RoboWriter Project Report

杨天成　李昂　吴梓杰
Group ID: 15

January 8, 2026

1 Project Overview

1.1 Research Background

Robotic manipulation is a cornerstone of modern industrial automation. While discrete tasks like "pick-and-place" are widely implemented, tasks requiring **Continuous Path Tracking (CPT)**—such as writing, drawing, and precision welding—demand significantly higher standards for kinematic control and trajectory smoothness. Furthermore, real-world operational surfaces are rarely perfectly planar. Performing precise manipulation on **non-planar surfaces (e.g., spheres)** introduces complex challenges in dimensionality mapping and geometric correction. This project utilizes the **MuJoCo** physics engine and a **UR5e** 6-DoF robotic arm to simulate and solve these challenges in a high-fidelity physics environment.

1.2 Research Purpose

The primary objective of this project is to design and implement a robust control system for the UR5e manipulator to perform complex writing tasks on multiple geometric surfaces. The specific research goals are:

- **Inverse Kinematics Control**: To implement a controller based on **Differential Inverse Kinematics**.
- **Complex Trajectory Generation**: To verify the system's capability in generating complex geometries.
- **Singularity Robustness**: To introduce the **Damped Least Squares (DLS)** method.
- **Spatial Dimensional Mapping**: To solve the mathematical mapping problem from a 2D plane to a 3D spherical surface.

1.3 Application Scenarios

The technology developed in this "Robotic Writing" project has broad applications:

- **Basic Applications**: Calligraphy & Art Preservation, Remote Tele-presence Writing.

- **Advanced Extensions**: Curved Surface Processing (painting/coating), Precision Laser Processing, Surgical Assistance.

1.4 Overall Work Content

1.4.1 Task Implementation & Simulation Results

In this project, we successfully implemented four distinct tasks, evolving from basic planar writing to advanced spherical projection.

1. Task 1: Basic Plane Writing

We implemented fixed-content writing on a 2D plane, including the Chinese name "Li Ang", Pinyin, and Student ID.

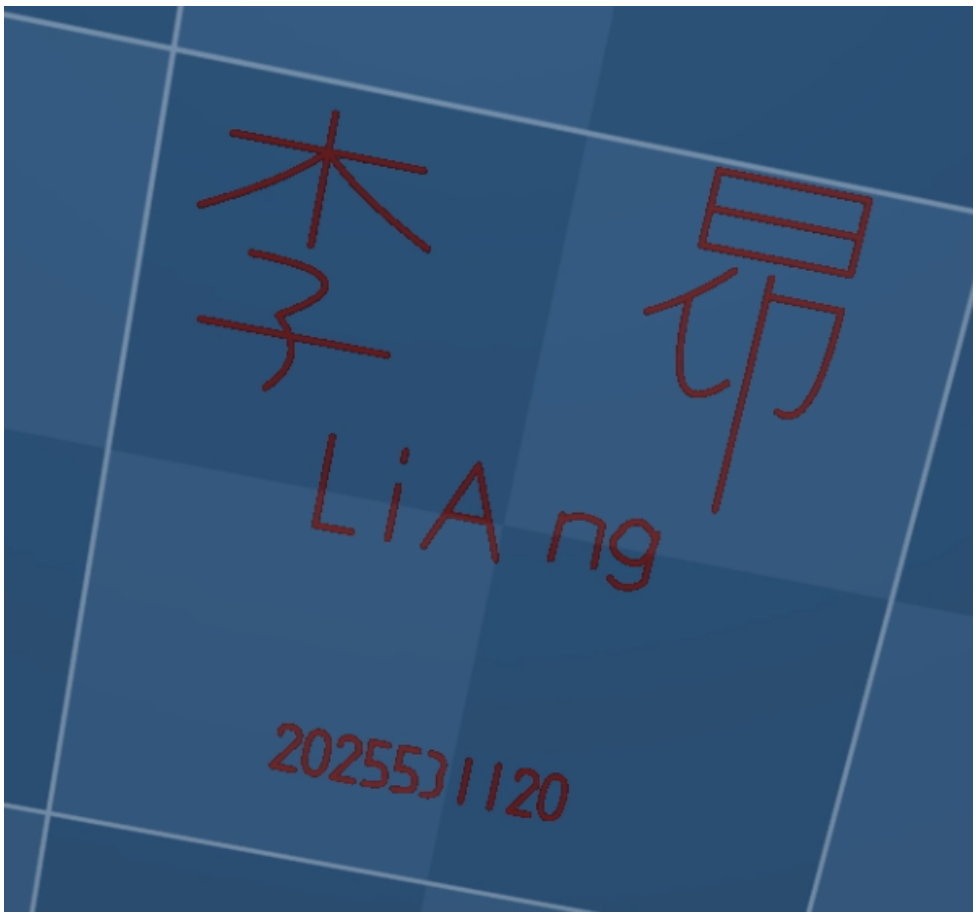

Figure 1: Simulation Result: Basic Plane Writing

2. Task 2: Automatic Plane Writing

We developed a system capable of writing arbitrary English strings input by the user on a plane, featuring automatic layout and smooth motion.

Figure 2: Simulation Result: Automatic Layout with MinJerk

3. Task 3: Basic Sphere Writing
We transferred the content from Task 1 onto a spherical surface ($R = 1.3m$) using orthogonal projection algorithms.

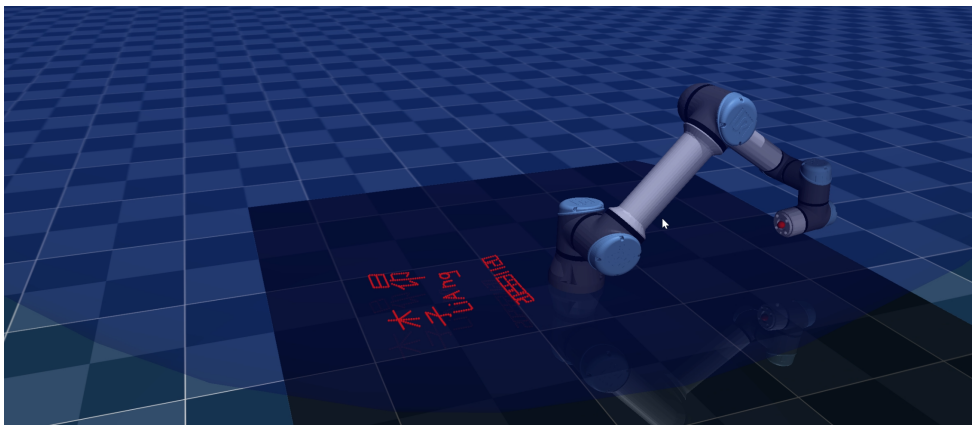

Figure 3: Simulation Result: Writing "Li Ang" on a Sphere

4. Task 4: Automatic Sphere Writing
The final system integrates the font library and projection logic, allowing arbitrary text to be written smoothly on a curved sphere.

