Planning and Control of an Autonomous Driving Vehicle in CARLA simulator

Sanket Khullar, Jasdeep Bajaj
MS, Mechancial Engineering, Texas A&M University
sanketkhullar@tamu.edu, jasdeepbajaj3@tamu.edu

Abstract

This project delves into the development of a sophisticated vehicle control system implemented in the Carla simulation environment. The system integrates both longitudinal and lateral control mechanisms to navigate a vehicle accurately through predefined waypoints in a simulated urban landscape. The primary objective is to create a comprehensive control system that ensures precise vehicle control by managing speed, acceleration, and steering. This system incorporates PID controllers for longitudinal and lateral control, facilitating smooth navigation through turns and straight paths. Key components include a global route planning mechanism that efficiently calculates the optimal path between waypoints and algorithms for precise control. Operating within the Carla simulation platform, this project aims to validate and demonstrate the efficiency of the control system in a realistic urban setting. Through this endeavor, the project seeks to contribute to the advancement of autonomous vehicle technology by showcasing a robust control system capable of navigating urban environments effectively. The successful implementation of this system signifies a significant stride toward enhancing autonomous vehicle capabilities in real-world urban transportation scenarios.

1 INTRODUCTION

1.1 Project Overview

The project involves developing a vehicle control system within the Carla simulation environment. It incorporates longitudinal and lateral control mechanisms along with route planning to navigate a vehicle through predefined waypoints. The system aims to demonstrate efficient and accurate vehicle control in a simulated urban environment.

1.2 Purpose

The primary purpose is to create a comprehensive control system that orchestrates both longitudinal (speed and acceleration) and lateral (steering) control of a vehicle. This system should exhibit smooth navigation and adherence to a defined path while considering the dynamic environment. This system aims to showcase efficient route planning and precise vehicle control.

The goals of the project include:

- **Route Planning:** Develop and implement a reliable global route planner using the A* algorithm to compute optimal paths based on randomly selected start and goal location in Carla Environment.
- **Precise Vehicle Control:** Integrate sophisticated planning algorithms and control mechanisms seamlessly into Carla's simulation platform for cohesive execution and validation.

- **Waypoint Following using PID and Pure Pursuit Controller:** Implement a PID-based longitudinal controller for speed regulation and a Pure Pursuit algorithm for precise lateral control (steering). Enabl the vehicle to accurately follow generated waypoints while maintaining appropriate speed and smooth steering adjustments.
- **Performance Evaluation of Planner and Controller in Simulated Environment:** Develop and apply a comprehensive evaluation metrics to assess the performance of the planner and controllers within Carla's simulated environment. Conducted thorough testing to evaluate path accuracy, control precision, and overall system reliability. Iterat and refin the system based on performance evaluations to enhance efficiency and accuracy within a controlled setting.

1.3 Scope

- **Global Route Planning:** Utilizing map data to determine feasible paths from a starting point to a randomly chosen destination.
- **Vehicle Control:** Implementing PID-based longitudinal control for speed maintenance and a Pure Pursuit algorithm for lateral control (steering).
- **Waypoint Following using PID and Pure Pursuit Controller:** Implement a PID-based longitudinal controller for speed regulation and a Pure Pursuit algorithm for precise lateral control (steering). Enabl the vehicle to accurately follow generated waypoints while maintaining appropriate speed and smooth steering adjustments.
- **Simulation Environment:** Utilizing Carla as a simulated environment to test and validate the developed navigation and control systems.

2 Literature Survey

In the domain of autonomous driving in the Carla simulation environment, existing research predominantly centers on reinforcement learning techniques, particularly Deep Q-Learning (DQN). This approach is tailored for real-time performance in varied scenarios and is scalable for extensive action and observation spaces. It emphasizes decentralized control and cooperative behavior among multiple agents, catering to the complexities of dynamic driving environments. Contrasting this, our project adopts different methodologies, including PID and Pure Pursuit Controllers for vehicle control, alongside the A* algorithm for path planning. This distinct combination of algorithms and the specific application within the Carla environment set our work apart from current studies, offering a unique perspective in the realm of autonomous driving simulations.

3 Code Overview

The code comprises several modules organized to create an autonomous vehicle control system within the CARLA simulator. It consists of modules for global route planning, longitudinal and lateral control, alongside utility functions for control signal generation and distance calculations.

