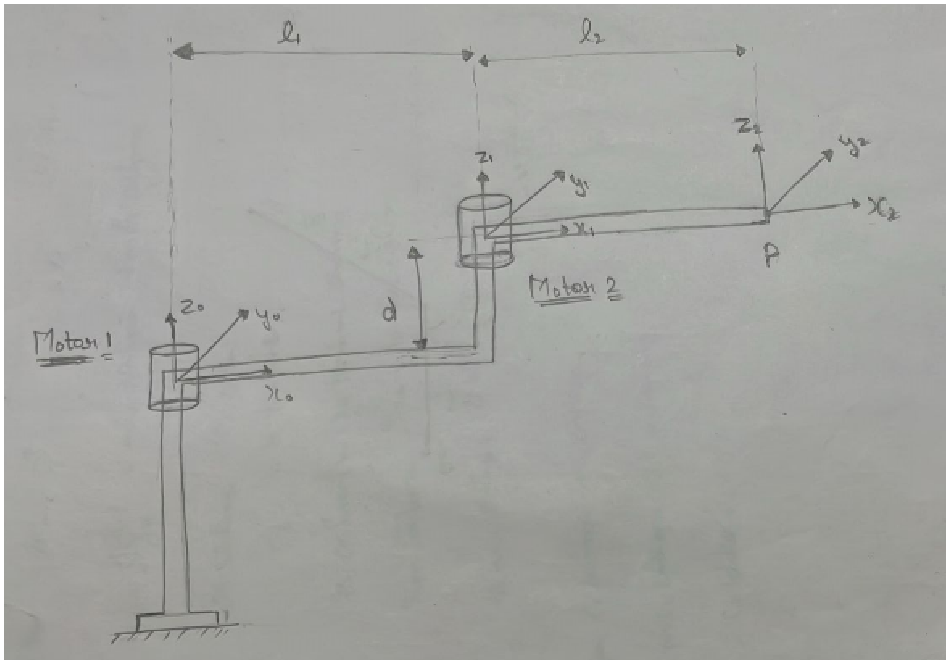
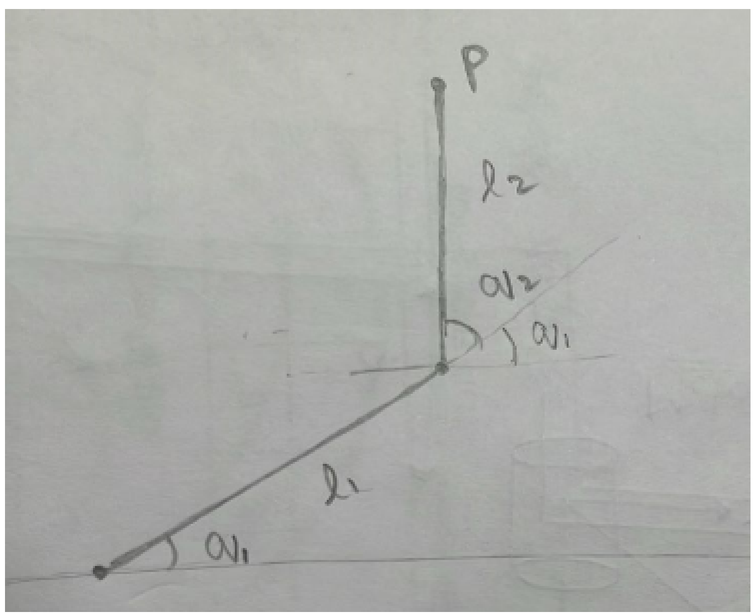
ME 639 - Introduction to Robotics
Mini Project II
Prof: Harish Palanthandalam Madapusi

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

$$J_0^2 = \begin{bmatrix} -l_2 S(q_1+q_2) - l_1 Sq_1 & -l_2 S(q_1+q_2) \\ l_2 C(q_1+q_2) + l_1 Cq_1 & l_2 C(q_1+q_2) \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 1 \end{bmatrix}$$

