Model Based Trajectory Control and Planning for a Magnetic Whiteboard Cleaning Robot

Nathan Sun[1], Xinxuan Tang[1], Xinwen Xu[1], Zhuowei Xu[1], Yunliang Zhao[1]
[1]Mechanical Engineering Department, Carnegie Mellon University, Pittsburgh, PA, USA

***Abstract*—The Whiteboard Intelligent and Programmable Erasing Robot (WIPER) is a magnetic adhesion, dual-motor-driven robot designed for autonomous whiteboard cleaning. The robot uses its erasers both as passive supports and as active end effectors. Initial experiments using a proportional integral derivative (PID) controller revealed limitations in position accuracy and robustness to disturbances. To address these issues, two advanced model-based control strategies were developed and experimentally evaluated: an improved Linear Quadratic Regulator (LQR) and a Model Predictive Controller (MPC). Real-time position tracking was implemented using an Intel RealSense camera combined with AprilTag detection, providing accurate state estimation for closed-loop control. The experimental results demonstrate that the LQR and MPC approaches outperform the baseline PID approach, with MPC being more robust. Furthermore, there is a computer vision based path planner to identify and plan the cleaning paths for WIPER to follow. This report presents the modeling, control design, and experimental evaluation of the improved WIPER control and planning system.**

I. Introduction

Whiteboards are widely used in classrooms, meeting rooms, and collaborative workspaces due to their ease of use, reusability, and low operational cost [1]. However, they require frequent manual cleaning, which can be labor-intensive and disruptive in high-usage environments. While self-cleaning whiteboards are commercially available, they tend to be prohibitively expensive and often require replacing existing installations. This creates a practical need for an add-on erasing system that is easy to deploy and capable of relieving users from repetitive cleaning tasks.

Unlike general-purpose cleaning or inspection robots [2], [3] that operate on horizontal floors or structured environments, a whiteboard erasing robot must function reliably on vertical surfaces under gravitational constraints [4]–[6]. It must not only maintain stable motion while adhering to the board but also perform effective erasing over unstructured, ink-marked regions with varying texture and reflectivity. Additionally, safe human–robot interaction is critical in shared environments, requiring the system to be compact, non-intrusive, and secure during operation [7].

To address these requirements, we propose WIPER—the Whiteboard Intelligent and Programmable Erasing Robot. WIPER is a compact autonomous robot designed to automate the whiteboard cleaning process in educational and office settings. It adheres to vertical whiteboard surfaces using neodymium magnets, which offer a balance of holding force and maneuverability. The robot is actuated by a pair of rear-mounted DC motors in a differential drive configuration, with passive front casters enhancing stability during motion.

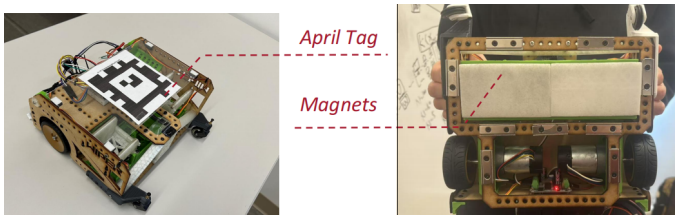

Fig. 1: Isometric (left) and bottom (right) views of WIPER highlighting the AprilTag marker, neodymium magnets, differential-drive wheels, front casters, melamine erasers, and chassis layout.

WIPER's erasing mechanism consists of two reusable melamine erasers mounted on servo-actuated arms. This setup enables selective engagement with the board surface for region-specific cleaning, reducing unnecessary wear and improving efficiency. The system supports programmable path execution, allowing it to follow predefined trajectories to target high-use zones or perform full-surface sweeps. Through consistent and autonomous operation, WIPER improves classroom turnover, reduces reliance on manual cleaning, and promotes a more efficient learning and working environment.

An Intel RealSense D435 depth camera is integrated into the system along with AprilTag markers to support visual localization and quadrant-based path planning. These components illustrate the system's extensibility and lay the groundwork for future integration of vision-guided cleaning strategies. However, the present work focuses on the underlying motion and velocity control framework, operating independently of external localization. The emphasis is on the accurate execution of preplanned trajectories using motor feedback and a model-based control approach. The localization setup, including AprilTags and the RealSense camera, is shown in Fig. 2.

This report presents the development and application of a linearized kinematic model in conjunction with a Linear Quadratic Regulator (LQR) to improve WIPER's trajectory tracking precision and responsiveness. The control framework is designed to accommodate motor asymmetry, dynamic uncertainty, and safety constraints. Broader considerations such as constraint enforcement and migration toward advanced techniques, including Time-Varying LQR (TVLQR) and Model Predictive Control (MPC), are discussed as part of the system's

Fig. 2: Experimental setup showing WIPER mounted on the whiteboard with AprilTag markers for localization and an Intel RealSense D435 camera for visual tracking. The cable provides power and data connectivity during testing.

continued refinement.

