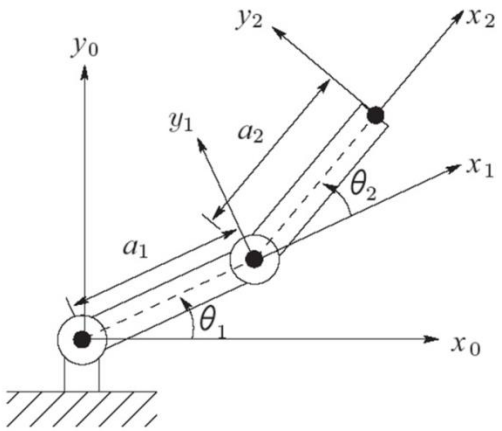
MAE 6245 (Spring 2020)

Robotic Systems

Assignment # 5 Solutions

1) For the manipulator shown below, do the following (assume a_1=a_2=10) (ignore the frames shown in the figure – use your own frame definition):



a) Find the closed form inverse kinematic solution (using whatever method you like) to find θ_1, θ_2 given x, y (no orientation given). [This should be a simpler problem than the one worked out in class, and you may be able to make use of similar identities to work out the solution.]. Also write a MATLAB function to compute the joint angles, given x, y (using the closed form inverse solution directly).

b) Use one of the numerical techniques shown in class (root finding or optimization) to find the solution of the equation – given x, y. Basically create another MATLAB function to compute the angles, when end effector position is given.

c) Write a function in MATLAB to use the brute force technique to find θ_1, θ_2 given particular values for x, y. Use the allowable range as: $\theta_1 \in [0°, 90°]$, $\theta_2 \in [-90°, 90°]$.

d) Choose a circle in the workspace of the manipulator. [One way to choose the circle is to plot the workspace and find an equation of a circle that fits within the space. You are free to choose the radius. Try to choose something big enough that it is not miniscule, and yet fits within the workspace.] Figure out a way to define several points (10 to 20 points) around the circle (preferably equi-distant).

e) Write another function (or include it in the main script) to plot the robot configuration for given values of θ_1, θ_2.

f) Now, for each point on the circle, use the function in part (a) to calculate the inverse solution and plot it. Your function would like this:

```
tic
for each point on the circle
        Find inverse kinematics solution
        Plot robot configuration (overwriting previous plot)
end
toc
```

Note that you should be able to calculate the next point on the circle automatically. This process will be very laborious if you do it by hand.

Also note that “tic” and “toc” are MATLAB commands that give the time to execute any function that is defined in between them. So this will give you the time that the function took to execute.

g) Now repeat the process with functions developed in (b) and (c).

What are the times to execute each of these? Can all of them be used for real-time solutions?

You have just developed 3 control techniques to make the manipulator move in a circle. Congratulations !

[25 points]

Solution

a)

Forward Kinematics :-

$$x = a_1 c\theta_1 + a_2 c(\theta_1 + \theta_2)$$
$$y = a_1 s\theta_1 + a_2 s(\theta_1 + \theta_2)$$

$$\therefore x^2 + y^2 = a_1^2 + a_2^2 + 2a_1a_2\left[c\theta_1\, c(\theta_1+\theta_2) + s\theta_1\, s(\theta_1+\theta_2)\right]$$
$$= a_1^2 + a_2^2 + 2a_1a_2\left[c(-\theta_1)\, c(\theta_1+\theta_2) - s(-\theta_1)\, s(\theta_1+\theta_2)\right]$$

Use $c(a+b) = ca\,cb - sa\,sb$

$$\therefore x^2 + y^2 = a_1^2 + a_2^2 + 2a_1a_2 c\theta_2$$

$$\therefore \theta_2 = \cos^{-1}\left(\frac{x^2 + y^2 - a_1^2 - a_2^2}{2a_1a_2}\right)$$

Also, $x = a_1 c\theta_1 + a_2 c\theta_1 c\theta_2 - a_2 s\theta_1 s\theta_2$

$$\Rightarrow (a_1 + a_2 c\theta_2)c\theta_1 + (-a_2 s\theta_2)s\theta_1 - x = 0$$