➢ **C++ code giving end effector position with respect to the base frame.**

```
#include <iostream>
#include <cmath>
#include <vector>

int main() {
    double q1_degrees, q2_degrees;
    int l1, l2;

    std::cout << "Rotation 1: ";
    std::cin >> q1_degrees;
    std::cout << "Rotation 2: ";
    std::cin >> q2_degrees;
    std::cout << "Length of Link 01: ";
    std::cin >> l1;
    std::cout << "Length of Link 02: ";
    std::cin >> l2;

    double q1 = q1_degrees * M_PI / 180.0; // Convert degrees to
radians
    double q2 = q2_degrees * M_PI / 180.0;

    std::vector<std::vector<double>> H01 = {
        {cos(q1), -sin(q1), 0, l1 * cos(q1)},
        {sin(q1), cos(q1), 0, l1 * sin(q1)},
        {0, 0, 1, 0},
        {0, 0, 0, 1}
    };

    std::vector<std::vector<double>> H12 = {
        {cos(q2), -sin(q2), 0, l2 * cos(q2)},
        {sin(q2), cos(q2), 0, l2 * sin(q2)},
        {0, 0, 1, 0},
        {0, 0, 0, 1}
    };
```

```cpp
    std::vector<std::vector<double>> H02(4, std::vector<double>(4));

    for (int i = 0; i < 4; ++i) {
        for (int j = 0; j < 4; ++j) {
            for (int k = 0; k < 4; ++k) {
                H02[i][j] += H01[i][k] * H12[k][j];
            }
        }
    }

    std::vector<std::vector<double>> P1 = {
        {0},
        {0},
        {0},
        {1}
    };

    std::vector<std::vector<double>> P00(4, std::vector<double>(1));

    for (int i = 0; i < 4; ++i) {
        for (int j = 0; j < 1; ++j) {
            for (int k = 0; k < 4; ++k) {
                P00[i][j] += H02[i][k] * P1[k][j];
            }
        }
    }

    // Print P00
    for (int i = 0; i < 4; ++i) {
        for (int j = 0; j < 1; ++j) {
            std::cout << P00[i][j] << "\n";
        }
    }

    return 0;
}
```

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

```
#include <iostream>
#include <cmath>

// Define functions for desired joint angles as a function of time
double theta1_desired(double t) {
    // I am considering that the manipulator will rotate by 3 degrees per
second, which gives me the following relation for theta2
    return 3.0 * t;
}

double theta2_desired(double t) {
    // For tracing the circle, link 2 needs to be at a fixed angle, which
I assumed to be 60 degrees
    return 60.0;
}

int main() {
    double t = 2.0; // Fixed time value

    // Defining link lengths
    double l1 = 10.0; // length of link 1
    double l2 = 11.5; // length of link 2

    // Small time interval for numerical differentiation
    double delta_t = 1e-6;

    // Calculate desired joint angles and their derivatives
    double theta1 = theta1_desired(t);
    double theta2 = theta2_desired(t);
    double theta1_dot = (theta1_desired(t + delta_t) - theta1_desired(t))
/ delta_t;
    double theta2_dot = (theta2_desired(t + delta_t) - theta2_desired(t))
/ delta_t;

    // Calculate tip position in x-direction
    double px = l1 * cos(theta1) + l2 * cos(theta1 + theta2);
```

```cpp
    // Calculate tip position in y-direction
    double py = l1 * sin(theta1) + l2 * sin(theta1 + theta2);

    // Calculate partial derivatives
    double a11 = -l1 * sin(theta1) - l2 * sin(theta1 + theta2);
    double a12 = -l2 * sin(theta1 + theta2);
    double a21 = l1 * cos(theta1) + l2 * cos(theta1 + theta2);
    double a22 = l2 * cos(theta1 + theta2);

    // Calculate the Jacobian matrix
    double J[2][2] = {{a11, a12}, {a21, a22}};

    // Define the angular velocity vector [theta1_dot, theta2_dot]
    double angular_velocity[2] = {theta1_dot, theta2_dot};

    // Calculate the end-tip linear velocity in x and y
    double linear_velocity[2] = {a11 * theta1_dot + a12 * theta2_dot,
                                 a21 * theta1_dot + a22 * theta2_dot};

    // Calculating the resulting end-tip angular velocity
    double end_tip_angular_velocity = sqrt(linear_velocity[0] *
linear_velocity[0] + linear_velocity[1] * linear_velocity[1]);

    // Print the result
    std::cout << "End-tip angular velocity: " << end_tip_angular_velocity
<< std::endl;

    return 0;
}
```

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