3.1 Purpose of Each Import and Library:

- **CARLA Python API:** Importing the CARLA Python API provides access to CARLA functionalities, enabling interaction with the simulator environment.
- **Network:** Used for graph representation and manipulation, essential for constructing the road network and performing route planning.
- **PID Controller Module:** Utilizes the Pure Pursuit algorithm for lateral control, calculating steering angles.
- **Math Libraries:** Utilized for mathematical calculations related to control signal generation, distance calculations, and waypoint transformations.

3.2 Description of the Main Function and Its Components:

- **Global Route Planner:** The main function initiates the process by constructing a road network topology using CARLA waypoints and converting it into a graph representation. It employs the A* search algorithm to trace a route from the starting point to the destination.
- **Longitudinal Controller:** Responsible for controlling the vehicle's speed, the main function incorporates a PID controller that computes throttle and brake signals to achieve and maintain the desired speed.
- **Lateral Controller (Pure Pursuit):** This component computes steering angles based on the Pure Pursuit algorithm. It selects target waypoints and calculates steering angles to guide the vehicle along the planned route.
- **Utility Functions:** These functions support the main components by generating control signals, calculating distances between waypoints, and facilitating mathematical operations essential for vehicle control.

4 Project Components

4.1 Global Route Planner Module

Purpose: This module is designed to create a topological representation of the road network using the CARLA waypoints. It constructs a graph-based representation allowing for efficient route planning.

Functions:

- *Build Road Network:* Converts CARLA waypoints into a graph structure, creating nodes and edges that represent the road layout.
- *A* Search Algorithm:* Implements the A* search algorithm to find the shortest path between the starting and ending waypoints.

4.2 Longitudinal Controller Module

Purpose: Manages the vehicle's longitudinal motion, primarily focusing on speed control through a PID controller.

Functions:

- *PID Controller:* Computes throttle and brake signals to maintain a set speed, ensuring smooth acceleration and deceleration.
- *Speed Regulation:* Adjusts the vehicle's speed based on the desired velocity and current speed feedback.

4.3 Lateral Controller Module

Purpose: Controls the vehicle's lateral motion by determining steering angles necessary to follow the planned route.

Functions:

- *Pure Pursuit Algorithm:* Implements the Pure Pursuit algorithm to calculate steering angles based on the target waypoints.
- *Target Waypoint Selection:* Identifies the appropriate waypoints and calculates the steering angles required to navigate the vehicle along the desired path.

4.4 Utility Functions Module

Purpose: Provides support functions and calculations required by other modules for seamless operation.

Functions:

- *Control Signal Generation:* Generates control signals such as throttle, brake, and steering angles required for vehicle control.
- *Distance Calculations:* Computes distances between waypoints and assists in determining the vehicle's position within the environment.
- *Mathematical Operations:* Performs necessary mathematical operations, aiding in waypoint transformations and other calculations needed for control and navigation.

Each module encapsulates specific functionalities necessary for autonomous vehicle control. These components work cohesively, enabling the vehicle to navigate within the CARLA simulator environment effectively.

5 Functionality and Implementation

5.1 Global Route Planner (global_route_planner.py)

Purpose: This module is responsible for providing a high-level route plan by building a topology and a graph representing the road map.

Implementation:

- *GlobalRoutePlanner class* initializes with a map and a sampling resolution, building a topology and graph in its constructor.
- *trace_route method* generates a route trace from an origin to a destination using the path search and turn decision functions.
- *_build_topology* extracts the road segments, waypoints, and paths from the map.
- *_build_graph* constructs a networkx graph representing the world map based on topology.
- *_find_loose_ends* identifies and adds unconnected road segments to the graph representation.
- *_lane_change_link* adds zero-cost links in the graph to represent availability for lane changes.
- *_path_search* utilizes A* search with a distance heuristic to find the shortest path between two waypoints.
- *_turn_decision* determines the appropriate turn decision (straight, left, right) based on edges and waypoints.

5.2 Lateral Controller (lateral_controller.py)

Purpose: Implements the Pure Pursuit Controller for lateral control.

Implementation:

- *PurePursuitController class* initializes with the vehicle's wheelbase (L) and a gain for calculating the lookahead distance (Kdd).
- *calc_steering_angle* calculates the steering angle based on alpha (angular difference) and lookahead distance.
- *get_target_wp_index* finds the index of the target waypoint in a list based on vehicle location and waypoints.
- *get_lookahead_dist* calculates the lookahead distance based on vehicle speed and target waypoint index.

