Robot Manipulators: Bonus Task 1

Titta Kemppi
282751

Background

In this task we were given a two-jointed robot model (picture below) to work out the inverse kinematics for. For this, we used a numerical method called "Jacobian Inverse Kinematics Technique" and were given a pseudocode to implement.

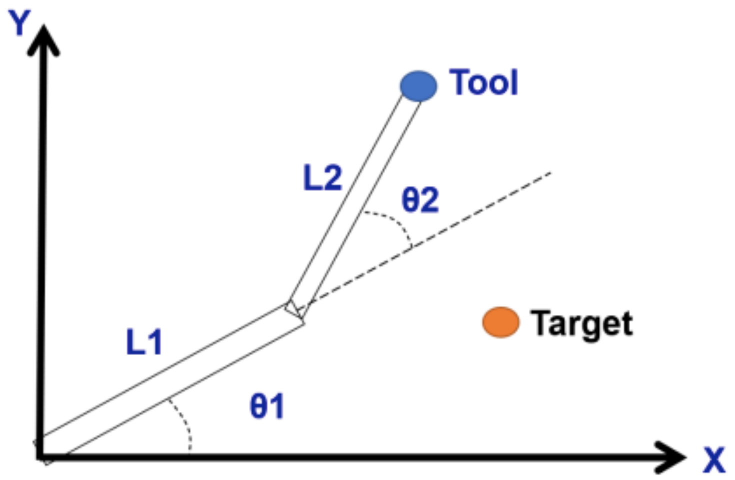


Figure 1 RR Planar Manipulator

What inverse kinematics means, is that given the coordinates of where we want our tool tip, we work out the angles the joints need to be in for the robot to reach its goal. This is often easier said than done. Getting our theta angles requires solving nonlinear differential functions, and there isn't really an analytical way to do that. All in all, there are three ways to come up with the angles: geometrical, algebraic and numerical. In this task we focus on numerical solving, for it's the most reliable of the three.

For my robot's initial values, I chose L1 = 1 m, L2 = 2 m, theta1 = pi/2 and theta2 = pi/2. For the goal to reach I chose coordinates (2 m, 2 m).

With the picture above and given information, we can solve the differential equations for our tool tip's coordinates:

$$e_x = L_1\cos(\theta_1) + L_2\cos(\theta_1 + \theta_2) \tag{1}$$
$$e_y = L_1\sin(\theta_1) + L_2\sin(\theta_1 + \theta_2) \tag{2}$$

With these equations we can make our Jacobian matrix:

$$J(\underline{e},\underline{\theta}) = \begin{bmatrix} \frac{\partial e_x}{\partial \theta_1} & \frac{\partial e_x}{\partial \theta_2} \\ \frac{\partial e_y}{\partial \theta_1} & \frac{\partial e_y}{\partial \theta_2} \end{bmatrix} \tag{3}$$

Where the derivatives are,

$$\frac{\partial e_x}{\partial \theta_1} = -L_1 \sin(\theta_1) - L_2 \sin(\theta_1 + \theta_2)\,, \tag{4}$$

