Controller Strategies for Trajectory Tracking - Comparative Study

Tirth Patel, Poojit Maddineni, Surabhi Gade, Keval Visaria

ME 5659 - Final Project Report
Professor- Sze Zheng Yong

April 14, 2025

Abstract

This project presents a comparative study of multiple control strategies for trajectory tracking in autonomous mobile robots. Leveraging 2D LIDAR-based SLAM for trajectory generation, we evaluate the performance of six controllers—PID, PD, MPC, Fast MPC — under varying speed conditions. The analysis focuses on tracking accuracy using cross track error as the key performance metric. Our findings offer insights into the trade-offs between classical and optimal control methods, highlighting their respective strengths, limitations, and suitability for real-world navigation tasks.

Keywords: Autonomous Mobile Robot, Trajectory Tracking, Non-linear Control, LIDAR SLAM, PID, MPC, Cross Track Error, Controller Comparison, Path Following

1 Background

Autonomous mobile robots must navigate complex, often unknown environments while maintaining accurate localization. Simultaneous Localization and Mapping (SLAM) plays a critical role in achieving this by enabling a robot to construct a map of its surroundings while simultaneously estimating its pose. Among various SLAM techniques, lidar-based SLAM is known for its accuracy and reliability in indoor and structured settings, making it a common choice for real-time robotic applications. Integrating SLAM with trajectory planning enables robots to make informed decisions about path following and obstacle avoidance in real time.

Once a trajectory is generated, the robot must track it effectively using suitable control strategies. Classical controllers like Proportional-Integral-Derivative (PID) and Proportional-Derivative (PD) are widely used due to their simplicity and ease of implementation, but they often lack robustness in dynamic or nonlinear environments. These methods may also struggle when faced with system constraints or rapidly changing conditions.

Model Predictive Control (MPC) and its variants, such as Fast MPC, address these limitations by solving an optimization problem at each time step, considering future states, constraints, and system dynamics. This predictive capability makes MPC highly effective for trajectory tracking in complex environments. Fast MPC, in particular, is designed for real-time applications with limited computational resources. By combining SLAM with advanced control strategies like MPC, and PID, this project aims to identify a robust and adaptive navigation system that performs reliably under varying environmental and system conditions.

2 Objectives

- Implement trajectory generation using 2D LIDAR-based SLAM data.
- Develop a non-linear model of a differential drive robot and analyze its state-space representation.
- Design and implement multiple control strategies: PID, PD, MPC, Fast MPC.
- Build corresponding Simulink models for selected controllers to support real-time validation.
- Compare controller performance across varying speeds using cross track error as the evaluation metric.
- Conduct ablation studies to identify limitations and trade-offs in control strategies for non-linear trajectory tracking.

3 Literature Review

Over the past decade, foundational work in mobile robotics emphasized the development of simulation platforms and early SLAM techniques. Early contributions such as those by Chen and Cheng [1] in 2010 introduced MATLAB-based simulators that provided essential tools for implementing and comparing SLAM algorithms like EKF, UKF, and FastSLAM. This groundwork was complemented by Suwannakom [7] in 2014, who applied a hybrid approach combining SLAM with fuzzy logic to enhance indoor localization and mapping, while J.Song and Gupta [6] in 2015 further pushed the envelope by proposing a shape adaptive coverage control algorithm that leveraged online sensor feedback for path adaptation in uncertain environments.

The mid-2010s witnessed a shift towards integrating robust control strategies with dynamic modeling. Sharma et al. [5] in 2016 developed a unified state-space model that combined both the dynamic and kinematic aspects of differential drive mobile robots, enabling effective trajectory tracking through model predictive control (MPC). This integration of dynamic modeling with predictive control was further advanced by Chen et al. [2] in 2020, who merged MPC with PID speed control to achieve time-optimal travel under physical constraints. Concurrently, benchmarking efforts by Nehate et al. [4] in 2021 underscored the practical effectiveness and versatility of these advanced control schemes.

In recent years, the research focus has gravitated towards enhancing robustness and accuracy by combining traditional control approaches with modern learning techniques. Most notably, Khan et al. [3] in 2025 introduced a deep reinforcement learning framework integrated with visual SLAM that significantly improved path tracking performance and system robustness in dynamic environments. Collectively, these studies trace an evolution from early simulator-based research and adaptive control strategies to sophisticated, learning-enhanced predictive control systems that are better suited to address the complex challenges of real-world mobile robotics.

4 Significance

The outcomes of this project are poised to significantly impact the field of autonomous robotics by providing a robust, scalable framework for navigation in unknown and dynamic environments. By combining SLAM-based trajectory generation with adaptive control, the system enables real-time localization and path tracking without prior maps or external positioning systems. This capability is crucial for applications such as warehouse automation, search and rescue, and autonomous delivery, where adaptability and precision are essential. Furthermore, the use of real sensor data and modular design lays the groundwork for future integration with learning-based algorithms, making the system well-suited for deployment in increasingly complex and unstructured environments.

