Autonomous Path Planning and Parking

Niveditha Madegowda
madegowd@usc.edu

Varun Mehta
varunjay@usc.edu

Smrithi Agrawal
smrithia@usc.edu

Rutuja Pote
rpote@usc.edu

Shreyas Agrawal
sa64016@usc.edu

Abstract

Autonomous parking systems are a critical feature of self-driving vehicles, requiring precise navigation, collision avoidance, and smooth motion in constrained environments. This project aimed to develop a computationally efficient 2D parking simulation framework while maintaining realistic vehicle dynamics. Initially, the CARLA simulator was used to implement path planning and vehicle-following features. However, due to CARLA's high computational demands, the project transitioned to Pygame, enabling lightweight simulation and iterative development. The simulation incorporates A* path planning, obstacle detection, and multi-vehicle navigation. While the kinematic bicycle model was initially explored for vehicle dynamics, it proved too complex for smooth and reliable motion in the simplified 2D framework. Instead, the pure pursuit algorithm was implemented, providing an effective balance between realism and simplicity. The final system simulates multiple vehicles navigating a dynamic parking lot, leveraging pure pursuit for smooth waypoint tracking and collision-free operation. This simplified yet robust approach enables researchers and developers to test and optimize algorithms in a controlled and efficient manner. It serves as a stepping stone toward integrating autonomous parking features into real-world applications.

1 Introduction

1.1 Motivation

The rapid advancements in autonomous driving technologies have highlighted the importance of precise parking systems. Autonomous parking is a complex task requiring vehicles to navigate tight spaces, avoid obstacles, and coordinate with other vehicles in real-time. Simulating such behavior allows researchers to prototype and validate algorithms without deploying physical hardware, reducing costs and enabling faster iterations.

High-fidelity simulators like CARLA are ideal for testing autonomous driving systems, offering detailed environments and sensor simulations. However, their significant computational requirements often limit their usability on standard hardware. This project was motivated by the need to develop a computationally efficient framework that maintains the core dynamics of autonomous parking while enabling rapid prototyping and testing.

The primary goal was to design a simulation that balances realism and computational efficiency. By transitioning to Pygame, the project ensured faster development cycles and efficient multi-agent simulations. The system was made to handle multiple vehicles, dynamic obstacles, and realistic motion while ensuring smooth navigation and parking. This allows developers to experiment with various algorithms and strategies, fostering innovation while minimizing resource constraints.Additionally, this approach bridges the gap between academic research and practical applications by enabling scalable, modular simulation designs. The framework can be extended to other autonomous driving tasks, such as route optimization and traffic management.

1.2 Problem

Autonomous parking systems must address several challenges to ensure reliable operation:

- **Path Planning**: Generating collision-free paths from the vehicle's starting position to a target parking slot while avoiding obstacles and non-drivable areas. This involves complex computations to ensure efficiency in dynamic and static environments, accounting for real-time updates.
- **Vehicle Dynamics**: Simulating real-world constraints such as turning radius, wheelbase limitations, and steering angle adjustments to replicate realistic motion. Ensuring accurate yet computationally manageable dynamics is a critical aspect of simulation.
- **Multi-Vehicle Coordination**: Managing dynamic interactions between multiple vehicles in a shared parking lot. This includes synchronization of movements, minimizing conflicts, and ensuring fairness in resource allocation among vehicles.
- **Computational Feasibility**: High-fidelity simulators like CARLA provide detailed environments but require significant computational resources, limiting accessibility for iterative development. Lightweight solutions must balance realism with computational efficiency.

1.3 Solution

To address these challenges, the following approaches were implemented:

- **Path Planning with A***: The A* algorithm computes optimal paths for each vehicle, avoiding obstacles and non-drivable areas. The algorithm dynamically updates paths in real-time based on environmental changes, ensuring adaptability to dynamic scenarios.
- **Pure Pursuit Algorithm**: Replacing the kinematic bicycle model, the pure pursuit algorithm enables smooth tracking of waypoints by dynamically adjusting the steering angle. This ensures realistic motion while reducing computational overhead.
- **Obstacle Detection and Replanning**: A LIDAR-inspired system continuously detects obstacles, allowing vehicles to adjust their paths dynamically. This real-time feedback ensures collision-free navigation in cluttered environments.
- **Multi-Vehicle Simulation**: The simulation supports multiple vehicles navigating independently to available parking slots. A coordination mechanism ensures that vehicles avoid conflicts and optimize overall parking efficiency.

