Optimization & Linear Models in Robotics: Trajectory Optimization with Lagrange Multipliers

1. Introduction

This report details the mathematical implementation of **Unit 1: Optimization & Linear Models** from the MIS syllabus within the Two SCARA Robot Collaboration project. Specifically, we apply **Lagrange Multipliers** to the problem of **Trajectory Optimization**.

The goal is to move the robot's end-effector in a **perfectly straight line** between two points in Cartesian space, minimizing the joint velocity norms (a proxy for energy). This contrasts with the default behavior of simply interpolating joint angles, which results in curved, unpredictable paths.

2. Mathematical Foundation

2.1 The Optimization Problem

We wish to find the optimal joint velocities $\dot{\theta}$ that satisfy the Cartesian velocity requirement (moving along the line) while minimizing the magnitude of joint motion.

Objective Function (f): Minimize the squared norm of joint velocities (Least Squares).

$$f(\dot{\theta}) = \frac{1}{2}\dot{\theta}^T\dot{\theta} = \frac{1}{2}(\dot{\theta}_1^2 + \dot{\theta}_2^2)$$

Equality Constraint (g): The Forward Kinematics velocity relationship must hold.

$$g(\dot{\theta}) = J(\theta)\dot{\theta} - \mathbf{v}_{target} = 0$$

Where:

- $\dot{\theta}$ is the vector of joint velocities $[\dot{\theta}_1, \dot{\theta}_2]^T$.
- $J(\theta)$ is the Jacobian matrix.
- $\mathbf{v}_{target}$ is the desired Cartesian velocity vector $[\dot{x}, \dot{y}]^T$.

2.2 Method of Lagrange Multipliers

To solve this constrained optimization problem, we define the **Lagrangian**:

$$\mathcal{L}(\dot{\theta}, \lambda) = f(\dot{\theta}) + \lambda^T g(\dot{\theta})$$

$$\mathcal{L} = \frac{1}{2}\dot{\theta}^T\dot{\theta} + \lambda^T(J\dot{\theta} - \mathbf{v})$$

Taking the partial derivatives and setting them to zero (Necessary Conditions for Extrema):

1. **Stationarity Condition**:

$$\frac{\partial \mathcal{L}}{\partial \dot{\theta}} = \dot{\theta} + J^T\lambda = 0 \implies \dot{\theta} = -J^T\lambda$$

2. **Constraint Condition**:

$$\frac{\partial \mathcal{L}}{\partial \lambda} = J\dot{\theta} - \mathbf{v} = 0 \implies J\dot{\theta} = \mathbf{v}$$

2.3 Solving for the Optimal Control Law

Substitute (1) into (2):

$$J(-J^T\lambda) = \mathbf{v}$$

$$-(JJ^T)\lambda = \mathbf{v}$$

$$\lambda = -(JJ^T)^{-1}\mathbf{v}$$

Substitute λ back into (1) to find $\dot{\theta}$:

$$\dot{\theta} = -J^T[-(JJ^T)^{-1}\mathbf{v}]$$

$$\dot{\theta} = J^T(JJ^T)^{-1}\mathbf{v}$$

This term $J^T(JJ^T)^{-1}$ is known as the **Moore-Penrose Pseudoinverse** ($J^\dagger$). For a non-redundant SCARA robot (square Jacobian), this simplifies to the standard inverse J^{-1}, but the formulation is

general for optimization.

Final Control Equation:

$$\dot{\theta}(t) = J(\theta(t))^{-1}\mathbf{v}(t)$$

3. Implementation Details

3.1 Algorithm

We typically implement this numerically over time steps Δt:

1. Define Start Point P_{start} and End Point P_{end}.
2. Define desired velocity $\mathbf{v} = (P_{end} - P_{start})/T$.
3. Loop from $t = 0$ to T:
   a. Calculate current Jacobian $J(\theta_{current})$.
   b. Solve for expected joint velocity: $\dot{\theta} = J^{-1}\mathbf{v}$.
   c. Integrate position: $\theta_{new} \approx \theta_{current} + \dot{\theta}\Delta t$.
   d. (Correction Step): Optionally correct for drift using Inverse Kinematics.

3.2 Code Structure

We have extended `kinematics.py` to include a `generate_straight_line_path` method.

Python Implementation Snippet

```python
def generate_straight_line_path(self, start_joints, end_coords, duration, dt=0.05):
    # ... setup ...
    for t in time_steps:
        # 1. Jacobian
        J = self.jacobian(theta1, theta2)

        # 2. Desired Cartesian V (Constant for straight line)
        v_cartesian = np.array([vx, vy])

        # 3. Optimization Solution (J_inv * v)
        # using pseudo-inverse for stability
        J_inv = np.linalg.pinv(J)
        omega = np.dot(J_inv, v_cartesian)

        # 4. Integrate
        theta1 += omega[0] * dt
        theta2 += omega[1] * dt
```

4. Comparison of Results

4.1 Linear Joint Interpolation (Old Method)

- **Method**: Linearly change angles from Start to End.
- **Path**: Arcs in Cartesian space.
- **Pros**: Computationally cheap, guaranteed to reach goal if IK exists.
- **Cons**: Unpredictable path, risk of collision.

4.2 Optimized Cartesian Path (New Method)

- **Method**: Differential Kinematics with Lagrange constraints.
- **Path**: Straight line.
- **Pros**: Predictable, efficient motion, "Optimization" compliant.
- **Cons**: Requires Jacobian inversion (singularity risk).

5. Conclusion

By applying the **Necessary Conditions for Extrema** and **Lagrange Multipliers** to the kinematic constraints, we have upgraded the robot's control system from basic joint interpolation to sophisticated Cartesian Trajectory Planning. This directly fulfills the requirements of **UNIT-1: Optimization & Linear Models**.