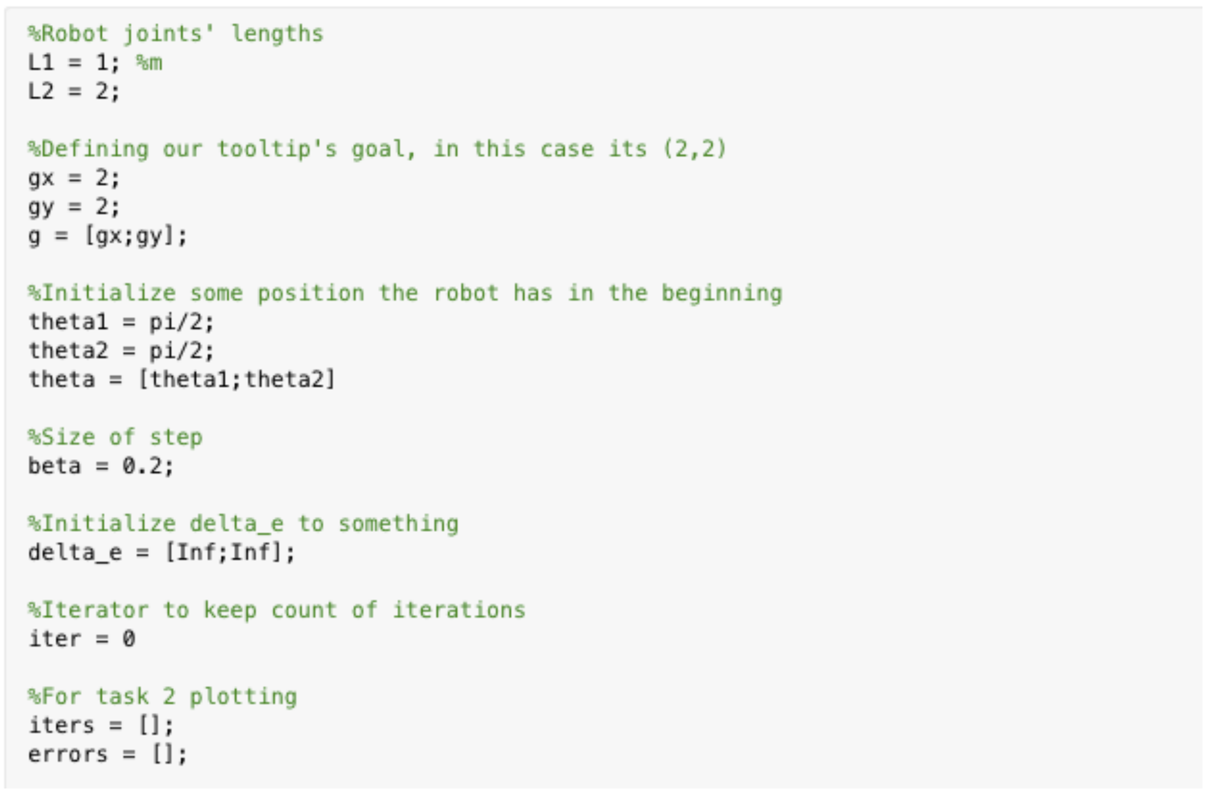
$$\frac{\partial e_x}{\partial \theta_2} = -L_2 \sin(\theta_1 + \theta_2)\,, \tag{5}$$

$$\frac{\partial e_y}{\partial \theta_1} = L_1 \cos(\theta_1) + L_2 \cos(\theta_1 + \theta_2)\,, \tag{6}$$

$$\frac{\partial e_y}{\partial \theta_2} = L_2 \cos(\theta_1 + \theta_2)\,. \tag{7}$$