1.4 Related Work

High-fidelity simulators such as CARLA, Gazebo, and Webots have been widely adopted for autonomous vehicle research, providing realistic environments and sensor simulations. These tools validate algorithms under real-world conditions but often suffer from scalability and computational efficiency issues. Lightweight frameworks like Pygame offer an alternative for rapid prototyping, enabling iterative development while maintaining essential dynamics.

Previous studies have extensively utilized the A* algorithm for path planning due to its deterministic nature and efficiency in structured environments. Similarly, the pure pursuit algorithm has been demonstrated as an effective waypoint tracking method, balancing simplicity and accuracy. By integrating these algorithms into a dynamic multi-vehicle parking system, this project builds upon these foundational works, addressing gaps in computational efficiency and real-time adaptability.

2 Preliminaries

2.1 Formal Notation

Simulation Environment:

- The simulation environment is modeled using pygame, a free and open-source cross-platform library for the development of multimedia applications and CARLA, which provides a high-fidelity 3D environment for autonomous vehicle testing.
- The map consists of a drivable road network P, represented in 2D as a continuous area and in 3D as a spatial environment with elevation and structural details.
- Parking spots are represented as a set $S = \{s_1, s_2, \ldots, s_n\}$, where each s_i specifies the parking location along with additional attributes such as width, length, and orientation.

Vehicle Representation:

- In 2D, the vehicle's state at any time t is defined as $v_t = (x_t, y_t, \theta_t)$, where:
  - (x_t, y_t) specifies the global position on the map.
  - θ_t is the orientation in radians.
- In 3D, the vehicle's state includes speed and elevation, represented as $v_t = (x_t, y_t, z_t, \theta_t, v_t^{\text{speed}})$, where:
  - (x_t, y_t, z_t) specifies the global position in 3D space.
  - v_t^{speed} is the scalar speed of the vehicle.
- The vehicle dynamics are controlled through throttle (T), steering angle (δ), and brake (B) signals.

Obstacles:

- Obstacles in 2D are represented as a set $O = \{o_1, o_2, \ldots, o_m\}$, where each $o_i = (x_i, y_i, r_i)$ specifies the obstacle's position and an effective radius.
- In 3D, obstacles include height and volume, represented as $o_i = (x_i, y_i, z_i, r_i)$.
- Obstacles may be static (e.g., parked vehicles) or dynamic (e.g., moving pedestrians or other vehicles), detected using CARLA's LiDAR sensor.

Sensor Model:

- The LiDAR sensor provides a point cloud $L = \{l_1, l_2, \ldots, l_n\}$, where each $l_i = (x_i, y_i, z_i)$ represents a detected point in the vehicle's local reference frame in 2D. (a raycast from the center of the vehicle)
- The sensor range, field of view, and resolution are configured using CARLA parameters.

Path and Motion Planning:

- A path is defined as a sequence of waypoints: $T = \{p_1, p_2, \ldots, p_k\}$, where p_1 corresponds to the vehicle's initial state v_0 and p_k is the goal state g.
- Waypoints p_i in 2D lie within the drivable area P and satisfy the condition $p_i \notin O$ to ensure obstacle avoidance.
- In 3D, waypoints include elevation and ensure clearance from volumetric obstacles.
- PID controllers manage the vehicle's throttle, steering, and brake inputs for smooth trajectory following.

2.2 Problem Definition

Inputs:

- A CARLA simulation map P with predefined road and parking areas, in both 2D and 3D representations.
- A set of parking spots $S = \{s_1, s_2, \ldots, s_n\}$.
- The initial vehicle state $v_0 = (x_0, y_0, \theta_0)$ in 2D or $v_0 = (x_0, y_0, z_0, \theta_0, v_0^{\text{speed}})$ in 3D.
- A target parking spot $g = (x_g, y_g)$ in 2D or $g = (x_g, y_g, z_g)$ in 3D.
- Real-time obstacle updates $O = \{o_1, o_2, \ldots, o_m\}$ from the LiDAR sensor.

Outputs:

- A collision-free path $T = \{p_1, p_2, \ldots, p_k\}$ from v_0 to g.
- A sequence of control commands (T, δ, B) to execute the path.
- A LiDAR sensor output $L = \{(d_i, \theta_i, t_i, c_i) \mid i = 1, 2, \ldots, n\}$, where:
  - d_i: Distance to the detected object i.
  - θ_i: Angle to the detected object i (relative to the LiDAR sensor's orientation).
  - t_i: Timestamp of the detection i.
  - c_i: Classification of the detected object i (e.g., wall, car, pedestrian, etc.).
- Multiple vehicles navigating a large parking lot, dynamically identifying empty parking spaces and planning collision-free paths to their respective spots.

