Indian Institute of Technology, Gandhinagar

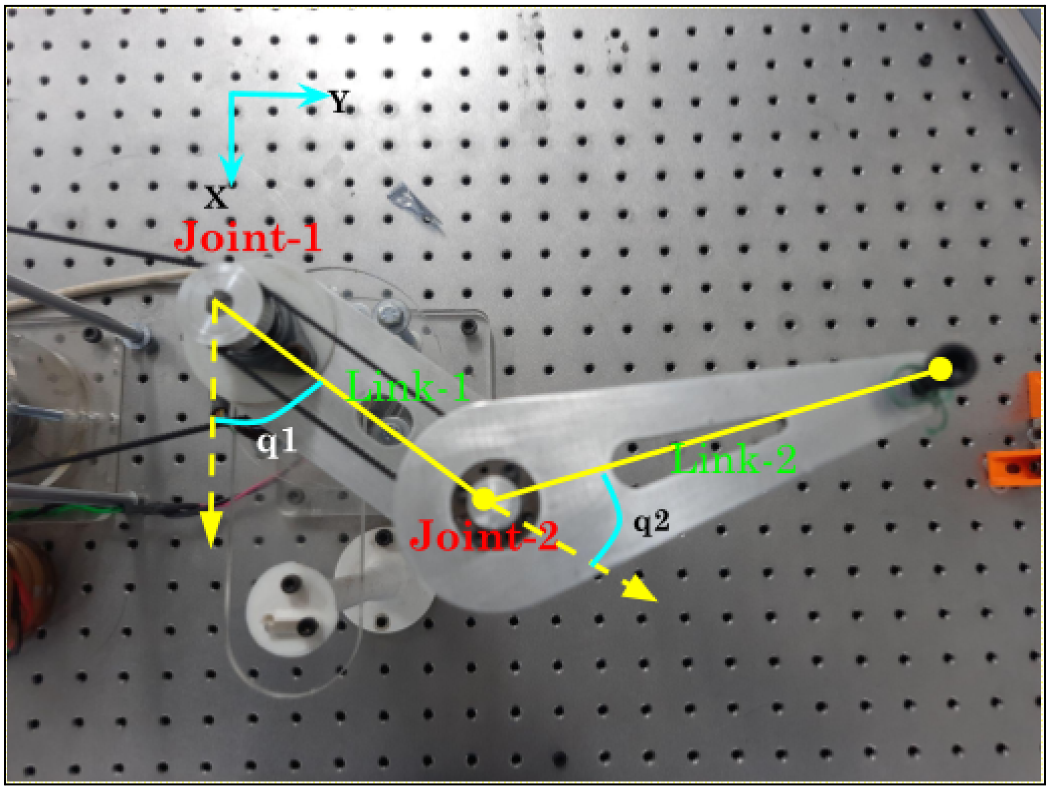
ME - 639
Introduction to Robotics

Mid-Sem Hardware Project Report

Authors:
Tejendra Patel - 19110209
Aryan Shah - 19110179
Sanskar Nalkande - 19110201
Dhruv Darda - 19110012
Kush Patel - 20110131
Ravi Dhorajia - 20110162
Harshil Safi - 20110073

Under the guidance of
Prof. Harish Palanthandalam-Madapusi

TASK-0



2R Elbow Manipulator

Known Parameters :-

Length of link 1(l_1) = 0.1m

Length of link 2(l_2) = 0.1m

Joint 1 angular velocity ($\dot{q}_1$) = 1 rad/s

Joint 1 angular velocity ($\dot{q}_2$) = 1 rad/s

DH Parameters :-

Link (i)	a_i	α_i	d_i	θ_i
1	l_1	0	0	θ_1
2	l_2	0	0	θ_2

where

θ_1 and θ_2 are variable

a_1 and a_2 are constants

Transformation Matrix :-

$$T_0^n = \begin{bmatrix} \cos(\theta_1+\theta_2) & -\sin(\theta_1+\theta_2) & 0 & l_1\cos\theta_1 + l_2\cos(\theta_1+\theta_2) \\ \sin(\theta_1+\theta_2) & \cos(\theta_1+\theta_2) & 0 & l_1\sin\theta_1 + l_2\sin(\theta_1+\theta_2) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

End Effector Position:-

$$x = l_1\cos\theta_1 + l_2\cos(\theta_1+\theta_2)$$
$$y = l_1\sin\theta_1 + l_2\sin(\theta_1+\theta_2)$$

Manipulator Jacobian:-

Linear

$$J_v = \begin{bmatrix} -l_1\, sin(\theta_1) - l_2\, sin(\theta_1+\theta_2) & -l_2\, sin(\theta_1+\theta_2) \\ l_1\, cos(\theta_1) + l_2\, cos(\theta_1+\theta_2) & l_2\, cos(\theta_1+\theta_2) \\ 0 & 0 \end{bmatrix}$$

Angular

$$J_\omega = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 1 \end{bmatrix}$$