5 Mathematical Modeling

5.1 Kinematics and Dynamics

Kinematic modelling for a differential drive robot:

We model the motion of a car-like mobile robot with two front steering wheels and two rear fixed wheels. To prevent slipping of the front wheels, they are steered using an *Ackermann steering* mechanism, which ensures correct turning geometry.

The configuration of the robot is represented as $q = (x, y, \phi, \psi)$, where (x, y) denotes the location of the midpoint between the rear wheels, ϕ is the car's heading direction, and ψ is the steering angle of the car, defined at the virtual wheel located at the midpoint between the front wheels.

The heading direction ϕ is perpendicular to the line connecting the rear axle midpoint and the center of rotation (CoR). The figure below illustrates the car turning at its minimum turning radius $r_{\min}$.

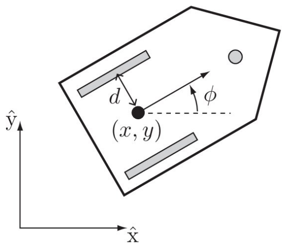

Figure 1: A di-drive robot consisting of two typical wheels and one ball caster wheel, shaded gray.

The control inputs to the system are:

- Forward speed v of the car at its reference point
- Angular speed ω of the steering angle

These inputs are constrained by physical limits:

$$v \in [v_{\min}, v_{\max}], \quad \omega \in [\omega_{\min}, \omega_{\max}], \quad \psi \in [\psi_{\min}, \psi_{\max}] \tag{1}$$

The general kinematic equations for the system are:

$$\dot{q} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\phi} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} v\cos\phi \\ v\sin\phi \\ \frac{v\tan\psi}{\ell} \\ \omega \end{bmatrix} \tag{2}$$

where ℓ is the wheelbase (distance between front and rear axles).

In a simplified kinematic model, the control input is the steering angle ψ itself (and not its rate). The configuration reduces to $q = (x, y, \phi)$, with control inputs (v, ψ), and the relation $\dot{\phi} = \frac{v}{\ell}\tan\psi$. The simplified equations become:

$$\dot{q} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} \cos\phi & 0 \\ \sin\phi & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{3}$$

The nonholonomic constraint implied by the system geometry leads to:

$$\dot{x} = v\cos\phi, \quad \dot{y} = v\sin\phi \tag{4}$$

The set of admissible velocity controls (v, ω) for the car robot forms a bow-tie shape in control space. The slope of the boundary lines is determined by the minimum turning radius. If the vehicle lacks a reverse gear, only the right-hand side of the bow-tie (positive v) is available.

6 Control Strategies

6.1 PID Controller

A Proportional-Integral-Derivative (PID) controller was implemented to regulate both the linear and angular velocities of the robot as it tracked the SLAM-generated reference trajectory. The control objective was to minimize both the positional and heading errors between the robot's current pose and the desired trajectory point at each time step.

The PID control law used is defined as:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\, d\tau + K_d \frac{de(t)}{dt} \tag{5}$$

where $e(t)$ denotes the error signal, and K_p, K_i, and K_d are the proportional, integral, and derivative gains, respectively.

Two independent PID loops were designed:

- The first loop regulates the linear velocity v, using the Euclidean distance to the next reference waypoint as the error.
- The second loop controls the angular velocity ω, based on the heading angle difference between the robot and the direction of the next waypoint.

The error terms were computed in real-time at each control update. A finite difference approximation was used to estimate the derivative of the error, and cumulative integration was employed for the integral term. The outputs of the controller were clamped to the robot's physical velocity limits to ensure feasibility.

Gain tuning was performed empirically in simulation to balance responsiveness and stability. The derivative component helped reduce oscillations during sudden changes in direction, while the integral term corrected for minor steady-state errors due to SLAM noise or sampling delays.

The PID controller exhibited satisfactory performance in smooth environments with consistent SLAM feedback. It offered fast convergence to the trajectory, minimal overshoot, and reliable heading correction for moderate curvature segments.

6.2 Model Predictive Control (MPC)

Model Predictive Control (MPC) was implemented as an alternative to PID to enhance trajectory tracking performance under uncertainty and constraints. Unlike PID, MPC optimizes future control actions over a finite prediction horizon by solving a constrained optimization problem at each time step.

The MPC formulation used in this project minimizes a quadratic cost function:

$$J = \sum_{k=1}^{N} \left\| \mathbf{x}_k - \mathbf{x}_k^{\text{ref}} \right\|_Q^2 + \left\| \mathbf{u}_k \right\|_R^2 \tag{6}$$

where $\mathbf{x}_k$ is the predicted state at time step k, $\mathbf{x}_k^{\text{ref}}$ is the reference trajectory point, and $\mathbf{u}_k$ is the control input. The matrices Q and R are used to weight the state tracking error and control effort, respectively.

