Autonomous Parallel Parking with MPPI

Shreyas Raorane
University of Pennsylvania

Eddie Shen
University of Pennsylvania

Yufeiyang Gao
University of Pennsylvania

Jerry Wang
University of Pennsylvania

***Abstract*—Autonomous parallel parking is a challenging maneuver that requires precise trajectory planning and execution in constrained environments. This project addresses the implementation of autonomous parallel parking for F1/10 scale cars using Model Predictive Path Integral (MPPI) control. The aim is to find optimal control inputs (velocity and steering angle) that guide the vehicle from an initial state to a target parking position while avoiding obstacles and respecting vehicle dynamics constraints. The project explores two approaches: direct reverse entry and three-point parking maneuver, leveraging MPPI's ability to sample and weight trajectories based on their expected costs.**

***Index Terms*—Autonomous Parking, Model Predictive Path Integral (MPPI), Parallel Parking, F1/10 Scale Cars**

GitHub Repository: https://github.com/Shreyas0812/parallel-parking-roboracer

I. Introduction

Autonomous parallel parking is a fundamental yet challenging capability for self-driving vehicles, requiring precise planning and control within highly constrained environments. Unlike regular lane-following or obstacle avoidance tasks, parallel parking demands centimeter-level accuracy, complex maneuvering under nonholonomic constraints, and the ability to switch seamlessly between forward and reverse motions. These requirements are further compounded by environmental uncertainties and tight spatial boundaries commonly found in urban settings.

In this work, we develop and evaluate an autonomous parallel parking system for F1/10-scale vehicles, using Model Predictive Path Integral (MPPI) control [1] to generate dynamically feasible trajectories that guide the vehicle from an initial pose to a designated parking slot. The problem is formulated as an optimal control task, where the goal is to compute a sequence of velocity and steering commands $\{u_t\}_{t=0}^{T-1}$ that minimize a cost function while respecting vehicle dynamics and avoiding obstacles. The system dynamics are represented as:

$$x_{t+1} = f(x_t, u_t) + w_t$$

where x_t is the vehicle state (position and orientation), u_t is the control input, f is the kinematic vehicle model, and $w_t \sim \mathcal{N}(0, \Sigma)$ captures process noise.

To capture the complex requirements of parallel parking, we design a cost function that combines multiple objectives: deviation from the target pose, distance from a reference trajectory, proximity to obstacles, and control effort. This formulation enables MPPI to evaluate thousands of candidate trajectories in real time and apply the first optimal control input while re-planning at each time step.

Our approach incorporates several enhancements to standard MPPI, including adaptive sampling, motion mode switching for forward and reverse planning, and early termination when the vehicle reaches the goal region. These customizations enable the controller to handle multi-point parking maneuvers in narrow slots and respond robustly to environmental uncertainty.

This work contributes a practical and deployable MPPI-based parallel parking framework tailored for resource-constrained embedded platforms, demonstrating reliable performance in real-world indoor test environments.

II. Related Work

Autonomous parking has been an active research area within robotics and autonomous vehicles for decades, with approaches evolving from simple rule-based methods to sophisticated optimization techniques. Early work by Paromtchik and Laugier [2] established foundations for rule-based parking systems using ultrasonic sensors and predefined maneuvers. These approaches, while computationally efficient, lacked adaptability to varied parking scenarios.

Path planning techniques for autonomous parking have subsequently evolved along several directions. Geometric approaches, as presented by Vorobieva et al. [3], generate reference paths using continuous curvature curves that satisfy vehicle kinematic constraints. While also being computationally efficient, these methods often struggle with obstacle avoidance in dynamic environments.

Optimization-based approaches have gained prominence in recent years. Ziegler and Stiller [4] applied model predictive control (MPC) to parking problems, explicitly handling constraints and minimizing control effort. However, traditional MPC struggles with the non-convexity inherent in parking problems with obstacles.