II. Background & Motivation

The WIPER system was developed to automate the repetitive and inconsistent task of whiteboard cleaning in educational and professional environments. While the hardware platform—featuring magnetic adhesion, dual-motor differential drive, and modular eraser mounts—demonstrated promising baseline functionality, the initial control implementation using a Proportional-Integral-Derivative (PID) controller revealed critical limitations. These included poor trajectory accuracy, inconsistent motion execution, and sensitivity to system disturbances, motivating the adoption of more robust model-based control strategies.

The deployed controller initially relied solely on an integral gain (K_i), which enabled WIPER to gradually correct positional drift but resulted in a slow response to disturbances and transient commands. Attempts to incorporate proportional and derivative gains (K_p and K_d) to achieve a more responsive PD control structure frequently led to instability. In particular, increasing these gains often caused reverse direction motion or trajectory overshoot, suggesting a mismatch between the low-level control behavior and the physical dynamics of the system.

Additional challenges stemmed from high system latency within the control loop. Control commands were computed on a host PC and transmitted via Bluetooth to an onboard Arduino Nano microcontroller responsible for actuator interfacing. This wireless link introduced significant delays, which—when combined with the limited computational resources of the microcontroller—resulted in poor closed-loop timing characteristics. Consequently, the system exhibited erratic path-tracking behavior and was unable to reliably follow even simple motion profiles.

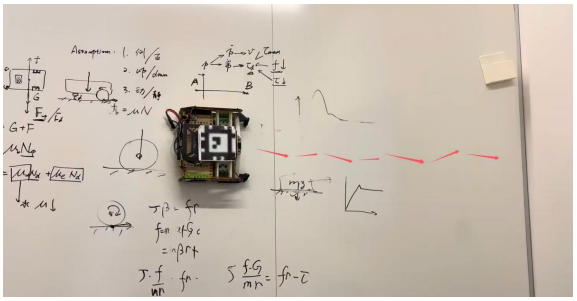

Fig. 3: Recorded trajectory of WIPER under the original PID controller. The motion is unstable and oscillatory due to poor gain tuning and latency in the control loop.

These observations motivated a shift away from reactive, loosely tuned feedback control toward a more principled, model-based approach. Linear Quadratic Regulator (LQR) control was adopted to enhance trajectory stability, leverage a simplified dynamics model, and prepare the control stack for integration with higher-level planners. The control strategy emphasized direct feedback through velocity commands to the motors, enabling more deterministic and tunable performance, even in the presence of communication-induced delays.

III. Dynamics Modeling

The WIPER robot is modeled as a planar differential-drive system with non-linear kinematics. Although its motion resembles that of the Dubins car model, which assumes forward-only motion at constant speed, the robot differs in its ability to operate bidirectionally. This capability is enabled by independently actuated, reversible motors, allowing in-place rotation, backward motion, and tighter path tracking. Such maneuverability is particularly beneficial for constrained tasks like whiteboard cleaning, where efficient reorientation and coverage are essential.

The continuous-time state and control vectors are defined as:

$$x = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}, \quad u = \begin{bmatrix} v \\ \omega \end{bmatrix}$$

where x, y are planar positions, θ is the robot's heading angle, v is the linear velocity, and ω is the angular velocity. The governing dynamics equations are given by:

$$\dot{x} = f(x, u) = \begin{bmatrix} v\cos(\theta) \\ v\sin(\theta) \\ \omega \end{bmatrix}$$

These equations assume ideal rolling without slip and serve as the foundation for both simulation and model-based control design.

To enable trajectory-dependent feedback control, the continuous-time dynamics are discretized using a fourth-order Runge-Kutta (RK4) method. Local linearization is performed around reference points (x_k, u_k), where the system Jacobians

A_k and B_k are computed numerically. These matrices describe the sensitivity of the state trajectory to small perturbations in state and input, and are central to the control formulation presented in Section IV.

IV. TVLQR Controller

A Time-Varying Linear Quadratic Regulator (TVLQR) controller was implemented to enable accurate trajectory tracking under nonlinear kinematic dynamics. Reference trajectories and associated feedforward inputs are first generated using geometric primitives and numerical differentiation. The nonlinear dynamics are discretized using a fourth-order Runge-Kutta method, and local linearizations are obtained by computing numerical Jacobians around reference states and inputs. A backward Riccati recursion is then performed to compute a time-varying sequence of state-feedback gains. The resulting controller combines these gains with feedforward commands to produce motor velocity targets at each timestep.

A. Reference Trajectory Generation

Reference trajectories are constructed from geometric motion primitives, with selectable modes including straight lines, circular arcs, and elbow turns. Each trajectory is discretized into $N = 50$ timesteps, yielding a sequence of reference states $\{x_{\text{ref},k}\}_{k=0}^{N-1}$ and corresponding feedforward control inputs $\{u_{\text{ref},k}\}_{k=0}^{N-1}$. The heading angle θ is defined such that $\theta = 0$ corresponds to upward, camera-facing motion. Finite differences of position and orientation are used to approximate the forward velocity v_k and angular velocity ω_k at each timestep.

