Optimal Trajectory Planning for a Robotic Arm

Abina Abey, Ayden Sojan, Benjamin Biju, Christin Siby

October 21, 2025

Abstract

Smooth trajectory planning is crucial for robotic manipulators to perform precise and efficient motions while minimizing mechanical wear. This project presents an optimal trajectory planning algorithm for a 2-degree-of-freedom (DOF) planar robotic arm, focusing on minimizing jerk — the rate of change of acceleration — to achieve smooth motion. The approach integrates concepts from partial differential equations, linear algebra, Laplace transforms, and optimization techniques. Using numerical methods and Python implementation, the project produces and visualizes an optimized joint trajectory that respects boundary conditions without hardware requirements.

1 Introduction

Robotic arms are widely used in automation, manufacturing, and medical applications, where smooth and precise movement is essential. Trajectory planning involves computing joint movements that guide the end-effector from a start to an end position efficiently and safely. High jerk in trajectories can cause vibrations, increased wear, and instability in control. Minimizing jerk ensures smoothness, resulting in natural and physically feasible motions. This project addresses trajectory planning for a simple 2-DOF planar robotic arm. The arm's motion is represented as functions of time joint angles, and the goal is to minimize the total jerk throughout the trajectory, while meeting the fixed start and end positions. The problem is formulated mathematically and solved via constrained numerical optimization. The final trajectories are visualized to validate smoothness and correctness. Once you're familiar with the editor, you can find various project settings in the Overleaf menu, accessed via the button in the very top left of the editor. To view tutorials, user guides, and further documentation, please visit our help library, or head to our plans page to choose your plan.

2 Methodology and System Modeling

The project is divided into five main modules,each handling a specific component of the trajectoryplanning process.

2.1 Setup and Imports

The first module lays the groundwork by importing the essential Python libraries. NumPy is used for efficient numerical computations, providing functions to manipulate arrays and perform mathematical operations required for representing trajectories and calculating derivatives. SciPy provides advanced optimization routines, including the Sequential Least Squares Programming (SLSQP) algorithm used here to solve the constrained optimization problem. Matplotlib is employed for plotting and visualizing the robotic arm's motion, allowing the user to see the effectiveness of the planned trajectory. Properly importing these libraries ensures that subsequent modules can perform mathematical operations, optimization, and visualization seamlessly.

2.2 Forward Kinematics

Forward kinematics transforms the robot's joint angles into the physical position of the end-effector in Cartesian coordinates. For the 2-DOF planar arm, given the two joint angles and the lengths of 1 the two links, this module calculates the precise x and y positions of the arm tip at each time step using trigonometric relationships. This is crucial for visualizing the arm's movement over time and verifying that the planned trajectory results in desired end-effector paths. Accurate forward kinematics also allows checking if the trajectory respects workspace constraints and helps interpret the physical meaning of the joint angle solutions produced by the optimizer.

2.3 Minimum Jerk Cost Function

The heart of the trajectory optimization lies in this module, which defines the objective function to be minimized. The jerk, defined as the third derivative of joint angles with respect to time, quantifies the smoothness of the motion. High jerk corresponds to sudden changes in acceleration that can cause mechanical shocks and vibrations. Using finite

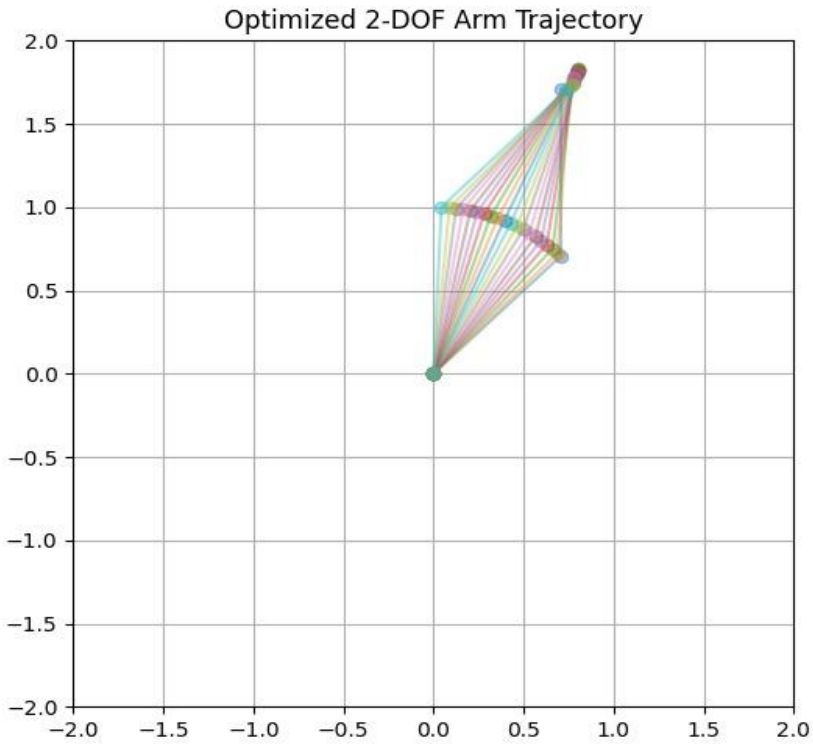
difference approximations, this module calculates jerk at each discretized time point for both joints. It then computes the sum of the squares of these jerk values across all time steps, forming a scalar cost value. Minimizing this cost encourages smooth trajectories with gradual acceleration changes, enhancing motion quality and robot lifespan.

2.4 Trajectory Constraints

Realistic trajectory planning must ensure that the robot begins and ends at desired configurations. This module implements equality constraints on the optimization problem that fix the initial and final joint angles to prescribed values. These constraints guaranty that the optimized trajectory starts exactly at the initial joint configuration and reaches the desired final configuration at the end of the time horizon. Without these constraints, the optimizer could produce mathematically smooth trajectories that do not meet practical positional requirements, making the solution unusable for real-world robotic tasks.

2.5 Optimization and Visualization

This module combines all previous components to execute the optimization process. It sets an initial guess for the joint trajectories, usually a simple linear interpolation between start and end angles, which guides the solver towards a feasible solution. Using the SLSQP algorithm, the jerk cost function is minimized subject to boundary constraints. After convergence, the module extracts the optimized joint trajectories and applies forward kinematics to compute arm positions over time. Finally, it visualizes the robot arm at multiple time steps using Matplotlib, providing a clear graphical representation of the smooth motion achieved. This visualization confirms the success of the optimization and facilitates qualitative assessment of the trajectory.



Result for the Graph

The graph showing the step response of the uncontrolled system clearly indicates an underdamped behavior. It exhibits a large overshoot and sustained oscillations, meaning the robotic arm's structure vibrates significantly when disturbed. This confirms that active vibration control is necessary to achieve stability and precision during operation.

Conclusion

The study successfully developed a mathematical and simulation framework for analyzing and controlling vibrations in a flexible robotic arm. The state-space and modal analyses accurately captured the system's natural vibration modes, while the Laplace-based control design verified the need for damping through a PD controller. Although the PDE simulation experienced numerical instability, it provided valuable insights into the limitations of explicit schemes. Overall, the project established a

strong foundation for future development of an adaptive vibration suppression system with improved numerical stability and optimized control gains.