The robot's dynamics were represented using the previously derived linear state-space model. At each control cycle, the current state was used as the initial condition, and a sequence of future control inputs was optimized subject to velocity and angular rate constraints:

$$v_{\min} \le v_k \le v_{\max}, \quad \omega_{\min} \le \omega_k \le \omega_{\max} \tag{7}$$

Only the first control input from the optimized sequence was applied to the robot, and the process was repeated at each time step in a receding horizon fashion. The optimization was solved using MATLAB's built-in quadratic programming solvers.

MPC demonstrated superior performance in scenarios with sharp turns and varying path curvature, where it maintained smooth control transitions while respecting input limits. Its ability to anticipate future errors and handle constraints made it more robust than PID in complex environments.

6.3 PD Controller

In addition to PID, a simpler Proportional-Derivative (PD) controller was implemented for baseline comparison. The PD controller eliminates the integral term to reduce computational load and prevent wind-up effects, especially in high-speed applications.

The control law is:

$$u(t) = K_p e(t) + K_d \frac{de(t)}{dt} \tag{8}$$

The PD controller was applied similarly to PID, with separate loops for position and orientation tracking. By removing the integral term, the system responded faster to changes in reference without accumulating error, making it well-suited for rapidly updating waypoints or short segments of trajectory.

While the PD controller generally provided faster settling times, it exhibited steady-state errors under long, curved trajectories or slow SLAM drift. It was particularly effective during short bursts of precise control or when localization confidence remained consistently high.

6.4 Fast Model Predictive Control (Fast MPC)

Fast Model Predictive Control (Fast MPC) was implemented as a time-efficient alternative to standard MPC for real-time applications. While conventional MPC solves an optimization problem over a prediction horizon at every control step, it can be computationally intensive, particularly for high-speed navigation. Fast MPC addresses this by simplifying the optimization process using precomputed matrices, warm-started solvers, and linearized dynamics.

The cost function remains identical to the standard MPC formulation:

$$J = \sum_{k=1}^{N} \left\| \mathbf{x}_k - \mathbf{x}_k^{\text{ref}} \right\|_Q^2 + \left\| \mathbf{u}_k \right\|_R^2 \tag{9}$$

but is solved more efficiently using lightweight solvers optimized for speed.

To enable real-time feasibility, the Fast MPC formulation made the following adjustments:

- Linear dynamics with constant matrices were assumed throughout the prediction horizon.
- Matrix factorizations were computed offline and reused at runtime.
- The prediction horizon was shortened to reduce the number of variables in the optimization.

This approach allowed us to maintain predictive control behavior with significantly reduced computational delay. It proved effective in high-speed scenarios where standard MPC introduced delays or failed to compute control inputs in time. Fast MPC offered smoother transitions and better adherence to the reference path under tight time constraints, making it ideal for real-time embedded control systems.

7 Simulation and Analysis

7.1 Test Setup

The evaluation pipeline was divided into two major stages: trajectory generation using MATLAB and control-based simulation in Python.

First, raw LIDAR scan data was processed in MATLAB using the built-in `lidarSLAM` module. The CSAIL NewCarmen dataset was used as the primary input, and a pose graph was constructed using incremental scan registration and loop closure detection. The final trajectory was optimized and smoothed using cubic B-spline interpolation. This refined trajectory served as the reference path for all controller evaluations.

The optimized trajectory was exported and loaded into a Python simulation environment, where a differential drive robot was simulated using a unicycle model. Each controller—PID, PD, MPC, and Fast MPC, implemented in Python and applied independently to the same trajectory.

For each run, the simulated robot started from a known initial pose and attempted to track the trajectory using one of the controllers. The key metric for evaluation was cross track error (CTE), defined as the lateral distance between the robot's actual position and the closest point on the reference trajectory at each timestep.

The simulation loop included the following steps:

1. Load and parse the reference trajectory.
2. Simulate the robot motion using the specified controller.
3. Log the robot's position and compute the cross track error at each timestep.
4. Plot trajectory overlays and CTE curves for comparison.

Multiple runs were conducted for each controller under varied speed conditions to analyze their performance. These results formed the basis of our ablation studies, highlighting the strengths, weaknesses, and operating limits of each control strategy in non-linear trajectory tracking.