5.3 Longitudinal Controller (longitudinal_controller.py)

Purpose: Implements the PID Longitudinal Controller for vehicle speed control.

Implementation:

- *get_speed* retrieves the speed of a vehicle.
- *PIDLongitudinalController class* initializes with PID gains and buffers for error handling.

- *run_step* executes a step of PID control based on target speed and current speed.
- *_pid_control* calculates the PID control action based on proportional, integral, and derivative error terms.

5.4 Utils (utils.py)

Purpose: Contains utility functions used across modules.

Implementation:

- *find_dist_veh* calculates the distance between a vehicle location and a target.
- *get_speed* retrieves the speed of a vehicle.
- *vector* computes the unit vector from one location to another.
- *control_signal* generates control signals for a vehicle using lateral and longitudinal controllers.

5.5 Main Function (main.py)

Purpose: Orchestrates the execution of the route planning and vehicle control based on the generated route.

Implementation:

- Loads the map, generates a route between random spawn points, and spawns a vehicle.
- Initializes longitudinal and lateral controllers.
- Iterates through waypoints, calculating control signals, and steering the vehicle until the final waypoint is reached.

6 Parameters and Configurations

6.1 Global Route Planner Parameters

Waypoint Resolution:

Explanation: Determines the distance between consecutive waypoints.

Effect of Change: Higher resolution leads to more waypoints and potentially more precise paths. However, it can increase computational load.

6.2 Pure Pursuit Controller Parameters

Lookahead Distance:

Explanation: Determines how far ahead the controller looks for the target waypoint.

Effect of Change: Increasing this distance can result in smoother, more gradual steering adjustments but may reduce responsiveness to sharp turns.

6.3 PID Longitudinal Controller Parameters

Proportional, Integral, and Derivative Gains (P, I, D):

Explanation: These gains affect how much weight is given to the current error, accumulated error, and rate of change of error in the control signal.

Effect of Change: Adjusting these gains alters the controller's response—increasing P can make it more reactive, while too much D might induce oscillations.

6.4 General Parameters

Vehicle Speed Limit:

Explanation: Defines the maximum speed the vehicle should achieve.

Effect of Change: Lower limits can slow down the vehicle's overall movement, while higher limits might cause safety issues or instability at curves or intersections.

6.5 Effects of Changing Parameters

Performance vs. Accuracy: Changing resolution, distance thresholds, or lookahead distances can affect the trade-off between computational performance and navigation accuracy.

Safety and Stability: Adjusting speed limits, or control gains, can impact the vehicle's stability, and safety margins.

Responsiveness: Parameters like lookahead distance and PID gains can influence how quickly the vehicle responds to changes in the environment or waypoints.

Resource Utilization: Changing parameters affecting the number of waypoints or simulation step size can impact computational resources required for planning and control.

7 CARLA Simulation

Carla is an open-source simulator designed for autonomous driving research. Its primary role in projects involving autonomous vehicles is to provide a realistic and configurable environment for testing algorithms, training models, and evaluating the performance of various autonomous driving systems.

7.1 Description and Role

Carla offers a 3D simulation environment that replicates real-world scenarios, including urban, suburban, and highway settings. It simulates sensor data, such as cameras, lidar, radar, GPS, and depth sensors, enabling developers to test and validate their algorithms in a virtual environment before deploying them in the real world. This simulation platform is pivotal in the development and validation of autonomous driving algorithms.

7.2 Carla Environment Setup and Usage

7.2.1 Environment Setup

Configuring Carla involves setting up the simulation environment, selecting maps, defining vehicle models, configuring sensor suites, and defining traffic and weather conditions. Developers can use Carla's Python API to interact with the simulation environment programmatically.

7.2.2 Usage within the Code

The code interacts with Carla through its API to spawn vehicles, pedestrians, and other objects, set weather conditions, define traffic rules, and control the ego vehicle. It uses Carla's functions and classes to retrieve sensor data, control vehicle movement, and simulate real-world scenarios.

7.3 Significance of Spawning Points, Maps, and Vehicles

7.3.1 Spawning Points

- Determine initial positions of vehicles, pedestrians, and other objects within the map.
- Essential for setting up scenarios and defining starting conditions for testing different algorithms.

7.3.2 Maps

- Carla provides various maps representing diverse urban, suburban, and highway environments.
- Choosing a map affects the scenarios and road layouts available for testing algorithms.

