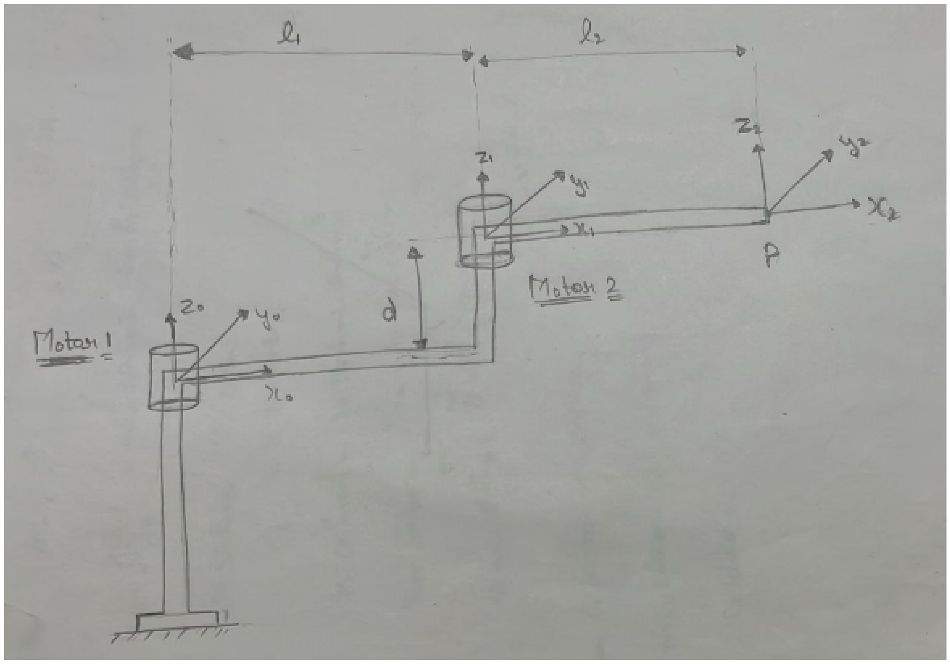
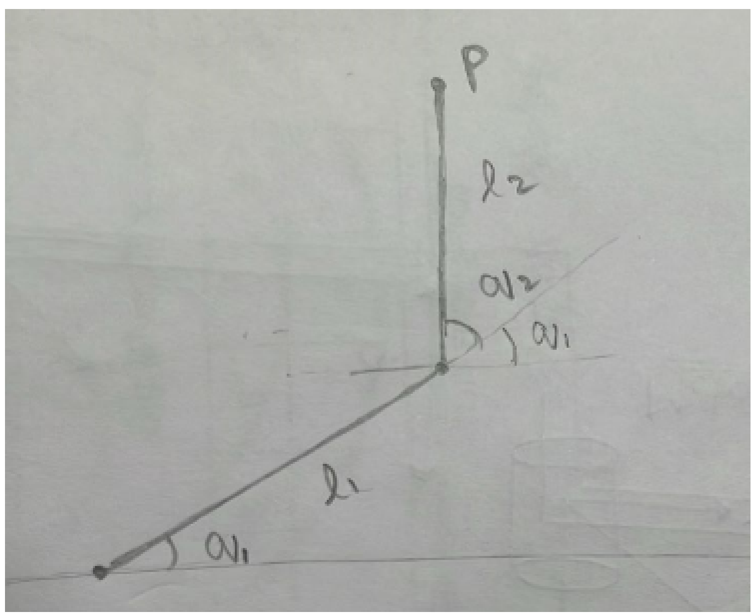
ME 639 - Introduction to Robotics
Mini Project II
Prof: Harish Palanthandalam Madapusi

Group Members:
Vedant Barde
Naman Varshney
Pankaj Khatti

Task0:



Side view of OSAKE



Top View of OSAKE

- **Essential Parameters:**
  - Length of link 1(L1) = 10 cm
  - Length of link 2(L2) = 11.5 cm
  - The interference d = 6.5 cm

- **DH Parameters:**

d (cm)	**θ (degree)**	**a (cm)**	**α (degree)**
6.5	$q1^*$	10	0
0	$q2^*$	11.5	0

Where

d is the translation along the z-axis to reach the common normal.

θ is rotation about the z-axis until the x-axis hits the common normal.

α is rotation about the x-axis until the z-axis becomes the rotation axis.

a is the translation along the x-axis to hit the common normal.

- **Resulting Homogeneous Transformation Matrix:**

$$H_0^2 = \begin{bmatrix} \cos(q1+q2) & -\sin(q1+q2) & 0 & L2.\cos(q1+q2) + L1.\cos(q1) \\ \sin(q1+q2) & \cos(q1+q2) & 0 & L2.\sin(q1+q2) + L1.\sin(q1) \\ 0 & 0 & 1 & d \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- **Jacobian Matrix:**

$$J = \begin{bmatrix} -L2.\sin(q1+q2)-L1.\sin(q1) & -L2.\sin(q1+q2) \\ L2.\cos(q1+q2)+L1.\cos(q1) & L2.\cos(q1+q2) \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 1 \end{bmatrix}$$