7.2 Evaluation Metrics

To quantitatively compare the performance of each controller, we employed the following evaluation metrics:

- **Cross Track Error (CTE):** CTE measures the perpendicular distance between the robot's current position and the nearest point on the reference trajectory. It is the primary metric for assessing how closely the robot follows the desired path. Lower CTE values indicate more accurate tracking. CTE was computed at each timestep and visualized over time to observe oscillations and deviations.
- **Overshoot and Settling Behavior:** For PID and PD controllers, we recorded overshoot near trajectory inflection points and evaluated how quickly the robot's position stabilized. Excessive oscillation was treated as a sign of poor gain tuning or lack of damping.

7.3 Ablation Study

The comparative evaluation of four controllers—**MPC**, **Fast MPC**, **PID**, and **PD**—was conducted across speed profiles of 1 m/s, 3 m/s, and 5 m/s. Performance was analyzed in terms of tracking accuracy, control smoothness, speed adherence, and computational behavior.

Accuracy:

- At 5 m/s, both MPC and Fast MPC maintained cross-track errors below 0.2 m.

- PD exhibited peak deviations exceeding 0.8 m and suffered from high oscillations.

- At 2 m/s, Fast MPC sustained tracking accuracy under 0.1 m, while PID had moderate fluctuations and PD continued to deteriorate.

Stability and Smoothness:

- Fast MPC produced the smoothest control profiles for both linear and angular velocity, even on sharp trajectory curves.

- PD showed aggressive, high-frequency oscillations in angular velocity (ω), often exceeding ± 3 rad/s.

- PID had moderate smoothness but showed abrupt changes in speed and heading, especially at higher velocities.

Speed Adherence:

- Both MPC and Fast MPC closely tracked target velocities throughout the trials.

- PID maintained reasonable adherence up to 3 m/s but struggled beyond that.

- PD exhibited poor adherence, with noticeable clipping and lag in speed control.

Computational Behavior:

- MPC delivered high accuracy at the cost of higher computation time, leading to occasional frame delays at 5 m/s.

- Fast MPC offered similar accuracy but with significantly reduced delay, making it more suitable for real-time applications.

