C-SPACE

INTRODUCTION

In this exercise, the goal is to produce a map representing the configuration space for a two degree of freedom robotic arm. In this Configuration Space, it is necessary to convert an obstacle defined in the Cartesian Space as a sphere into an obstacle to avoid for the joints positions. In fact, C-space is an n-dimensional space (with n equal to the numer of the joints of the robot) where robot position is defined not by the position of its end effector in the x-y(-z) coordinate, but with the position of its joints. This space is often more useful to use because robot is not modeled as a point but instead it is necessary to consider its kinematics relationships in path planning. However, this is not always so easy to do in the cartesian space because there are some particular configurations, called kinematics singularities, that need to be taken into account considering joint positions/speeds.
In the given exercise, a 2 d.o.f manipulator must reach a final position avoiding contact with the obstacle. This obstacle needs to be converted into a correspondent obstacle in C-space.
One of the easiest way to do so, and this is what is done in this example, its to define grid points for the configuration space and check if, for every of these points in the space (that corresponds to a position of the robot) there is collision. If there is no collision, the point represents an available configuration, otherwise it is not available.

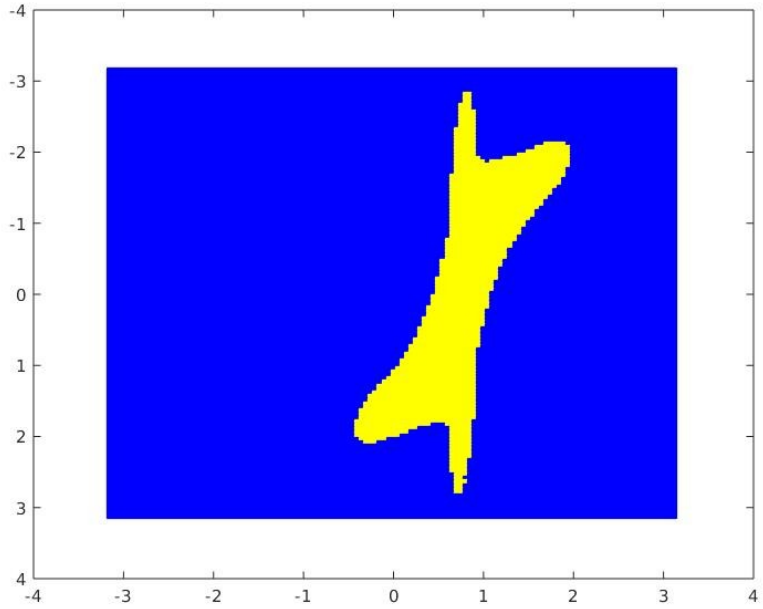
CODE DESCRIPTION

Firstly, the robot has been defined using the function rigidBodyTree. To do so, given parameters of links length and starting position are defined. Then the robot is composed starting from the base (revolute joint 1), to link 1 and joint 2 (revolute) and finally link 2 with the fixed joint corresponding to the end effector.
Additional inverse kinematics is define (it is used to compute approximate value of joints q1, q2 corresponding to starting position and final position in the x-y plane) together with collisions properties both of the robot and of the sphere.

After this initialization, the core of the exercise starts. First of all, two vectors, from -pi to pi representing values for joint q1 (theta0) and joint q2 (theta1) are defined, with a step of 0.05, which is sufficiently precise to define a C-space for our purpose.

As explained before, the algorithm consists in checking for every point in the grid (which is a point described by coordinate (q1,q2)) if for the correspondent position of the robot there is collision or not. This is done with 2 for loops, one to move along theta0 direction and one to move along theta1 direction.
At every iteration, joints position vector q is computed together with correspondent pose of link1 and link2 through robot.getTransform function. S
With the use of checkCollision function, at every iteration collision between link 1 and obstacle and link 2 and obstacle is detected. If one of the two link is in collision, the correspondent point in the plot is plotted in yellow; viceversa, if there is no collision, it

is plotted in blue (a square marker is used to reproduce exactly as it is the figure of the C-space shown during lecture).
A final adjust is done with the gca (current axes or chart) in order to reverse the Y axis and obtain the same image. Final results is the following:



