ASSIGNMENT 2

1. Forward Kinematics (General l_1, l_2)

For a 2-link planar arm:

$$\begin{aligned} x(t) &= l_1 \cos(q_1(t)) + l_2 \cos(q_1(t) + q_2(t)) \\ y(t) &= l_1 \sin(q_1(t)) + l_2 \sin(q_1(t) + q_2(t)) \end{aligned}$$

These equations are valid for **any** l_1, l_2.

2. Linear Joint-Space Trajectory (General Case)

Let:

- q_{1s}, q_{2s} = start angles
- q_{1e}, q_{2e} = end angles
- T = total motion time

$$\begin{aligned} q_1(t) &= q_{1s} + \frac{t}{T}(q_{1e} - q_{1s}) \\ q_2(t) &= q_{2s} + \frac{t}{T}(q_{2e} - q_{2s}) \end{aligned}$$

Joint velocities

$$\begin{aligned} \dot{q}_1(t) &= \frac{q_{1e} - q_{1s}}{T} \\ \dot{q}_2(t) &= \frac{q_{2e} - q_{2s}}{T} \end{aligned}$$

➡️ **Velocities are constant and non-zero at start and end.**

3. Smooth Polynomial Joint-Space Trajectory (Cubic)

We use a **cubic polynomial** per joint:

$$q_i(t) = a_{0i} + a_{1i}t + a_{2i}t^2 + a_{3i}t^3$$

Boundary Conditions

$$\begin{aligned} q_i(0) &= q_{i,s} \\ q_i(T) &= q_{i,e} \\ \dot{q}_i(0) &= 0 \\ \dot{q}_i(T) &= 0 \end{aligned}$$

Solved Coefficients (General)

$$\begin{aligned} a_{0i} &= q_{i,s} \\ a_{1i} &= 0 \\ a_{2i} &= \frac{3(q_{i,e} - q_{i,s})}{T^2} \\ a_{3i} &= -\frac{2(q_{i,e} - q_{i,s})}{T^3} \end{aligned}$$

Final Expression

$$\boxed{q_i(t) = q_{i,s} + 3\frac{(q_{i,e} - q_{i,s})}{T^2}t^2 - 2\frac{(q_{i,e} - q_{i,s})}{T^3}t^3}$$

Velocity (Smooth!)

$$\dot{q}_i(t) = 6\frac{(q_{i,e} - q_{i,s})}{T^2}t - 6\frac{(q_{i,e} - q_{i,s})}{T^3}t^2$$

➡️ Zero velocity at start and end ✔️

4. End-Effector Trajectory (Using General l_1, l_2)

Substitute either linear or polynomial $q_1(t), q_2(t)$ into:

$$x(t) = l_1 \cos(q_1(t)) + l_2 \cos(q_1(t) + q_2(t))$$
$$y(t) = l_1 \sin(q_1(t)) + l_2 \sin(q_1(t) + q_2(t))$$

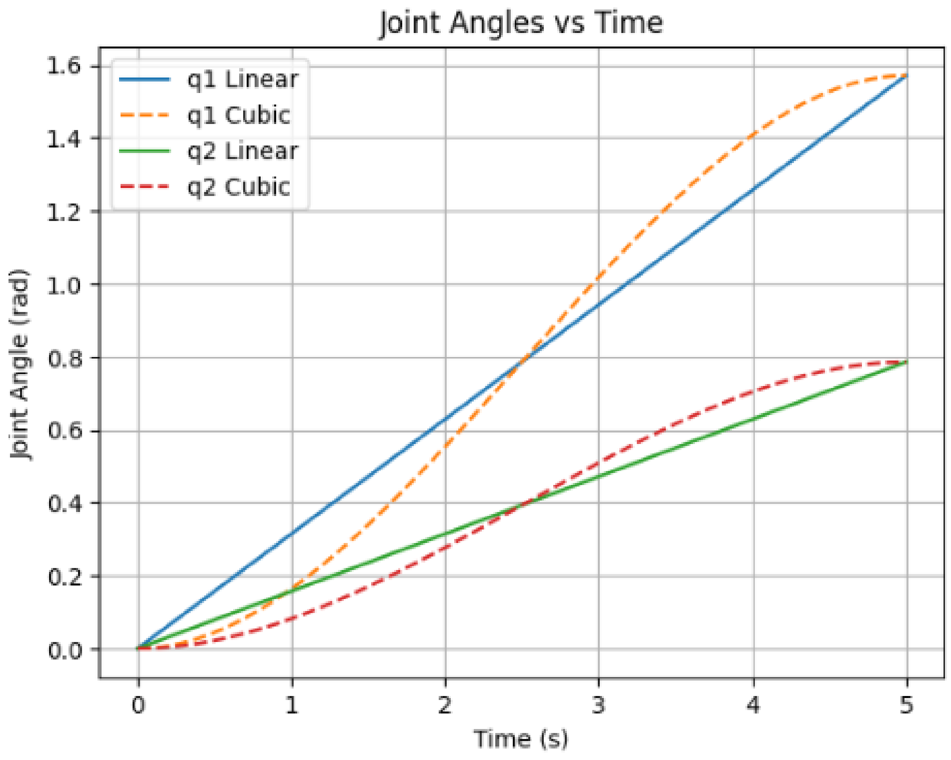
5. Python Example (Fully General)

```
import numpy as np

import matplotlib.pyplot as plt


# Parameters

l1, l2 = 1.5, 1.0

q1s, q2s = 0.0, 0.0

q1e, q2e = np.pi/2, np.pi/4

T = 5.0


t = np.linspace(0, T, 500)


# Linear trajectory

q1_lin = q1s + (q1e - q1s) * t / T

q2_lin = q2s + (q2e - q2s) * t / T


# Cubic polynomial trajectory

def cubic(qs, qe, t, T):

  return qs + 3*(qe-qs)*(t**2)/(T**2) - 2*(qe-qs)*(t**3)/(T**3)
```

```
q1_poly = cubic(q1s, q1e, t, T)
q2_poly = cubic(q2s, q2e, t, T)

# Plot joint angles
plt.figure()
plt.plot(t, q1_lin, label='q1 linear')
plt.plot(t, q1_poly, '--', label='q1 cubic')
plt.plot(t, q2_lin, label='q2 linear')
plt.plot(t, q2_poly, '--', label='q2 cubic')
plt.xlabel('Time [s]')
plt.ylabel('Joint Angle [rad]')
plt.legend()
plt.grid()
plt.show()
```

6. Short Discussion

Linear joint-space trajectories are simple to generate but produce discontinuities in joint velocity at the start and end of motion. In contrast, polynomial trajectories ensure zero velocity boundary conditions, resulting in smoother motion profiles. Smooth trajectories reduce mechanical stress, vibrations, and tracking errors in real robotic systems. Polynomial trajectories are therefore more suitable for physical robots, especially when operating at higher speeds or with sensitive payloads. While linear interpolation may be acceptable for simulation, smooth trajectories are preferred in practical applications.


Joint Angles vs Time
q1 Linear
q1 Cubic
q2 Linear
q2 Cubic
Joint Angle (rad)
Time (s)
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
0
1
2
3
4
5