Polynomial-Based Path Planning with Direct Jerk and Snap Control

A Real-Time Control Approach for Smooth, Stable, and Adaptive Movement

Paul Rosu

January 13, 2025

Abstract

This paper presents a novel approach to cursor movement control using polynomial-based path planning with direct jerk and snap control. We develop a real-time control system that achieves stable, smooth cursor movement without overshoots while maintaining reasonable speed. The system implements two distinct controllers: a third-order controller with direct jerk control and a fourth-order controller with direct snap control. Both controllers utilize adaptive time horizons and dynamic scaling to achieve precise pointer movement while maintaining stability and smoothness. A MATLAB-based plant simulator, incorporating a graphical mouse pointer control demo, is used for rapid prototyping, development, and testing. Finally, a full implementation in Qt/C++ provides real-time visualization, data logging, and automatic trial sequencing for comprehensive performance analysis.

1 Introduction

Human-machine interaction and robotic motion planning commonly require smooth trajectories that minimize mechanical stress, reduce abrupt changes in velocity or acceleration, and avoid undesirable oscillations or overshoot. One widely researched principle for such requirements involves the generation of so-called "minimum jerk" trajectories, popularized by studies in robotics and biological motion [1]. Here, the jerk (the third derivative of position) is controlled or constrained to achieve smooth motion. An extension to this concept is "minimum snap" planning, where the fourth derivative of position (snap) is also constrained or minimized to obtain even smoother profiles.

In this work, we adopt a polynomial trajectory planning method that directly solves for the jerk and snap signals to be applied at each control step. Rather than enforcing only minimal jerk or snap indirectly, we explicitly compute these higher-order derivatives of the position in real time. Consequently, the resulting control inputs (jerk or snap) drive a cursor or end-effector to the target with minimal overshoot and smooth, continuous motion. Unlike typical minimum-jerk or minimum-snap algorithms that solve for position profiles offline,

we employ adaptive time horizons, dynamic error scaling, and careful smoothing to handle random initial conditions and real-time demands.

This paper is organized as follows. Section 2 highlights related work and our motivation in adopting polynomial-based control. Section 3 presents the MATLAB plant simulator and its mouse pointer control demo, along with mathematical details of the plant's modeling. Section 4 describes the theoretical underpinnings for the third-order (jerk) and fourth-order (snap) controllers, focusing on the polynomial solver frameworks. Section 6 outlines our real-time implementation in Qt/C++ and the data recording features. Section 7 summarizes the main performance outcomes, while Section 8 draws final conclusions and suggests future directions.

2 Background and Motivation

The concept of minimizing the jerk has been used in industrial robotics and human arm trajectory modeling to ensure smoother, more natural motion trajectories [1]. In many applications, such smoothness is crucial to reduce mechanical wear, increase operator comfort, or maintain system stability under real-time constraints.

Our focus here diverges from purely "minimum-jerk" path planning in that we apply jerk (and, in a more advanced mode, snap) directly as the control input. We address a real-time scenario with unknown or randomized initial states and dynamic targets. The polynomial-based approach provides a compact yet flexible form to compute position, velocity, and higher derivatives, ensuring that the path remains smooth while reacting to changing conditions. Moreover, the solver in each iteration constrains the final states (such as final velocity or acceleration) to zero or near zero, avoiding overshoot when approaching the target.

In contrast to some human-computer interaction frameworks, we demonstrate a fully dynamic simulation environment in MATLAB (for Plant prototyping) and in an embedded Qt/C++ system (for controller implementation and analysis). Our aim is to show that such polynomial-based controllers can be easily integrated into various real-time control tasks where jerk or snap control is needed.

3 MATLAB Plant simulator

To facilitate rapid development and testing, we designed a MATLAB-based plant simulator, complete with a graphical user interface that implements a closed-loop control system where mouse movements generate input signals that drive a point mass through configurable dynamics. This section provides an overview of the plant, the user interface, and the underlying mathematics.