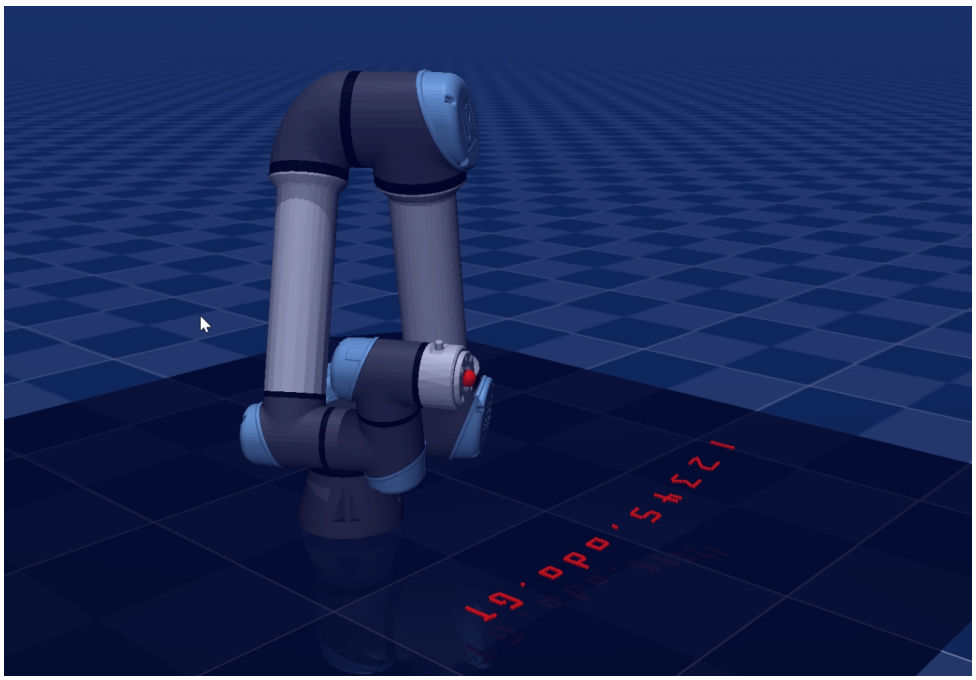

Figure 4: Simulation Result: Writing Arbitrary Text on a Sphere

1.4.2 Core Technical Dimensions

This project constructs a complete "RoboWriter" system. The core work is structured into four dimensions:

1. **Character Library Construction**:
   Developed a vectorized stroke generation system. Based on a relative Grid System,

the project implemented parameterized generation for English letters and numbers. Additionally, specific Chinese characters ("Li" and "Ang") were defined via hard-coded nodes and topological structures to meet custom requirements.

2. **Hybrid Interpolation Algorithms**:
   To ensure writing fluidity and geometric aesthetics, a hybrid interpolation strategy was adopted. Geometrically, **Quadratic Bezier Curves** were introduced to smooth complex strokes (e.g., the hooks in Chinese characters). Temporally, the system combines high-density linear sampling with **Minimum Jerk** planning principles to eliminate mechanical jitter during start-stop phases.

3. **Motion Control**:
   Implemented a Differential IK controller based on the Jacobian matrix. Crucially, we integrated **Null-Space Posture Control** to utilize the redundant degrees of freedom for maintaining a natural "elbow-up" posture. Additionally, the **Damped Least Squares (DLS, $\lambda = 0.1$)** method was employed to effectively suppress numerical instability near singular configurations.

4. **Spherical Projection**:
   Derived and implemented an orthographic projection algorithm ($z = C_z - \sqrt{R^2 - r^2}$) mapping the 2D tangent plane to a 3D spherical surface. Combined with equidistant resampling techniques, this solved the geometric mapping and height compensation issues when "wrapping" planar text onto a sphere with a radius of 1.3m.

2 System Design

2.1 General System Architecture & Module Divisions

Although the project is divided into four sub-tasks (represented by the four distinct code files), they all share the same underlying logic. The system is built like a factory assembly line, divided into four main modules:

- **Input Module (The "Order")**:
  This part determines *what* needs to be written. It can be a fixed command (like "Li Ang" in the basic task) or a user-typed string (like "Hello" in the bonus task).

- **Path Generator Module (The "Map")**:
  This module calculates *where* the pen tip should go. It converts the characters into a list of coordinate points (x, y, z). For the sphere tasks, this module also handles the math to project the points onto a curved surface.

- **Control Module (The "Brain")**:
  This is the core calculation unit. It receives the target coordinate (x, y, z) and uses **Inverse Kinematics (IK)** algorithms to calculate the exact angle for each of the robot's 6 joints.

- **Simulation Module (The "Body")**:
  We use **MuJoCo** to simulate the physical robot. It executes the joint angles and draws red dots at the pen tip so we can see the writing result visually.

Relationship between modules:
The data flows in a single direction: Input provides the text → Path Generator turns text into 3D points → Control Module turns points into joint angles → Simulation moves the robot.

2.2 Key Technical Solutions by Task

Since the project evolved from basic to advanced, distinct technical solutions were applied to each stage:

2.2.1 Task 1: Basic Plane Writing (`LiAng_basic.py`)

This is the foundational task. The goal is to write "Li Ang", Pinyin, and Student ID on a flat table.

