Assignment 2: Joint-Space Trajectory Generation and Smoothness Analysis

Problem

2-link planar robotic arm.

Moving the robot from a start joint configuration to an end joint configuration over a fixed time T.

Start configuration: (q1_start, q2_start)

End configuration: (q1_end, q2_end)

Total motion time: T

Linear Joint- Space Trajectory

Linear interpolation: q(t) = q_start + (q_end− q_start)*t/T

Smooth polynomial (cubic) Trajectory

Cubic polynomial: q(t)=a0 +a1 t+a2 t2+a3 t3

Boundary conditions:

- $q(0) = q_{start}$
- $q(T) = q_{end}$
- $\dot{q}(0) = 0$
- $\dot{q}(T) = 0$

Coefficients

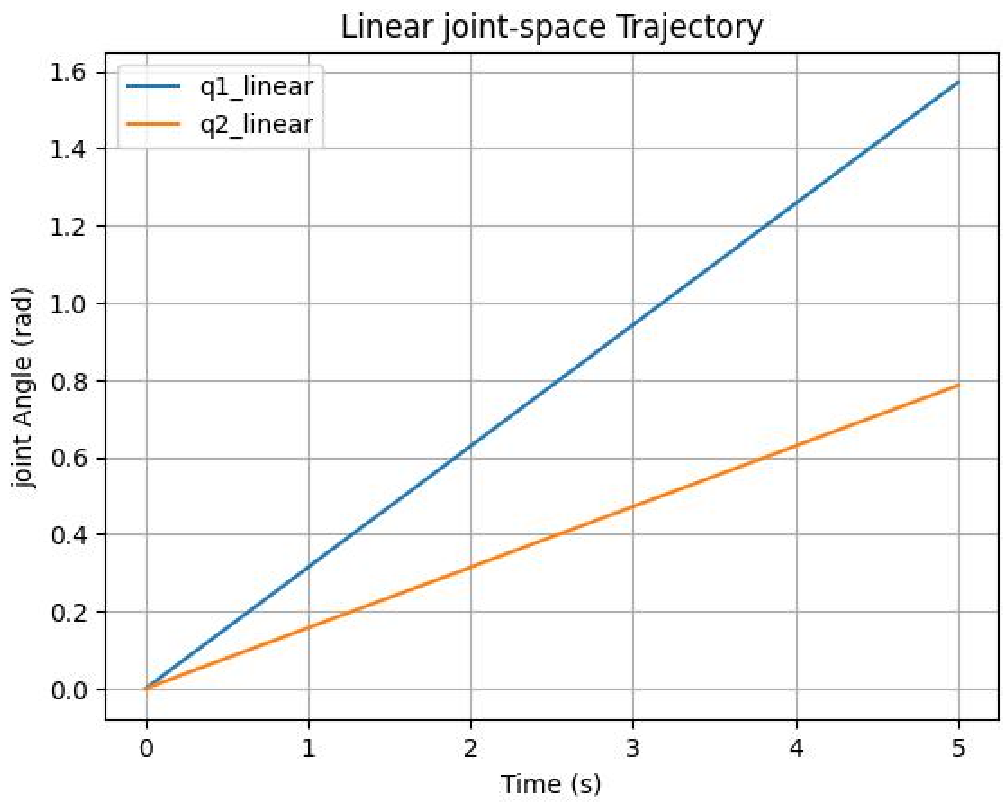
$$
\begin{aligned}
a_0 &= q_{start} \\
a_1 &= 0 \\
a_2 &= \frac{3(q_{end} - q_{start})}{T^2} \\
a_3 &= -\frac{2(q_{end} - q_{start})}{T^3}
\end{aligned}
$$