The Model Predictive Path Integral (MPPI) control framework, developed by Williams et al. [5], offers advantages for non-linear stochastic systems by sampling multiple control trajectories and combining them based on their expected costs. This approach has shown promise in agile autonomous driving applications but has seen limited application to the specific challenges of parallel parking.

Recent work by Takehara et al. [6] has explored reinforcement learning approaches for autonomous parking, allowing systems to learn optimal policies through interaction with simulated environments. While promising, these methods typically require large amounts of training data and may struggle with generalization to unseen scenarios.

Our work builds particularly on the MPPI implementations by Williams et al. and extends them to address the specific challenges of parallel parking in constrained environments, with particular attention to creating robust fallback strategies when initial trajectories become infeasible.

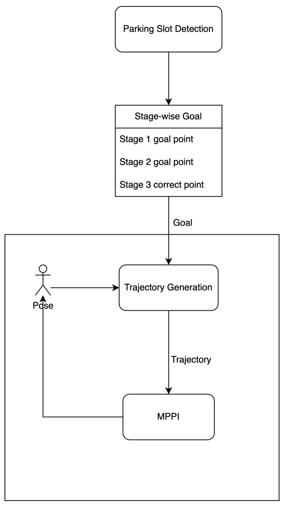


Fig. 1: Overview of our method structure

III. Approach

Our implementation of autonomous parallel parking using MPPI follows a systematic approach that leverages sampling-based trajectory optimization while incorporating key domain-specific constraints and heuristics. Generally, our approach can be splitted into Parking Space Detection, Trajectory Generation and MPPI Follower. The Parking Space Detection given the map and determines the stage-wise goal points automatically, Trajectory Generation produces the corresponding trajectories from initial car position along each goal points to the end and MPPI Follower execute the trajectories to reach the goal. The overview of the stucture is in fig 1.

A. Vehicle Dynamics Modeling

We employ a bicycle model, as shown in 2, to represent the dynamics of F1/10 cars, capturing the essential non-holonomic constraints while maintaining computational efficiency. This model relates the control inputs (velocity and steering angle) to the state evolution (position and orientation) through a set of differential equations. We augment this model with appropriate noise terms to account for uncertainties in actuation and environmental factors.

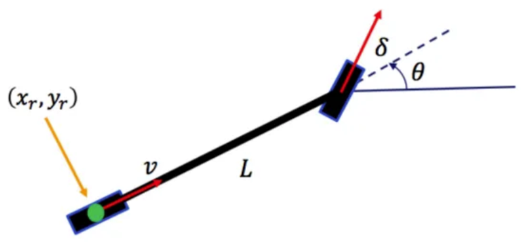


Fig. 2: Bicycle model

B. Parking Space Detection Module

For this module, the goal is to find regions (contours) in the binary image that contain enough long gaps (white pixels) horizontally or vertically, which can represent parking spaces. Also it should generate reference waypoints that help the car park inside that slot.

This module takes in the binary occupancy map and gets the contour of the obstacle. The contour is examined by scanning its interior area for continuous sequences of free space. If the length of the gap is within the a threshold $[min\_gap, max\_gap]$, the region is marked as a valid parking slot.

The result is visualized as in 3. The gap is marked green, and reference waypoints are marked red. The red boundary is the detected contour for the map. This allows both quantitative evaluation and easy visual verification of potential parking areas within the map. We are given a binary map $M \in \{0, 255\}^{H\times W}$, where:

- 0 represents free space,
- 255 represents obstacles.

Step 1: Find valid parking slots

It extracts the largest contour in the binary map and analyzes each contour segments to find the gap. Contour segment refers to the disconnected blocks in the contour (separated by obstacles). $p1$ and $p2$, the inner two corner points of the gap, can be extracted from the segment. We define the direction of the parking space to be that parallel to the longer side of the wall. This direction is obtained with $v = (p1-p2)/norm(p1-p2)$. The middle point for the inner side $m = (p1+p2)/2$. Besides the location of the parking space, the entrance direction is also crucial for our task, for example, whether the slot is open upward or downward. To examine the direction, we take the vector k perpendicular to the parking space direction. k can be either positive or negative. Then we take a few trial points starting from m and extend to the positive and negative side of v. Whichever side with trial points in free cells is considered as the parking space opening direction. The width w of the slot is obtained by counting the number of pixels from $p1$ to $p2$ along the parking space direction. We only consider one as a valid parking space if $w \in [w_{min}, w_{max}]$. The depth d of the gap is obtained by counting the number of pixels from

$p1$ on the direction of k until it reaches the free cell. Finally, we have the information about the parking space $\{p1, p2, k, v, d, w, m\}$.