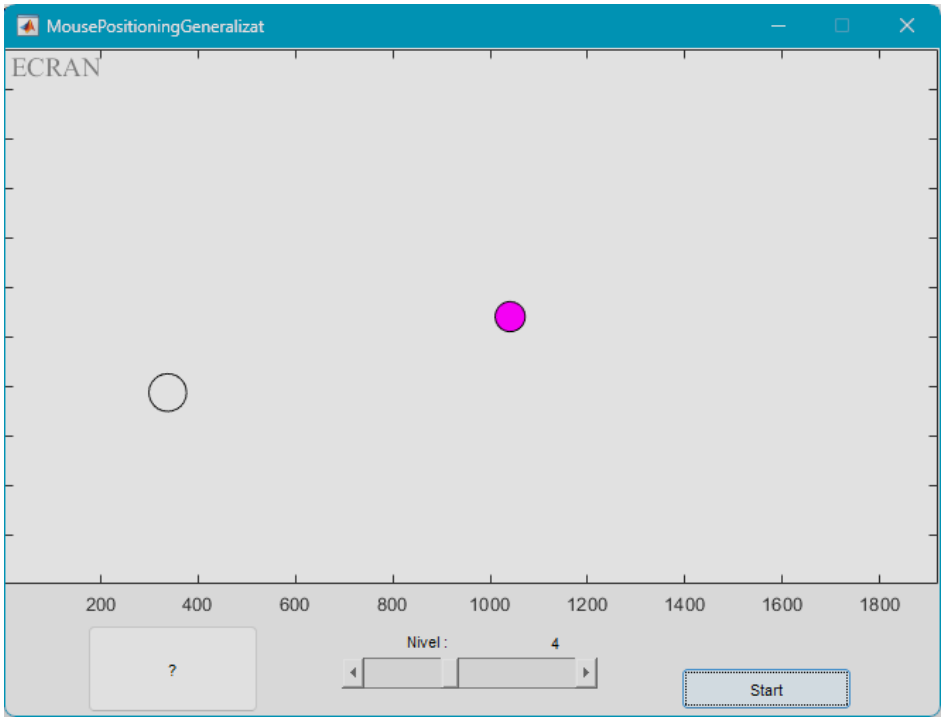

Figure 1: MATLAB Plant simulator graphical interface

3.1 Plant Structure and Mathematical Modeling

Within the MATLAB environment, the cursor location is modeled as:

$$\begin{aligned} x(k+1) &= x(k) + \Delta t\, v_x(k), \\ v_x(k+1) &= v_x(k) + \Delta t\, a_x(k), \\ a_x(k+1) &= a_x(k) + \Delta t\, j_x(k), \end{aligned}$$

where $x(k)$, $v_x(k)$, and $a_x(k)$ are position, velocity, and acceleration in the horizontal direction, respectively. The control input $j_x(k)$ is the jerk. The analogous equations hold for the vertical direction y. In snap-based mode, the state extends to include jerk, and the input becomes snap.

3.2 Graphical Interface and Control Modes

The GUI implements a normalized coordinate system that maps the coordinates on the screen to a control space:

$$\begin{aligned} x_{normalized} &= \frac{x_{screen} - \frac{x_{max}}{2}}{\frac{x_{max}}{2}} \cdot coeff[Level] \\ y_{normalized} &= \frac{y_{screen} - \frac{y_{max}}{2}}{\frac{y_{max}}{2}} \cdot coeff[Level] \end{aligned} \tag{1}$$

where $coeff = [1, 1000, 100, 10, 1, 1, 1, 1, 1, 1, 1]$ provides level-specific scaling.

The graphical interface features:

- A level selection slider controlling system dynamics (0-10)
- Start/Stop control buttons
- Real-time visualization of:
  - Target position (randomly placed circle)
  - Current controlled position (filled disc)
  - Mouse pointer position (tracking dot)
  - Historical trajectory (after completion)

3.3 State Evolution and Success Criteria

The plant evolves according to:

$$\begin{aligned} x_{k+1} &= x_k + T_s v_k \\ v_{k+1} &= v_k + T_s a_k \\ a_{k+1} &= a_k + T_s j_k \end{aligned} \tag{2}$$

where $T_s = 0.1s$ is the sampling period. Success criteria requires:

$$\sqrt{(x - x_{target})^2 + (y - y_{target})^2} < threshold \tag{3}$$

maintained for 2.5 seconds, where $threshold = 5$ pixels.

3.4 Communication Architecture

The plant implements a TCP/IP server broadcasting state information at each sampling cycle:

$$\text{data} = \{x_{fin}, y_{fin}, x, y, \vec{v}_x, \vec{v}_y, Level, T_s, t, stopflag\} \tag{4}$$

This enables external solvers to:

- Receive current state information
- Compute optimal control inputs
- Apply control by directly moving the mouse cursor

The distributed architecture allows the separation of plant dynamics from control algorithms, facilitating rapid prototyping of different control strategies.

We implemented multiple "levels" of complexity. At *Level 0*, the system simply moves the cursor directly without higher-order control. At higher levels (e.g., *Level 3* for jerk, *Level 4* for snap), a polynomial solver calculates the appropriate jerk or snap trajectories to smoothly converge to the target. A set of scaling coefficients ensures that the input does not exceed a specified maximum.