To ensure smooth and physically feasible motion, the reference trajectory is time-parameterized using acceleration-continuous profiles based on a trapezoidal velocity shape. The velocity v_t increases linearly to a nominal cruising speed v_0, remains constant during the mid-phase, and then decelerates symmetrically to rest. The piecewise definition of v_t is:

$$v(t) = \begin{cases} a_{\max} t, & 0 \le t < t_a \\ v_0, & t_a \le t < t_d \\ v_0 - a_{\max}(t - t_d), & t_d \le t \le T \end{cases} \tag{1}$$

where

$a_{\max}$ is the maximum allowable acceleration,
$t_a = \dfrac{v_0}{a_{\max}}$ is the acceleration phase duration,
$t_d = T - t_a$ is the deceleration onset time,
T is the total trajectory duration.

The linear displacement $s(t)$ is obtained by integrating $v(t)$, and the reference position at timestep k is determined by evaluating the curve at arc length $s_k = s(k\Delta t)$, where $\Delta t = T/N$. Angular velocity ω_k is computed similarly by differentiating the heading angle over time, and the feedforward input at each step is given by:

$$u_{\text{ref},k} = \begin{bmatrix} v_k \\ \omega_k \end{bmatrix} \tag{2}$$

B. Control Design

At each timestep k, the nonlinear dynamics are discretized using RK4 with $\Delta t = 0.05$. Around each reference point, we compute local linearizations using numerically approximated Jacobians:

$$A_k = \left.\frac{\partial f}{\partial x}\right|_{x_k, u_k}, \quad B_k = \left.\frac{\partial f}{\partial u}\right|_{x_k, u_k}$$

Analytical forms of these Jacobians (derived in Section III) are:

$$A_k = \begin{bmatrix} 0 & 0 & -v\sin(\theta) \\ 0 & 0 & v\cos(\theta) \\ 0 & 0 & 0 \end{bmatrix}, \quad B_k = \begin{bmatrix} \cos(\theta) & 0 \\ \sin(\theta) & 0 \\ 0 & 1 \end{bmatrix}$$

Using these matrices, we solve a backward Riccati recursion to obtain time-varying gains K_k. The objective is to minimize the following quadratic cost:

$$\begin{aligned} J = \sum_{k=0}^{N-1} \Big(& (x_k - x_{\text{ref},k})^T Q (x_k - x_{\text{ref},k}) \\ & + (u_k - u_{\text{ref},k})^T R (u_k - u_{\text{ref},k}) \\ & + \alpha (u_{k,0} - v_{\text{schedule},k})^2 \\ & + 0.01\, u_{k,1}^2 \Big) \\ & + (x_N - x_{\text{ref},N})^T Q_f (x_N - x_{\text{ref},N}) \end{aligned} \tag{3}$$

where

$$Q = \text{diag}(1, 1, 10), \quad R = \text{diag}(0.1, 0.1), \quad Q_f = 10 \cdot Q$$

Each term in J has the following meaning:

- $(x_k - x_{\text{ref},k})^T Q (x_k - x_{\text{ref},k})$: penalizes deviation from the desired state.
- $(u_k - u_{\text{ref},k})^T R (u_k - u_{\text{ref},k})$: penalizes deviation from the nominal control input.
- $\alpha (u_{k,0} - v_{\text{schedule},k})^2$: enforces adherence to a desired linear velocity profile.
- $0.01\, u_{k,1}^2$: regularizes angular velocity to prevent aggressive turning.
- $(x_N - x_{\text{ref},N})^T Q_f (x_N - x_{\text{ref},N})$: terminal cost penalizing final state error.

The optimal control law is given by:

$$u_k = u_{\text{ref},k} - K_k (x_k - x_{\text{ref},k})$$

Heading errors $\theta_k - \theta_{\text{ref},k}$ are wrapped to $[-\pi, \pi]$ to avoid discontinuities.

C. Execution and Actuation

The computed $[v, \omega]$ commands are converted to motor RPMs as follows:

$$v_L = v + \frac{L}{2}(-\omega), \quad v_R = v - \frac{L}{2}(-\omega)$$

$$\text{RPM}_{\{1,2\}} = \frac{v_{\{L,R\}}}{2\pi r} \cdot 60$$

where $L = 0.215$ m and $r = 0.03$ m. A negative sign is applied to ω to correct for asymmetrical motor mounting.