We know that, solution of "$a\,c\theta + b\,s\theta - c = 0$" is : $\theta = \text{atan2}(b, a) \pm \text{atan2}\left(\sqrt{a^2 + b^2 - c^2}, c\right)$

$$\therefore \theta_1 = \frac{\text{atan2}(-a_2 s\theta_2, a_1 + a_2 c\theta_2) \pm \text{atan2}\left(\sqrt{(a_1 + a_2 c\theta_2)^2 + (a_2 s\theta_2)^2 - x^2}, x\right)}{}$$

Matlab code to calculate solution:

```
close all;
clear all;

%% Define Link Lengths
a1 = 10;
a2 = 10;

%% Define End Effector Position (assume 2D for planar manipulator)
x = 10*sqrt(2);
y = 10*sqrt(2);

%% Inverse Kinematics Solution
theta2 = acos((x^2+y^2-a1^2-a2^2)/(2*a1*a2))
theta1_1 = atan2(-a2*sin(theta2),a1+a2*cos(theta2)) +
atan2(sqrt((a1+a2*cos(theta2))^2 + (a2*sin(theta2))^2 - x^2), x)          %
First Solution
theta1_2 = atan2(-a2*sin(theta2),a1+a2*cos(theta2)) -
atan2(sqrt((a1+a2*cos(theta2))^2 + (a2*sin(theta2))^2 - x^2), x)          %
Second Solution
```

Example solution for x=y=10*sqrt(2): theta1 = ±0.7854, theta2 = 0

b) Use Optimization to Solve for this

Main Script

```
close all;
clear all;

%% Define Link Lengths
a1 = 10;
a2 = 10;

%% Define End Effector Position
x = 20/sqrt(2);
y = 20/sqrt(2);

%% Define Initial Values
theta1_init = 0.4;

%% Define Function Handle
fun1 = @inverseEqn1;

%% Compute Inverse Using Optimization
theta1 = fminunc(fun1,theta1_init)
% Find The Other Angle
theta2 = acos((x-a1*cos(theta1))/a2) - theta1
```

Function Definition

```
function [f] = inverseEqn1(theta1)
    x = 20/sqrt(2);
    y = 20/sqrt(2);
    a1 = 10;
    a2 = 10;

    f = abs(a1*sin(theta1) + a2*sin(acos((x-a1*cos(theta1))/a2)) - y);
```

Example solution for x=y=10*sqrt(2): theta1 = 0.7854, theta2 = 0

Note that the above is an inelegant way of defining the function, since it requires re-declaration of the fixed parameters and end effector positions. There is a way to pass these values are arguments to the function and then the way we define handles and function calls will also change – try it for fun !

c) Use the forward kinematics equations defined earlier.

Main Script

```
close all;
clear all;

%% Define Link Lengths
a1 = 10;
a2 = 10;

%% Define Joint Angle Values
theta1 = 0.7845;
theta2 = 0;

%% Compute End Effector Position
pos = fwdKin(a1,a2,theta1,theta2)

%% Plot Workspace
% Open Figure
figure(1);
hold on;
for theta1 = 0:0.1:pi/2
    for theta2 = -pi/2:0.1:pi/2
        pos = fwdKin(a1,a2,theta1,theta2);
        scatter(pos(1),pos(2),'b');
    end
end
```

Function

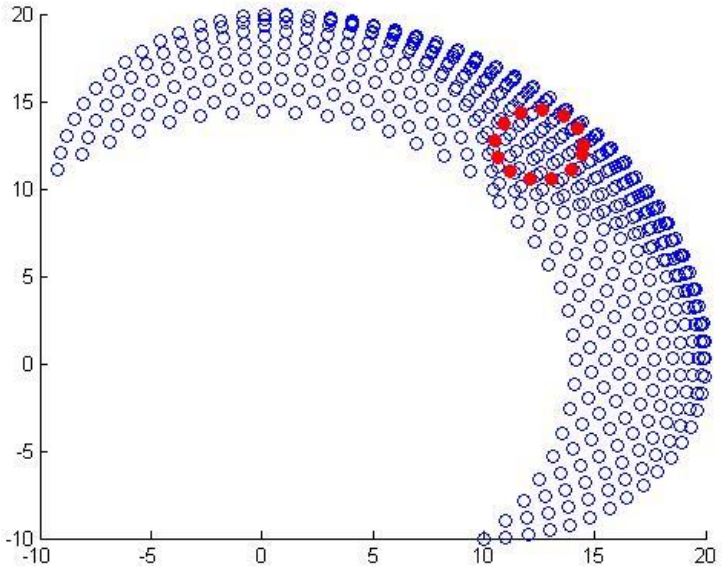
```
function [pos] = fwdKin(a1,a2,theta1,theta2)
    % Define End Effector Position Using Forward Kinematics Solution
    x = a1*cos(theta1) + a2*cos(theta1+theta2);
    y = a1*sin(theta1) + a2*sin(theta1+theta2);

    pos = [x; y];
```