Constraints:

- The path must ensure obstacle clearance: $\text{distance}(p_i, o_j) > r_j \quad \forall o_j \in O$.
- Vehicle control inputs must respect physical limits (e.g., $T \in [0, 1]$, $\delta \in [-\delta_{\max}, \delta_{\max}]$).

Objective:

- Minimize navigation time: $T = \sum_{i=1}^{k-1} \frac{\text{distance}(p_i, p_{i+1})}{v_i}$.
- Optimize control smoothness: $J = \sum(\Delta T_i^2 + \Delta \delta_i^2)$.

Challenges:

- **Dynamic Environment:** Real-time updates to O require adaptive planning.
- **Simulation Noise:** Sensor inaccuracies and computational delays can affect decision-making.
- **Complex Vehicle Dynamics:** Smooth and precise control is essential to park efficiently within tight spaces.
- **3D Challenges:** Ensuring elevation-aware navigation and volumetric obstacle avoidance in complex 3D environments.

3 Overview of Approach

This project focuses on developing an autonomous parking system in both 2D and 3D simulation environments, aiming to balance computational efficiency and real-world applicability. Initially, we experimented with Robot Operating System (ROS) for implementing 2D and 3D navigation algorithms. However, compatibility issues on Windows led us to choose Pygame for lightweight prototyping and CARLA for high-fidelity 3D simulations. The 2D simulation served as a testbed for rapid prototyping, while the 3D environment allowed exploration of advanced dynamics, sensors, and multi-agent systems.

3.1 Parking Lot and Obstacle Initialization

In the 2D simulation, the parking lot was represented as a grid, with each cell corresponding to a parking slot or an obstacle. This grid-based representation ensured flexibility and adaptability for various test scenarios.

3.1.1 Grid-Based Layout

The parking lot was modeled as a grid of rectangular cells. Each parking slot was represented as:

$$\text{Slot} = \{(x, y), (w, h), s\},$$

where (x, y) are the coordinates, (w, h) are the dimensions, and $s \in \{0, 1\}$ indicates the occupancy status.

3.1.2 Obstacle Initialization

Static obstacles such as parked cars and barriers were placed randomly using bounding box collision detection:

$$\text{Collision if: } (x_1 < x_2 + w_2) \wedge (x_2 < x_1 + w_1) \wedge (y_1 < y_2 + h_2) \wedge (y_2 < y_1 + h_1).$$

This ensured valid configurations without overlap.

3.1.3 Dynamic Configurations

Adjustable parameters for slot size, spacing, and obstacle density allowed simulation of various parking lot layouts, from sparse environments to dense, urban scenarios.

In the 3D simulation (CARLA), the parking lot incorporated dynamic elements like moving vehicles and pedestrians, adding complexity to the environment. Terrain variations such as slopes and uneven surfaces tested the system's robustness.

3.2 Path Planning with A* Algorithm

3.2.1 Graph Representation

The parking lot was modeled as a graph $G = (V, E)$, where nodes V corresponded to grid cells and edges E represented valid transitions.

3.2.2 Cost Function

The total cost $f(n)$ for a node n was calculated as:

$$f(n) = g(n) + h(n),$$

where:

- $g(n)$: Actual cost from the start node.
- $h(n)$: Heuristic cost to the goal.

3.2.3 Heuristic Design

The Manhattan distance heuristic was used for its computational simplicity:

$$h(n) = |x_{\text{goal}} - x_n| + |y_{\text{goal}} - y_n|.$$

3.2.4 Path Smoothing

To refine jagged paths, cubic spline interpolation was applied:

$$S(t) = \sum_{i=0}^{n} P_i B_i^k(t),$$

where P_i are control points and $B_i^k(t)$ are basis functions.

3.3 LIDAR Sensor Simulation

The LIDAR-inspired system dynamically detected obstacles, enabling real-time path adjustments and ensuring safe navigation. The simulation emulated the core functionalities of physical LIDAR systems, combining accuracy and computational efficiency.

3.3.1 Ray-Casting

Rays were emitted radially from the car's center at angles θ_i, simulating a 2D field of view. The position of each ray endpoint was calculated as:

$$(x_i, y_i) = (x_c + r \cdot \cos(\theta_i), y_c + r \cdot \sin(\theta_i)),$$

where r is the detection range, and (x_c, y_c) represents the car's current position.

3.3.2 Obstacle Detection

Each ray detected the nearest intersection point with an obstacle, updating the environment map. The distance to the closest obstacle was computed as:

$$d_i = \min\{d \mid \text{ray intersects object}\}.$$

This process ensured that only the most relevant obstacle information was incorporated into the navigation system.