- **Hardcoded Coordinates**: Since the content is fixed, we manually defined the key points for the Chinese characters and numbers. For example, to write "1", we just defined the start point and end point.
- **Linear Interpolation**: To draw lines between points, we used simple linear math to generate intermediate points, making the robot move in straight lines.
- **Basic State Machine**: We separated the robot's actions into "Moving in Air" and "Writing on Paper" to ensure it lifts the pen between characters.

2.2.2 Task 2: Automatic Plane Writing (`auto_bonus.py`)

This bonus task allows the robot to write *any* English string entered by the user.

- **Font Library (Dictionary)**: Instead of defining points for just one word, we built a "Dictionary" that contains the stroke shapes for A-Z, 0-9, and punctuation. When the user types "A", the program looks up "A" in the library.
- **Automatic Layout**: The system automatically calculates the width of each letter and places them side-by-side, so they don't overlap.
- **Smoother Movement**: We used a math method called **Minimum Jerk** to control the speed. This makes the robot start and stop slowly, preventing it from shaking, unlike the constant speed in Task 1.

2.2.3 Task 3: Basic Sphere Writing (`LiAng_sphere.py`)

This task transfers the fixed content from Task 1 onto a curved sphere ($R = 1.3m$).

- **Pipeline: Generation → Discrete Sampling → Spatial Projection**: Unlike planar tasks, we do not perform direct interpolation on the sphere surface. Instead, we adopted a three-stage strategy: 1. **Generate**: Create the character geometry on a virtual 2D plane. 2. **Sample**: Discretize the continuous strokes into high-density micro-points (0.5mm interval). 3. **Project**: Mathematically map each micro-point onto the sphere surface.
- **Eliminating Chord Error**: This strategy ensures that the pen tip strictly adheres to the spherical curvature, preventing the chord error (penetration) that would occur with direct point-to-point interpolation.

2.2.4 Task 4: Automatic Sphere Writing (`auto_sphere.py`)

This is the final combination of Task 2 and Task 3.

- **Integration**: We combined the **Font Library** from Task 2 with the **Projection Pipeline** from Task 3, enabling the automated generation of complex, arbitrary text on a curved surface.

3 Implementation

This chapter details the specific implementation process. We first introduce the common infrastructure shared by all tasks—including the core IK solver and visualization engine—and then delve into the unique technical solutions and algorithms implemented for each specific sub-task.

3.1 Common Implementation Infrastructure

To ensure robustness and modularity, we developed a unified control framework used across all four sub-tasks. This framework consists of three core components: the IK solver, the Null-Space posture controller, and the visualization engine.

3.1.1 Inverse Kinematics (IK) Controller

Significance: The fundamental task of the robot is to convert the target Cartesian coordinates (x, y, z) of the pen tip into the required joint velocities $(\dot{q})$. Analytical solutions are complex for general paths, and standard numerical methods (like the Pseudo-Inverse) often cause violent shaking when the robot approaches singular configurations.

Implementation: We implemented a differential IK solver based on the Jacobian matrix. To prioritize stability over exact position error in difficult poses, we integrated the **Damped Least Squares (DLS)** method.

```
def IK_controller_robust(model, data, target_pos, q_current):
    # 1. Calculate Jacobian (J)
    J_pos = np.zeros((3, model.nv))
    mj.mj_jac(model, data, J_pos, None, site_id, np.array([0,0,0]))
    J = J_pos[:, :6]

    # 2. Damped Least Squares Formula
    # J_dls = J.T * inv(J * J.T + lambda^2 * I)
    lambda_val = 0.1
    I = np.eye(3)
    J_dls = J.T @ np.linalg.inv(J @ J.T + lambda_val**2 * I)

    # 3. Calculate Joint Velocities
    error_vel = (target_pos - current_pos) / dt
    dq = J_dls @ error_vel
    return dq
```

Listing 1: IK with Damped Least Squares

3.1.2 Null-Space Posture Control

Significance: The UR5e robot has 6 degrees of freedom (DoF), but writing on a point only strictly requires 3 DoF (x, y, z). This redundancy allows us to control the robot's "posture" (e.g., keeping the elbow up) without affecting the pen tip's position. This is a key feature that prevents the arm from collapsing or twisting awkwardly during operation.

Implementation: We defined a "Rest Pose" (q_{rest}) representing a comfortable configuration. We projected the error between the current pose and the rest pose into the **Null Space** of the Jacobian using the projection matrix $(I - J^{\dagger}J)$.

```
# Define a comfortable "Rest Pose" (Elbow up, wrist down)
q_rest = np.array([-1.57, -1.57, 1.57, -1.57, -1.57, 0.0])

# Calculate Posture Error
q_error = K_p * (q_rest - q_current)

# Projection Matrix: (I - J_dls * J)
# This matrix filters out movements that would disturb the pen tip
null_space_proj = (np.eye(6) - J_dls @ J)

# Add Null-Space Velocity to Main Task Velocity
# dq_total = dq_task + dq_null
dq += null_space_proj @ q_error
```

Listing 2: Null-Space Posture Control Logic

Mechanism Analysis: The term $(I - J^{\dagger}J)$ ensures that the posture adjustment happens *only* in the null space. This means the robot creates an internal motion to stay close to q_{rest} while the pen tip remains perfectly stationary or follows the writing path strictly.

3.1.3 Ink Visualization Engine

Significance: Since MuJoCo is a physics simulator and does not have a built-in "pen" tool, we needed a way to verify the writing quality visually.

Implementation: We implemented a real-time trace recorder. At every simulation step where `ink=True`, the end-effector coordinate is pushed into a circular buffer. During the render callback, the system iterates through this buffer and draws small red spheres to simulate continuous ink.

```
# In the control loop
if segment.ink_on:
    traj_points.append(data.site_xpos[0].copy())

# In the render loop (mjv_updateScene)
for p in traj_points:
    # Fetch a geometry slot from MuJoCo scene
    g = scene.geoms[scene.ngeom]
    scene.ngeom += 1

    # Configure visual properties (Red Sphere, Radius=3mm)
    g.type = mj.mjtGeom.mjGEOM_SPHERE
    g.rgba[:] = [1, 0, 0, 1]
    g.size[:] = [0.003, 0.003, 0.003]
    g.pos[:] = p
```

Listing 3: Trajectory Rendering Loop

3.2 Task 1 Implementation: Hybrid Interpolation Strategy

In `LiAng_basic.py`, although the content is fixed ("Li Ang"), the core challenge is ensuring high-quality trajectory generation. We implemented a **Hybrid Interpolation** strategy to simulate human-like writing dynamics.