Performance at 2 m/s

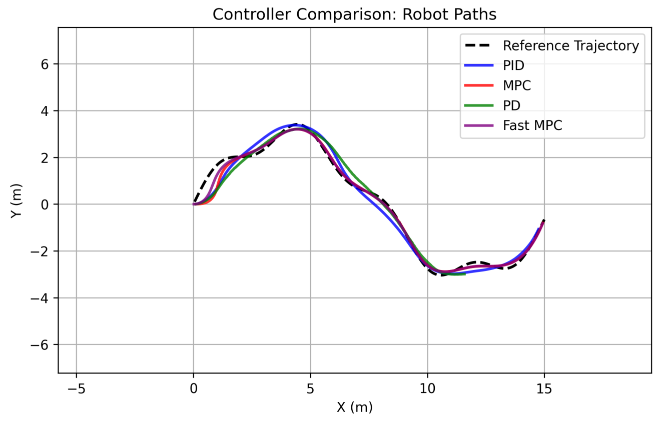

Figure 2: Controller Output Comparison at 1 m/s

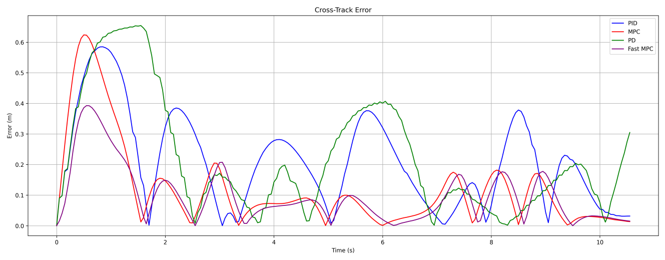

Figure 3: Cross Track Error over Time at 2 m/s

Performance at 3 m/s

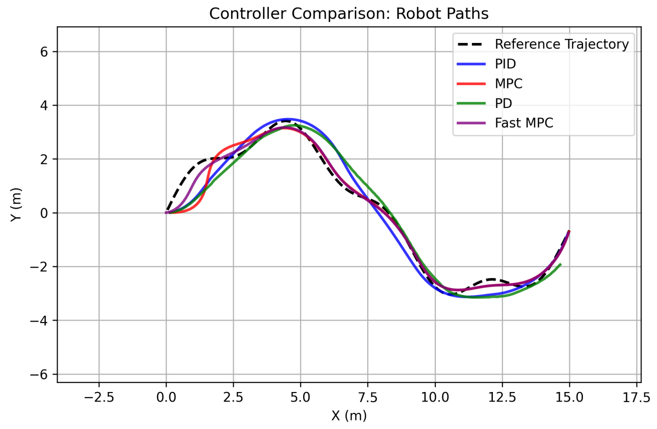

Figure 4: Controller Output Comparison at 3 m/s

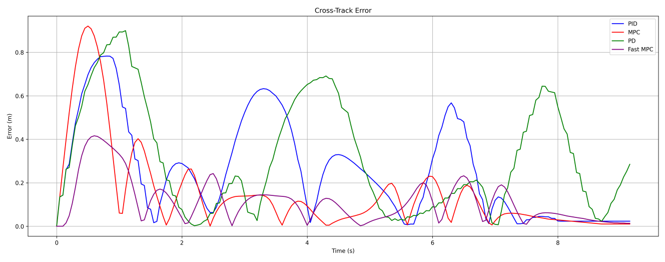

Figure 5: Cross Track Error over Time at 3 m/s

Performance at 5 m/s

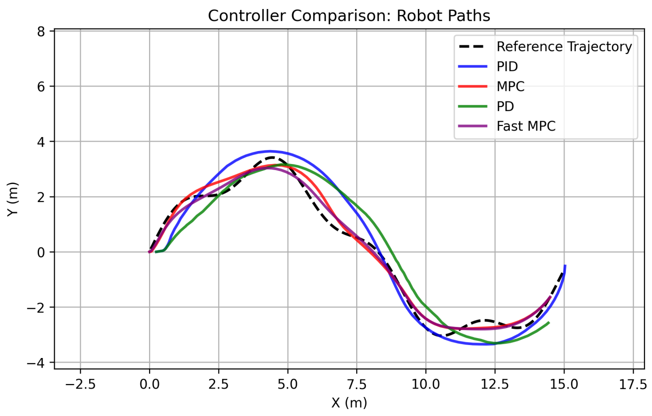

Figure 6: Controller Output Comparison at 5 m/s

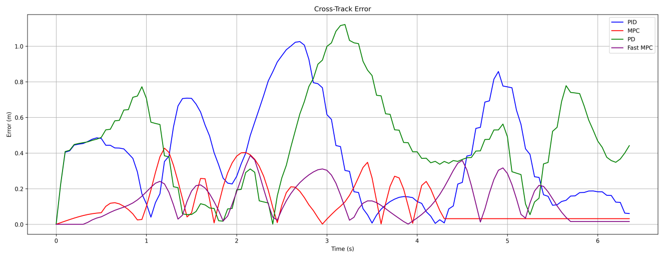

Figure 7: Cross Track Error over Time at 5 m/s

Summary of Controller Performance

In summary, Fast MPC emerges as the most balanced and reliable controller across all tested velocities. It combines high tracking accuracy, excellent stability, and smooth control commands with efficient runtime, making it suitable for both high-speed and low-speed applications. While MPC provides strong tracking at high speeds, its computational overhead limits scalability for real-time use. PID is serviceable up to 3 m/s, but beyond that, its stability and accuracy deteriorate. PD control, though simple, proves unsuitable for speeds above 2 m/s due to poor accuracy and noisy control outputs. For applications requiring both performance and responsiveness—such as agile mobile robots or autonomous navigation—Fast MPC is the optimal choice.

7.4 Results and Analysis

The trajectory tracking performance of each controller was evaluated by overlaying the robot's path on the SLAM-generated reference trajectory and analyzing the deviation. The controllers were tested under identical initial conditions and trajectory data to ensure consistency. The key metric used for performance evaluation was cross track error (CTE), supported by qualitative inspection of path adherence.

PID Controller: The PID controller demonstrated stable tracking with moderate convergence to the reference trajectory. However, noticeable oscillations occurred at the beginning of the path, especially in the angular component. This behavior indicates a delay in correcting heading errors, potentially due to the integral term accumulating slowly under rapidly changing curvature. Once the robot aligns with the trajectory, it maintains close adherence with slight lag during directional shifts.

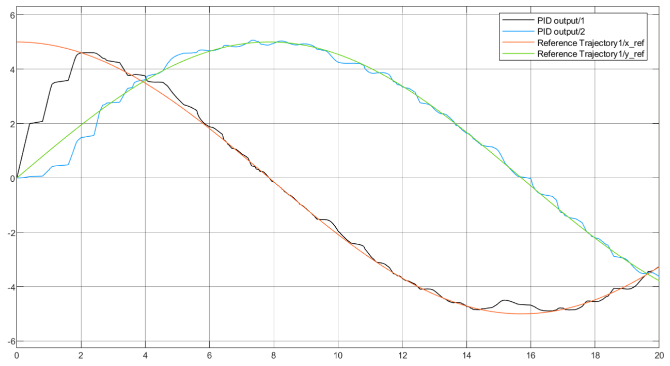

Figure 8: Trajectory tracking with PID controller

- **X-Axis Output**
  - Peak Values: 4.609, -4.683
  - Peak Times: 2.269 s, 14.000 s
- **Y-Axis Output**
  - Peak Values: 5.053, 5.051
  - Peak Times: 6.880 s, 7.843 s
  - Approx. Settling Time: 8.479 s

PD Controller: The PD controller responded more aggressively to changes in heading, showing faster initial convergence compared to PID. However, the lack of an integral term resulted in higher steady-state errors. The robot's path remained offset from the reference in some sections, especially along the vertical curve, highlighting limitations in long-term correction. Despite that, the smoother output curve suggests lower overshoot and better damping under high-speed conditions.