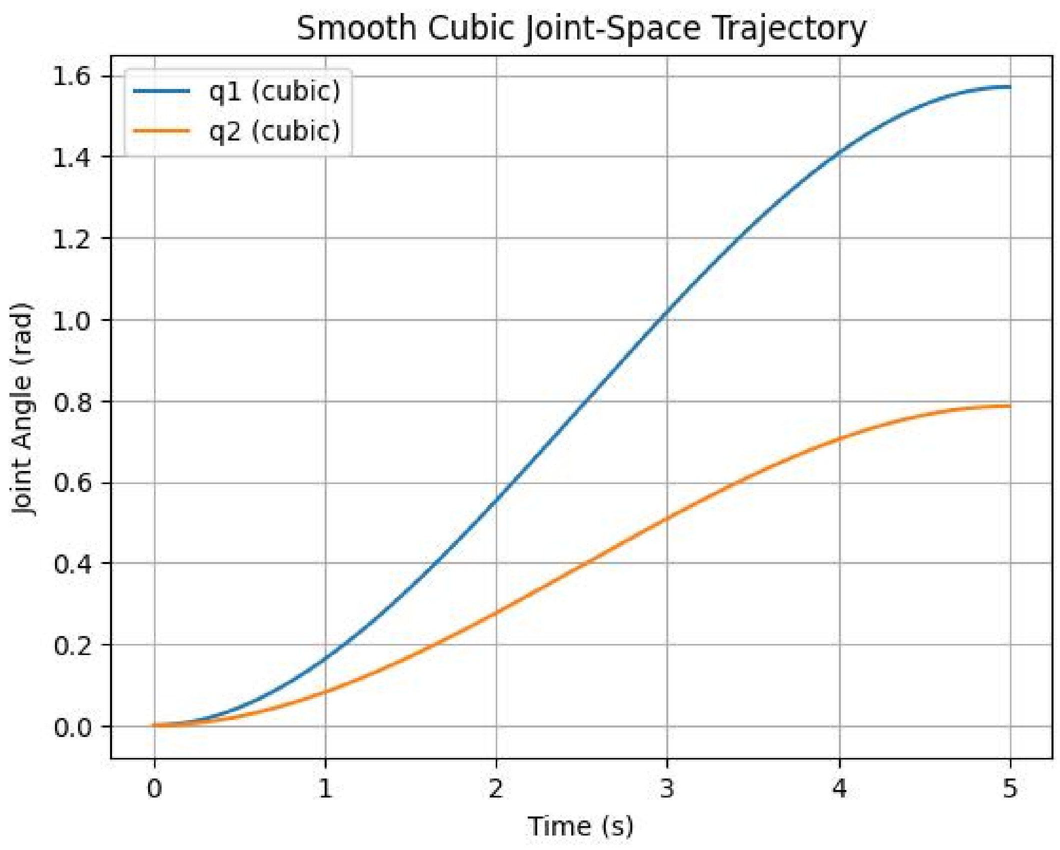
Python script:

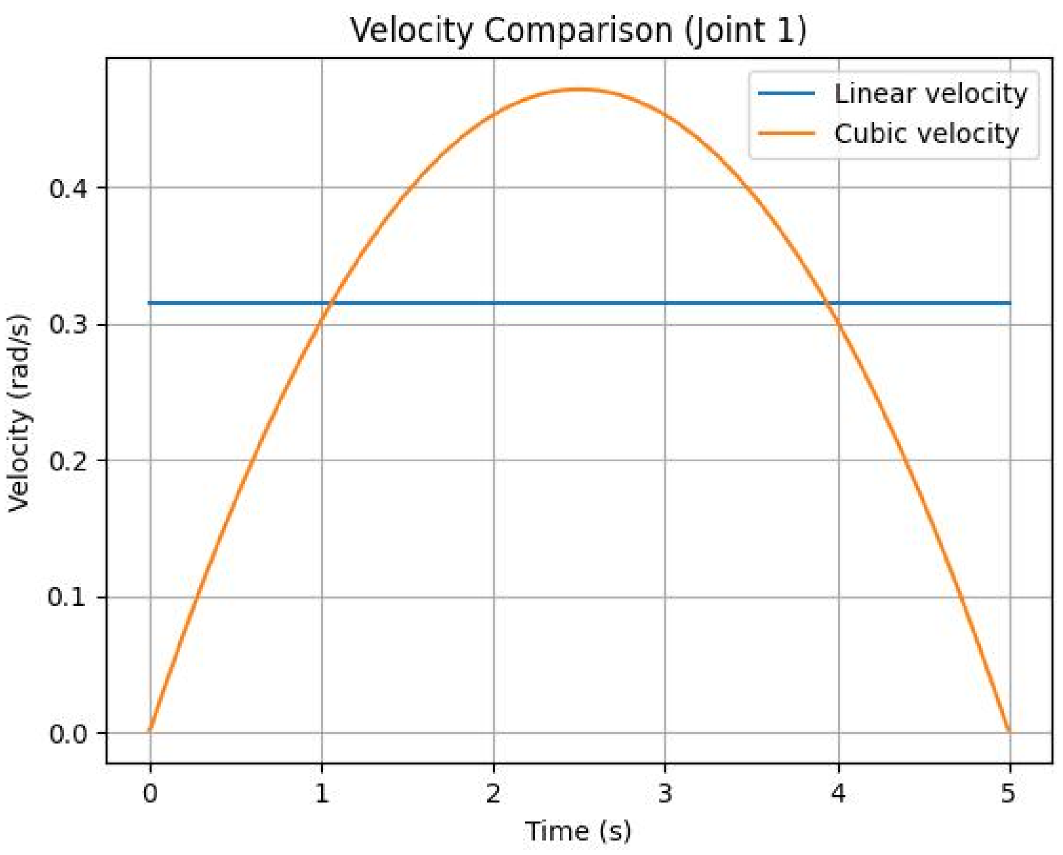
```
import numpy as np

import matplotlib.pyplot as plt


# time parameters

T = 5.0

N = 500

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


# joint start and end angles

q1_start , q1_end = 0.0, np.pi/2

q2_start , q2_end = 0.0, np.pi/4


# Linear trajectories

q1_linear = q1_start + (q1_end - q1_start) * (t / T)

q2_linear = q2_start + (q2_end - q2_start) * (t / T)


# plotting

plt.figure()
plt.plot(t, q1_linear, label='q1_linear')

plt.plot(t, q2_linear, label='q2_linear')

plt.xlabel("Time (s)")

plt.ylabel("joint Angle (rad)")

plt.title("Linear joint-space Trajectory")

plt.legend()

plt.grid()

plt.show()


"""# **Polynomial trajectory** **generation**"""


def cubic_trajectory(q_start, q_end, t, T):

  a0 = q_start

  a1 = 0

  a2 = 3 * (q_end - q_start) / (T ** 2)

  a3 = -2 * (q_end - q_start) / (T **3)

  return a0 + a1*t + a2*t**2 + a3*t**3


# cubic trajectory

q1_cubic = cubic_trajectory(q1_start, q1_end, t, T)

q2_cubic = cubic_trajectory(q2_start, q2_end, t, T)


plt.figure()

plt.plot(t, q1_cubic, label="q1 (cubic)")
plt.plot(t, q2_cubic, label="q2 (cubic)")

plt.xlabel("Time (s)")

plt.ylabel("Joint Angle (rad)")

plt.title("Smooth Cubic Joint-Space Trajectory")

plt.legend()

plt.grid()

plt.show()


# Joint velocity comparison (joint 1)

q1_linear_vel = np.gradient(q1_linear, t)

q1_cubic_vel = np.gradient(q1_cubic, t)


plt.figure()

plt.plot(t, q1_linear_vel, label="Linear velocity")

plt.plot(t, q1_cubic_vel, label="Cubic velocity")

plt.xlabel("Time (s)")

plt.ylabel("Velocity (rad/s)")

plt.title("Velocity Comparison (Joint 1)")

plt.legend()

plt.grid()

plt.show()


# Joint velocity comparison (joint 1)

q2_linear_vel = np.gradient(q2_linear, t)
```