Algorithm 1 Detect Entrance Gaps in Binary Map

```
Require: Binary map image $I$, minimum gap width threshold $w_{min}$, upper threshold $w_{max}$
Ensure: List of detected entrance gaps, visualization image
 1: Find external contours in $I$
 2: if no contours found then
 3:     return empty list, color version of $I$
 4: end if
 5: Select the largest contour $C$
 6: Convert $I$ to color image $V$ and draw $C$ in red
 7: Approximate $C$ with polygon $C_{approx}$ using Douglas-Peucker algorithm
 8: Initialize empty list $Gaps$
 9: for each edge $(p_1, p_2)$ in $C_{approx}$ do
10:     Compute edge length $L \leftarrow \|p_2 - p_1\|$
11:     if $L < w_{min}$ or $L > w_{max}$ then
12:         continue
13:     end if
14:     Compute midpoint $m$ and normalized direction vector $\vec{v}$
15:     Compute perpendicular vector $\vec{k}$
16:     for each sign $s \in \{-1, 1\}$ do
17:         $t \leftarrow m + 10 \cdot (s \cdot \vec{k})$
18:         if $t$ in bounds and $I[t] = 255$ then
19:             Set $\vec{k} \leftarrow s \cdot \vec{k}$ (points inward)
20:             break
21:         end if
22:     end for
23:     $m_{inner} \leftarrow m + 5 \cdot \vec{k}$, $m_{deep} \leftarrow m + 20 \cdot \vec{k}$
24:     if both $m_{inner}$ and $m_{deep}$ are in free space then
25:         Compute entrance rectangle corners using $\vec{p}$
26:         Call CALCULATEGAPDEPTH on rectangle
27:         Add gap info to $Gaps$ list
28:         Draw entrance on visualization image $V$
29:     end if
30: end for
31: return $Gaps$, $V$
```

Step 2: Generate reference waypoints for parking

We have 7 reference waypoints around the parking space as shown in 3.

$$pt_1 = m + (d * k/2)$$

$$pt_2 = pt_1 + (waypoint\_offset * v)$$

$$pt_3 = pt_1 - (waypoint\_offset * v)$$

$$pt_4 = p1 + d * k + (gap\_offset * k)$$

$$pt_5 = pt_4 - (waypoint\_offset * v)$$

$$pt_6 = pt_5 - (waypoint\_offset * v)$$

$$pt_7 = pt_3 + (gap\_offset * k) + (gap\_offset * k/2)$$

pt_1 is the center of the parking space. pt_2 and pt_3 are the two points aligned with the center point but with a tunable offset. pt_4, pt_5, pt_6 are points in a row outside the parking slot. pt_7 is the point between the boundary of the parking spot and the outer waypoints, and it is aligned with pt_2 in the direction of k. The waypoint visualization can be found in 3.

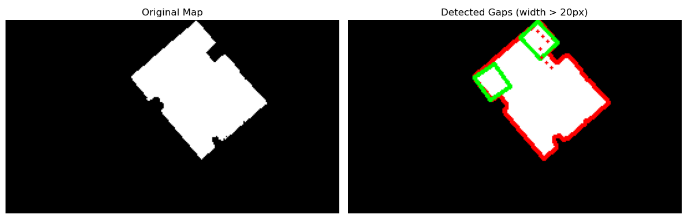


Fig. 3: Gap detection and waypoint generation

C. Trajectory Generation Module

The Trajectory Generation Module constructs a dynamically feasible reference trajectory for the MPPI controller to follow, using the stage-wise goal points provided by the previous planning module. This module ensures that the resulting path is smooth and collision-free, avoiding sharp turns, unnecessary oscillations, or trajectories that intersect static obstacles or walls. Such undesirable paths can hinder the MPPI controller's convergence or lead to unstable behavior.