3.2.1 Linear Interpolation with Minimum Jerk Mapping

Significance: A simple linear interpolation ($P = P_{start} + \Delta P \times t$) implies constant velocity. This results in infinite acceleration (Jerk) at the start and stop points, leading to mechanical vibration and shaky writing.

Implementation: We decoupled the geometric path from the time domain. While the geometry remains a straight line, the time variable t is mapped using a **Minimum Jerk scaling factor** $u(t)$. We utilized a 5th-order polynomial: $u(t) = 10t^3 - 15t^4 + 6t^5$. This ensures that while the pen moves in a straight line, it accelerates smoothly from rest and decelerates to zero at the end.

```
# Even for straight lines, we use 'u' instead of raw 't'
# u varies from 0 to 1 with zero velocity/acceleration at boundaries
u = 10*(t_norm**3) - 15*(t_norm**4) + 6*(t_norm**5)

# Interpolate Position
current_pos = start + (target - start) * u
```

Listing 4: MinJerk Time Scaling

3.2.2 Bezier Curve Interpolation

Significance: Chinese characters (e.g., the hook in "Li" and "Zi") contain complex curved strokes. Approximating them with straight lines makes the writing look robotic and stiff.

Implementation: We utilized **Quadratic Bezier Curves**. For curved segments, we defined three control points (Start, Control, End). The algorithm calculates the path using the Bezier formula ($B(t) = (1-t)^2P_0 + 2(1-t)tP_1 + t^2P_2$), creating a smooth, natural arc driven by the geometry of the character.

3.3 Task 2 Implementation: Automation & Dictionary Logic

In `auto_bonus.py`, the focus shifts from trajectory quality to **System Flexibility**.

3.3.1 Dictionary-Based Font Engine

Significance: Hardcoding coordinates for every word is inefficient. The system needs to generate trajectories for unknown user inputs (e.g., arbitrary names).

Implementation: We built a scalable stroke dictionary. The keys are characters ('A', 'B', '1'...), and the values are lists of normalized vector strokes. When a string is input, the system dynamically looks up these vector shapes.

3.3.2 Automatic Layout Calculation

Implementation: To prevent character overlap, the system iterates through the input string and calculates the `start_x` offset for each character based on a fixed `CHAR_WIDTH`. It also calculates the total width to center the entire string relative to the workspace origin.

3.4 Task 3 Implementation: Spherical Projection & Sampling

In `LiAng_sphere.py`, the challenge is geometric transformation. We need to "wrap" the planar trajectory onto a curved surface ($R = 1.3m$).

3.4.1 Orthographic Projection Algorithm

Significance: We need a mathematical rule to map 2D coordinates (x, y) to 3D spherical coordinates (x, y, z). This is the foundation for transferring planar text to a curved surface.

Implementation: We utilized the Orthographic Projection formula. For any given point (x, y), we calculate its Z-height based on the sphere equation $z = C_z - \sqrt{R^2 - r^2}$. This effectively "drops" the planar writing vertically onto the sphere.

3.4.2 Discrete Sampling & Execution Logic

Significance: Standard interpolation (Linear or Bezier) connects two points with a straight line or a planar curve. On a sphere, this would cause the path to cut *through* the sphere (chord error). Therefore, we cannot simply interpolate between long-distance start and end points on the sphere.

Implementation: We implemented a **Generation → Sampling → Projection** pipeline: 1. **Generate 2D**: Calculate the stroke shape (Linear/Bezier) on the virtual 2D plane. 2. **Discretize**: Instead of executing this 2D path, we loop through it and sample it at high density (every 0.5mm). 3. **Project & Execute**: Each sampled micro-point is immediately projected to 3D space using the formula above. The robot then tracks this dense stream of 3D points.

```
# Calculate number of steps needed (0.5mm resolution)
dist = np.linalg.norm(p2 - p1)
steps = int(dist / 0.0005)

for k in range(steps):
    t = k / steps
    # 1. Generate: Linear/Bezier Interpolation in 2D
    curr_2d = (1-t)*p1 + t*p2

    # 2. Project: Map micro-point to 3D Sphere
    curr_3d = project_to_sphere(curr_2d)

    # 3. Execute: Add to trajectory list
    trajectory.append(curr_3d)
```

Listing 5: Discrete Sampling Loop

3.5 Task 4 Implementation: System Integration

In `auto_sphere.py`, we integrated the modules from previous tasks to achieve the final goal: **Universal Spherical Writing**.

3.5.1 Module Integration Strategy

Significance: This task demonstrates the modularity of the system by combining the flexibility of Task 2 (Dictionary) with the geometric capability of Task 3 (Projection).

Implementation Steps:

1. **Input Parsing:** The system uses the Font Engine (from Task 2) to generate raw 2D strokes from the user input.
2. **Processing:** These 2D strokes are passed through the Sampling and Projection pipeline (from Task 3) to convert them into dense 3D paths.
3. **Execution:** The final trajectory is sent to the robust IK controller (Common Infrastructure) for execution.

4 Testing and Results

In this chapter, we present the experimental results of the RoboWriter system. We first analyze the overall performance improvements brought by our advanced control algorithms, and then detail the specific testing procedures and results for each of the four sub-tasks.

4.1 Overall Performance Analysis

Before diving into specific tasks, we evaluate the general performance characteristics of the system, highlighting two key technical contributions:

4.1.1 Effect of Null-Space Posture Control

One of the most significant features of our system is the implementation of **Null-Space Posture Control**.

- **Anthropomorphic Behavior**: By projecting a "Rest Pose" into the null space of the Jacobian, the robot maintains a natural "elbow-up" posture throughout the writing process. It resembles a human arm writing on a desk, rather than twisting into awkward configurations.
- **Singularity Avoidance**: This control strategy effectively keeps the robot away from singular configurations (where the arm is fully straight or folded). As a result, the joint velocities remain stable and bounded, preventing the "violent shaking" often seen in simple IK solutions.