The control loop runs every 5 ms. The controller uses camera-based localization (AprilTags) when available, falling back to IMU-based dead reckoning otherwise. A proximity gate ensures the step counter advances only when the robot is sufficiently close to the current reference:

$$\|x_k - x_{\text{ref},k}\| < 0.02 \text{ m}$$

D. TVLQR Results

A series of six test trials were conducted to evaluate the performance of the TVLQR controller on 0.5-meter horizontal trajectories. Each test iteratively addressed a specific failure mode:

- **Test 1:** Fast loop rate (0.001 s delay) caused overshooting due to motor lag.
- **Test 2:** Increased delay to 0.01 s improved stability, but angular correction remained weak.
- **Test 3:** Increased heading weight in the cost matrix: $Q = \text{diag}(1, 1, 10)$, improving directional control.
- **Test 4:** Tight 1 cm proximity threshold stalled progress. Relaxed to 2 cm for smoother step transitions.
- **Test 5:** AprilTag occlusion caused a large IMU drift. Relocating the tag prevented the loss of visual feedback.
- **Test 6:** Final configuration with $\Delta t = 0.05$, $N = 30$ yielded the most stable and smooth performance.

Figure 4 shows the recorded trajectory for a representative trial using the final configuration. The robot's wheel velocity difference was affected by the gravity and the offline planner could not make real-time adjustments to correct to the reference path from left to right with minimal lateral deviation and consistent heading alignment.

While the TVLQR controller demonstrated improved tracking over PID, its offline gain schedule and lack of adaptive speed control limited robustness. These findings motivated the adoption of a receding-horizon MPC framework, enabling online optimization and real-time disturbance rejection.

V. Model Predictive Control

While the TVLQR controller provided improved tracking and speed control over PID, its reliance on a precomputed, open-loop gain schedule limited adaptability under disturbances and localization drift, especially during gravity affected horizontal climbing.

To address these limitations, a Model Predictive Control (MPC) framework was developed that reoptimizes control actions online over a moving horizon. Meanwhile, in order to further enhance the performance of the controller, we utilized the CasADi [8] library to linearize the original dynamic equations, which is an open-source symbolic computation framework designed particularly for numerical simulation and optimization of nonlinear dynamic systems.

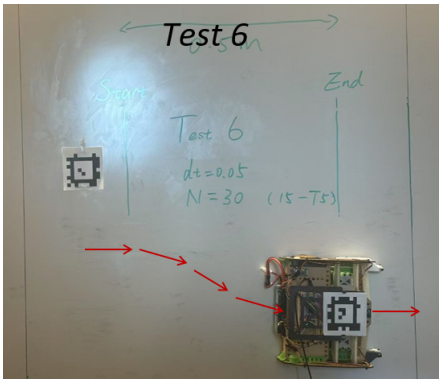

Fig. 4: Trajectory of WIPER under TVLQR on a 0.5 m horizontal path (left to right). The robot demonstrates stable convergence to reference states under tuned parameters. The red arrows represent the trajectory and headings of WIPER over time.

A. Control Formulation

The MPC controller discretizes the robot's nonlinear dynamics using the RK4 method. Around each reference point (x_k, u_k), local linearizations are computed to generate time-varying dynamics:

$$x_{k+1} = A_k x_k + B_k u_k$$

At each control step, a finite-horizon quadratic program with a horizon set at 20 steps is solved to minimize the following cost:

$$\begin{aligned} J = \sum_{k=0}^{N-1} \Big(& (x_k - x_{\text{ref},k})^T Q (x_k - x_{\text{ref},k}) \\ & + (u_k - u_{\text{ref},k})^T R (u_k - u_{\text{ref},k}) \\ & + \alpha (u_{k,0} - v_{\text{schedule},k})^2 \\ & + 0.001\, u_{k,1}^2 \Big) \\ & + (x_N - x_{\text{ref},N})^T Q_f (x_N - x_{\text{ref},N}) \end{aligned} \tag{4}$$

where $v_{\text{schedule},k}$ is the nominal feedforward velocity, and α is an optional regularization weight. The matrices are:

$$Q = \text{diag}(1, 1, 10), \quad R = 0.1 \cdot I, \quad Q_f = 50 \cdot Q$$

Wheel velocity constraints are imposed to ensure actuator limits are respected:

$$v_{\text{wheel},\{L,R\}} \in \left[\frac{\text{rpm}_{\min}}{60} \cdot 2\pi r, \frac{\text{rpm}_{\max}}{60} \cdot 2\pi r \right]$$

where $r = 0.03$ m is the wheel radius and $L = 0.215$ m is the wheelbase.

The optimization is solved at each step using OSQP.

B. Execution and Actuation

At runtime, a moving window of future reference points of size $N = 20$ is extracted based on the robot's current state. The MPC solver outputs the first control input u_0, which is then converted into motor RPM commands same in Section IV-C. Control updates are issued every 50 ms.

C. MPC-Based Trajectory Tracking Experiment

1) Simulation Test: We developed a simple simulation algorithm to verify the effectiveness of the algorithm. In this study, four motion modes are evaluated: straight horizontal, an elbow L-shape (horizontal then vertical), a quarter-circle arc, and a sine curve trajectory. Noise is modeled by limiting the maximum wheel speed to 500rpm, enforcing a left-wheel efficiency of approximately 90% relative to the right wheel, inducing a stall below 100rpm, and injecting a random disturbance every 0.05m of travel with greater variance in the y-direction than in the x-direction to emulate camera measurement errors.