Depending on the scenario, the system selects between two trajectory generation strategies:

- `generateAckermannWaypoints()`: Uses a kinematic Ackermann steering model to simulate feasible vehicle paths, ideal for structured or narrow environments.
- `generate_s_curve_waypoints()`: Employs a smoother interpolation method, generating gentle curvature changes suitable for open spaces or when minimal control input variation is desired.

Given an initial vehicle pose (x_0, y_0, θ_0) and a target pose (x_g, y_g, θ_g), the Ackermann-based model iteratively computes the vehicle's motion as follows:

$$x_{t+1} = x_t + v\cos(\theta_t) \cdot \Delta t$$
$$y_{t+1} = y_t + v\sin(\theta_t) \cdot \Delta t$$
$$\theta_{t+1} = \theta_t + \frac{v}{L}\tan(\delta_t) \cdot \Delta t$$

where:

- v is the forward velocity,
- L is the vehicle's wheelbase,
- δ_t is the steering angle computed by a lookahead-based controller:

$$\delta_t = \text{clip}\left(\arctan\left(\frac{2L\sin(\theta_g - \theta_t)}{\ell_d}\right), -\delta_{\max}, \delta_{\max}\right)$$

Here, ℓ_d is a fixed lookahead distance, and $\delta_{\max}$ bounds the maximum steering angle. Trajectory generation halts when the pose error—both positional and angular—falls below a predefined threshold.

The logic for trajectory generation and control rollout is summarized in Algorithm 2. The generated trajectory serves as a reference path, which the MPPI controller tracks using the system dynamics and cost function feedback. The process repeats stage-by-stage until the global goal is reached.

Algorithm 2 Trajectory generation and tracking

```
Require: Initial pose $x_{init}$, Stage-wise goal set $G_{goal}$, Current state $x_S$, Generator $T(x_{init}, p_{goal})$, Controller $MPPI(x_S)$, Vehicle dynamics $f$
1: Hyperparameter: Goal pose tolerance $\tau_{goal}$
2: $p_{goal} \sim G_{goal}$                          ▷ Sample next stage goal
3: $x_{\text{ref}} \leftarrow T(x_{init}, p_{goal})$        ▷ Generate reference trajectory
4: $x_0 \leftarrow x_{init}$
5: while $||x_k - p_{goal}|| \geq \tau_{goal}$ do
6:     $u_k \leftarrow MPPI(x_k, x_{\text{ref}})$          ▷ Compute optimal control
7:     $x_{k+1} \leftarrow f(x_k, u_k)$                ▷ Update state with dynamics
8: end while
9: Repeat for next stage goal
```

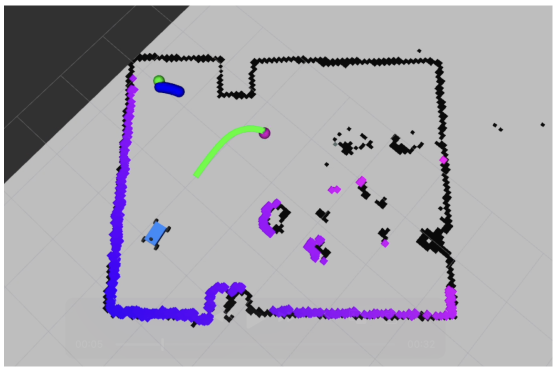


Fig. 4: Trajectory Generation from car postion to stage-wise goal point

D. MPPI Control Implementation

Model Predictive Path Integral (MPPI) control is a sampling-based trajectory optimization method designed to handle nonlinear systems with complex cost structures. At its core, MPPI aims to solve the stochastic optimal control problem:

$$\mathbf{U}^* = \arg\min_{\mathbf{U}} \mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(0,\Sigma)} \left[\sum_{t=0}^{T-1} C(\mathbf{x}_t, \mathbf{u}_t + \boldsymbol{\epsilon}_t) \right]$$

where:

- $\mathbf{U} = (\mathbf{u}_0, \ldots, \mathbf{u}_{T-1})$ is the control sequence over a horizon T,
- $\boldsymbol{\epsilon}_t \sim \mathcal{N}(0, \Sigma)$ is zero-mean Gaussian noise added to each control input,
- $C(\cdot)$ is a cost function defined over states and control inputs.

At each iteration, the algorithm proceeds as follows:

1) Generate K control sequences by adding noise to the previous optimal sequence.
2) Simulate each trajectory using the system dynamics f, i.e., $\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t)$.
3) Evaluate the cost S^k of each trajectory using a weighted cost function.
4) Compute weights using:

$$w^k = \frac{\exp\left(-\frac{1}{\lambda}S^k\right)}{\sum_{j=1}^{K} \exp\left(-\frac{1}{\lambda}S^j\right)}$$

5) Update the control sequence:

$$\mathbf{U} \leftarrow \mathbf{U} + \sum_{k=1}^{K} w^k \boldsymbol{\epsilon}^k$$

6) Apply the first control input $\mathbf{u}_0$, and repeat at the next step.

Cost Function Design: The effectiveness of MPPI is strongly determined by the cost function. Ours incorporates multiple objectives relevant to autonomous parking:

- **Terminal state error:** Encourages the final state x_N to match the desired goal x_r.
- **Trajectory tracking:** Penalizes deviation from a reference trajectory throughout the horizon.
- **Obstacle avoidance:** Applies large penalties to states close to known obstacles.
- **Constraint violations:** Penalizes control inputs that exceed actuation or dynamic limits.

The total cost for a trajectory is:

$$J = C_{\text{state}} \Bigg((x_N - x_r)^T Q_N (x_N - x_r) + \sum_{k=0}^{N-1} (x_k - x_r)^T Q (x_k - x_r) \Bigg) + C_{\text{obs}} \cdot \sum_{k=0}^{N-1} \text{Obs}(x_k)$$

where:

- Q_N and Q are weight matrices for terminal and running costs,
- C_{state} and C_{obs} are scalar weights,
- $\text{Obs}(x_k)$ is the obstacle cost based on distance to the nearest obstacle:

$$Obs(\mathbf{x_k}) = \frac{\alpha}{\min_i \|\mathbf{x_k} - \mathbf{o}_i\|^2 + \delta}$$

Our MPPI Customizations: We extend the baseline MPPI controller with several domain-specific improvements:

- **Reverse Mode Switching:** If the goal lies behind the vehicle's heading, MPPI automatically switches to a reverse control mode.

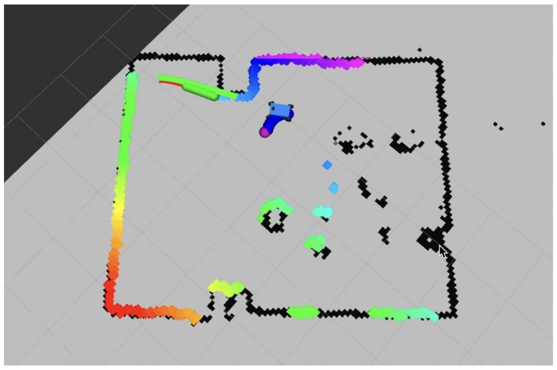


Fig. 5: Car stop at first stage goal point

- **Early Termination:** The algorithm stops sampling once the trajectory reaches the goal within a threshold, saving compute and improving stability.
- **Obstacle-aware Planning:** The cost function includes proximity-based penalties that strongly discourage paths intersecting static obstacles.