7.3.3 Vehicles

- Carla allows spawning different types of vehicles with customizable attributes such as speed, behavior, and sensor configurations.
- Vital for simulating traffic scenarios, interactions between vehicles, and testing autonomous driving algorithms in various environments.

7.4 Significance within the Simulation

- **Realistic Scenario Testing:** Carla's realistic environment enables testing algorithms in scenarios like traffic congestion, diverse weather conditions, and complex road layouts.
- **Algorithm Validation:** Developers can validate perception, planning, and control algorithms in a safe and controlled environment before real-world deployment.
- **Training and Evaluation:** Carla allows for training machine learning models using simulated data and evaluating their performance under different conditions, enhancing the development of robust autonomous systems.

8 Algorithms Used

8.1 A* Algorithm (Planner)

The A* algorithm is a pathfinding algorithm widely used in robotics and game development for finding the shortest path between nodes in a graph. It operates by exploring nodes in a way that minimizes the total cost of the path.

Mathematical Explanation:

- *Heuristic Function:* A* uses a heuristic function $f(n) = g(n) + h(n)$ to estimate the total cost from the start node to the goal node through the current node n. $g(n)$ is the cost of the path from the start node to node n and $h(n)$ is the heuristic (estimated) cost from node n to the goal node.
- *Open and Closed Sets:* A* maintains two lists: the open set (nodes to be evaluated) and the closed set (nodes already evaluated).
- *Algorithm Steps:* Start with the initial node and add it to the open set. Continue by choosing the node with the lowest f value, exploring its neighbors, updating their g and f values, and moving the current node to the closed set. Terminate when the goal node is reached or when no path is available.

8.2 Implementation of the A* Algorithm in Route Planning

Initialization: The class initializes by creating a graph representation of the road network, with nodes representing waypoints and edges representing connections between waypoints. It builds this graph by parsing the map and extracting road segments, junctions, lanes, and connections between them.

Trace Route: The trace_route method is the core function used to find the shortest path. It uses the A* algorithm (_path_search method) within the constructed graph to find the optimal path from the origin to the destination. It iterates through waypoints, considering different road options (e.g., lane change, intersections) while ensuring the path is continuous and navigable.

Topology & Graph Construction: The road network is broken down into segments with defined entry and exit waypoints. These segments are represented as nodes and edges in the graph, considering factors like intersection presence, lane changes, and road options (e.g., left/right turns, straight paths).

Lane Change Links: The _lane_change_link method adds zero-cost links in the graph to indicate the availability of lane changes. It identifies adjacent lanes and adds edges representing lane changes, ensuring the shortest path considers these possibilities.

Heuristics & Decision Making: The class employs heuristics to calculate distances between nodes (_distance_heuristic) and makes decisions at each waypoint to determine the optimal road option (e.g., going straight, turning left/right, lane following). These decisions are based on factors like angle deviations, intersection presence, and the road's topology.

Path Tracing & Optimization: The path traced by the trace_route method ensures continuity by considering waypoints, edges, and road options iteratively. It optimizes the path by evaluating the road network's structure and conditions at each step, allowing for smooth navigation from origin to destination.

8.3 PID Longitudinal Controller

The Proportional-Integral-Derivative (PID) controller is a widely used feedback control mechanism. The longitudinal (speed) controller maintains a desired velocity by adjusting the throttle or braking of the vehicle.

Mathematical Explanation:

- *PID Equation:* The control signal $u(t)$ at time t is calculated using the equation

$$u(t) = K_p \cdot Error + K_i \cdot \sum Error + K_d \cdot \frac{d(Error)}{dt}$$

  where K_p, K_i, and K_d are the proportional, integral, and derivative gains, respectively.
- *Controller Actions:* Includes proportional action responding to the current error, integral action accumulating past errors, and derivative action predicting future errors based on their rate of change.

8.4 Pure Pursuit Lateral Controller

The Pure Pursuit algorithm guides a vehicle to follow a desired path by computing the steering angle needed to reach a lookahead point on that path.