As shown in Fig. 5, in the noise-free simulations (top row), MPC and TVLQR trajectories nearly coincide and both track the reference path with negligible differences. When noise is introduced (bottom row), MPC maintains a smooth, stable path with only minor deviations, whereas TVLQR exhibits larger oscillations and drift. This clearly demonstrates that MPC offers significantly greater robustness and stability in the presence of disturbances.

Under identical disturbance and constraint conditions across multiple trajectories, we quantified RMSE (Root Mean Square Error), IAE (Integral of Absolute Error), MaxE (Maximum Error), and Control Effort for both LQR and MPC models, using these metrics as criteria for controller performance evaluation.

MPC demonstrates superior performance over TVLQR across all evaluated metrics. It achieves 53-60% lower RMSE in straight and L-shape trajectories (0.0358/0.0417 vs 0.0762/0.1050) and reduces maximum tracking error by 71% in complex paths (0.0951 vs 0.3296). The integrated absolute error (IAE) is consistently better for all motion patterns, confirming MPC's tracking advantage.

2) Real Test: To validate the effectiveness of the proposed Model Predictive Control (MPC) strategy, we conducted an experiment in which the WIPER robot was commanded to follow a predefined 1-meter vertical trajectory on the whiteboard surface. The reference trajectory was designed to maintain a constant heading angle while translating downward at a uniform speed.

As shown in Fig. 6, the top row presents snapshots of the robot's motion captured at 10-second intervals. The robot maintained firm adhesion to the vertical whiteboard and executed smooth, continuous motion throughout the trial. The bottom plots compare the actual trajectory against the reference in all three state dimensions (x, y, and θ). The WIPER closely followed the planned path, with only minor deviations in heading and lateral drift, which remained within acceptable margins for successful erasing.

TABLE I: Performance Comparison of MPC and TVLQR Controllers in Simulator

Trajectory Mode	**RMSE**		**IAE**	
	MPC	TVLQR	MPC	TVLQR
Straight horizontal	0.0358	0.0762	0.3721	0.9367
L-shape (horiz. + vert.)	0.0417	0.1050	0.5030	1.1755
Quarter-circle arc	0.0570	0.0641	0.8310	1.1442
Sine curve	0.0944	0.0767	1.4788	1.6750

Trajectory Mode	**MaxE**		**Effort**	
	MPC	TVLQR	MPC	TVLQR
Straight horizontal	0.0721	0.0878	1.8238	0.9972
L-shape (horiz. + vert.)	0.0951	0.3296	2.9472	14.0603
Quarter-circle arc	0.1549	0.0925	2.9200	2.4621
Sine curve	0.4248	0.1347	24.3751	13.6543

Additionally, the whiteboard surface was cleanly erased along the entire path, confirming that the robot's motion and arm actuation were well synchronized. This experiment demonstrates that the MPC controller enables reliable trajectory tracking under real-world conditions and ensures consistent cleaning performance, validating both the control design and mechanical robustness of the WIPER system.

VI. Path Planner

To enable autonomous traversal across a desired path, we developed a simple yet effective path planning module based on visual input. Rather than generating paths from a global map, our system extracts an explicit path from a pre-existing visual line (e.g., a black marker line on a whiteboard) captured by a front-facing RGB-D camera.

A. Image-Based Path Extraction

We implemented a vision-based path extraction pipeline using the RealSense D435 RGB-D camera, which provides both color and depth frames. The color image is first converted to grayscale and thresholded to generate a binary mask that highlights the inked regions. Morphological operations are applied to reduce noise and close gaps. The processed mask is then skeletonized to extract a 1-pixel-wide path centerline.

Endpoints along the skeleton are identified based on the number of neighboring nonzero pixels, and a greedy search algorithm is used to traverse the skeleton between endpoints, producing an ordered sequence of pixel coordinates representing the path.

However, due to the freehand nature of the drawn curve, the grayscale intensity along the ink trace exhibits local discontinuities, especially in low-pressure strokes and near overlapping regions. These variations result in fragmented skeletons or spurious branches, introducing inaccuracies in the extracted path geometry. Fig. 7 shows an example of a successfully extracted path contour in red.