Velocity (Linear & Angular) :-

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} -l_1\, sin(\theta_1) - l_2\, sin(\theta_1+\theta_2) & -l_2\, sin(\theta_1+\theta_2) \\ l_1\, cos(\theta_1) + l_2\, cos(\theta_1+\theta_2) & l_2\, cos(\theta_1+\theta_2) \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 1 \end{bmatrix} * \begin{bmatrix} \dot{q_1} \\ \dot{q_2} \end{bmatrix}$$

Representative Configuration:-

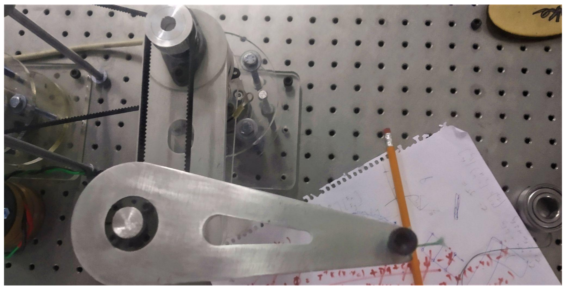
1)

Link (i)	a_i	α_i	d_i	θ_i
1	0.1	0	0	0^o
2	0.1	0	0	90^o

$$v = \begin{bmatrix} -0.1 & -0.1 \\ 0.1 & 0 \\ 0 & 0 \end{bmatrix} * \begin{bmatrix} \dot{q_1} \\ \dot{q_2} \end{bmatrix} = \begin{bmatrix} -0.2 \\ 0.1 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 0.1 \\ 0.1 \\ 0 \end{bmatrix}$$

$$J = \begin{bmatrix} -0.1 & -0.1 \\ 0.1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 1 \end{bmatrix}$$



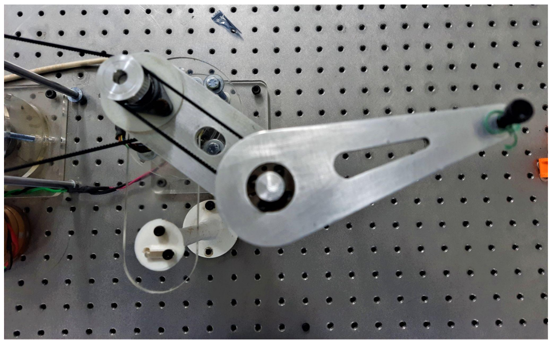
2)

Link (i)	a_i	α_i	d_i	θ_i
1	0.1	0	0	45^o
2	0.1	0	0	45^o

$$v = \begin{bmatrix} -0.171 & -0.1 \\ 0.071 & 0 \\ 0 & 0 \end{bmatrix} * \begin{bmatrix} \dot{q_1} \\ \dot{q_2} \end{bmatrix} = \begin{bmatrix} -0.271 \\ 0.071 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 0.07071 \\ 0.17071 \\ 0 \end{bmatrix}$$

$$J = \begin{bmatrix} -0.171 & -0.1 \\ 0.071 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 1 \end{bmatrix}$$



3)

Link (i)	a_i	α_i	d_i	θ_i
1	0.1	0	0	45°
2	0.1	0	0	90°

$$v = \begin{bmatrix} 0 & 0.07071 \\ 0.1414 & 0.07071 \\ 0 & 0 \end{bmatrix} * \begin{bmatrix} \dot{q_1} \\ \dot{q_2} \end{bmatrix} = \begin{bmatrix} 0.07071 \\ 0.21211 \\ 0 \end{bmatrix} \qquad \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 0.1414 \\ 0 \\ 0 \end{bmatrix}$$

$$J = \begin{bmatrix} 0 & 0.07071 \\ 0.1414 & 0.07071 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 1 \end{bmatrix}$$

Task 1:

How the code works:

In the outer loop, we assign the value to desired angles θ_1 and θ_2. These assigned values are calculated in python by making the virtual robot follow the desired trajectory and storing the values of angles. The angular velocities are also stored in a separate array along with these angles. In a loop, the value of desired angles and desired velocities are updated by iterating through the array of the stored trajectory. Then, the desired current is calculated using the following controller:

$$I_D = k_p(\theta_D - \theta) + k_d\left(\dot{\theta_d} - \dot{\theta}\right)$$

Here, θ_D is the desired value of the angle, and θ is the current value of the angle obtained from the actuators. Then this value of the desired angle is controlled by the inner loop, which was already implemented. We wrote the code for the outer loop based on the above formula.

Key results and insights:

When the robot is fed a ∞ shaped trajectory, we see that it roughly follows the trajectory, but it deviates slightly at the ends where it needs to make sharp angle changes. We predicted that this behaviour could be the result of the following.

We see that the inner loop, which has already been implemented, runs at the same frequency as the outer loop. We need the inner loop to run at a much higher frequency than the outer loop for precise results. When the motor needs to make sharp angle changes, the variations of current it faces are large. Since the variation of electric current is large, the current needs to be calculated at a higher speed than the speed at which the angles are calculated. Since we are calculating everything in one loop at the same speed, errors are induced since we have not accounted for the variation in electric current. These errors accumulate gradually, and the trajectory followed by the end effector deviates from the trajectory fed into the code.