Mathematical Explanation:

- *Algorithm Steps:* Includes finding the lookahead point on the planned path, determining the steering angle based on the vehicle's position relative to this point, and adjusting the steering angle to direct the vehicle towards the lookahead point.
- *Steering Angle Calculation:* The steering angle δ is calculated using trigonometry, considering the vehicle's wheelbase L, the distance l_d between the vehicle's rear axle and the lookahead point, and the angle α between the vehicle's heading direction and the line to the lookahead point. The equation for the steering angle is:

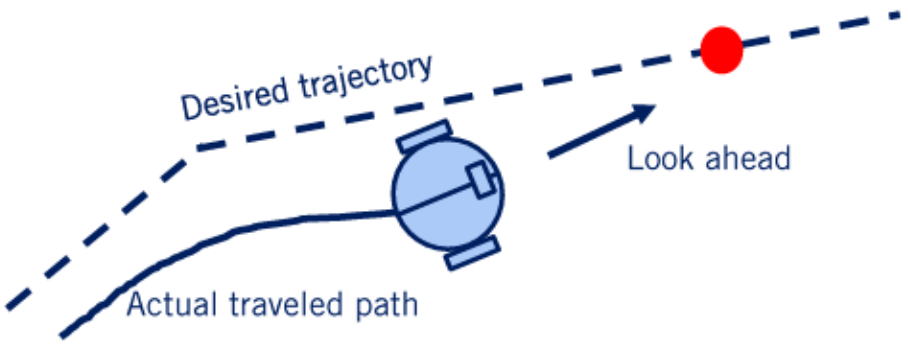
$$\delta = \tan^{-1}\left(\frac{2L\sin(\alpha)}{l_d}\right)$$

Operation and Interaction in the Code:

- *A* Planner:* Generates the optimal path based on the map and the goal, providing waypoints for the vehicle to follow.
- *Longitudinal and Pure Pursuit Controllers:* These controllers receive waypoints from the planner and adjust vehicle speed and steering angle, respectively, to follow the desired path.

Control Signals' Significance:

- *Speed Control Signal:* Regulates the vehicle's velocity to adhere to the planned path and avoid overshooting or lagging.
- *Steering Control Signal:* Directs the vehicle along the planned path by adjusting the steering angle based on the lookahead point, ensuring smooth and accurate trajectory tracking.



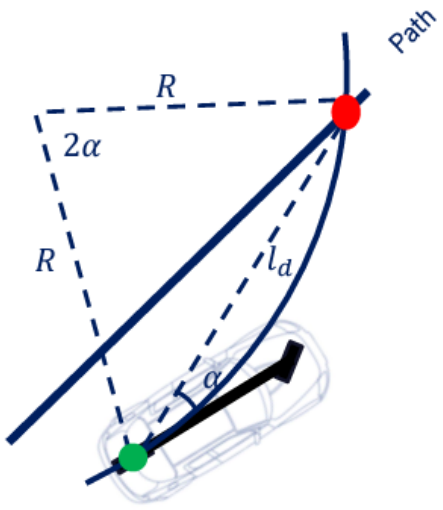
From the *law of sines*:

$$\frac{l_d}{\sin 2\alpha} = \frac{R}{\sin\left(\frac{\pi}{2} - \alpha\right)}$$

$$\frac{l_d}{2\sin\alpha\cos\alpha} = \frac{R}{\cos(\alpha)}$$

$$\frac{l_d}{\sin\alpha} = 2R$$

$$\kappa = \frac{1}{R} = \frac{2\sin\alpha}{l_d}$$ Path curvature



8.5 Speed control during turns

The code uses the curvature radius (R) to determine if the vehicle is on a straight road or a turn. It does this by calculating:

$$R = \frac{l_d}{2L\sin\alpha}$$

The absolute value of R is compared against a straight road threshold. If R is greater than the threshold, the vehicle is on a straight road. Otherwise, it's identified as being on a turn.

Behavior on a Straight Road: On a straight road, vehicles typically accelerate to maintain a consistent speed. The absence of curves or bends allows the vehicle to optimize its velocity, ensuring a smoother and more efficient movement along the path.

Behavior on a Turn: When navigating a turn, vehicles typically undergo a reduction in speed to ensure safe and stable maneuvering. This alteration in speed is essential to manage the change in direction without compromising safety or stability.

9 Execution and Output

9.1 Main Execution Flow

9.1.1 Initialization

- Load the map and environment (like Carla) and initialize the vehicle's starting position.

9.1.2 Path Planning

- Use the A* algorithm to generate a path from the vehicle's current position to the desired destination.
- Obtain a series of waypoints that form the planned path.

9.1.3 Control Loop

- Start a control loop that executes at regular intervals (e.g., every few milliseconds).

9.1.4 Vehicle Control

- Use the PID longitudinal controller to adjust the vehicle's speed based on the waypoints' locations.
- Use the Pure Pursuit controller to determine steering angles and keep the vehicle on the planned path.