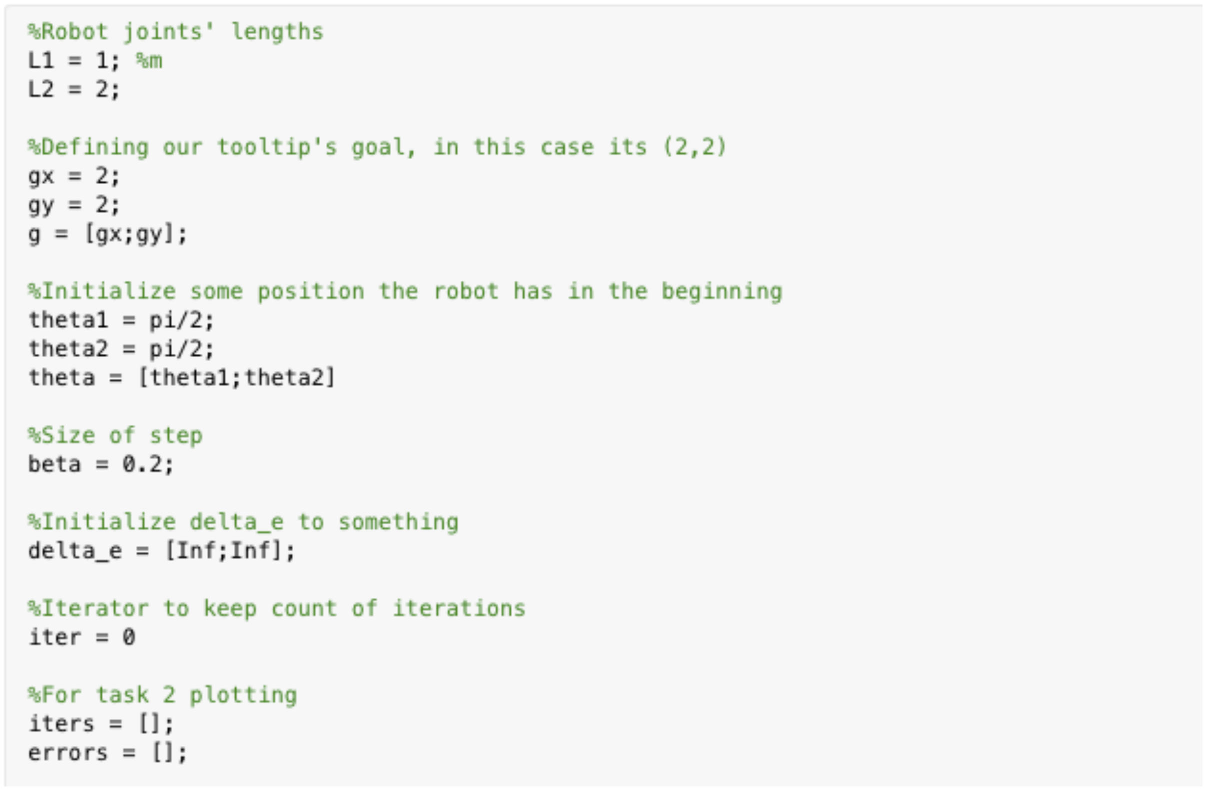
Solution in MATLAB

Here are the snippets of code from my implementation of the algorithm, separated into given tasks. First, initializations for the first task:



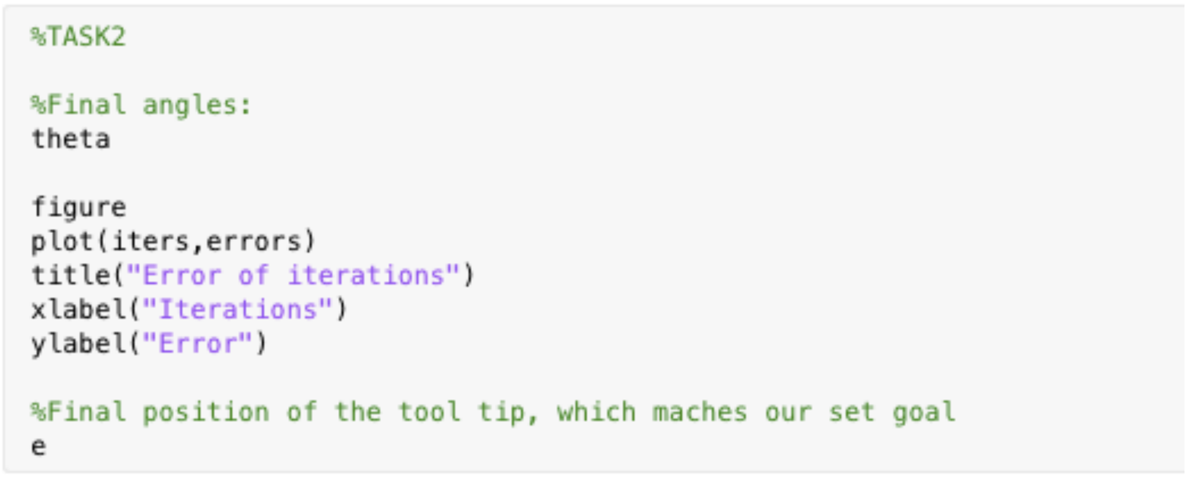
```
%Robot joints' lengths
L1 = 1; %m
L2 = 2;

%Defining our tooltip's goal, in this case its (2,2)
gx = 2;
gy = 2;
g = [gx;gy];

%Initialize some position the robot has in the beginning
theta1 = pi/2;
theta2 = pi/2;
theta = [theta1;theta2]

%Size of step
beta = 0.2;

%Initialize delta_e to something
delta_e = [Inf;Inf];

%Iterator to keep count of iterations
iter = 0

%For task 2 plotting
iters = [];
errors = [];
```