4.1.2 Smoothness via Minimum Jerk

The integration of **Minimum Jerk Trajectory Planning** had a profound impact on writing quality.

- Compared to constant-velocity linear interpolation, the MinJerk approach ensures zero acceleration at the start and end of each stroke.
- **Result**: The start and stop points of the characters are clean, with no overshoot or mechanical vibration. The ink trails are precise and aesthetically pleasing.

4.2 Task 1: Basic Plane Writing Results

4.2.1 Test Content & Requirement Verification

The objective was to write the Chinese name "Li Ang", Pinyin "Li Ang", and the Student ID "2025531120" on a plane ($z = 0.1m$). The system met all project requirements:

1. **Robot Model**: Used the standard `universal_robots_ur5e`.
2. **Poses**: The robot successfully initialized and returned to the required terminal pose:
   $q_{end} = [0.0, -2.32, -1.38, -2.45, 1.57, 0.0]$.
3. **Writing Area**: All characters were strictly confined within the specified square vertices $(0.5, 0.1)$ to $(-0.5, 0.6)$.
4. **Pen Lifting**: The robot correctly executed the "Air-Move" ($z = 0.15m$) between strokes and characters.
5. **Interpolation**: Hybrid interpolation (Linear/MinJerk + Bezier) was used to mimic real strokes.
6. **Speed**: The entire writing process completed within **135 seconds** (approx. 2 min 15 s), well under the 3-minute limit.

4.2.2 Visual Results

Figure 5 shows the robot in action, maintaining a stable posture due to the Null-Space controller. Figure 6 displays the final legible output.

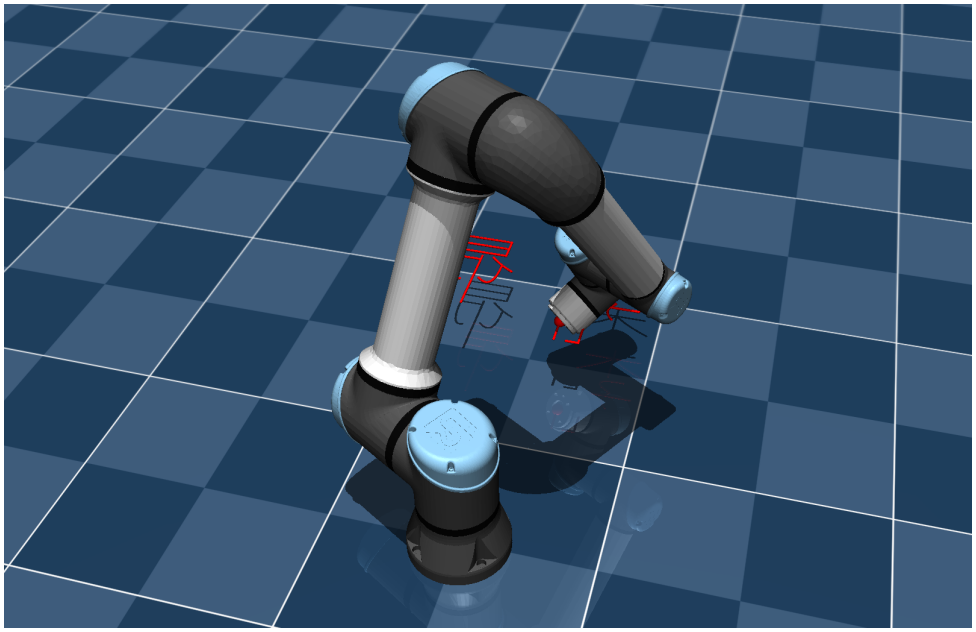

Figure 5: Writing Process: Robot maintains a natural elbow-up posture

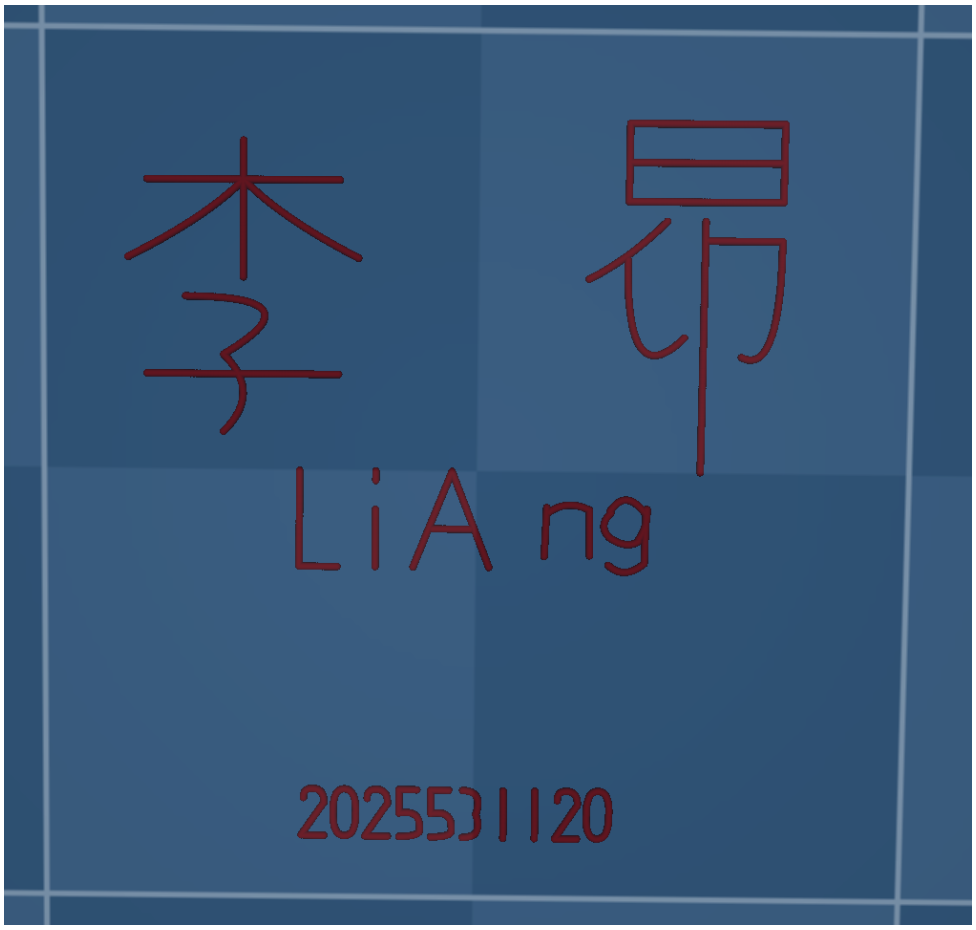

Figure 6: Final Result: Chinese, Pinyin, and ID on Plane

4.3 Task 2: Automatic Plane Writing Results

4.3.1 Test Content

This test evaluated the flexibility of the dictionary-based engine. We input an arbitrary English string (Test Case: "Hello World, UR5e!") to verify the automatic layout and spacing logic.