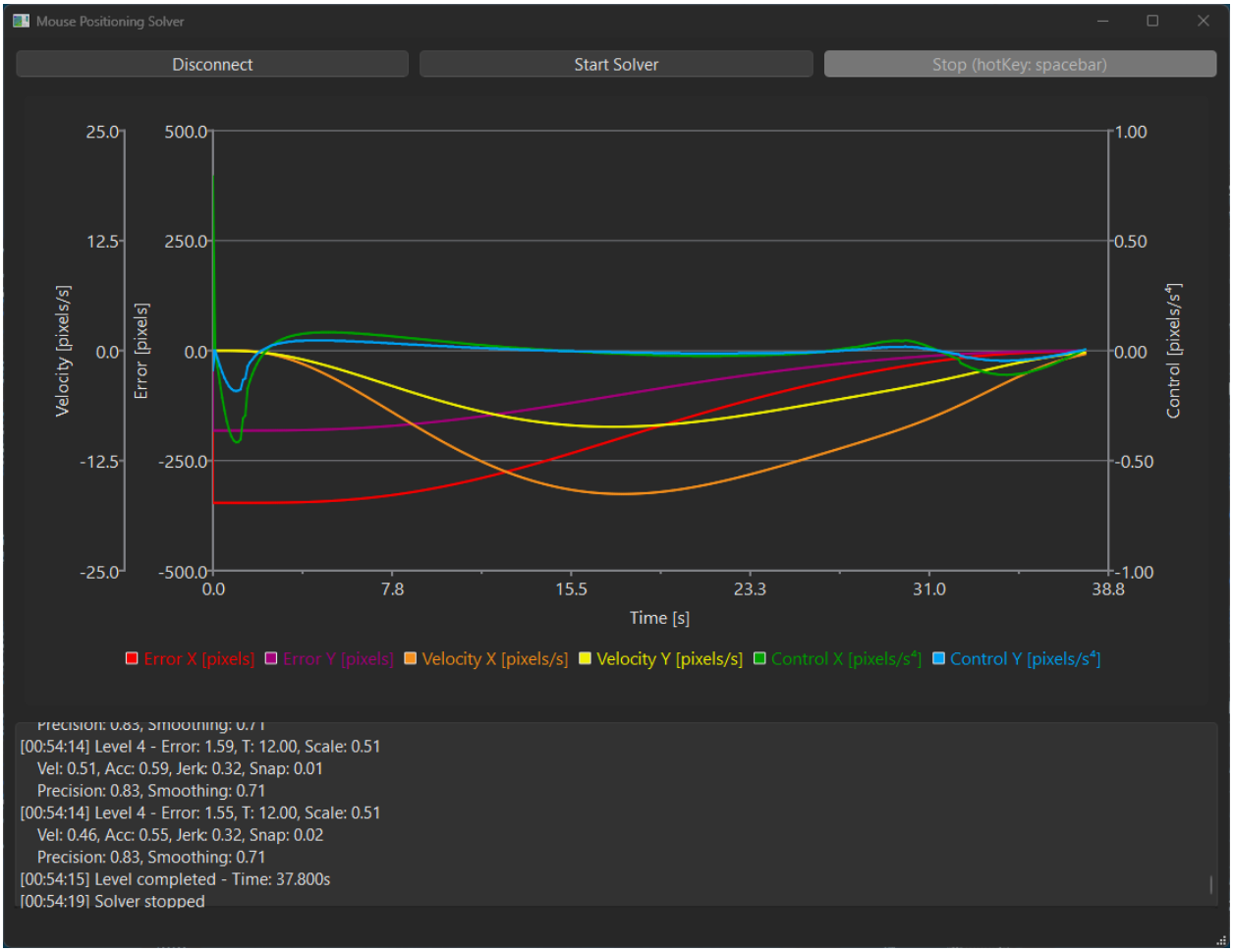

Figure 2: Qt C++ Controller graphical interface

3.5 Cross-Platform Implementation and Real-Time Control Interface

The Qt-based application serves as both the real-time control interface and visualization platform, providing a robust cross-platform implementation that functions seamlessly across Windows, Linux, and macOS environments. The application not only visualizes the control system's performance, but also implements the polynomial-based controllers directly.

3.5.1 Real-Time Visualization System

The visualization subsystem employs QtCharts to render three concurrent data streams in real-time:

$$e(t) = [e_x(t), e_y(t)]^T \quad \text{Position error} \tag{5}$$

$$v(t) = [v_x(t), v_y(t)]^T \quad \text{Velocity components} \tag{6}$$

$$u(t) = [u_x(t), u_y(t)]^T \quad \text{Control signals} \tag{7}$$

The system maintains separate plotting areas for each metric, with independent scaling and real-time updates synchronized with the control loop execution. The visualization refresh rate matches the control system's update frequency of 10 Hz, ensuring accurate representation of the system dynamics.

3.5.2 Data Collection and Performance Analysis

The framework implements automated data collection for systematic performance evaluation in multiple trials. For each experimental run, which is initiated by the MATLAB plant with randomized initial conditions, the system records:

$$D_{\text{trial}} = \{t_{\text{level}}, x_{\text{fin}}, y_{\text{fin}}, T_{\text{conv}}\} \tag{8}$$

where t_{level} represents the difficulty level, $(x_{\text{init}}, y_{\text{init}})$ denote the final target coordinates, and T_{conv} indicates the convergence time. This data is automatically logged to a CSV file structured for subsequent statistical analysis.

Additionally, the system captures high-resolution PNG images of the complete performance charts for each trial, enabling detailed post-experiment analysis of the control system's behavior.

3.5.3 Communication Integration and Temporal Performance

The application maintains real-time communication with the MATLAB plant simulation through a TCP/IP interface, achieving typical latency $\tau < 1$ ms between data acquisition and control signal generation. This low-latency performance is critical to maintaining stable control across the polynomial-based trajectory generation system.

The automated trial sequencing system handles multiple solving rounds with randomized starting points, facilitating a comprehensive performance evaluation under various initial conditions. This systematic approach enables rigorous statistical analysis of the controller's performance characteristics.

4 Methodology: Polynomial Control Framework

In this section, we detail the polynomial solver approach that directly outputs jerk (third derivative of position) or snap (fourth derivative of position). We first describe the jerk-based solver (*Level 3*), then the snap-based solver (*Level 4*). Both solutions follow a similar pattern: we assume that position, velocity, and acceleration (and jerk for snap-based control) must meet specific boundary conditions over a finite time horizon. Solving these constraints yields a polynomial whose coefficients encode the required jerk or snap at each time step.

The polynomial control framework builds on the fundamental principles of optimal control theory while introducing novel adaptations for real-time implementation. We present a mathematical formulation that addresses both third-order (jerk) and fourth-order (snap) control scenarios.