Example solution: for theta1 = 0.7854, theta2 = 0: x = 14.1548, y = 14.1294

d) We plot the workspace to roughly determine the location of the circle and the radius that would fit. Then use the coordinates of the points around the circumference of the circle: $x = x_{center} + r\cos(\varphi)$; $y = y_{center} + r\sin(\varphi)$; where φ is the angle that varies from 0 to 360 degrees.

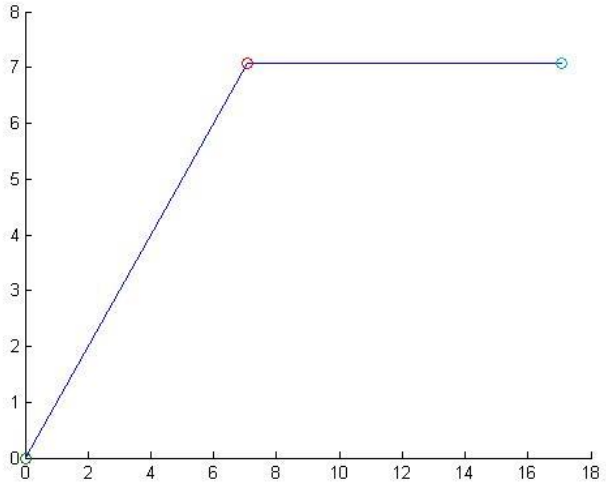
```
%% Define Circle
circleCenter = [12.5; 12.5];
radius = 2;
% Rotate Around the Circumference of the Circle
for phi = 0:0.5:2*pi
    xCircle = radius*cos(phi) + circleCenter(1);
    yCircle = radius*sin(phi) + circleCenter(2);
    scatter(xCircle,yCircle,'filled','r');
end
```



e) To define the links, we can plot lines joining the end points of the links. We know the base and the end effector position for given angles. We only need to define the position of joint

2. There are several ways to do this, but one way is to use the forward kinematics solution with a2 = 0.

```
%% Plot Manipulator
theta1 = 0.7845;
theta2 = -0.7845;
% Define Position of Joint Locations
joint1 = [0; 0];
joint2 = fwdKin(a1,0,theta1,theta2);
joint3 = fwdKin(a1,a2,theta1,theta2);
% Plot
figure(2);
hold on;
plot([joint1(1),joint2(1),joint3(1)],[joint1(2),joint2(2),joint3(2)]);
```



Parts (f) and (g) are straightforward application of code developed earlier. The time for execution would typically be (1) closed form solution, (2) numerical solution, and (3) brute force solution in increasing order of time required.

1) For the problem solved in class using “fminunc” function, solve the same problem using “fmincon”. Note that this is the function for constrained optimization, so you will first need to pose the problem as a constrained optimization problem [Hint: We discussed both formulations in class]. Compare results with the “correct” solution found in class.

[15 points]

Solution

Constrained Problem Formulation

Min. 1

w.r.t. θ_1

s.t. $j(\theta_1) = 0$

fmincon usage:

$\theta_1 = fmincon(fun, \theta_{1,init}, A, B, Aeq, Beq, LB, UB, NONLCON)$

Note:
* 'fun' is a fn handle as defined earlier but now we want to return 1 instead of $j(\theta_1)$
* $\theta_{1,init}$ is initial starting point
* A, B, Aeq, Beq refer to equality/inequality linear constraints; but we don't have any (Try to see what we can put as values)
* LB, UB are lower and upper bounds (can set them to very low and very high numbers)
* NONLCON is a function that will accept θ_1 and return $j(\theta_1)$