Algorithm 3 Model Predictive Path Integral (MPPI) Control

```
Require: Previous control sequence $\mathbf{U}_{0:T-1}$, dynamics model $f$, cost function $C$, number of samples $K$, temperature $\lambda$, goal position $p_{goal}$
Ensure: Optimal control $u_0$ to apply at current step
 1: if $p_{goal}$ is behind the car then
 2:     adjust control mode (e.g., switch to reverse)
 3: end if
 4: for each step $k$ do
 5:     $\epsilon^k_{0:T-1} \sim \mathcal{N}(0, \Sigma)$                ▷ Sample control noise
 6:     $\mathbf{U}^k = \mathbf{U}_{0:T-1} + \epsilon^k_{0:T-1}$   ▷ Generate noisy control sequence
 7:     $\mathbf{X}^k \leftarrow f(\mathbf{x}_0, \mathbf{U}^k)$                ▷ Simulate trajectory
 8:     $S^k = C(\mathbf{X}^k, \mathbf{U}^k)$                 ▷ Evaluate trajectory cost
 9: end for
10: $w^k = \frac{\exp(-\frac{1}{\lambda}S^k)}{\sum_{j=1}^K \exp(-\frac{1}{\lambda}S^j)}$                ▷ Compute weights
11: $\mathbf{U}_{0:T-1} \leftarrow \mathbf{U}_{0:T-1} + \sum_{k=1}^K w^k \epsilon^k_{0:T-1}$     ▷ Update control sequence
12: $u_0 \leftarrow \mathbf{U}_0$                          ▷ Apply first control input
13: if goal proximity is within threshold then
14:     terminate early
15: end if
```

6 shows the direct reverse with MPPI. The full video can be found: https://youtube.com/shorts/FtsG7Xy8tEs

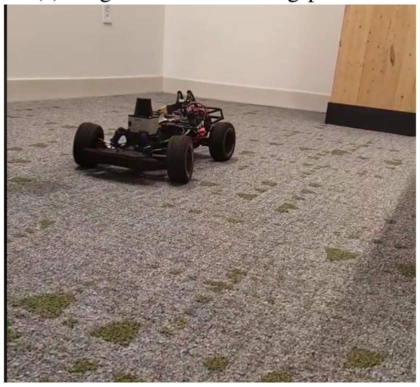
E. Fallback Mechanisms

A key contribution of our approach is the development of robust fallback mechanisms when MPPI fails to find feasible trajectories. We implement the following.

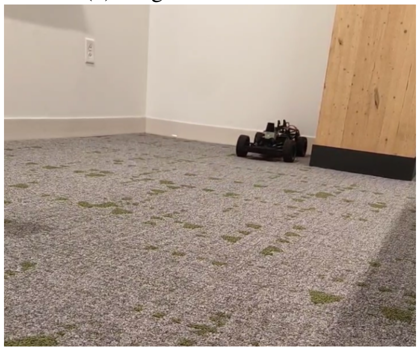
- Multi-point maneuvering when direct approaches fail
- Temporary retreating to create space for more effective approaches
- Gradual refinement of position through iterative micro-adjustments when near the target

These fallback strategies are triggered by specific conditions detected during the optimization process and ensure that the system can recover from challenging initial conditions.

(a) Stage 1 reverse starting position



(b) Stage 1 reverse MPPI



(c) Stage 1 reverse MPPI accomplished

Fig. 6: Stage 1 reverse with MPPI

IV. Results

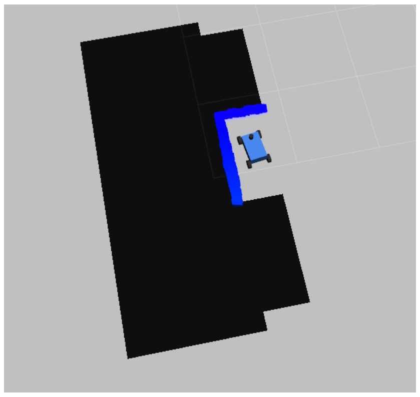
A. Hardware

We evaluate our autonomous parallel parking system on the NVIDIA Jetson Orin Nano platform, using a SICK LiDAR sensor for environment perception and SLAM for real-time map generation.