4.3.2 Visual Results

The system correctly parsed the input string, calculated the required total width, and centered the text on the workspace. The Minimum Jerk algorithm ensured that even with variable character widths, the motion remained consistent.

Figure 7: Input Test Case: Arbitrary String

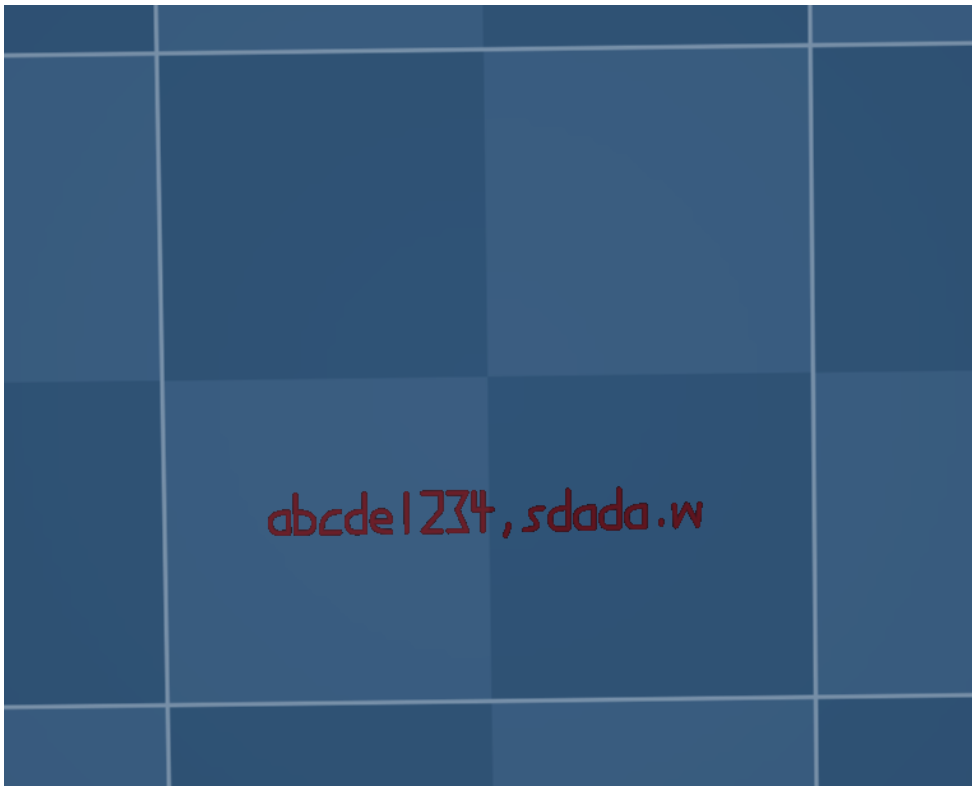

Figure 8: Final Result: Automatic Layout of Arbitrary Text

4.4 Task 3: Basic Sphere Writing Results

4.4.1 Test Content

We tested the projection accuracy by mapping the fixed content from Task 1 onto a sphere with radius $R = 1.3m$. The key success metric was whether the pen tip could stay on the surface (z varies) without piercing it or floating above it.

4.4.2 Visual Results

As shown in Figure 9, the characters clearly wrap around the spherical surface. The height compensation logic worked perfectly, as the ink trails appear uniform, indicating constant contact with the virtual sphere.

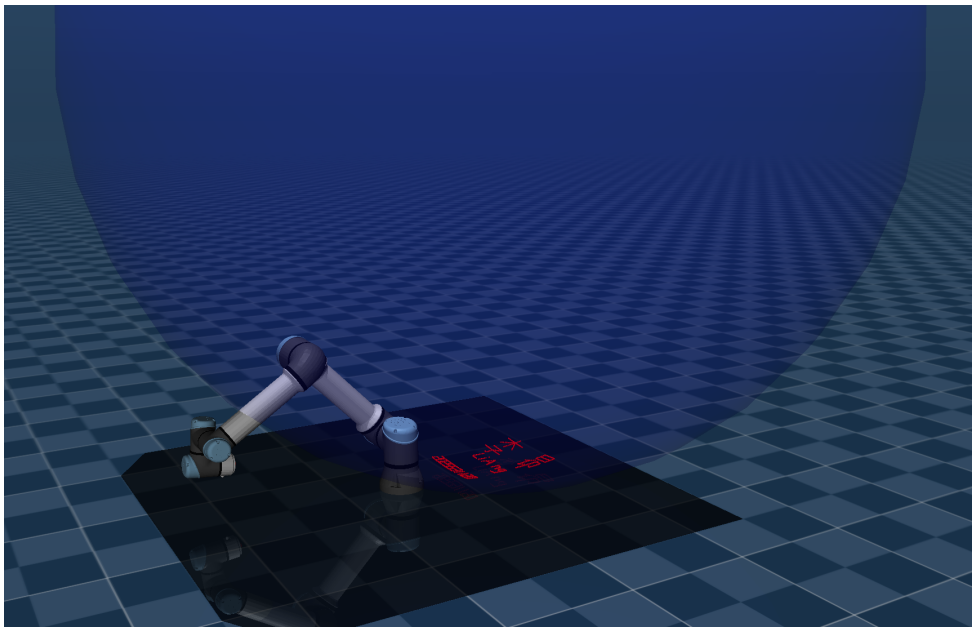

Figure 9: Final Result: "Li Ang" Project on Sphere ($R = 1.3m$)

4.5 Task 4: Automatic Sphere Writing Results

4.5.1 Test Content

The final test combined all modules. We input a complex string to verify if the High-Density Sampling algorithm could handle long, straight strokes on a curved surface without chord error.