But since the ∞ shape is symmetrical in shape, there are errors at the ends, but ultimately the end effector returns to the original position from which it started. This is clearly visible in the video.

Task 2:

In this task, we had to create a rigid surface for the robot to apply force. As seen in the picture, we implemented a virtual wall in the code and tied the robot's end effector to a load cell via a string. The load cell is connected to an HX-711 amplifier which is then connected to an Arduino Mega board. We use it to get the values of load on our pc. The load cell was first calibrated using a known weight, and then this calibration was used to measure the force values.

We first write the code for a trajectory to move the bot from the initial position to the point at which we need to apply a force. When the robot reaches the virtual surface, the string gets taut, and its motion is restricted in one direction.

When the robot completes its trajectory and reaches the desired point, we use the following equations to get the value of desired torques for the desired forces. Once we have the values for desired torque, we can calculate the desired current using the formulae given below.

We know:

$$\tau = J^T \cdot F$$

In this case:

$$\tau = \begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix}, \text{ and } F = \begin{bmatrix} F_x \\ F_y \end{bmatrix}$$

Since our robot is remotely drive, the Jacobian will be:

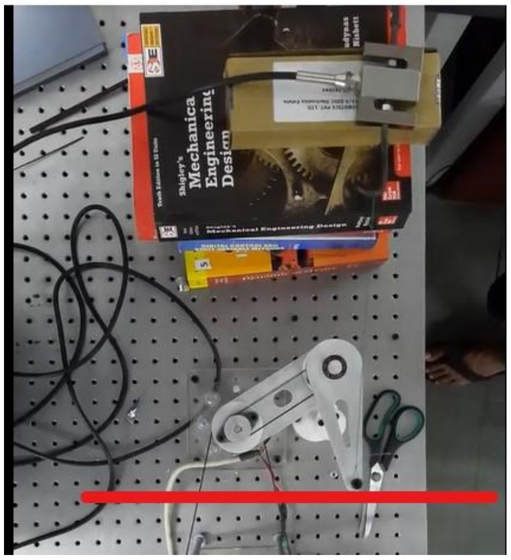
$$\text{And, } J = \begin{bmatrix} -l_1 \cdot \sin(\theta_1) & -l_2 \cdot \sin(\theta_1 + \theta_2) \\ l_1 \cdot \cos(\theta_1) & l_2 \cdot \cos(\theta_1 + \theta_2) \end{bmatrix}$$

Substituting, we get the following expressions:

$$\tau_1 = -l_1 \cdot \sin(\theta_1) \cdot F_x + l_1 \cdot \cos(\theta_1) \cdot F_y$$

$$\tau_2 = -l_2 \cdot \sin(\theta_1 + \theta_2) \cdot F_x + l_2 \cdot \cos(\theta_1 + \theta_2) \cdot F_y$$

$$I_{D1} = \frac{\tau_1}{k_t} \text{ and } I_{D2} = \frac{\tau_2}{k_t}$$



Key results and insights:

While deriving the above expression, we have ignored the forces due to friction. But in our case, the frictional forces are not negligible and significantly impact our readings. We were unable to model friction appropriately to incorporate that into our equations. Hence our desired value of force and output value of force from the load cell differ.

We put $|F_x|$= 0 N and $|F_y|$= 5 N. But using a load cell, the measured value of output force was about 3 N. This difference in results is due to the friction that was not incorporated into our equations of motion.

Task 3:

We need to make the end effector act like it is attached to the end of a spring with the neutral position at (x_0,y_0). Hence at this position, the virtual forces acting should be given by the formula:

$F_x = k(x - x_0)$

$F_y = k(y - y_0)$

Here, x and y are coordinates of tip of end effector which are given by the expression:

$x = l_1 \cdot \cos(\theta_1) + l_2 \cdot \cos(\theta_1 + \theta_2)$ and $y = l_1 \cdot \sin(\theta_1) + l_2 \cdot \sin(\theta_1 + \theta_2)$

Hence we have the expressions for F_x and F_y :

$F_x = k(l_1 \cdot \cos(\theta_1) + l_2 \cdot \cos(\theta_1 + \theta_2)\ - x_0)$

$F_y = k(l_1 \cdot \sin(\theta_1) + l_2 \cdot \sin(\theta_1 + \theta_2) - y_0)$

We use the above expressions to calculate the forces and use these forces to calculate the torques in each iteration using the formulae in task 2. Once we have the torques for each position, we calculate the desired current using the formula: $\tau = k_t \cdot I_D$

Now that we have the values of desired current, we calculate these values in outer loop and feed these values into the inner loop. The inner loops makes the microcontroller provide the required current and the end tip behaves like a spring

Key results and insights:

When the end effector reaches close enough to the neutral point, the required torque become very small hence the current also becomes small. But there is also static friction present in the apparatus. Hence, at these small currents, the motor becomes unable to move the links. Hence the end effector remains at its position which is close to the neutral point but is not precisely at the neutral point.

Another effect of the friction is that it acts as a damper on the spring. Hence the spring does not oscillate about the point but it damps down and comes at rest at a point close enough to the neutral point for the frictional force to dominate. We can clearly see this type of behavior in the video.