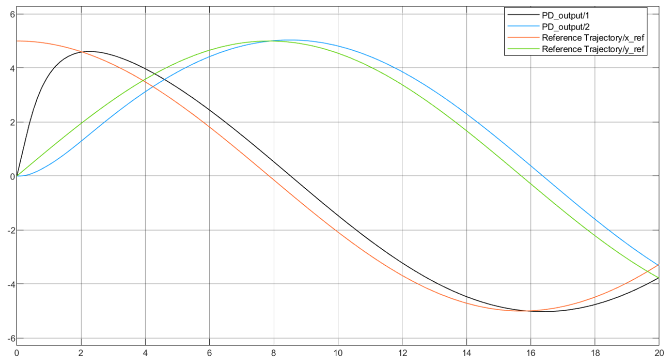

Figure 9: Trajectory tracking with PD controller

- **X-Axis Output**

  - Peak Value: 4.613
  - Peak Time: 2.269 s

- **Y-Axis Output**
  - Peak Value: 5.038
  - Peak Time: 8.507 s

MPC Controller: Model Predictive Control showed superior anticipation of trajectory curvature. It maintained close proximity to the reference path throughout the simulation, with minimal overshoot and smoother transitions between segments. The MPC controller was particularly effective in handling changes in curvature, balancing responsiveness and stability. However, its performance comes at the cost of higher computational load, noticeable in runtime when compared to PD and PID.

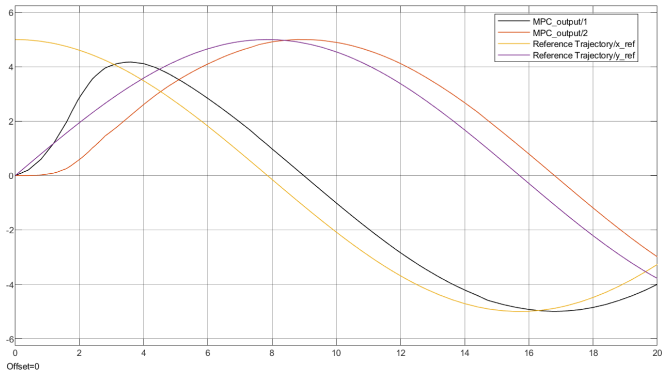

Figure 10: Trajectory tracking with MPC controller

- **X-Axis Output**
  - Peak Value: 4.173
  - Peak Time: 3.600 s

- **Y-Axis Output**
  - Peak Value: 5.000
  - Peak Time: 8.970 s

Fast MPC Controller: Fast MPC achieved a balance between computation time and trajectory tracking performance. While not as accurate as full MPC, its path closely followed the reference in both axes. Initial alignment was slightly delayed, and it showed minor drift at turning points. Nevertheless, the overall trajectory remained within acceptable bounds, validating its utility for real-time applications where speed is critical.