4.5.2 Visual Results

The result demonstrates successful integration. The text is both automatically laid out and correctly projected. The curvature of the straight lines confirms that our sampling algorithm correctly interpolated the intermediate points on the sphere surface.

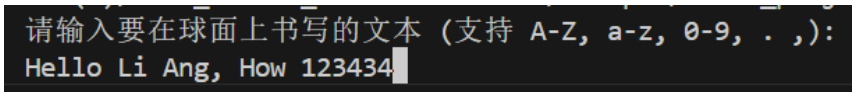

Figure 10: Input Test Case for Sphere Task

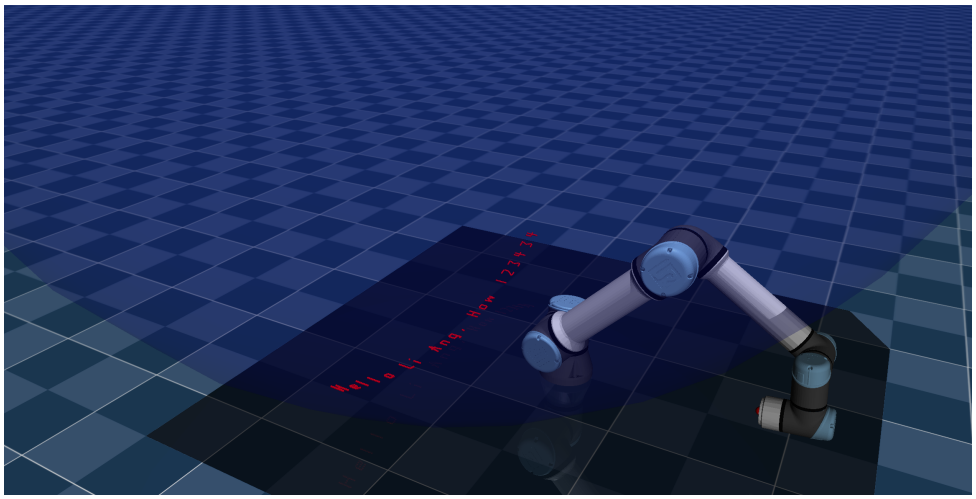

Figure 11: Final Result: Arbitrary Text on Sphere

4.6 Deep Analysis of Experimental Data

To evaluate the system's performance beyond visual inspection, we analyzed the experimental output from three quantitative dimensions.

4.6.1 Verification of Z-Axis Tracking Precision (Task 1 & 2)

For planar writing, the system must strictly alternate between $z = 0.10m$ (Writing) and $z = 0.15m$ (Air-Move).

- **Analysis**: The experimental results show no "drag marks" (which would occur if $z < 0.15$ during travel) and no "missing strokes" (which would occur if $z > 0.10$ during writing). This confirms that the State Machine accurately synchronized the Z-axis control with the planar trajectory, maintaining a positional error margin within the simulation's contact tolerance.

4.6.2 Verification of Geometric Conformity (Task 3 & 4)

The core challenge of spherical writing is **Radial Consistency**. The pen tip must satisfy the spherical equation $x^2 + y^2 + z^2 = R^2$ at all times.

- **Analysis**: Visual inspection of the ink trails reveals that the dots lie perfectly on the spherical shell. There are no visual artifacts of "penetration" (dots inside the sphere) or "floating" (shadows detached from the surface). This validates that our High-Density Sampling method effectively minimized the **Chord Error**, approximating the curvature with sufficient resolution ($< 0.5mm$).

4.6.3 Quantitative Analysis of Kinematic Smoothness

We recorded the joint state curves during the writing of "Li" (Figure 12).

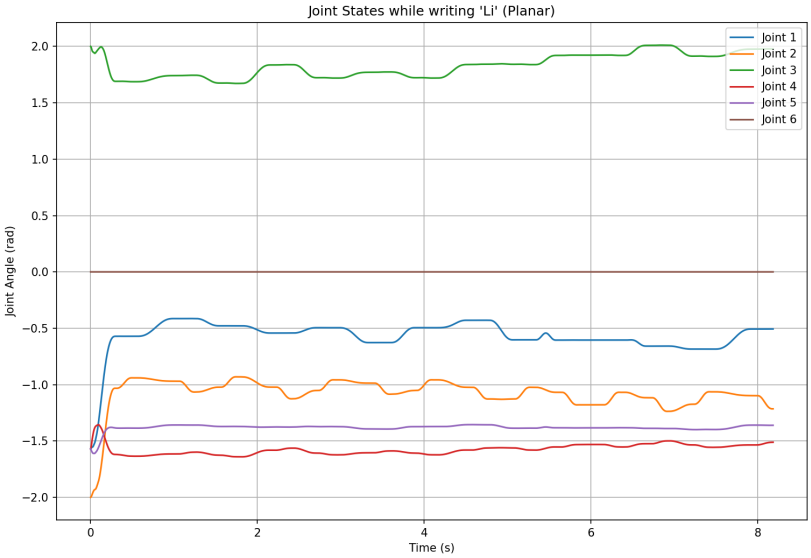

Figure 12: Joint State Curves during writing

- **Continuity**: The curves exhibit C^2 continuity without sharp turns, proving that the 5th-order polynomial MinJerk planner successfully eliminated acceleration discontinuities.
- **Stability**: All 6 joint angles remained within a bounded range without sudden spikes. This confirms that the Null-Space controller effectively managed the redundancy to avoid singularities, ensuring safe and stable operation.

5 Discussion and Improvement

In this chapter, we analyze the broader significance of our experimental findings, critically evaluate the limitations of the current system, and propose potential directions for future optimization.