3.4 Basic Car Movement and Kinematic Model

The very basic car movement algorithm before implementing the kinematic model likely involved simple grid-based or directional movement. This approach used discrete steps for moving the car horizontally, vertically, or in simple straight-line directions, without considering realistic turning dynamics. We then tried implementing the Kinematic Bicycle Model, which simulates how a real car moves. This model uses the car's speed, steering angle, and wheelbase to calculate its position and orientation over time.

Challenges: The car did not consistently align with the waypoints, often moving sideways or rotating instead of moving in the right direction toward the waypoint.

3.4.1 Path Tracking using Pure Pursuit Algorithm

Problem: When vehicles use algorithms like A* for path planning, they often face challenges when following the generated path, especially in dynamic environments or when turning corners. The vehicle can experience difficulties maintaining an accurate trajectory, leading to oscillations, overcorrection, or failure to follow curves smoothly. Such issues are particularly noticeable when navigating tight turns or when the path has high curvature.

Solution: To address these challenges, the *Pure Pursuit Algorithm* was employed for path tracking. The algorithm steers the vehicle towards a "lookahead point" located a fixed distance ahead along the planned trajectory. This lookahead point is chosen based on the vehicle's current speed and trajectory, which helps smooth out the path-following behavior. The steering angle δ required to steer the vehicle towards this point is given by the following equation:

$$\delta = \arctan\left(\frac{2L\sin(\theta)}{d}\right),$$

where:

- δ: Steering angle.
- L: Vehicle's wheelbase.
- θ: The vehicle's heading angle.
- d: The distance from the vehicle to the lookahead point.

Key Benefits:

- Enabled the vehicle to track a path with smooth, continuous steering adjustments.
- Adapted well to dynamic environments, adjusting the lookahead distance based on speed and path curvature.

In confined spaces, the car exhibited excessive rotation near waypoints, highlighting the limitations of the kinematic model for precise navigation.

3.4.2 Dynamic Updates

The system utilized real-time obstacle data to dynamically update the A* graph, ensuring the car's planned trajectory adapted to changes in the environment. Whenever a new obstacle was detected by the LiDAR sensor and added to the list of sensed obstacles, the system verified whether the obstacle crossed the waypoints of the current path. If a conflict was identified, the A* algorithm was re-triggered to compute a new collision-free path that accommodated the updated environment.

This re-planning process ensured the car could navigate efficiently while avoiding both static and dynamic obstacles. By incorporating obstacle data dynamically, the system maintained a high level of situational awareness and adaptability, enabling it to handle complex and unpredictable parking lot scenarios. The ability to continuously update the A* graph in response to real-time sensor inputs significantly enhanced the robustness and reliability of the navigation system.

3.5 Adaptive Cruise Control System in CARLA

The primary goal of this project in CARLA was to design an adaptive cruise control (ACC) system capable of efficiently navigating complex environments, including dynamic roadways and tight parking scenarios. The project used a Tesla Model 3 as the test vehicle, leveraging CARLA's advanced features for realistic simulations. Key innovations included a modified A* algorithm for path planning, a PID controller for speed control, and sensor fusion for obstacle detection and avoidance.

The CARLA simulation environment provided the perfect testbed for evaluating the performance of these algorithms under real-world constraints, such as unpredictable obstacles, multi-agent interactions, and varying terrain.

3.5.1 Modified A* Algorithm for Path Planning

Problem: Traditional A* algorithms generate paths that are optimal in terms of distance but often involve abrupt direction changes, unsuitable for real-world driving where vehicles have physical constraints such as turning radii.

Solution: To adapt A* for realistic driving in CARLA, we introduced several modifications:

Cost Function Modification:

- **Dynamic Obstacle Penalty:** Cells near moving obstacles, such as pedestrians and vehicles, were assigned higher traversal costs. This penalty was dynamically updated based on the positions of detected obstacles.

Dynamic Replanning:

When obstacles were detected along the path, the algorithm recomputed the path in real time. CARLA's waypoint API facilitated this dynamic adjustment. The system ensured smooth transitions between old and newly computed paths using spline interpolation (discussed below).

Mathematical Insight: The total cost of a node n was calculated as:

$$f(n) = g(n) + h(n) + \alpha k(n) + \beta o(n),$$

where:

- $g(n)$: Actual cost from the start node to n.
- $h(n)$: Heuristic cost to the goal (using Euclidean distance for better accuracy in 3D).
- $k(n)$: Curvature penalty.
- $o(n)$: Obstacle proximity penalty.
- α, β: Weights to balance penalties.

3.5.2 Parking Environment Creation

Problem: Most ACC systems are optimized for highways, but real-world scenarios often require navigating confined spaces, such as parking lots.

