ENPM662 - Fall 2022

Homework - 04

Due: 13th November 2022 Points/Weightage: 5 points

Trajectory Generation

In Homework - 03, you modeled the forward position kinematic equations of the Panda robot using the Denavit-Hartenberg representation. In most practical applications, the inverse position kinematics of robots is desired, to be able to program them to perform a specific task. However, the inverse kinematics problem can have multiple solutions and often doesn't result in closed-form solutions. In this assignment, your task is to compute the inverse position kinematics of the Panda robot to **draw a circle of radius 10 cm within 5 seconds** using the inverse velocity kinematics approach (Inverse Jacobian).

The robot's end-effector has a pen (length of 10 cm, see Fig. 1) rigidly mounted on it such that it points along the 'z-axis' of the end-effector frame (Frame {n}). Joint 3 of the robot is locked **(θ_3 = 0)** and cannot move, such that the Jacobian matrix is a square matrix of size 6x6.

Also, the robot is already moved to the configuration shown below (in Fig. 1). The pen is in contact with the wall at point 'S' and is perpendicular to it. The joint angles for this initial configuration (in rad) are,

```
q(at t=0) = [q1, q2, q4, q5, q6, q7]^T
          = [0.0, 0.0, pi/2, 0.0, pi, 0.0]^T
```

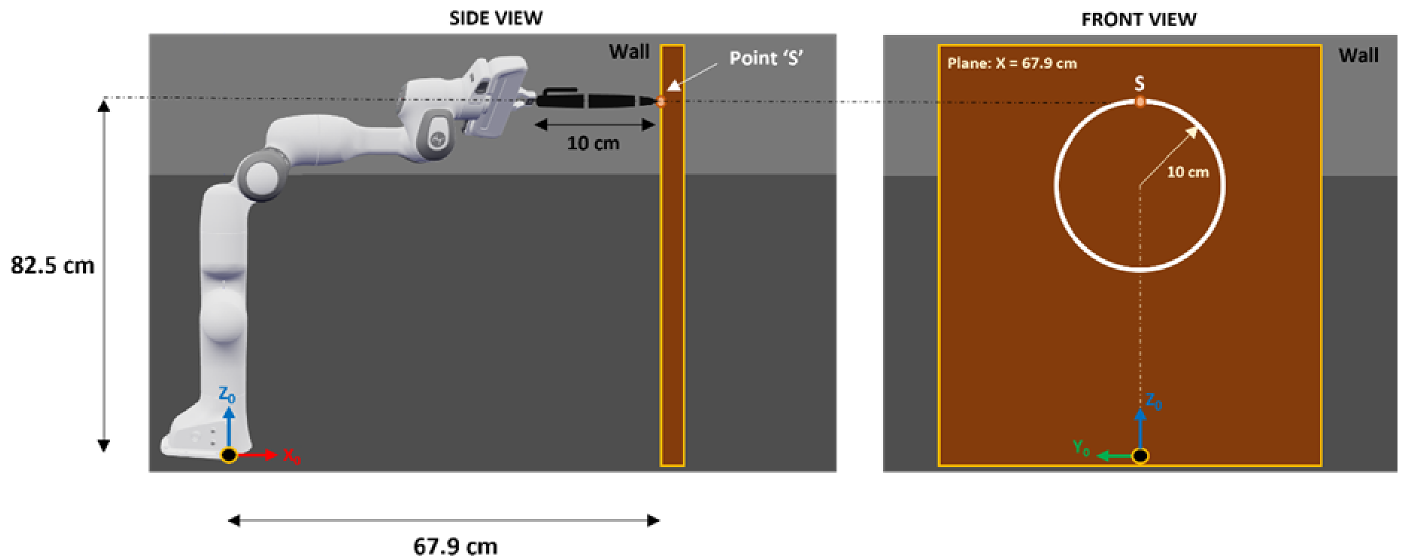

Fig. 1: Panda Robot with the pen (Side and Front Views)

Approach

- Set up the Jacobian matrix for the robot, using any ONE of the methods discussed in class (Lecture 8).
- Write down the equation for the desired circle trajectory **w.r.t. the robot's base frame.** From this equation, obtain the desired velocity trajectory of the end-effector w.r.t the base.
- Obtain the joint angular velocities for each data point on the circle using inverse velocity kinematics. Plug in the **numeric values for the '*a*' and '*d*' parameters** in the computation of the Jacobian matrix at each iteration (values are given in Fig. 2).
- Perform numerical integration on the joint angular velocities to obtain the corresponding joint angles.
- Using forward position kinematics equations, obtain the end-effector position w.r.t the base and plot this, to obtain the circle as shown in Fig. 1.

NOTE: You are NOT allowed to use inbuilt functions from any library that directly give you the Jacobian matrix and the Homogeneous transformation matrices. However, you may use inbuilt functions that assist you in setting up the above, such as taking derivatives, computing cross-products, matrix inversion (regular/pseudo-inverse), etc.