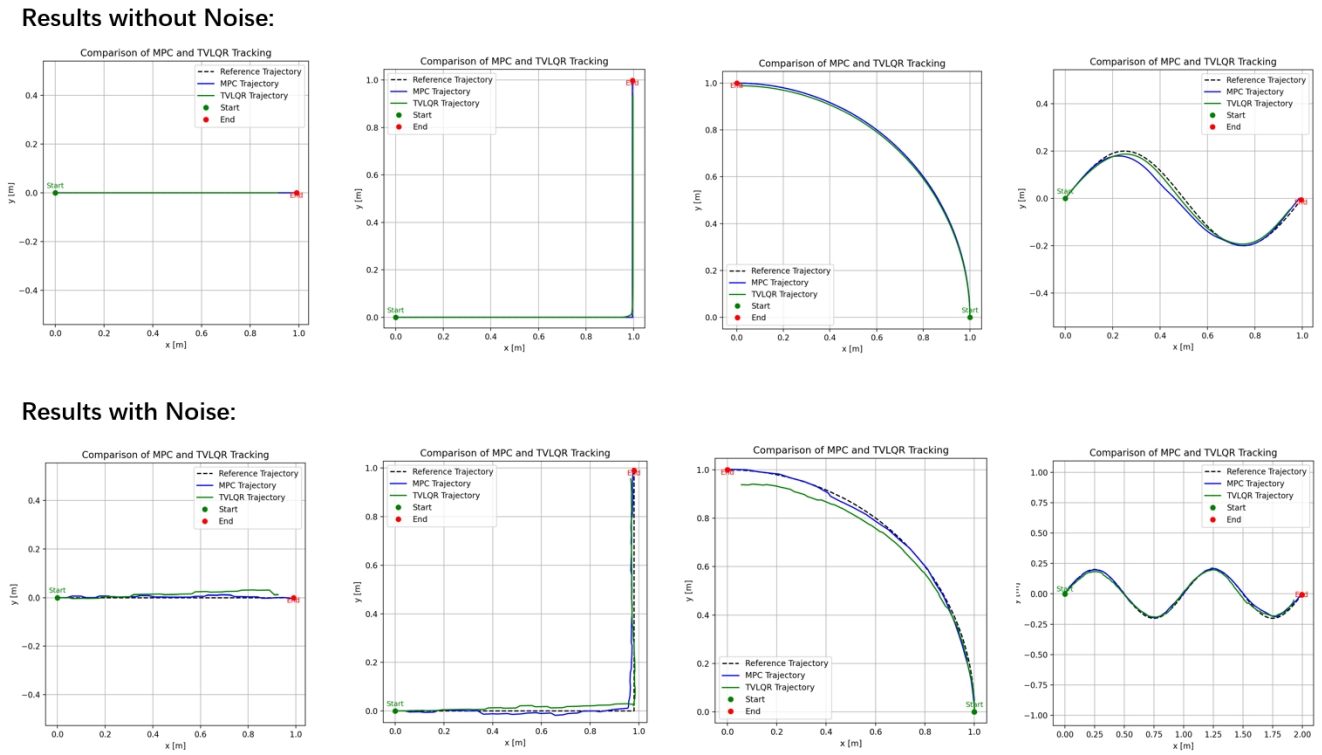

Fig. 5: Comparison of MPC (blue) and TVLQR (green) trajectories against the reference path (dashed) across six motion modes. Top row: noise-free conditions yield nearly identical tracking performance. Bottom row: with measurement noise, MPC maintains a smoother, more stable trajectory while TVLQR exhibits larger oscillations and drift.

B. 3D Reconstruction and Time Parameterization

Each 2D pixel along the extracted path is back-projected into 3D space using the depth map and intrinsic calibration of the camera. The resulting point cloud forms a spatial trajectory, which is then time-parameterized using a trapezoidal velocity profile. The profile consists of a linear acceleration phase up to a nominal velocity v_0, a constant-speed segment, and a deceleration phase back to zero velocity. Acceleration is limited to a maximum value $a_{\max}$, ensuring physically realizable motion.

This process yields a sequence of timestamped 3D waypoints that define the reference trajectory for downstream planning or control modules.

C. Limitations in Deployment

While the vision-based pipeline performed well in simulation and controlled indoor conditions, several challenges emerged during real-world deployment. The deprojected 3D trajectory could not be robustly transferred to the WIPER's onboard system due to multiple factors, including inconsistent depth readings, misalignment between the camera and robot frames, and unmodeled calibration offsets.

Moreover, the variability in marker visibility and the inherent inconsistency in human-drawn curves caused localized grayscale inconsistencies. These led to breaks or ambiguities in the extracted skeleton, further degrading the accuracy of the resulting path. As a result, the vision-based trajectory was ultimately not used in the final hardware experiments, and manually defined geometric paths were employed instead.

VII. Implementation

A. Serial Communication

To support real-time control, we transitioned from an HC-06 Bluetooth module to a direct USB serial connection between the robot and host PC. This upgrade eliminated Bluetooth-induced latency and unreliability, enabling fast and deterministic communication. The system now achieves stable transmission rates above 20 Hz, suitable for model-based controllers such as TVLQR. The communication string was optimized for compactness and parsing efficiency, further increasing throughput and reducing processing delays on the Arduino side.

B. State Estimation and Localization

The robot's control system supports hybrid localization from two sources: an onboard IMU and a RealSense D435 camera using AprilTag detection. The IMU provides low-latency, continuous updates via accelerometer-based dead reckoning, while the AprilTag tracker offers more accurate and drift-free pose measurements when tags are visible.