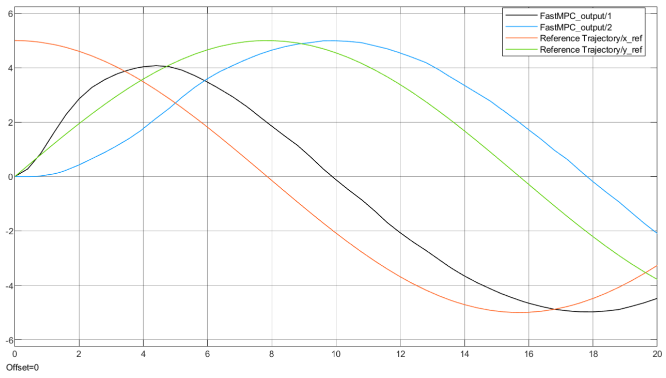

Figure 11: Trajectory tracking with Fast MPC controller

- **X-Axis Output**
  - Peak Value: 4.070
  - Peak Time: 4.367 s

- **Y-Axis Output**
  - Peak Value: 4.989
  - Peak Time: 9.929 s

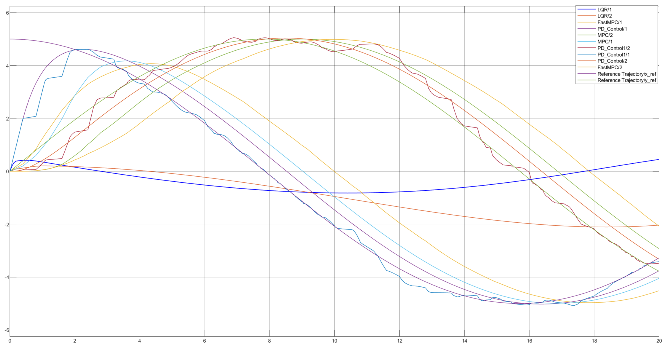

Figure 12: Comparison of trajectory tracking performance across all controllers

Summary: Among all the controllers, MPC performed best in terms of accuracy and smoothness. PD offered fast reaction time but lacked correction over longer durations. PID was stable but prone to initial oscillation. Fast MPC proved viable for constrained environments. These observations form the basis of our ablation study and support the selection of hybrid or adaptive control strategies for real-world deployment.

8 MATLAB and Simulink

The system design and simulation were partially implemented using MATLAB and Simulink. MATLAB was used for processing LIDAR data, generating SLAM-based trajectories, and preparing the reference path. Simulink was used to model and simulate control strategies for testing and validation in a visual, block-based environment.

PID Controller Block

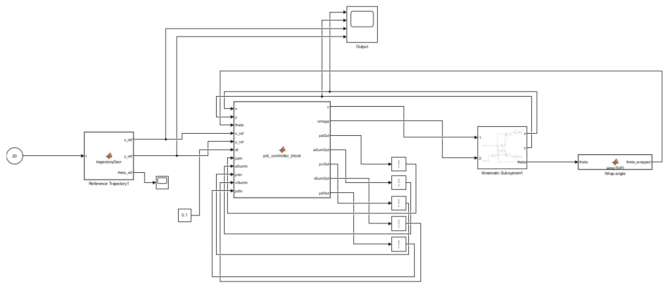

Figure 13: PID controller structure

PD Control Block

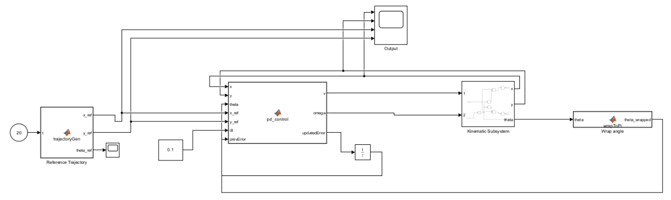

Figure 14: PD Controller structure

MPC Control Block

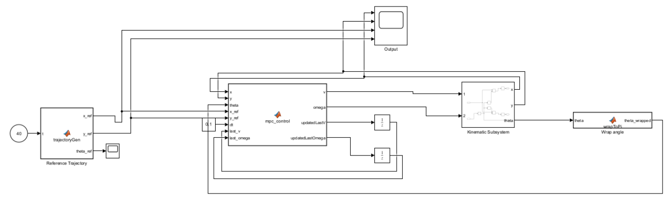

Figure 15: MPC Controller structure

FastMPC Control Block

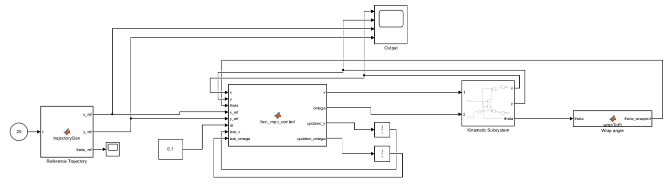

Figure 16: Fast MPC Controller structure

Final Large Controller

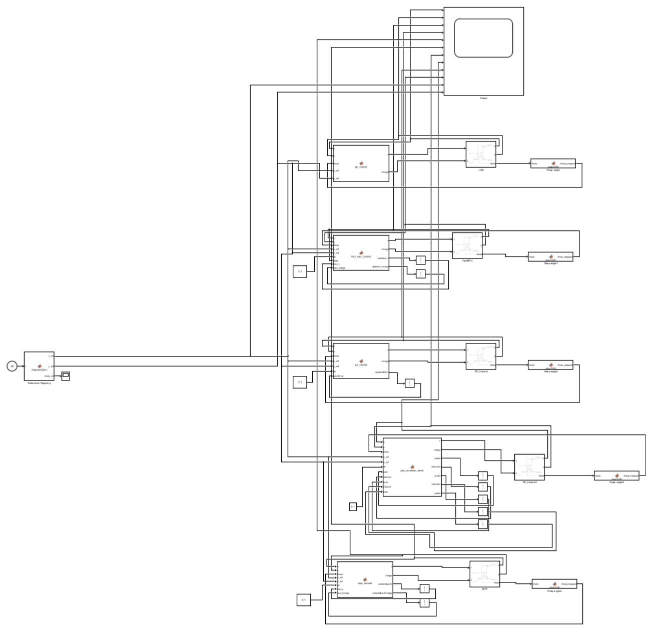

Figure 17: Combined controller architecture for comparative analysis

These block diagrams visualize how the control logic and trajectory data are integrated into a real-time simulation framework. They were useful for debugging and tuning controller parameters interactively.