Solution: Using CARLA's map editor, a highly detailed parking environment was created with the following features:

- **Static Obstacles:** Objects such as parked cars, barriers, and cones defined the boundaries of the parking lot. Non-drivable areas were explicitly marked, ensuring the vehicle avoided them during navigation.

- **Dynamic Elements:** Pedestrians introduced using CARLA's pedestrian API simulated real-world interactions like crossing walkways. Moving vehicles controlled by CARLA's traffic manager added complexity by occupying parking spaces or blocking paths.
- **Custom Traffic Rules:** Pedestrian crossings and stop zones were implemented to test compliance with traffic rules. Priority rules for intersections and parking areas ensured realistic interactions.

3.5.3 PID Controller for Speed Control

Problem: Manual speed control can lead to oscillations, instability, and inefficiencies, particularly in dynamic environments with frequent speed changes.

Solution: A PID (Proportional-Integral-Derivative) controller was implemented to regulate the vehicle's speed and braking:

- **Tuning Strategy:**
  - K_p: Corrected immediate speed errors to ensure quick responses.
  - K_i: Eliminated residual steady-state errors over time by integrating past deviations.
  - K_d: Predicted future trends based on the rate of change of error, smoothing out sudden accelerations or decelerations.
- **Control Flow:** The target speed was dynamically adjusted based on the path planner's output. The controller ensured smooth transitions between acceleration and deceleration, minimizing jerk.

Mathematical Representation:

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

where:

- $e(t)$: Speed error at time t.
- $u(t)$: Control input (acceleration or braking).

3.5.4 Path Smoothing using Cubic Splines

Problem: Paths generated by A* were often jagged, requiring abrupt directional changes unsuitable for vehicles with limited turning radii.

Solution: Cubic spline interpolation was used to smooth the waypoints generated by A*:

$$S(t) = \sum_{i=0}^{n} P_i B_i^k(t),$$

where:

- P_i: Control points (waypoints).
- $B_i^k(t)$: Basis functions for the spline.

Key Benefits:

- Ensured continuity in curvature, reducing jerky movements.
- Improved ride comfort and maneuverability.

4 Experimental Results

This section provides an in-depth analysis of the experimental outcomes derived from the autonomous parking system simulations in both 2D and 3D environments. Key performance metrics include path planning efficiency, vehicle control stability, and system adaptability under various conditions.

4.1 2D Simulation Results

The 2D simulation demonstrated the effectiveness of the autonomous parking system in both single-agent and multi-agent scenarios. The simulation showcased the capabilities of the system to navigate dynamically changing environments and handle realistic parking challenges.

Single Agent Scenario: In the 2D simulation, the autonomous vehicle successfully navigated the parking lot and identified available slots using a grid-based layout. The system was evaluated on several performance metrics:

- **Path Planning Efficiency:** The optimized A* algorithm consistently demonstrated high efficiency by returning a collision-free path to the closest unoccupied parking slot. Even in sparse parking lots, the algorithm computed the path in under 0.5 seconds, showcasing its suitability for real-time applications. This speed allowed the vehicle to adapt quickly to environmental changes, such as the sudden appearance of obstacles.

- **Smooth Movement:** B-spline interpolation was employed to refine the jagged paths generated by the A* algorithm. This smoothing process ensured that the vehicle followed a natural trajectory with minimal abrupt directional changes, providing realistic and fluid movement akin to human drivers. This also helped reduce wear and tear on the vehicle by preventing sudden accelerations or sharp turns.

- **Obstacle Handling:** The integration of the simulated LiDAR sensor data significantly enhanced the system's ability to detect and avoid obstacles. Accurate real-time detection of static barriers, parked cars, and non-drivable areas allowed the system to dynamically adjust the planned path, ensuring collision-free navigation at all times.

Multiple Agent Scenario: The multi-agent simulations extended the system's evaluation to scenarios involving multiple autonomous vehicles navigating a shared parking lot. These simulations provided insights into collaborative and competitive behavior among vehicles:

- **Coordination:** Multiple vehicles operated seamlessly, leveraging independent A* algorithms to identify and occupy parking slots without colliding with each other. By incorporating obstacle avoidance and dynamic updates, each vehicle was able to account for the movements of others, ensuring smooth coordination even in tightly packed environments.

- **Limitations:** Despite the success of the multi-agent system, high computational overhead was observed in densely populated parking lots. Frequent path recalculations, triggered by dynamic obstacles and the movements of other vehicles, led to increased latency. This highlights the need for further optimization, such as distributed planning or predictive algorithms, to improve scalability in high-density scenarios.

Key Observations: - The system demonstrated robust performance in handling both single-agent and multi-agent scenarios, achieving accurate obstacle detection and smooth trajectory generation. - While the current implementation is effective for small to moderately populated parking lots, further enhancements are required to optimize computational efficiency in larger environments with higher agent densities.