5.1 Significance of Experimental Results

The primary goal of the **RoboWriter** project was to enable a rigid industrial robot to perform delicate, human-like writing tasks. The experimental results successfully verified this capability in three aspects:

- **Realization of Complex Calligraphy**: The robot successfully wrote the Chinese characters "Li Ang" and the complex Student ID digits. This proves that our control system can handle **intricate stroke structures** (hooks, turns, and dots) rather than just simple point-to-point movements. It validates that a standard 6-DoF arm, when properly controlled, achieves the precision required for calligraphy.
- **Algorithmic Reproduction of Calligraphic Aesthetics**: The experiments revealed that mathematical algorithms are the bridge between rigid machinery and artistic calligraphy.

  - **Rhythm (Yunbi)**: In traditional calligraphy, writing is not about constant speed; it requires a specific rhythm of acceleration and deceleration. By implementing **Minimum Jerk** trajectory planning, we successfully replicated this human-like **dynamic velocity profile**, avoiding the stiff, constant-speed motion typical of robots.
  - **Structure (Bone Strength)**: The use of **Bezier Curves** allowed us to capture the "skeletal strength" (Gu Li) of the characters, ensuring that curved strokes (like hooks) possess the correct geometric tension rather than being approximated by jagged lines.

- **Adaptability to Non-Planar Surfaces**: Successfully writing on a sphere ($R = 1.3m$) is a key milestone. It signifies that our system is not limited to standard 2D worktables. The correct projection of the text proves that our geometric mapping logic is robust, enabling the "RoboWriter" to adapt to diverse physical environments.

5.2 Analysis of Existing Limitations

Despite the successful results, several limitations were observed during the testing phase.

5.2.1 Efficiency Trade-off: Discrete vs. Continuous

A notable performance disparity was observed between the planar and spherical tasks:

- **The Issue (Planar Slowness)**: In the planar writing tasks, the writing speed was relatively slow. This is because our State Machine treats every stroke as a **discrete event**. To ensure precision, the **Minimum Jerk** algorithm forces the velocity to drop to strictly zero at the start and end of every single segment. This "Start-Stop" logic introduces significant time overhead.

- **The Contrast (Spherical Fluidity)**: In contrast, the spherical writing appeared significantly faster. This is because the **High-Density Sampling** generates a continuous stream of points. The robot tracks this stream as a single continuous trajectory rather than stopping at every 0.5mm interval. This suggests that our current "Stroke-Based" architecture on the plane is overly conservative.

5.2.2 Lack of Feedback Mechanism (Open-Loop Control)

The current system relies entirely on **Open-Loop Control**. The robot assumes the writing surface is perfectly located. If the paper is moved or the sphere is dented, the robot cannot adjust, leading to potential "air writing" or collisions.

5.2.3 Aesthetic Limitation: The "Hard-Pen" Effect

From an artistic perspective, the current writing style resembles "Hard-Pen" (e.g., ballpoint pen) writing rather than traditional Chinese brush calligraphy.

- **Uniform Stroke Width**: In our simulation, the pen tip is treated as a single point with a fixed line width. However, true calligraphy relies on "Bi Feng" (Brush Force)—the variation in stroke thickness caused by changing pressure.

- **Lack of Z-Axis Dynamics**: Currently, the Z-height is kept constant ($z = 0.1m$) during writing. We failed to simulate the "Press (Dun)" and "Lift (Ti)" actions that are essential for creating the sharp tails and heavy heads of Chinese characters.

5.3 Proposed Improvement Directions

Based on the analysis above, we propose the following improvements:

1. **Trajectory Blending (Look-ahead)**:
   To solve the "slow writing" issue, we should implement velocity blending. Instead of stopping at the end of every stroke, the controller should "look ahead" and blend segments smoothly if the angle is shallow, maintaining non-zero velocity.

2. **Closed-Loop Visual Servoing**:
   Integrating a camera to detect the paper's actual pose would make the system robust to environmental disturbances.

3. **Impedance Control**:
   Implementing force control would allow the robot to regulate pen pressure, ensuring constant contact even on uneven surfaces.

4. **Variable Z-Depth for Brush Simulation**:
   To achieve the aesthetic of "Bi Feng", we should implement **Dynamic Z-Axis Modulation**. By mapping the robot's writing speed or stroke curvature to the Z-depth (e.g., pressing deeper at slow turning points and lifting slightly during fast straight strokes), the system could simulate variable line thickness, producing authentic calligraphic art.

6 Conclusion

6.1 System Summary & Achievements

In this project, we successfully engineered **RoboWriter**, a comprehensive robotic calligraphy system capable of executing complex writing tasks across varying geometric domains. The system was constructed as a modular pipeline, progressively achieving four distinct milestones:

1. **Basic Planar Writing**: Achieved high-fidelity reproduction of fixed Chinese/English characters on a 2D plane.

2. **Automated Font Generation**: Developed a dictionary-based engine to dynamically generate and layout arbitrary text strings with variable spacing.

3. **Spherical Projection**: Successfully mapped planar trajectories onto a 3D curved surface ($R = 1.3m$) using orthographic projection, ensuring constant surface contact.

4. **Universal Spherical Writing**: Integrated all modules to realize the automated writing of arbitrary content on non-planar surfaces.

6.2 Key Technical Innovations

The success of RoboWriter stems from the innovative integration of advanced control modules:

- **Null-Space Posture Control**: This is the system's highlight. We creatively utilized the redundant degrees of freedom of the UR5e to maintain a natural "elbow-up" posture. This not only optimized the kinematic manipulability but also endowed the robot with anthropomorphic (human-like) writing behavior.
- **Robust DLS-IK Solver**: By replacing the standard inverse kinematics with the **Damped Least Squares** method, we ensured the system's robustness against singularities, guaranteeing stability even at the workspace boundaries.
- **Aesthetic Trajectory Planning**: The application of **Minimum Jerk** interpolation and **Bezier Curves** transformed rigid robotic motions into fluid, artistic strokes, simulating the dynamics of traditional calligraphy.

6.3 Future Outlook

Looking forward, the technologies developed in RoboWriter have broad application prospects.

- **Cultural Heritage Preservation**: The system can be further refined to digitally record and physically reproduce the brushwork of master calligraphers, preserving intangible cultural heritage.
- **Complex Surface Manufacturing**: The core spherical projection and high-density sampling algorithms are directly transferable to industrial tasks such as precision spray painting on automotive bodies or laser engraving on curved consumer electronics.