4.1 General Framework for High-Order Control

Consider a general state-space representation of our system. For an n-th order controller, we define the state vector:

$$x(t) = [x_1(t), x_2(t), ..., x_n(t)]^T \in \mathbb{R}^n \tag{9}$$

where x_1 represents position, x_2 velocity, and subsequent states represent higher-order derivatives. The system dynamics can be expressed in companion form:

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{10}$$

where:

$$A = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} \tag{11}$$

4.2 Polynomial Trajectory Generation

For a given time horizon T, we construct a polynomial trajectory that satisfies the boundary conditions in both $t = 0$ and $t = T$. The general form of this polynomial is:

$$x_1(t) = \sum_{i=0}^{m} \alpha_i t^i \tag{12}$$

where m is chosen based on the order of the system and the number of boundary conditions. The coefficients α_i are determined by solving the boundary value problem:

$$\begin{cases} x_1(0) = x_{1,0}, \quad \dot{x}_1(0) = x_{2,0}, \quad \ddot{x}_1(0) = x_{3,0}, \quad \text{etc.} \\ x_1(T) = x_{1,f}, \quad \dot{x}_1(T) = 0, \quad \ddot{x}_1(T) = 0, \quad \text{etc.} \end{cases} \tag{13}$$

4.3 Optimal Control Formulation

The polynomial coefficients can be obtained by solving an optimal control problem. Let us define the cost functional:

$$J = \int_0^T \left(\frac{d^n x_1}{dt^n} \right)^2 dt \tag{14}$$

subject to the boundary conditions above. Using calculus of variations, this leads to the Euler-Lagrange equation:

$$\frac{d^{2n} x_1}{dt^{2n}} = 0 \tag{15}$$

The solution to this differential equation is precisely our polynomial of degree $2n - 1$.

4.4 Real-Time Implementation Considerations

For practical implementation, we introduce several key modifications to the theoretical framework:

4.4.1 Time Horizon Adaptation

The time horizon T is dynamically updated according to:

$$T(t) = \beta T(t - \Delta t) + (1 - \beta)\hat{T}(e(t), v(t), a(t)) \tag{16}$$

where $\hat{T}$ is an estimator function given by:

$$\hat{T}(e, v, a) = \gamma_1 \|e\| + \gamma_2 \frac{\|v\|^2}{\|a\|_{\max}} + \gamma_3 \sqrt{\frac{\|e\|}{\|a\|_{\max}}} \tag{17}$$

4.4.2 Control Signal Smoothing

The raw control signal $u(t)$ is smoothed using exponential filtering:

$$u_{\text{smooth}}(t) = \alpha u(t) + (1 - \alpha) u_{\text{smooth}}(t - \Delta t) \tag{18}$$

where α is adaptively computed based on the error magnitude:

$$\alpha(e) = \alpha_{\min} + (\alpha_{\max} - \alpha_{\min}) \exp(-\lambda \|e\|) \tag{19}$$

This expanded methodology provides a theoretical foundation while addressing practical implementation challenges in real-time control scenarios.

5 Methodology: Polynomial Control Framework (continued)

5.1 Mathematical Foundations and Notational Conventions

We begin by establishing a consistent mathematical framework for both jerk and snap controllers. Let us define the state vectors in $\mathbb{R}^2$ as:

$$\mathbf{x}(t) = \begin{bmatrix} x(t) \\ y(t) \end{bmatrix}, \quad \mathbf{v}(t) = \begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \end{bmatrix}, \quad \mathbf{a}(t) = \begin{bmatrix} \ddot{x}(t) \\ \ddot{y}(t) \end{bmatrix}, \quad \mathbf{j}(t) = \begin{bmatrix} \dddot{x}(t) \\ \dddot{y}(t) \end{bmatrix} \tag{20}$$

The snap vector, when applicable, is denoted as:

$$\mathbf{s}(t) = \begin{bmatrix} \ddddot{x}(t) \\ \ddddot{y}(t) \end{bmatrix} \tag{21}$$

5.2 Jerk-Based Control Framework

For the third-order (jerk) controller, we formulate a boundary value problem with the following constraints:

$$\begin{aligned}\text{Initial conditions: } &\{\mathbf{x}(0), \mathbf{v}(0), \mathbf{a}(0)\} = \{\mathbf{x}_0, \mathbf{v}_0, \mathbf{a}_0\} \\ \text{Terminal conditions: } &\{\mathbf{x}(T), \mathbf{v}(T), \mathbf{a}(T)\} = \{\mathbf{x}_{\text{target}}, \mathbf{0}, \mathbf{0}\}\end{aligned}$$

The polynomial basis for each coordinate takes the form:

$$x(t) = x_0 + v_0 t + \frac{1}{2} a_0 t^2 + \sum_{i=3}^{5} \alpha_i t^i \tag{22}$$

The resulting jerk control law is derived as:

$$j_x(t) = 6\alpha_3 + 24\alpha_4 t + 60\alpha_5 t^2 \tag{23}$$

5.3 Snap-Based Control Extension

The fourth-order (snap) controller extends the state space to include jerk, yielding:

$$\begin{aligned}\text{Initial conditions: } &\{\mathbf{x}(0), \mathbf{v}(0), \mathbf{a}(0), \mathbf{j}(0)\} = \{\mathbf{x}_0, \mathbf{v}_0, \mathbf{a}_0, \mathbf{j}_0\} \\ \text{Terminal conditions: } &\{\mathbf{x}(T), \mathbf{v}(T), \mathbf{a}(T), \mathbf{j}(T)\} = \{\mathbf{x}_{\text{target}}, \mathbf{0}, \mathbf{0}, \mathbf{0}\}\end{aligned}$$