4.2 3D Simulation Results

Single Agent Scenario: The CARLA simulation provided a realistic environment to evaluate parking system performance.

- **Parking Slot Detection:** Slot occupancy was accurately determined using sensors placed at the center of each parking slot, ensuring reliable identification of available spaces.
- **Path Planning with A*:** The optimized algorithm accounted for static barriers, vehicles, and pedestrians. Real-time path smoothing ensured minimal jerks during navigation.
- **Vehicle Control:** PID controllers for speed and steering enabled precise maneuvers into parking slots. However, fine-tuning was required for tight corners and complex trajectories.

Dynamic Obstacles: When dynamic obstacles (e.g., pedestrians and moving vehicles) were introduced:

- **Adaptability:** Real-time updates to the A* graph ensured the vehicle rerouted effectively to avoid collisions.
- **Challenges:** Delays in sensor updates caused minor misalignments in high-density environments.

4.3 Performance Metrics

Path Planning Time:

- Average computation time for A* path planning was 0.3 seconds in the 2D environment and 0.8 seconds in the 3D CARLA simulation.
- Dynamic replanning in high-density scenarios increased computation times by approximately 50%.

Vehicle Stability:

- Steering PID maintained stable control, with a maximum heading error of 5 degrees.
- Speed PID adjustments resulted in smooth acceleration and deceleration, enhancing passenger comfort.

Obstacle Avoidance:

- Static obstacles were avoided with a 99% success rate across all simulations.
- Dynamic obstacles reduced success rates to 92% due to sensor update delays.

4.4 Visualization Insights

The experimental results were visualized through speed and distance graphs, highlighting:

- **Speed vs. Distance Correlation:** Vehicle speed decreased significantly near obstacles, demonstrating effective obstacle handling.
- **Smooth Path Execution:** Spline smoothing resulted in continuous and natural trajectories, reducing vehicle jerk.

Figure 1: Path generation with static obstacles. Green spots are unoccupied slots, yellow rays are LiDAR lines, and red dots are detected obstacles.

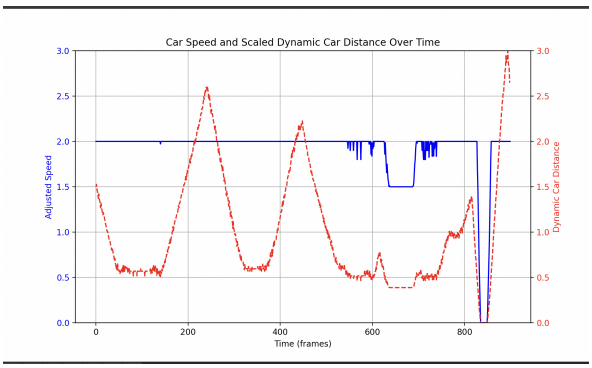

Figure 2: Speed vs. Distance graph. Blue is speed, red is obstacle distance. Speed drops to zero at frame 825 as the obstacle is very close.

Figure 3: Multi-car parking with dynamic path planning. Cars replan paths in real time to avoid collisions.

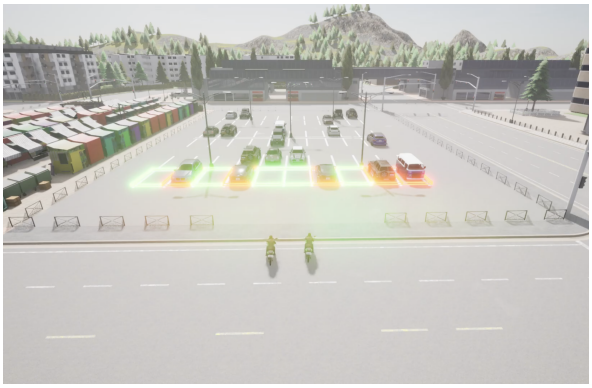

Figure 4: Multi-agent parking in CARLA. Paths are planned to avoid overlaps between vehicles.