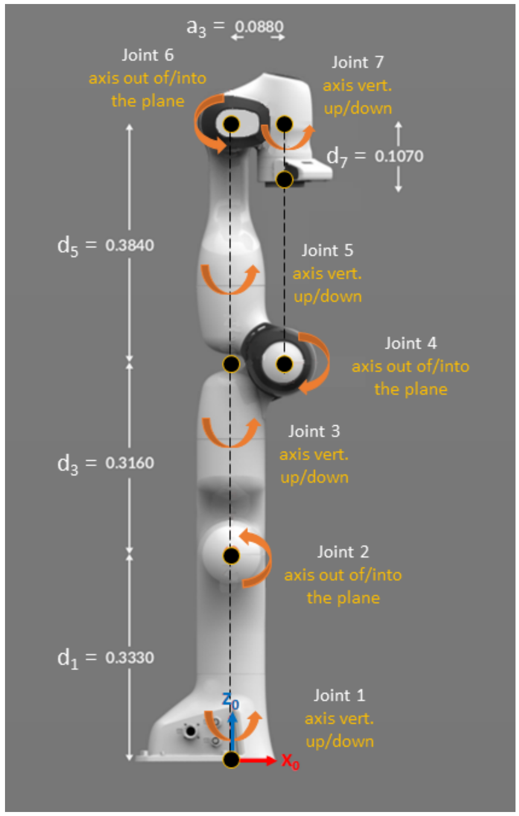

Fig. 2: Panda Robot - Home configuration

(Lengths shown are in **meters**)

Deliverables

- A **PDF** report (either hand-written or typed) containing the following. For results that are large or tedious to write manually, you may simply include a screenshot of that matrix as displayed in the terminal output after running your code.
  - The updated **D-H table** and diagram showing the robot's **D-H frames** to reflect the locked Joint 3 and the pen on the end-effector.
  - A brief explanation of how you set up the Jacobian matrix in generic form using **any one** of the two methods (from Lecture 8). Write all z and o components needed to compute the Jacobian (if using the first method) or z components and general form of partial derivatives (if using the second method).
    - Print this Jacobian matrix to the terminal.

- All the steps and equations involved in the process, from obtaining the equations of the desired circle trajectory w.r.t robot's base to obtaining the corresponding joint angles through numerical integration.
- The **picture of the plot output** of the robot's forward kinematics equations, that should be a circle as shown in Fig. 1.
    - (Optional) You may include a series of pictures at different data points or include a Drive link of a GIF/video of an animated plot.
- Name your report: ***<your-directoryID>_hw4_report.pdf*** [Note: Directory ID ≠ UID!]

- All **codes** used (Make sure the code(s) submitted run **without any errors**!)
    - The code(s) **must** print the initially set up parametric Jacobian matrix to the terminal (use *pprint* from SymPy to pretty print your matrices). **NOTE**: As you loop through plotting the different data points on the circle, you will simply plug in the joint angle values to obtain a numerical Jacobian matrix at each instant.
    - The code(s) **must** plot the computed forward kinematics end-effector X, Y, and Z coordinates (should resemble a circle)
        - Use Matplotlib to draw the circle as a series of points. See this link. Generate **a 3D scatter plot**.
        - If the plot seems elliptical, change your plot figure's axes limits such that they are equal in length.
    - Include a *readme* file that briefly describes **how** to run your code(s) and if **any dependencies** need to be installed. Use **Markdown (`.md`) format** for creating the readme file.
- Submit the above in a .zip file with the name: ***<your-directoryID>_hw4.zip***
    - Folder Structure:
        - *<your-directoryID>_hw4.zip / readme.md*
        - *<your-directoryID>_hw4.zip / <your-directoryID>_hw4_report.pdf*
        - *<your-directoryID>_hw4.zip / code —> should contain all codes (.py) used*

Supplementary Material

D-H Parameter definitions :

a_i: Distance from z_{i-1} to z_i axes along the x_i axis.

d_i: Distance from origin O_{i-1} to the intersection of the z_{i-1} and x_i axes along the z_{i-1} axis.

α_i: Angle between z_{i-1} and z_i axes about the x_i axis.

θ_i: Angle between x_{i-1} and x_i axes about the z_{i-1} axis.

Homogeneous Transformation Matrix between Links 'i' and 'i-1' :

$$T_{i-1}^{i} = \begin{bmatrix} cos\theta_i & -cos\alpha_i sin\theta_i & sin\alpha_i sin\theta_i & a_i cos\theta_i \\ sin\theta_i & cos\alpha_i cos\theta_i & -sin\alpha_i cos\theta_i & a_i sin\theta_i \\ 0 & sin\alpha_i & cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Inverse Velocity Kinematics equation:

$$\dot{q} = J^{-1}(q) * \dot{X}$$

where,

$$\dot{X} = [v_x, v_y, v_z, \omega_x, \omega_y, \omega_z]^T$$

(angular velocity components are 0 in our case)

Numerical Integration:

$$q_{next} = q_{current} + \dot{q}_{next}\Delta t$$

where,

$$\Delta t = \frac{T}{N}$$

T is the total time (sec)

N is the no. of data points to plot

Note on Singularities:

As discussed in class, the Jacobian matrix may become singular at certain values of joint angles and hence the inverse wouldn't exist. If you encounter this, you may do the following:

- Instead of taking the inverse of the Jacobian matrix, try computing the pseudo-inverse (Moore-Penrose) of the matrix (see numpy.linalg.pinv).
- In the above given initial joint angles, replace some of the '0.0' with very small values (such as 0.0001).