Code Repository

The full implementation of this project, including all controllers, SLAM code, and simulation environments, is available on GitHub at: `https://github.com/Pateltirths1012/Control-Strategies-for-Trajectory-Tracking.git`

The repository contains:

- MATLAB code for SLAM and trajectory generation
- Python implementations of all controllers (PID, PD, MPC, Fast MPC)
- Simulink models as described in this report
- Simulation results and evaluation scripts
- Dataset information and processing utilities

9 Conclusion

9.1 Major Design Decisions

The project's success was predicated on several key design decisions. First, the selection of 2D LIDAR-based SLAM for trajectory generation proved effective for indoor environments with structured features. Second, the decision to implement multiple control strategies rather than optimizing a single approach allowed for comprehensive comparison and situation-specific deployment recommendations. Third, using cross track error as the primary evaluation metric provided a consistent basis for controller comparison across varying speeds. Finally, the modular software architecture facilitated seamless integration between MATLAB's sensor processing capabilities and Python's flexible simulation environment.

9.2 Challenges

Several technical challenges were encountered during implementation. The non-linear dynamics of the differential drive robot complicated the control design, particularly for methods that assume linearity. Additionally, optimizing the Fast MPC algorithm to maintain performance while reducing computational overhead required careful tuning of the prediction horizon and constraint formulation. Finally, achieving consistent, real-time performance across all controllers proved challenging at high speeds (5 m/s), where computational delays became non-negligible for complex algorithms.

9.3 Key Learnings

This project yielded several important insights into autonomous robot control. First, we observed that while MPC provides superior tracking performance, its computational requirements may outweigh benefits in resource-constrained applications. Second, we found that PID control, despite its simplicity, remains effective for moderate speeds and can be augmented with adaptive gain scheduling for improved performance. Third, we discovered that Fast MPC offers an excellent compromise between computational efficiency and tracking accuracy, making it suitable for deployment on embedded systems with limited processing power.

9.4 Future Scope

Future work will focus on several promising directions. First, implementing adaptive control strategies that dynamically select between controllers based on speed, trajectory complexity, and computational resources. Second, extending the SLAM system to incorporate visual-inertial odometry for enhanced localization in feature-poor environments. Third, integrating reinforcement learning approaches for parameter tuning and control policy optimization. Finally, developing a hardware implementation on a scaled prototype to validate simulation results and assess real-world performance under varying environmental conditions.

10 Contributions

- Tirth Patel - 2D LidarSLAM, Matlab code, Simulink
- Poojit Maddineni - Controllers implementation
- Surabhi Gade - PID, PD, MPC Controllers Documentation, Literature review
- Keval Visaria - Differential Drive mathematical modelling, Fast MPC and Documentation

References

[1] Chen Chen and Yinhang Cheng. "MATLAB-based simulators for mobile robot Simultaneous Localization and Mapping". In: *2010 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE)*. 2010, pp. V4-386–V4-389. DOI: 10.1109/ICACTE.2010.5579471.

[2] Shu-ping Chen et al. "MPC-based path tracking with PID speed control for high-speed autonomous vehicles considering time-optimal travelMPC PID ". In: *Journal of Central South University* 27 (Dec. 2020). DOI: 10.1007/s11771-020-4561-1.

[3] S. Khan et al. "Deep reinforcement learning and robust SLAM based robotic control algorithm for self-driving path optimization". In: *Frontiers in Neurorobotics* 18 (2025). DOI: 10.3389/fnbot.2024.1428358.

[4] Chinmay Nehate et al. "Implementation and Evaluation of SLAM Systems for a Mobile Robot". In: *2021 IEEE 18th India Council International Conference (INDICON)*. 2021, pp. 1–6. DOI: 10.1109/INDICON52576.2021.9691761.

[5] Rahul Sharma, Daniel Honc, and František Dušek. "Predictive Control Of Differential Drive Mobile Robot Considering Dynamics And Kinematics". In: June 2016, pp. 354–360. DOI: 10.7148/2016-0354.

[6] J. Song and S. Gupta. "SLAM based shape adaptive coverage control using autonomous vehicles". In: *2015 10th System of Systems Engineering Conference (SoSE)*. 2015, pp. 141–146. DOI: 10.1109/SYSOSE.2015.7151959.

[7] A. Suwannakom. "Adaptive control performance of a mobile robot using hybrid of SLAM and fuzzy logic control in indoor environment". In: *2014 International Electrical Engineering Congress (iEECON)*. 2014, pp. 1–4. DOI: 10.1109/iEECON.2014.6925859.