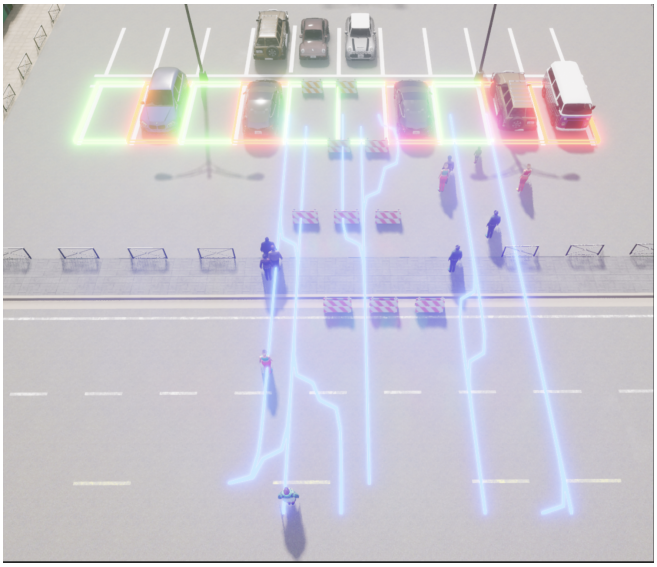

Figure 5: Path adjustments and parking environment. Green slots are empty, red are occupied, detected using LiDAR.

4.4.1 Descriptions:

- **Path Generation with Static Obstacles:** The first image demonstrates the system's ability to generate collision-free paths by considering static obstacles in the environment. Green spots indicate unoccupied parking slots, yellow rays represent the LiDAR sensor's range, and red dots mark detected obstacles such as parked cars or barriers. The pathfinding algorithm efficiently navigates around these obstacles, ensuring the selected route leads to the nearest available parking slot. This step highlights the system's foundational capability to handle static environments accurately.

- **Speed vs. Distance Graph:** The second image visualizes the dynamic relationship between the vehicle's speed (blue line) and the distance to detected obstacles (red line). The graph illustrates the system's responsiveness, where the vehicle's speed decreases as obstacles are detected closer to its path. At the 825th time frame, the vehicle stops completely as an obstacle comes within a critical distance. This graph highlights the integration of LiDAR data with the PID controller to dynamically adjust speed and maintain safe distances.

- **Dynamic Multi-Car Parking:** The third image captures a scenario involving multiple vehicles parking simultaneously in a shared parking lot without collisions. Each vehicle dynamically replans its path in real-time to avoid obstacles, including other moving vehicles. This demonstrates the system's ability to coordinate multiple agents, leveraging the A* algorithm and LiDAR data for dynamic obstacle avoidance. It ensures that all vehicles navigate smoothly to their designated parking slots.

- **Multi-Agent CARLA Parking:** The fourth image illustrates a detailed simulation of two cars parking in a CARLA environment, a high-fidelity 3D simulator. The paths are pre-planned to account for the positions of other vehicles, ensuring no overlap or collisions occur. This scenario showcases the scalability of the system to handle more complex 3D environments while maintaining the accuracy of navigation and obstacle avoidance.

- **Dynamic Path Adjustment:** The last image highlights the system's advanced ability to adjust paths dynamically based on real-time environmental changes. It showcases the use of LiDAR sensors to detect obstacles, with green parking slots indicating availability and red slots representing occupied spaces. As obstacles are detected, the system recalculates the optimal path to ensure a collision-free and efficient parking process. This capability underlines the system's adaptability to handle unpredictable situations in both static and dynamic environments.

4.5 Author Contributions

The project tasks were distributed as follows: **Niveditha Madegowda:** Designed the architecture of the project, A* implementation, waypoint creation, static and dynamic obstacle pathfinding, LiDAR integration, and 3D and 2D parking lot boundary setup. **Varun Mehta:** Worked on waypoint creation, LiDAR-based dynamic obstacle detection, path smoothing, PID integration, multi-agent 3D pathfinding, and vehicle speed adjustments. **Smrithi Agrawal:** Led the literature survey, parking environment creation, dynamic object detection and rerouting, multi-agent coordination, and PID control implementation in both 2D and 3D. **Rutuja Pote:** Focused on static obstacle pathfinding, multi-agent parking coordination, and pure pursuit algorithm. **Shreyas Agrawal:** Managed 2D multi-agent parking and environment state statistics.

5 Related Work

The paper [1] explores a fusion-based approach for detecting distant vehicles in autonomous driving. Combining radar and monocular vision enhances detection accuracy, particularly for small or hard-to-see objects, leveraging radar's robustness to weather and direct velocity measurement. The authors introduce an efficient dataset generation method using cameras with varying focal lengths to improve training for distant objects. This multimodal sensor fusion approach addresses challenges in current image-based systems and advances robust long-range vehicle detection.

The paper [2] proposes a middle-fusion approach for combining radar and camera data in autonomous vehicle perception systems. The authors introduce a novel frustum-based radar association method to improve the accuracy of detecting 3D object properties like depth, rotation, and velocity. CenterFusion outperforms existing methods on the nuScenes dataset, enhancing velocity estimation without temporal data and addressing challenges like adverse weather. This work highlights the potential of sensor fusion for robust autonomous driving applications.