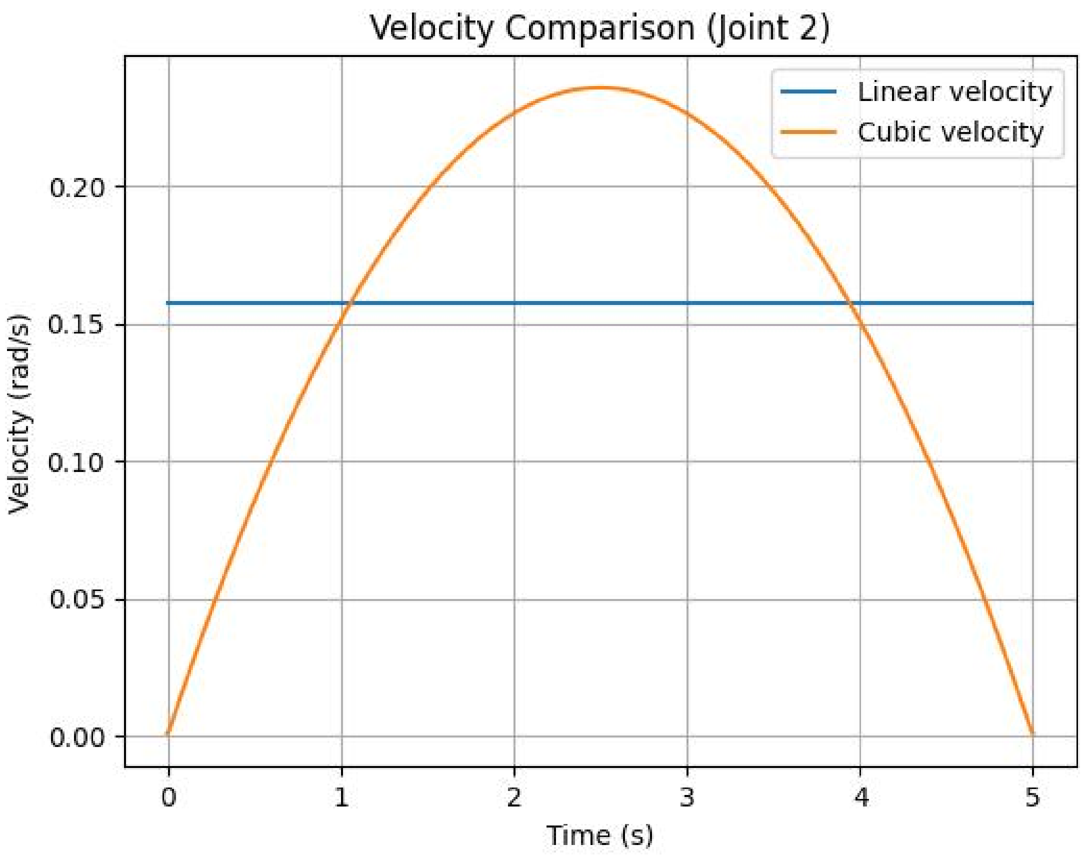
```
q2_cubic_vel = np.gradient(q2_cubic, t)


plt.figure()

plt.plot(t, q2_linear_vel, label="Linear velocity")

plt.plot(t, q2_cubic_vel, label="Cubic velocity")

plt.xlabel("Time (s)")

plt.ylabel("Velocity (rad/s)")

plt.title("Velocity Comparison (Joint 2)")

plt.legend()

plt.grid()

plt.show()
```

Plots:


Linear joint-space Trajectory
q1_linear
q2_linear
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


Smooth Cubic Joint-Space Trajectory
q1 (cubic)
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


Velocity Comparison (Joint 1)
Linear velocity
Cubic velocity
Velocity (rad/s)
0.0
0.1
0.2
0.3
0.4
0
1
2
3
4
5
Time (s)


Velocity Comparison (Joint 2)
Linear velocity
Cubic velocity
Velocity (rad/s)
0.00
0.05
0.10
0.15
0.20
0
1
2
3
4
5
Time (s)

Difference between linear and smooth Trajectory

- *Linear joint-space trajectories change joint angles at a constant rate, which results in abrupt changes in velocity at the start and end of motion.*
- *These sudden velocity changes lead to very high or infinite accelerations, making linear trajectories impractical for real robots.*
- *Smooth polynomial trajectories ensure zero velocity at the beginning and end of motion, leading to gradual acceleration and deceleration.*
- *This results in smoother joint movements and eliminates sharp discontinuities in motion.*

Importance of smoothness for real robots

- *Smoothness is important because robotic actuators have limits on velocity, acceleration, and torque.*
- *Respecting these limits reduces mechanical stress, prevents actuator overload, and improves motion accuracy and system safety.*

Jaypal (240494)