B. Experiment

We implemented and tested our approach in both simulation and real-world hardware environments. In simulation, we evaluated performance across three distinct parking scenarios:

two manually defined slots within a custom-designed map, and one slot extracted from a 2D LiDAR-based scan of a study room in Amy Gutmann Hall. The figure for slot1 and slot2 are shown below in the fig 7a and fig 7b respectively. As for the AGH study room map please refer to the fig 4. To assess the effectiveness of our approach, we compared


(a) Slot 1 Map

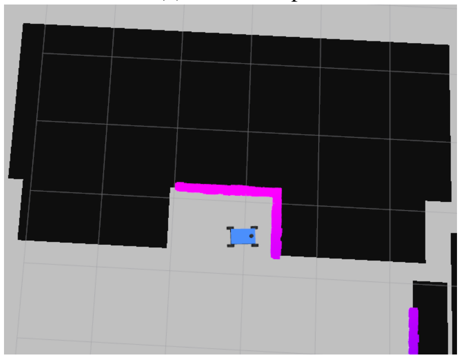

(b) Slot 2 Map

Fig. 7: Driving modes: First driving backward to the waypoint that close to and provid a better starting point, and doing a forward driving to park the vehicle into the destination

our MPPI-based parking system against a pure pursuit baseline. The performance metrics, including position error and yaw error, are reported in Table I, Table II, and Table III, corresponding to Manual Slot 1, Manual Slot 2, and the Amy Gutmann Hall map, respectively. The results demonstrate that the MPPI controller consistently outperforms pure pursuit, particularly in terms of final pose accuracy. This improvement stems from MPPI's ability to explicitly optimize over both obstacle avoidance and yaw correction through its cost function. While pure pursuit can navigate into the parking slot, it lacks fine-grained control over the final orientation, resulting in larger yaw errors and less precise alignments with the desired pose.

The manually constructed simulation maps facilitated rapid development and debugging due to their smooth and predictable boundaries. In contrast, the Amy Gutmann Hall environment introduced greater complexity and realism. Notably, a fixed beam positioned near the parking wall created a narrow and constrained slot, offering a valuable stress test for our planner's obstacle avoidance and trajectory precision capabilities.

On the hardware side, deploying the system required extensive tuning of both sensor calibration and control parameters. Despite these challenges, the system achieved reliable performance in approximately 75 percent of trials. One significant limitation encountered on hardware was the latency of the MPPI control loop, which made high-speed operation difficult. To mitigate this, we operated the vehicle at a low speed to ensure that control updates had sufficient time to compute and execute effectively in real time.

Furthermore, our hardware experiments primarily followed a two-stage maneuver: the vehicle first reversed to a staging point near the parking slot, then executed a forward motion into the slot (as illustrated in Fig.8b and Fig.8a). This strategy was chosen based on the observation that MPPI exhibited instability during backward motion—particularly during tight turning maneuvers—which led to overshooting and poor trajectory correction. These limitations made reverse-only parking infeasible in cluttered environments, as the controller was often unable to reorient the vehicle quickly enough to avoid collisions. However dispite the instability, our mppi could still be able to successfully do a reverse parking for a less constraint terrain as shown in fig 6 on the real car.

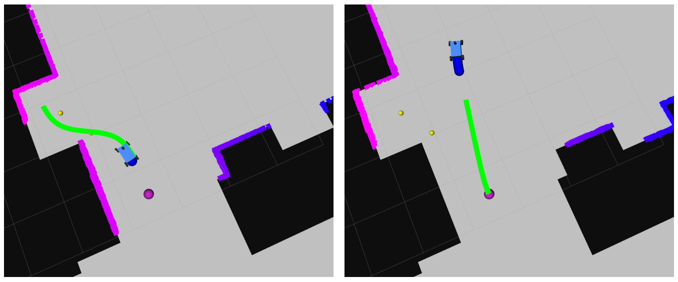

(a) Forward driving (b) Reverse driving

Fig. 8: Driving modes: First driving backward to the waypoint that close to the slot and provide a better starting point. Then doing a forward driving to park the vehicle into the desire position.