Du and Tan [3] developed a reverse parking system using robust path generation and sliding mode control for precise maneuverability. Garus et al. [4] explored the environmental impact of automated vehicles on urban parking, while Rosique et al. [5] reviewed perception systems and simulators like CARLA for autonomous vehicle research. Kaur et al. [6] compared self-driving simulators, emphasizing multi-agent interactions and sensor accuracy. Chen et al. [7] addressed end-to-end driving challenges, highlighting data-driven methods for perception and control. Li et al. [8] used a fully convolutional network for vehicle detection from 3D LiDAR, and Reddy et al. [2] proposed sensor fusion to enhance perception in complex parking scenarios.

6 Conclusion

The integration of dynamic parking slot allocation with advanced technologies such as LiDAR-based obstacle detection, the A* algorithm for pathfinding, and sophisticated obstacle avoidance mechanisms represents a significant advancement in modern vehicular systems. By utilizing LiDAR, the system effectively identifies and responds to dynamic obstacles in real time, ensuring safety and adaptability in varying environments. The A* algorithm provides an efficient and reliable method for determining optimal paths, while advanced motion management guarantees smooth navigation along these paths.

This comprehensive approach not only optimizes parking efficiency but also enhances the overall user experience by ensuring seamless and collision-free vehicle movement. As autonomous systems continue to evolve, these techniques lay the groundwork for more intelligent, responsive, and user-friendly parking solutions. However, certain limitations were observed during the development and testing phases. **Scalability** became a concern, as the system struggled with high computational loads in large, complex parking lots. Additionally, while static obstacles were handled efficiently, **dynamic obstacle handling** posed challenges due to sensor latency, affecting real-time responsiveness. Furthermore, the use of a simplified vehicle model limited the accuracy of tight maneuver simulations, particularly in more constrained spaces, which highlights the need for more sophisticated vehicle dynamics modeling.

In conclusion, while there are areas for improvement, the integration of these advanced technologies marks a significant step towards more efficient, intelligent, and autonomous parking systems.

7 Future Work

Advancements in path planning will focus on integrating machine learning with algorithms like A* for adaptive and real-time decision-making in complex environments. Enhanced LiDAR and sensor fusion will combine high-resolution sensors and diverse data sources, improving obstacle detection, reliability, and performance under various conditions.

Scalability and multi-vehicle coordination will involve developing systems for managing multiple vehicles through distributed algorithms and V2V/V2I communication, ensuring efficient parking and congestion management. Finally, the integration of YOLO with camera footage will enhance object detection, providing richer environmental awareness and precision through deep learning and real-time processing.

In addition to these advancements, several other improvements are recommended:

- Integration of advanced sensor fusion techniques (e.g., LiDAR and camera) for improved dynamic obstacle detection.
- Adoption of more realistic vehicle models to simulate complex maneuvers.
- Parallelized path planning algorithms to improve scalability in multi-agent scenarios.

These innovations will drive smarter, safer, and more scalable autonomous systems.

References

[1] W. M. Simon Chadwick and P. Newman, "Distant vehicle detection using radar and vision,"

[2] N. Reddy, M. Vo, and S. G. Narasimhan, "Carfusion: Combining point tracking and part detection for dynamic 3d reconstruction of vehicles," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1906–1915, 2018.

[3] X. Du and K. K. Tan, "Autonomous reverse parking system based on robust path generation and improved sliding mode control," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 3, pp. 1225–1237, 2015.

[4] A. Garus, A. Rodríguez, A. Mourtzouchou, L. dell'Olio, B. Alonso Oreña, and B. Ciuffo, "Automated vehicles and the urban parking paradigm: Environmental implications and citizen preference," *Transportation Research Part D: Transport and Environment*, vol. 136, p. 104437, 2024.

[5] F. Rosique, P. J. Navarro, C. Fernández, and A. Padilla, "A systematic review of perception system and simulators for autonomous vehicles research," *Sensors*, vol. 19, no. 3, 2019.

[6] P. Kaur, S. Taghavi, Z. Tian, and W. Shi, "A survey on simulators for testing self-driving cars," in *2021 Fourth International Conference on Connected and Autonomous Driving (MetroCAD)*, pp. 62–70, 2021.

[7] K. C. B. J. A. G. Li Chen, Penghao Wu and H. Li, "End-to-end autonomous driving: Challenges and frontiers," *arXiv preprint arXiv:2306.16927v2*, 2023. `https://arxiv.org/abs/2306.16927v2`.

[8] T. X. T. Li, B.; Zhang, "Vehicle detection from 3d lidar using fully convolutional network," *arXiv 2016*. arXiv:1608.07916.