.get(i) - q\_random)) % array of Tree A

end

min(diff\_mat);

index = find(diff\_mat == min(diff\_mat));

q\_near = Ta.get(index(1));

indexnear = index(1);

end

%% limit

% The LIMIT function finds an intermediate configuration q\_int from between q\_near and q\_target,

% such that q\_int is at a distance of step\_length from q\_near.

% If the distance between q\_near and q\_target is less than step\_length, it would return q\_target

function [q\_int] = limit( q\_random, q\_near, step\_length)

disp("limit called")

delta\_q = q\_random - q\_near;

delta\_q = limitAngle(delta\_q);

step = step\_length \* delta\_q / norm(delta\_q) ;

flag = step\_length < norm(delta\_q);

if(flag == true)

q\_int = q\_near + step;

else

q\_int = q\_random;

end

end

%% limit Angle

function [Ta\_mat] = limitAngle(Ta\_mat)

[m, n] = size(Ta\_mat); %m rows, n columns

for i = 1: m

for j = 1: n

if (Ta\_mat(i,j) >= pi )

Ta\_mat(i,j) = Ta\_mat(i,j) - 2\*pi;

elseif (Ta\_mat(i,j) < - pi )

Ta\_mat(i,j) = Ta\_mat(i,j) + 2\*pi ;

end

end

end

end

Code – part 6– developing rrt algorithm function:

function [] = main()

clc;

%rng(1);

disp("main called")

q\_start =[0, -0.9273 ,0.9273];

q\_goal = [0 , 0.9273 , -0.9273];

Ta = tree(q\_start);

Tb = tree(q\_goal);

success = false;

max\_nodes = 1000;

step\_length = 0.1;

% step\_length = 1;

q\_connect=[];

result2 = false;

result = false;

for i = 1: max\_nodes

q\_rand = random\_node\_gen(); %% find the next point to extend to

[result, Ta, q\_target] = rrt\_extend\_single\_func(Ta, q\_rand, step\_length); % extend from Ta to this new point using this step length

if(result == true)

disp("1st True")

[result2, Tb, q\_connect] = rrt\_extend\_multiple\_func(Tb, q\_target,step\_length);

if (result2 == true) % connected the two trees

disp("2nd True")

success = true;

break;

end

end

[Ta,Tb] = swap(Ta, Tb);

end

plotFinal(success, q\_connect, Ta, Tb);

end

%% random\_config()

function[q\_rand] = random\_node\_gen()

disp("random\_node\_gen called");

a = 0;

b = 2 \* pi;

q\_rand = a + ( b-a ).\* rand(1,3);

for i = 1: length(q\_rand)

if (q\_rand(1,i) >= pi) % if value is greater than 180, we get a negative degree

q\_rand(1,i) = -( b - q\_rand(1,i));

end

end

end

%% swap

function [Ta,Tb] = swap(Ta, Tb)

temp = Ta;

Ta = Tb ;

Tb = temp ;

disp("swapped");

end

Code – part 7– plotting the result

function [] = plotFinal(success, q\_connect, Ta, Tb)

disp("plotFinal called");

nodesA = nnodes(Ta);

nodesB = nnodes(Tb);

pathFromA = zeros(nodesA ,3);

pathFromB = zeros(nodesB ,3);

if success == true

for nA = nodesA :-1:1 %% tracing back each parent from final node

elem = Ta.getparent(nA);

if(elem == 0)

parent = -1 ;

else

parent = Ta.get(elem);

end

pathFromA(nodesA - nA + 1,:) = parent;

end

for nB = nodesB :-1:1

elem = Tb.getparent(nB);

if(elem == 0)

parent = -1;

else

parent = Tb.get(elem) ;

end

pathFromB(nodesB - nB + 1,: )= parent;

end

pathFromA

pathFromB

pathFromA(nodesA,:) = []; % remove the last element i.e. [-1,-1,-1]

pathFromB(nodesB,:) = [];

path = [pathFromB; q\_connect; pathFromA] % make the whole path

plotThePath(path);

else

disp("no plot");

for i=1 : nodesA

pathFromA(i,:) = Ta.get(i);

end

plotThePath(pathFromA)

for i=1 : nodesB

Tb.get(i);

pathFromB(i,:) = Tb.get(i);

end

plotThePath(pathFromB)

end

end

%%

function [] = plotThePath(path)

% path

% length(path)

disp("plotThePath called");

figure();

title('final Path');

% Fix the axis limits.

xlim([-20 20]);

ylim([-20 20]);

% Set the axis aspect ratio to 1:1.

axis square

viscircles([8,0],2);

hold on;

% Clear the axes.

viscircles([8,8],2);

hold on;

viscircles([8,-8],2);

hold on;

a = 5;

x\_initial = 0;

y\_initial = 0;

for q = 1 : size(path,1)

q;

theta1 = path(q,1);

theta2 = path(q,2);

theta3 = path(q,3);

x\_final1 = a\*cos(theta1);

y\_final1 = a\*sin(theta1);

x\_final2 = a\*cos(theta1) + a\*cos(theta1 +theta2);

y\_final2 = a\*sin(theta1) + a\*sin(theta1 +theta2);

x\_final3 = a\*cos(theta1) + a\*cos(theta1 +theta2) + a\*cos(theta1 +theta2 + theta3);

y\_final3 = a\*sin(theta1) + a\*sin(theta1 +theta2) + a\*sin(theta1 +theta2 + theta3);

line([x\_initial,x\_final1],[y\_initial,y\_final1],'Color',"red");

hold on ;

line([x\_final1,x\_final2],[y\_final1,y\_final2],'Color',"blue");

hold on ;

line([x\_final2,x\_final3],[y\_final2,y\_final3],'Color',"green");

hold on;

end

end