TABLE I: Comparison of Position and Yaw Error Between Our method and Pure Pursuit for Manual slot 1

Method	Position Error (m)	Yaw Error (°)
Our method	0.26	0.27
Pure Pursuit	0.4	0.72

TABLE II: Comparison of Position and Yaw Error Between Our method and Pure Pursuit for Manual slot 2

Method	Position Error (m)	Yaw Error (°)
Our method	0.21	0.32
Pure Pursuit	0.41	0.65

TABLE III: Comparison of Position and Yaw Error Between Our method and Pure Pursuit for AGH study room

Method	**Position Error (m)**	**Yaw Error (°)**
Our method	0.37	0.4
Pure Pursuit	0.5	0.63

C. Videos:

- autonomous parking task in simulation: https://youtu.be/hYVC5iGvg9w
- Reverse-first parallel parking: https://youtu.be/64MqaUWPIyM
- Reverse parking: https://youtube.com/shorts/k9jr5C9r-qE?feature=share

V. Discussion and Future Work

Our current system successfully performs parallel parking in a variety of non-trivial scenarios. However, the resulting parking positions, while feasible, are often sub-optimal in terms of lateral alignment and distance from the curb. Fine-tuning the trajectory generation and cost function components of the MPPI controller will be essential to improve final positioning accuracy.

At present, the MPPI planner is executed within a Docker container, which introduces additional latency due to container overhead and GPU communication delays. This latency becomes especially noticeable during high-frequency control loops, where real-time performance is critical. As part of future work, we aim to reduce this overhead either by moving to a native execution environment or by optimizing the Docker setup to minimize timing bottlenecks.

We observe a substantial sim-to-real gap, especially in vehicle speed control and trajectory adherence. The simulated model does not always reflect the physical limitations and friction characteristics of the real vehicle, leading to overshooting behaviors—particularly in strategies that involve moving forward and then reversing into the parking space. Addressing this discrepancy will require improved system identification, domain randomization in simulation, or incorporating learned dynamics models that adapt online.

In future iterations, we also intend to evaluate additional planning approaches (e.g., hybrid A* or lattice-based planners) as fallbacks, incorporate onboard perception feedback for dynamic obstacle avoidance during parking, and extend testing to include more complex parking scenarios such as angled and perpendicular parking.

References

[1] G. Williams, A. Aldrich, and E. Theodorou, “Model predictive path integral control using covariance variable importance sampling,” *arXiv preprint arXiv:1509.01149*, 2015.

[2] I. Paromtchik and C. Laugier, “Autonomous parallel parking of a non-holonomic vehicle,” in *Proceedings of Conference on Intelligent Vehicles*, 1996, pp. 13–18.

[3] H. Vorobieva, N. Minoiu-Enache, S. Glaser, and S. Mammar, “Geometric continuous-curvature path planning for automatic parallel parking,” in *2013 10th IEEE INTERNATIONAL CONFERENCE ON NETWORKING, SENSING AND CONTROL (ICNSC)*, 2013, pp. 418–423.

[4] J. Ziegler, P. Bender, T. Dang, and C. Stiller, “Trajectory planning for bertha — a local, continuous method,” in *2014 IEEE Intelligent Vehicles Symposium Proceedings*, 2014, pp. 450–457.

[5] G. Williams, A. Aldrich, and E. Theodorou, “Model predictive path integral control: From theory to parallel computation,” *Journal of Guidance, Control, and Dynamics*, vol. 40, pp. 1–14, 01 2017.

[6] R. Takehara and T. Gonsalves, “Autonomous car parking system using deep reinforcement learning,” in *2021 2nd International Conference on Innovative and Creative Information Technology (ICITech)*, 2021, pp. 85–89.