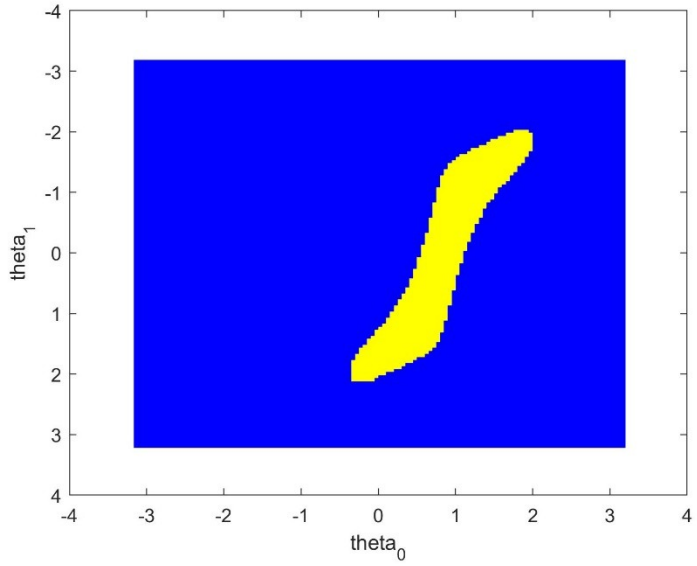
RESULTS ANALYSIS AND FINAL CONSIDERATIONS

Obtained result shows that with this very simple method, it is very easy to shift from Cartesian Space to Configuration Space. Of course, reducing the step size of the theta0 and theta1 vector, a more accurate mapping can be obtained, but already with a stepsize of 0.01 the computational effort increases and it requires a relatively high amount of time.
Also, before realizing this code with the use of Robotics Toolbox (I was unable to make it work because I had MATLAB r2019a and r2019b is required), I did another implementation, trying to reproduce the robot as if it was compose by two connected lines forming a kinematic chains. I report the script here:

```
clear all
close all
theta_0_axis = -(-pi:0.05:pi);
theta_1_axis = -(-pi:0.05:pi);

%% robot parameters
%initial condition q1, q2 = 0;
q1 = 0;
q2 = 0;
L1 = 0.5;
L2 = 0.5;
joint_0 = [0; 0];
joint_1 = joint_0 + [L1*cos(q1);L1*sin(q1)];
end_effector = joint_1 + [L2*cos(q1+q2);L2*sin(q1+q2)];

obstacle = circle(0.5,0.5,0.2);
obs = false(length(theta_0_axis),length(theta_1_axis));

%robot = rigidBodyTree('DataFormat','column','MaxNumBodies',3);
```

```
for i=1:length(theta_1_axis)        %along the vertical
    for j=1:length(theta_0_axis)    %along the horizontal
        q1 = theta_0_axis(j);
        q2 = theta_1_axis(i);
        joint_1 = [L1*cos(q1);L1*sin(q1)];
        end_effector = joint_1 + [L2*cos(q1+q2);L2*sin(q1+q2)];
        collisionStatus_1 = detect_collision(joint_0, joint_1, obstacle);
        collisionStatus_2 = detect_collision(joint_1, end_effector, obstacle);
        % collision!
        if (collisionStatus_1 || collisionStatus_2)
            plot(theta_0_axis(j),theta_1_axis(i),'-
s','MarkerEdgeColor','yellow','MarkerFaceColor','yellow')
            h = gca;
            h.YDir = 'reverse';
            hold on
            obs(j,i) = true;
        % no collision!
        else
            plot(theta_0_axis(j),theta_1_axis(i),'-
s','MarkerEdgeColor','blue','MarkerFaceColor','blue')
            hold on
            h = gca;
            h.YDir = 'reverse';
        end
    end
end

xlabel('theta_0'), ylabel('theta_1');



%sphere
function h = circle(x,y,r)

th = 0:pi/50:2*pi;
xunit = r * cos(th) + x;
yunit = r * sin(th) + y;
h = polyshape(xunit, yunit);

end



%% collision function
function exit_flag = detect_collision(q1,q2, P)

line = [q1(1) q1(2); q2(1) q2(2)];

[in,out] = intersect(P,line);

if sum(in) == 0
    exit_flag = 0;
else
    exit_flag = 1;
end
end
```

This code, which is basically the same as the one before, declares the sphere object as a polygon, link 1 corresponds to a line connecting joint 0 and joint1 and link 2 corresponds to a line connecting joint 1 and end effector. Collision function is the same used for exercise 5.1. I wanted to report also this code to highlight an interesting consideration. The produce C-space with this code is the following:



You can see that even if this C-space is very similar to the correct one produced with the first code, two important features are missing with respect to the previous one: this is because this model does not take into consideration the width of the links: when the arm is in a position where joint 1 is close to the obstacle, there are some configurations where the link 2 touches it! This of course must be avoided and the correct C-space is the one obtained with the first code and not this one