The system monitors the freshness of the camera-based state, prioritizing it when recent (within 100 ms) and falling back to IMU-based estimation when unavailable. This design

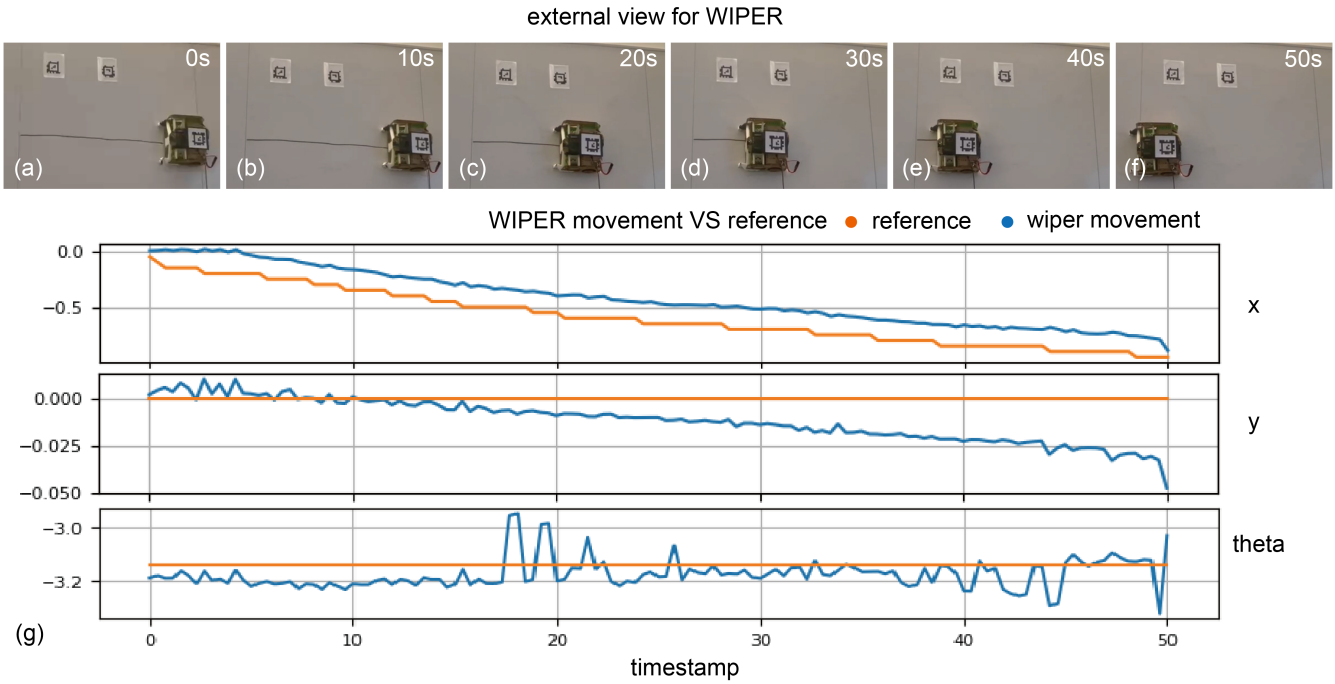

Fig. 6: MPC-based trajectory tracking experiment. (a)–(f): External snapshots of the WIPER robot during a 1-meter vertical erasing task, captured at 10-second intervals. The robot maintains stable vertical motion while adhering to the whiteboard surface. (g): Comparison of actual and reference trajectories along x, y, and θ axes over time. The blue curves represent the WIPER's measured states, and the orange curves denote reference values. The robot closely follows the planned trajectory in x, exhibits minor deviations in y, and maintains bounded orientation error in θ, confirming effective tracking and successful surface cleaning under MPC control.

allows the robot to maintain accurate and robust localization even during temporary tag occlusion or camera dropouts.

C. Real-Time Control Threading

Control is executed in a dedicated real-time loop running at approximately 100 Hz. To maintain consistent timing, the control loop uses a unified timer and schedules its sleep duration dynamically to correct for runtime variability. Other components, such as AprilTag tracking and serial communication, run in separate threads, synchronized via thread-safe shared memory structures.

This architecture ensures consistent controller output intervals, prevents race conditions, and isolates sensor updates from control computation, enhancing both performance and reliability under multi-threaded execution.

VIII. Results

TABLE II: Comparison of Control Strategies on WIPER

Type	Linear Velocity (cm/s)	Trajectory Following	Destination
PID	10.65	Medium	Overshot
TVLQR	4.82	Poor	Missed
MPC	2.65	Good	Reached

In this section, performance of PID, TVLQR, and MPC implemented on WIPER will be discussed.

To quantitatively compare the performance of the three control strategies—PID, TVLQR, and MPC—a representative trial was conducted using a 1-meter vertical trajectory. Key evaluation metrics included linear velocity magnitude, trajectory adherence, and whether the robot successfully reached the intended destination. Table II summarizes the results.

The PID controller achieved the highest speed but lacked stability, leading to destination overshoot. TVLQR offered better orientation control but underperformed in trajectory adherence due to its offline nature and sensitivity to disturbances. MPC, while slower, successfully reached the goal with precise tracking and minimal deviation.

