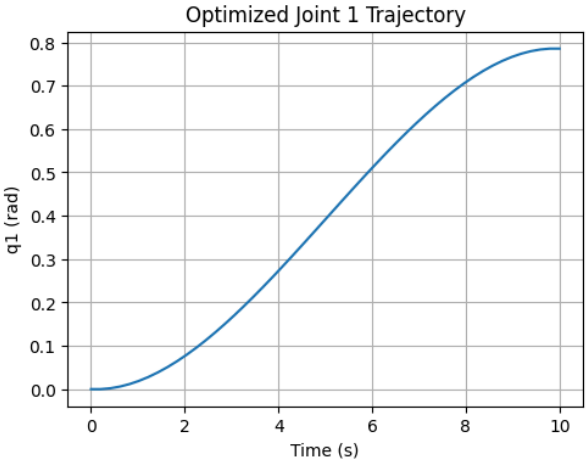
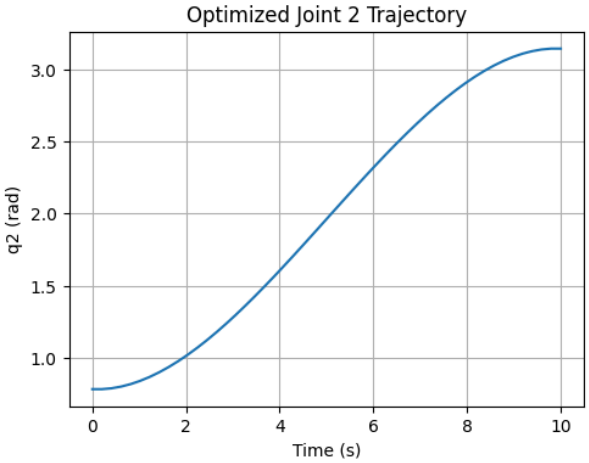
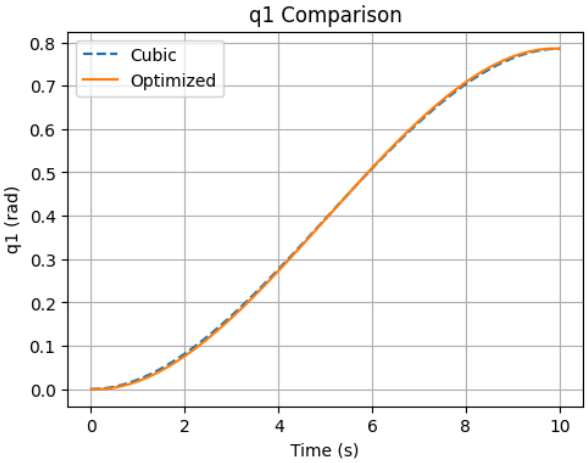
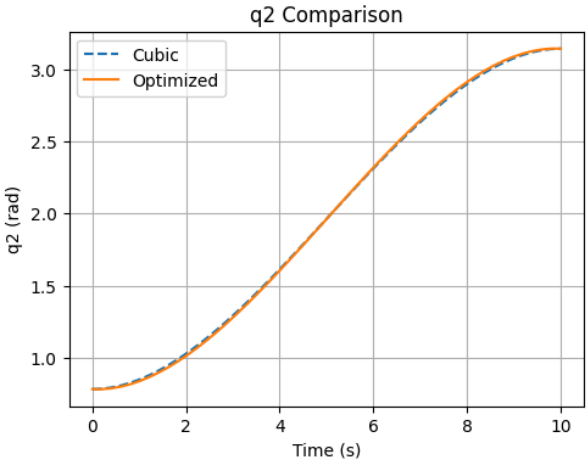
In [7]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize

# Parameters
T = 10
N = 50
dt = T / (N - 1)
q1_start, q2_start = 0, np.pi / 4
q1_end,   q2_end   = np.pi / 4, np.pi
t = np.linspace(0, T, N)
# Cost Function (Acceleration Squared)
def cost(q):
    q1 = q[:N]
    q2 = q[N:]

    q1_ddot = np.diff(q1, n=2) / dt**2
    q2_ddot = np.diff(q2, n=2) / dt**2

    return np.sum(q1_ddot**2) + np.sum(q2_ddot**2)
# Constraints
constraints = []
# Position boundary constraints
constraints += [
    {'type': 'eq', 'fun': lambda q: q[0] - q1_start},
    {'type': 'eq', 'fun': lambda q: q[N-1] - q1_end},
    {'type': 'eq', 'fun': lambda q: q[N] - q2_start},
    {'type': 'eq', 'fun': lambda q: q[2*N-1] - q2_end},
]
# Velocity boundary constraints
constraints += [
    {'type': 'eq', 'fun': lambda q: (q[1] - q[0]) / dt},
    {'type': 'eq', 'fun': lambda q: (q[N-1] - q[N-2]) / dt},
    {'type': 'eq', 'fun': lambda q: (q[N+1] - q[N]) / dt},
    {'type': 'eq', 'fun': lambda q: (q[2*N-1] - q[2*N-2]) / dt},
]
# Initial Guess (Linear Trajectory)
q1_init = np.linspace(q1_start, q1_end, N)
q2_init = np.linspace(q2_start, q2_end, N)
q_init = np.hstack((q1_init, q2_init))
# Optimization
result = minimize(cost, q_init, constraints=constraints, method='SLSQP')
q1_opt = result.x[:N]
q2_opt = result.x[N:]
# Cubic Polynomial Trajectory (Comparison)
def cubic_traj(q0, qf, T, t):
    a0 = q0
    a1 = 0
    a2 = 3 * (qf - q0) / T**2
    a3 = -2 * (qf - q0) / T**3
    return a0 + a1*t + a2*t**2 + a3*t**3

q1_cubic = cubic_traj(q1_start, q1_end, T, t)
q2_cubic = cubic_traj(q2_start, q2_end, T, t)
```

```
# Plot Optimized Trajectories
plt.figure(figsize=(12,4))
plt.subplot(1,2,1)
plt.plot(t, q1_opt)
plt.title("Optimized Joint 1 Trajectory")
plt.xlabel("Time (s)")
plt.ylabel("q1 (rad)")
plt.grid()

plt.subplot(1,2,2)
plt.plot(t, q2_opt)
plt.title("Optimized Joint 2 Trajectory")
plt.xlabel("Time (s)")
plt.ylabel("q2 (rad)")
plt.grid()
plt.show()
# Comparison Plots
plt.figure(figsize=(12,4))
plt.subplot(1,2,1)
plt.plot(t, q1_cubic, '--', label="Cubic")
plt.plot(t, q1_opt, label="Optimized")
plt.title("q1 Comparison")
plt.xlabel("Time (s)")
plt.ylabel("q1 (rad)")
plt.legend()
plt.grid()

plt.subplot(1,2,2)
plt.plot(t, q2_cubic, '--', label="Cubic")
plt.plot(t, q2_opt, label="Optimized")
plt.title("q2 Comparison")
plt.xlabel("Time (s)")
plt.ylabel("q2 (rad)")
plt.legend()
plt.grid()
plt.show()
```






q1 Comparison
Cubic
Optimized
q1 (rad)
Time (s)
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
2
4
6
8
10


q2 Comparison
Cubic
Optimized
q2 (rad)
Time (s)
1.0
1.5
2.0
2.5
3.0
0
2
4
6
8
10

The optimized trajectory moves the robot joints from the start position to the end position smoothly. The joint angle plots show smooth curves without sudden changes. Optimization reduces sharp motion by minimizing a cost function. After optimization, the motion becomes more gradual and stable. The cost value after optimization is lower than before optimization. This means the robot uses less effort and moves more smoothly. Therefore, optimization improves the overall quality of the robot’s motion.