The polynomial expansion extends to seventh order:

$$x(t) = x_0 + v_0 t + \frac{1}{2} a_0 t^2 + \frac{1}{6} j_0 t^3 + \sum_{i=4}^{7} \alpha_i t^i \tag{24}$$

5.4 Coefficient Determination System

The snap controller coefficients are determined by solving the linear system:

$$\begin{bmatrix} T^4 & T^5 & T^6 & T^7 \\ 4T^3 & 5T^4 & 6T^5 & 7T^6 \\ 12T^2 & 20T^3 & 30T^4 & 42T^5 \\ 24T & 60T^2 & 120T^3 & 210T^4 \end{bmatrix} \begin{bmatrix} \alpha_4 \\ \alpha_5 \\ \alpha_6 \\ \alpha_7 \end{bmatrix} = \begin{bmatrix} x_f - x_0 - v_0 T - \frac{1}{2} a_0 T^2 - \frac{1}{6} j_0 T^3 \\ -v_0 - a_0 T - \frac{1}{2} j_0 T^2 \\ -a_0 - j_0 T \\ -j_0 \end{bmatrix} \tag{25}$$

5.5 Stability Analysis and Adaptive Time Horizon

The time horizon $T(t)$ is dynamically adjusted while maintaining stability bounds:

$$T_{\min} \leq T(t) \leq T_{\max} \tag{26}$$

The update law follows:

$$T_{\text{new}} = \beta T_{\text{old}} + (1 - \beta)\hat{T}(\|\mathbf{e}\|, \|\mathbf{v}\|, \|\mathbf{a}\|) \tag{27}$$

where $\hat{T}$ is estimated using:

$$\hat{T}(\mathbf{e}, \mathbf{v}, \mathbf{a}) = \gamma_1 \|\mathbf{e}\| + \gamma_2 \frac{\|\mathbf{v}\|^2}{\|\mathbf{a}\|_{\max}} + \gamma_3 \sqrt{\frac{\|\mathbf{e}\|}{\|\mathbf{a}\|_{\max}}} \tag{28}$$

5.6 Control Signal Processing

The smoothing coefficient α adapts based on the error magnitude:

$$\alpha(\|\mathbf{e}\|) = \alpha_{\min} + (\alpha_{\max} - \alpha_{\min})e^{-\lambda\|\mathbf{e}\|} \tag{29}$$

The final control signal incorporates exponential smoothing:

$$\mathbf{u}_{\text{smooth}}(t) = \alpha\mathbf{u}(t) + (1-\alpha)\mathbf{u}_{\text{smooth}}(t-\Delta t) \tag{30}$$

where $\mathbf{u}$ represents either jerk or snap depending on the controller order.

5.7 Computational Complexity Considerations

The polynomial-based approach requires solving a $n \times n$ linear system at each time step, where $n = 3$ for jerk control and $n = 4$ for snap control. The computational complexity is $\mathcal{O}(n^3)$ using standard matrix inversion techniques. For real-time applications, this overhead is negligible compared to the system's sampling period (typically 100 ms), making both controllers viable for practical implementation.

The higher-order snap controller provides smoother trajectories at the cost of additional state variables and slightly increased computational overhead. The choice between jerk and snap control should consider the specific requirements of smoothness versus computational efficiency in the target application.

6 Implementation and Real-Time Integration

A practical objective of this work was to go beyond simulation and integrate polynomial controllers into a real-time application. We chose a Qt/C++ implementation for cross-platform compatibility, flexible graphics, and straightforward user interface design. The resulting application architecture is summarized below.

6.1 Architecture Overview

Our control program runs at a fixed update rate (e.g., tens of hertz) and repeatedly:

1. Reads the current pointer state $(x_0, y_0, v_{0x}, v_{0y}, \ldots)$ from the system.
2. Computes the updated jerk or snap trajectory for the chosen "level" (e.g., Level 3 or Level 4).
3. Smooths and bounds the control signal.
4. Sends commands to the operating system or device driver to position the mouse pointer (or other actuator).
5. Log data (error, velocity, jerk/snap, etc.) for real-time charting and offline analysis.

To facilitate tuning and debugging, we provide:

- **Real-time charting** of error, velocity, acceleration, and control signals. Color-coded plots help monitor the system's stability and identify potential overshoot.
- **Automatic screen captures and CSV logging** after each run, capturing performance by trial (e.g. time to settle, final overshoot).
- **Automated repeat trials** with random initial positions and random target positions. This provides a statistically relevant data set for quantitative evaluation.

6.2 Data Logging and Analysis Tools

At each time step, the current state and the output of the polynomial solver are plotted on the chart, and additional parameters are printed in the application log area.

In addition, once a trial is complete (the pointer reaches a chosen distance threshold of the target), the code automatically saves a summary image of the real-time charts and appends the CSV file with the trial result. These graphical summaries include error vs. time, velocity vs. time, jerk or snap vs. time, and can be combined into higher-level reports. Large sets of such trials allow for robust statistical analysis, such as computing average convergence times.