Importantly, the MPC controller offers significantly greater tuning flexibility. Parameters such as horizon length, cost weights, and constraint bounds can be dynamically adjusted to balance speed, accuracy, and robustness. Currently the accuracy overweighted the wheel speed. This flexibility leaves considerable room for further improvement through future parameter optimization and state estimation enhancements.

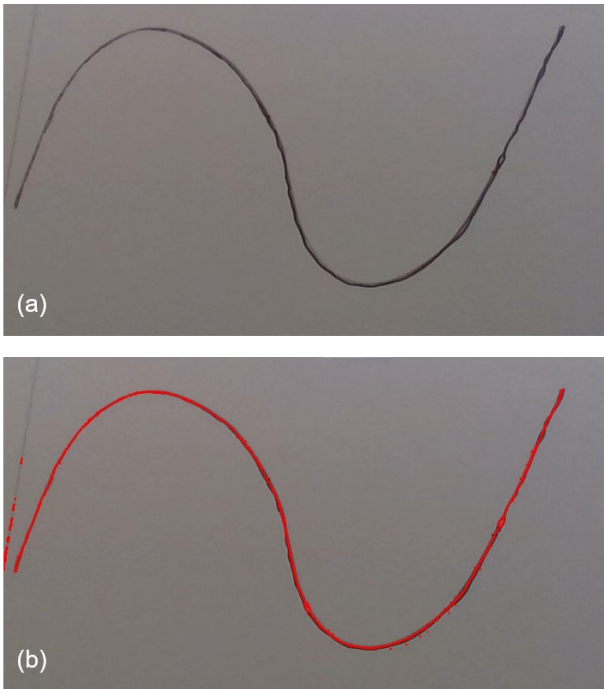

Fig. 7: Comparison between raw hand-drawn input and extracted path. (a) Original image of a freehand trajectory drawn on the whiteboard. (b) Result of image-based path extraction using skeletonization and endpoint-to-endpoint tracing. The red overlay represents the ordered trajectory computed for downstream 3D reconstruction and motion planning. Note the minor artifacts on the left due to discontinuous grayscale regions.

IX. Future Steps

While the proposed control framework demonstrates promising results, several challenges remain to be addressed for full deployment:

- **MPC Parameter Optimization:** Although the current MPC implementation successfully tracks reference trajectories, its performance is sensitive to cost weights, horizon length, and dynamic constraints. Future work will involve systematic tuning and auto-optimization techniques (e.g., grid search or Bayesian optimization) to balance speed, accuracy, and robustness under varying surface and loading conditions.
- **Path Planner Integration:** The CV-based path planner, which extracts paths from user-drawn lines, encountered projection errors when mapping 3D coordinates into the robot's operating frame. This limited its deployment, necessitating hardcoded trajectories instead. Future steps include solving the camera-frame alignment and calibration issues to enable fully autonomous path generation from visual input.
- **IMU Drifting:** Current when depth camera's capture fails, WIPER fallbacks to IMU. Future versions may incorporate sensor fusion via Kalman filters or implement re-localization strategies to improve robustness in long-term operation.
- **Scalability and Usability:** To move toward real-world adoption, the software stack will be refactored into a modular ROS-based architecture, enabling plug-and-play trajectory execution and dynamic environment adaptation.

REFERENCES

[1] V. Armstrong, S. Barnes, R. Sutherland, S. Curran, S. Mills, and I. Thompson, "Collaborative research methodology for investigating teaching and learning: the use of interactive whiteboard technology," *Educational review*, vol. 57, no. 4, pp. 457–469, 2005.

[2] V. Prabakaran, M. R. Elara, T. Pathmakumar, and S. Nansai, "Floor cleaning robot with reconfigurable mechanism," *Automation in Construction*, vol. 91, pp. 155–165, 2018.

[3] E. Prassler, A. Ritter, C. Schaeffer, and P. Fiorini, "A short history of cleaning robots," *Autonomous Robots*, vol. 9, pp. 211–226, 2000.

[4] W. R. Provancher, S. I. Jensen-Segal, and M. A. Fehlberg, "Rocr: An energy-efficient dynamic wall-climbing robot," *IEEE/ASME Transactions on Mechatronics*, vol. 16, no. 5, pp. 897–906, 2010.

[5] Y. Fang, S. Wang, Q. Bi, D. Cui, and C. Yan, "Design and technical development of wall-climbing robots: A review," *Journal of Bionic Engineering*, vol. 19, no. 4, pp. 877–901, 2022.

[6] S. Nansai and R. E. Mohan, "A survey of wall climbing robots: recent advances and challenges," *Robotics*, vol. 5, no. 3, p. 14, 2016.

[7] M. Falahi and M. Jannatifar, "Using orthogonal basis functions and template matching to learn whiteboard cleaning task by imitation," in *ICCKE 2013*, pp. 289–294, IEEE, 2013.

[8] J. A. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi: A symbolic framework for nonlinear optimization and optimal control." https://web.casadi.org, 2023. Version 3.6.3, accessed January 2024.