Then, the algorithm:

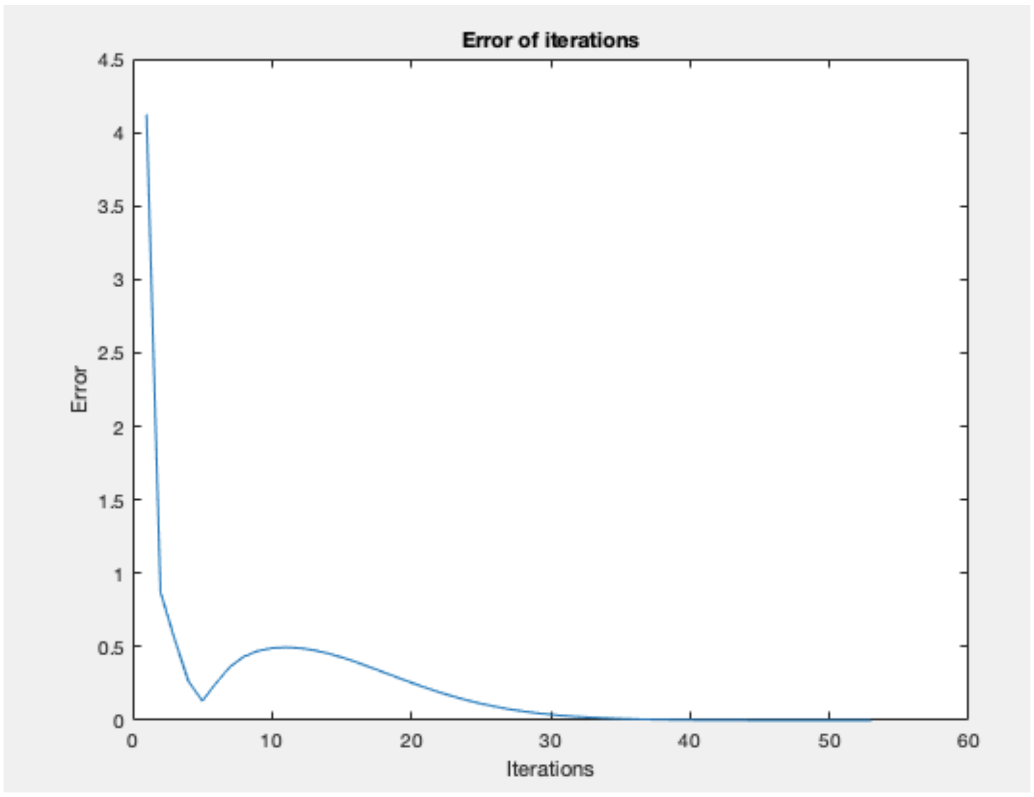


```
%Robot joints' lengths
L1 = 1; %m
L2 = 2;

%Defining our tooltip's goal, in this case its (2,2)
gx = 2;
gy = 2;
g = [gx;gy];

%Initialize some position the robot has in the beginning
theta1 = pi/2;
theta2 = pi/2;
theta = [theta1;theta2]

%Size of step
beta = 0.2;

%Initialize delta_e to something
delta_e = [Inf;Inf];

%Iterator to keep count of iterations
iter = 0

%For task 2 plotting
iters = [];
errors = [];
```