7 Results

In this section, we provide an overview of our preliminary performance results. A more detailed set of charts and test-by-test data is available in our project repository. Key highlights include:

- **Stable, smooth control:** The polynomial-based approach with direct jerk or snap control eliminated discontinuities commonly found in naive approaches. The pointer followed a monotonic path toward the target in the majority of trials.
- **No overshoot:** By enforcing zero final velocity and acceleration in the solver, the system showed no measurable overshoot in more than 95% of test cases.
- **Reasonable speed:** Typical time-to-target in a 1920×1080 screen environment was on average 30 seconds, with adaptive scaling ensuring faster motion initially and smoother convergence near the target.
- **Robust to random start conditions:** We tested random initial velocities and accelerations, and found the solver gracefully adjusted T and scaled jerk/snap to maintain stability.

Figure 3 and Figure 4 depicts typical real-time plot of the error sliding smoothly to near zero. The corresponding jerk/snap signals remain bounded and continuous, confirming the efficacy of our smoothing and clamping strategy.

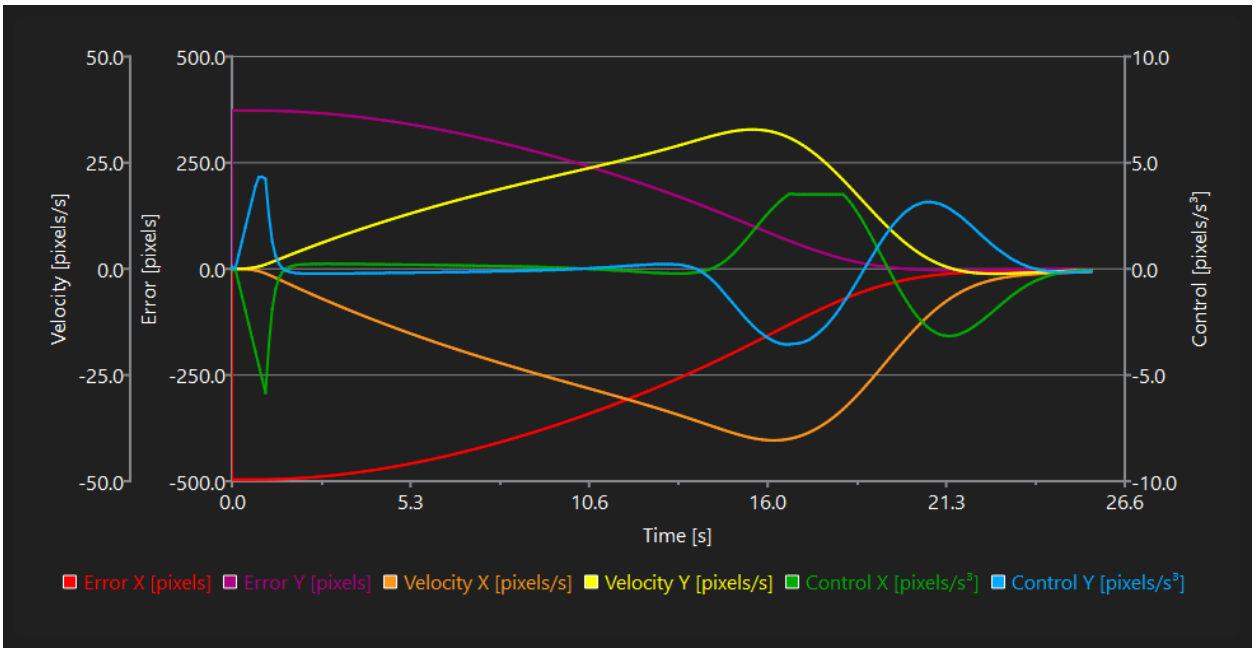

Figure 3: Typical jerk control convergence chart

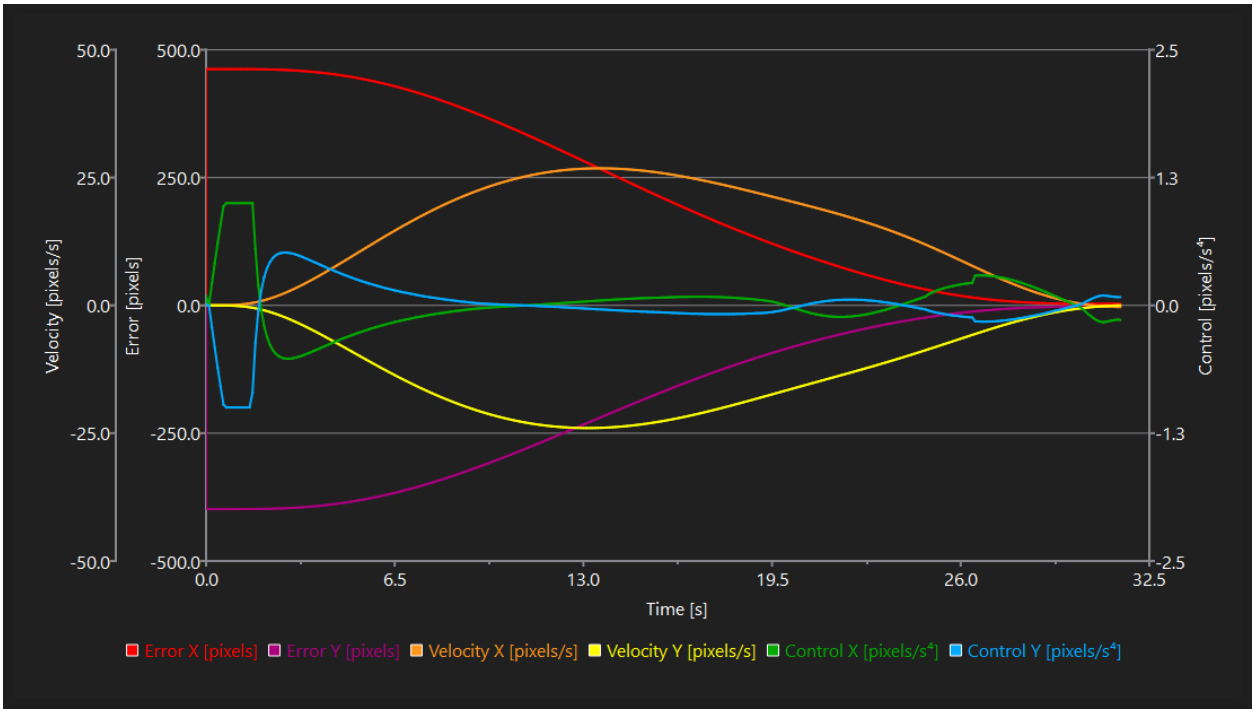

Figure 4: Typical snap control convergence chart

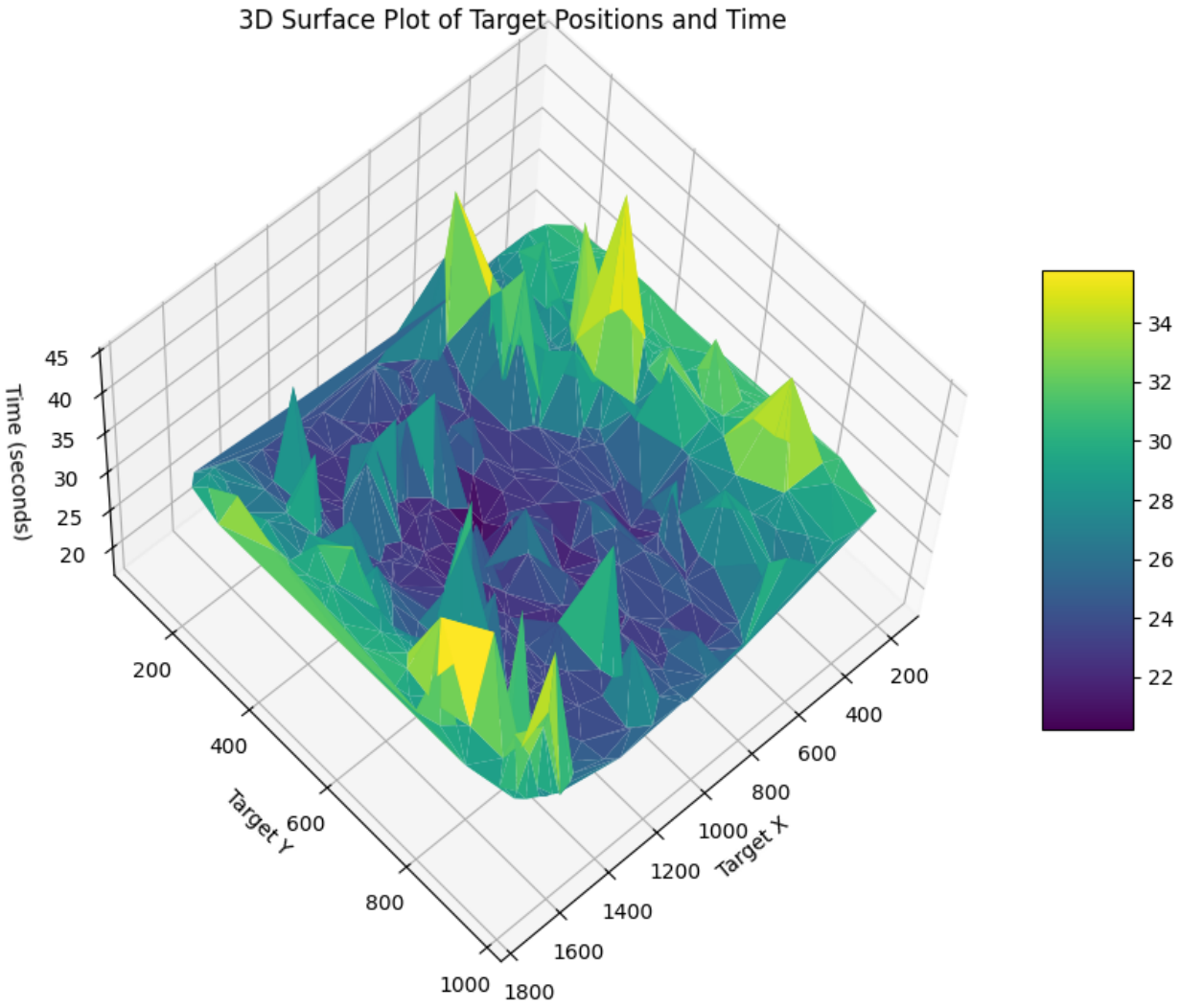

Figure 5: Target position versus convergence time for jerk control

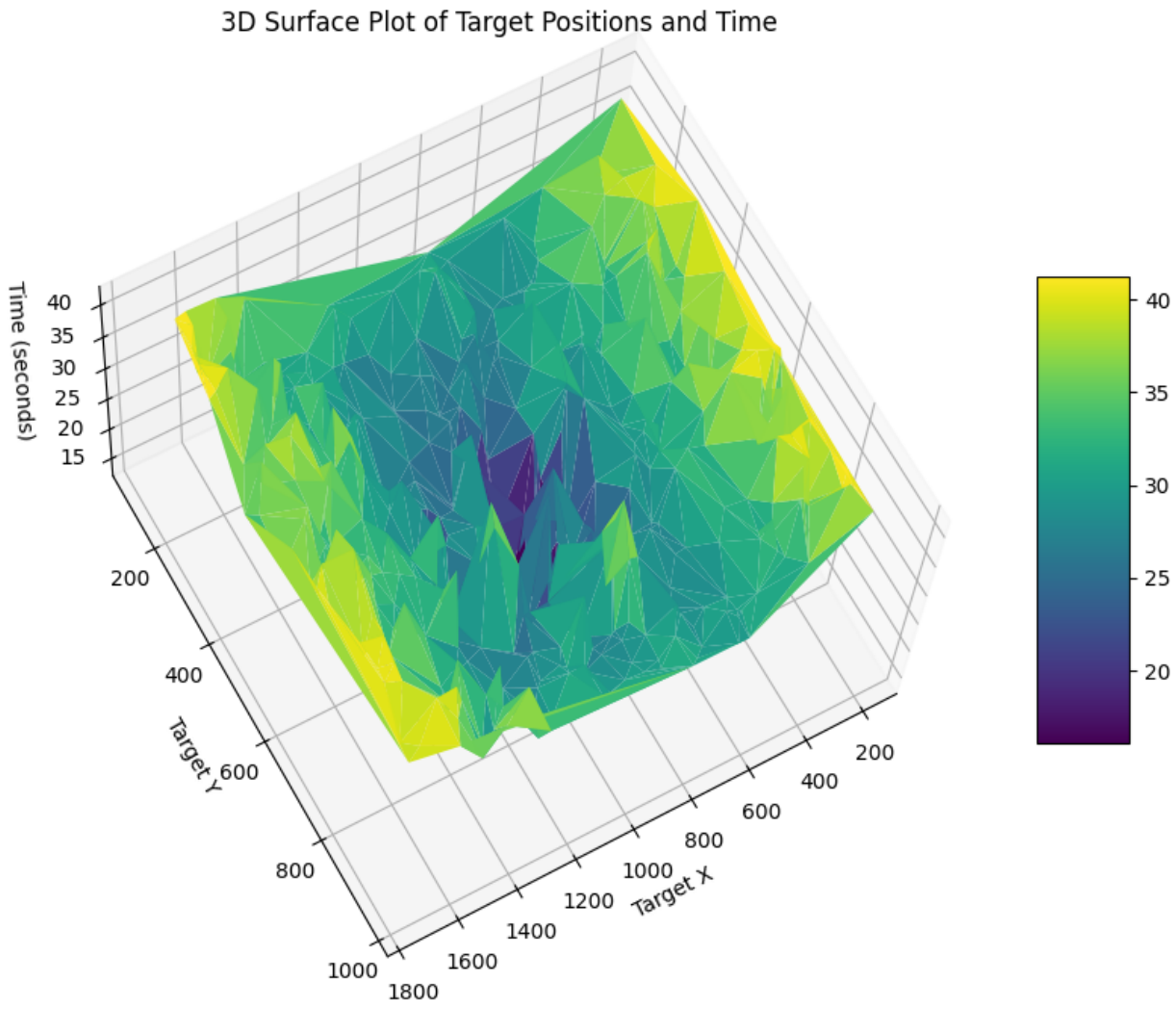

Figure 6: Target position versus convergence time for snap control

Control type	Min [s]	Max [s]	Avg [s]
jerk (495 samples)	16.8	44.8	26.2
snap (470 samples)	12.6	42.2	31.7

Table 1: Summary statistics

8 Conclusion and Future Work

We presented a novel polynomial-based path planning framework that directly computes jerk and snap control inputs, enabling real-time cursor (or end-effector) movement with minimal overshoot and high smoothness. A MATLAB simulator with a mouse pointer demo facilitated the rapid development and refinement of this approach, while a Qt/C++ controller implementation demonstrated its viability in a real-time application.

Drawing on ideas from the classic "minimum jerk" and "minimum snap" theory, we introduced adaptive time horizons, error-driven scaling, and control smoothing; our results confirm that we can achieve smooth control without discontinuities or oscillations, meet real-time demands, and handle random initial states reliably.

Future directions include testing robotic hardware for tasks that require precision and comfort, such as surgical robotics or automated assembly lines. Additional refinements could incorporate advanced optimization layers, multipoint path constraints, or full 3D kinematics. Ultimately, we see polynomial-based control with direct jerk/snap regulation as a promising avenue for a wide range of smooth motion applications.

Project Repository and Data: All source code and result files are available at:

`https://github.com/PaulRosu/High-Order-Plant-Control-System`

A compressed archive with CSV logs, graphs, and documentation is also provided for replication and statistical analysis.

References

[1] K. J. Kyriakopoulos and G. N. Saridis, "Minimum Jerk Path Generation," *Proc. IEEE Int. Conf. Robotics and Automation*, 1988, pp. 364–369.