➢ Mathematical Calculation for Obtaining Jacobian Matrix:

$$R_0^1 = \begin{bmatrix} cq_1 & -sq_1 & 0 \\ sq_1 & cq_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad d_0^1 = \begin{bmatrix} l_1 cq_1 \\ l_1 sq_1 \\ d \end{bmatrix}$$

$$R_1^2 = \begin{bmatrix} cq_2 & -sq_2 & 0 \\ sq_2 & cq_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad d_1^2 = \begin{bmatrix} l_2 cq_2 \\ l_2 sq_2 \\ 0 \end{bmatrix}$$

$$H_0^1 = \begin{bmatrix} cq_1 & -sq_1 & 0 & l_1 cq_1 \\ sq_1 & cq_1 & 0 & l_1 sq_1 \\ 0 & 0 & 1 & d \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$H_1^2 = \begin{bmatrix} cq_2 & -sq_2 & 0 & l_2 cq_2 \\ sq_2 & cq_2 & 0 & l_2 sq_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$H_0^2 = H_0^1 H_1^2$$

$$= \begin{bmatrix} cq_1 & -sq_1 & 0 & l_1 cq_1 \\ sq_1 & cq_1 & 0 & l_1 sq_1 \\ 0 & 0 & 1 & d \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cq_2 & -sq_2 & 0 & l_2 cq_2 \\ sq_2 & cq_2 & 0 & l_2 sq_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$H_0^2 = \begin{bmatrix} c(q_1+q_2) & -s(q_1+q_2) & 0 & l_2 c(q_1+q_2) + l_1 cq_1 \\ s(q_1+q_2) & c(q_1+q_2) & 0 & l_2 s(q_1+q_2) + l_1 sq_1 \\ 0 & 0 & 1 & d \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$d_0^2 = \begin{bmatrix} l_2 c(q_1+q_2) + l_1 cq_1 \\ l_2 s(q_1+q_2) + l_1 sq_1 \\ d \end{bmatrix}$$

J_0^n where n is no. of joints ($6 \times n$ matrix)

For OSAKE, $n = 2$

$$J_0^2 = \begin{bmatrix} R_0^0 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \times (d_2^0 - d_0^0) & R_1^0 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \times (d_2^0 - d_1^0) \\ R_0^0 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} & R_1^0 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \end{bmatrix}$$

R_0^0 = identity ~~de~~ matrix

d_0^0 = zero matrix

$$J_0^2 = \begin{bmatrix} -l_2 s(q_1+q_2) - l_1 sq_1 & -l_2 s(q_1+q_2) \\ l_2 c(q_1+q_2) + l_1 cq_1 & l_2 c(q_1+q_2) \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 1 \end{bmatrix}$$

- **Python code giving end effector position with respect to the base frame.**

```
import numpy as np
import math

q1 = math.degrees(float(input("Rotation 1: ")))
q2 = math.degrees(float(input("Rotation 2: ")))
l1 = int(input("Length of Link 01: "))
l2 = int(input("Length of Link 02: "))

H01 = [[np.cos(q1), -np.sin(q1), 0, l1*np.cos(q1)],
       [np.sin(q1), np.cos(q1), 0, l1*np.sin(q1)],
       [0, 0, 1, 0],
       [0, 0, 0, 1]]

H12 = [[np.cos(q2), -np.sin(q2), 0, l2*np.cos(q2)],
       [np.sin(q2), np.cos(q2), 0, l2*np.sin(q2)],
       [0, 0, 1, 0],
       [0, 0, 0, 1]]
H02 = np.dot(H01, H12)

P1 = [[0],
      [0],
      [0],
      [1]]

P00 = np.dot(H02, P1)

print(P00)
```

- **Python code giving resulting end effector velocity using Jacobian Matrix (J).**

```
import math

theta1 = input(float("Rotation Angle 1: "))
theta2 = input(float("Rotation Angle 2: "))

# Define a time variable that will get changed that many times as the
loop containing the code runs and time proceeds
#For now, I have taken a fixed value of t to be 2 sec
t = 2.0;

# Define functions for desired joint angles as a function of time
def theta1_desired(t):
    # I am considering that the manipulator will rotate by 3 degrees
per second, which gives me the following relation for theta2
    theta1 = 3.0 * t
    return theta1

def theta2_desired(t):
# For tracing the circle, link 2 needs to be at a fixed angle, which I
assumed to be 60 degrees
    theta2 = 60
    return theta2

# Defining link lengths
l1 = 10   # length of link 1
l2 = 11.5   # length of link 2

#Defining Joint Angles
delta_t = 1e-6   # Small time interval for numerical differentiation
theta1_dot = (theta1_desired(t + delta_t) - theta1_desired(t)) /
delta_t
theta2_dot = (theta2_desired(t + delta_t) - theta2_desired(t)) /
delta_t


# Calculate tip position in the x-direction
```

```python
px = l1 * math.cos(theta1) + l2 * math.cos(theta1 + theta2)

# Calculate tip position in the y-direction
py = l1 * math.sin(theta1) + l2 * math.sin(theta1 + theta2)

# Calculate partial derivatives
a11 = -l1 * math.sin(theta1) - l2 * math.sin(theta1 + theta2)
a12 = -l2 * math.sin(theta1 + theta2)
a21 = l1 * math.cos(theta1) + l2 * math.cos(theta1 + theta2)
a22 = l2 * math.cos(theta1 + theta2)

# Calculate the Jacobian matrix
J = [[a11, a12], [a21, a22]]

# Define the angular velocity vector [theta1_dot, theta2_dot]
angular_velocity = [[theta1_dot],
                    [theta2_dot]]

# Calculate the end-tip linear velocity in x and y
linear_velocity = [[a11 * theta1_dot + a12 * theta2_dot],
                   [a21 * theta1_dot + a22 * theta2_dot]]

# Calculating the resulting end-tip angular velocity
end_tip_angular_velocity = math.sqrt(linear_velocity[0] ** 2 +
linear_velocity[1] ** 2)

# Print the result
print(f"End-tip angular velocity: {end_tip_angular_velocity}")
```

Link for LinkedIn Submission:
https://www.linkedin.com/feed/update/urn:li:activity:7114668876956676096/