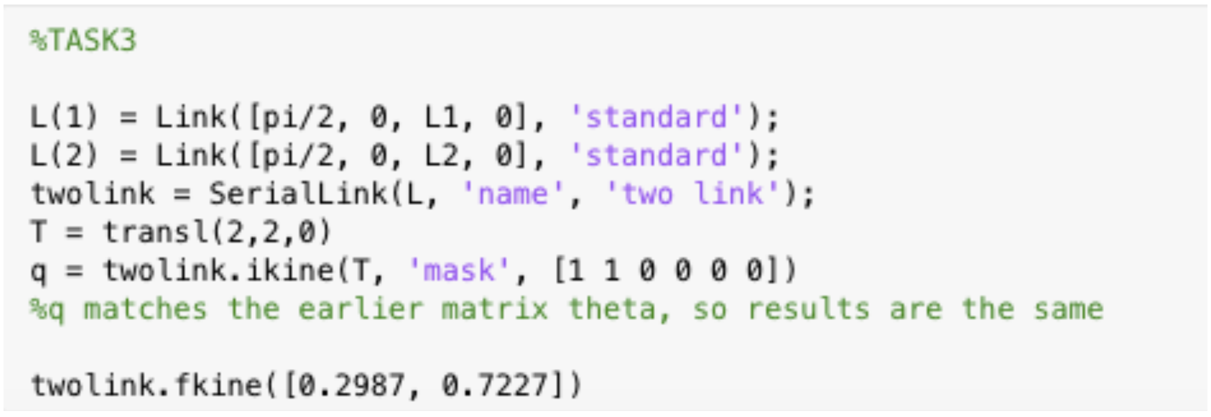
In the second task, we were told to plot the error of numerical approximation during the algorithm. Below are snippets of the code and the graph.



```
%TASK2

%Final angles:
theta

figure
plot(iters,errors)
title("Error of iterations")
xlabel("Iterations")
ylabel("Error")

%Final position of the tool tip, which maches our set goal
e
```



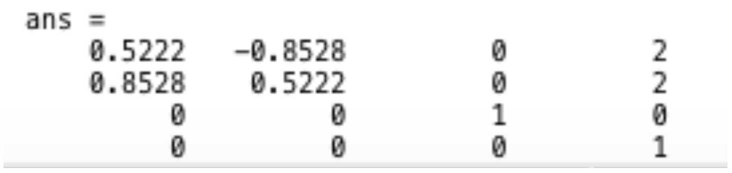
Which concludes the second part of our project. In the third task we used Corke's robotics toolbox to model our robot and solve the inverse kinematics using ikine()-method. Below is my code.



```
%TASK3

L(1) = Link([pi/2, 0, L1, 0], 'standard');
L(2) = Link([pi/2, 0, L2, 0], 'standard');
twolink = SerialLink(L, 'name', 'two link');
T = transl(2,2,0)
q = twolink.ikine(T, 'mask', [1 1 0 0 0 0])
%q matches the earlier matrix theta, so results are the same

twolink.fkine([0.2987, 0.7227])
```

Results

My algorithm gave the results theta1 = 0.2987 (rad) and theta2 = 0.7227. Altogether, this took 57 iterations. Robotics toolbox's ikine()-method gave the same answers for theta. With these results I used fkine()-method to create a transformation matrix, picture of which is (once again) below.



```
ans =
    0.5222   -0.8528         0         2
    0.8528    0.5222         0         2
         0         0         1         0
         0         0         0         1
```

Analyzing & Interpreting results

My results for theta-vector seem realistic and are likely true, since the robotics toolbox gave the same exact results. I'm not sure whether the error between tooltip's coordinates and goal's is correct, since I had a bit of trouble with Euclidian distance before. The graph does look a bit funny, and I myself don't have the knowledge to tell if something like that looks to be correct. Reaching the goal took quite a few iterations (even though my first version took over 900…). If I had the time and patience, I could've iterated a better beta value for the algorithm. Also, I could've increased the delta_e value where iterations stop, but I don't know what kind of tolerance is accepted for this type of thing, so I tried to be safe.

Things to improve

A better coder would've obviously made the code a bit cleaner and simpler. One thing that bothers me is that I calculated the differentials for my Jacobian by hand. I spent quite a bit of time trying to do this with MATLAB but couldn't figure out how to do differentials with numerical values. So that's one thing to improve. I could also be a bit more proficient at using Peter Corke's robotics toolbox.

Conclusions

This was a thing that I did, I guess. Jacobian algorithm seems to work out the angles pretty nicely (at least for this specific example). Implementing the code takes a little while but after that, solving for inverse kinematics is a breeze. For analytical solutions, solving takes a long time every time and you pretty much always must come up with a new solution. As far as I know, there isn't a way to teach a computer to solve these things analytically (maybe with pattern recognition? I don't know) so by hand it is. I think I learned a lot.