9.2 Progression through Waypoints

The vehicle starts at a designated point and progresses through the waypoints provided by the planner. At each iteration of the control loop, the vehicle adjusts its speed and steering angle to move towards the next waypoint. The controllers ensure the vehicle adheres to the planned path by adjusting its behavior based on the current position, desired speed, and the lookahead point from the Pure Pursuit algorithm.

10 Results and Comparisons

For the A* algorithm, waypoint resolution and pathfinding heuristics parameters were tuned in the code. Adjusting waypoint resolution impacted the precision of path planning and computational load.

In the PID controller, proportional, integral, and derivative gains were the key tunable parameters. Altering these gains affected the vehicle's response to speed regulation, where high proportional gain increased responsiveness, but excessive derivative gain caused oscillations.

In the Pure Pursuit controller, the lookahead distance was a critical parameter. Tuning this distance affected steering adjustments and the vehicle's ability to respond to sharp turns as shown in the plots attached below.

10.1 Town 1 with non-ideal parameter values

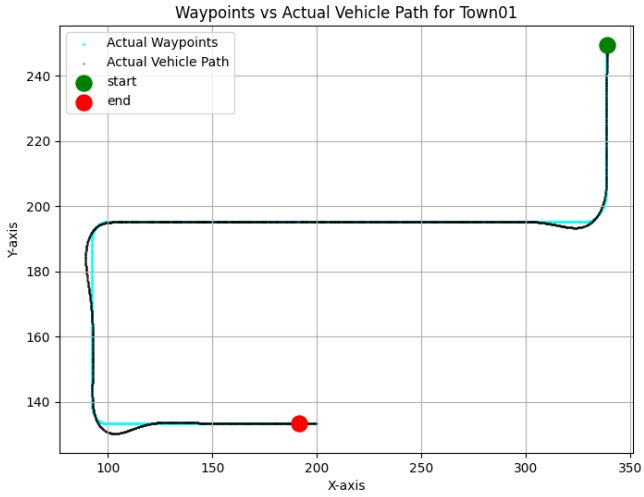


Figure 1: Plot showing the waypoints and the path followed by the vehicle

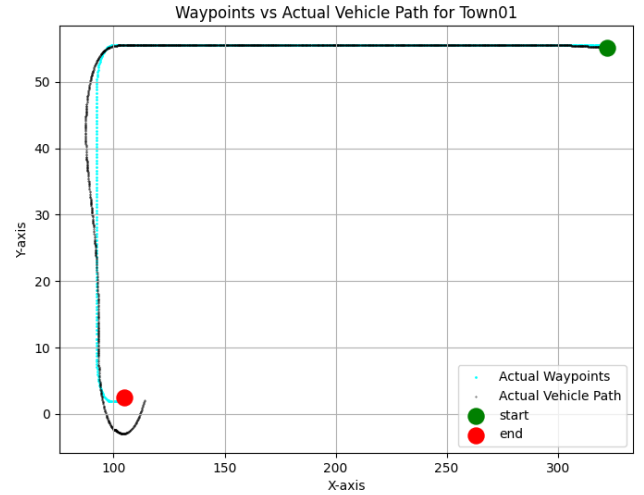


Figure 2: Plot showing the waypoints and the path followed by the vehicle

10.2 Town 2 with non-ideal parameter values

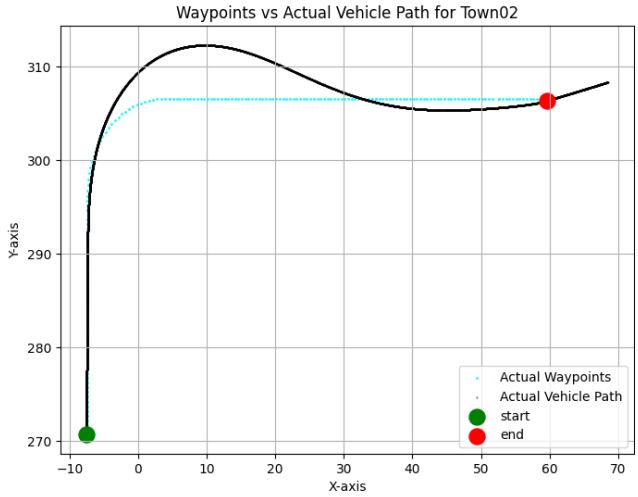


Figure 3: Plot showing the waypoints and the path followed by the vehicle

10.3 Town 3 with non-ideal parameter values

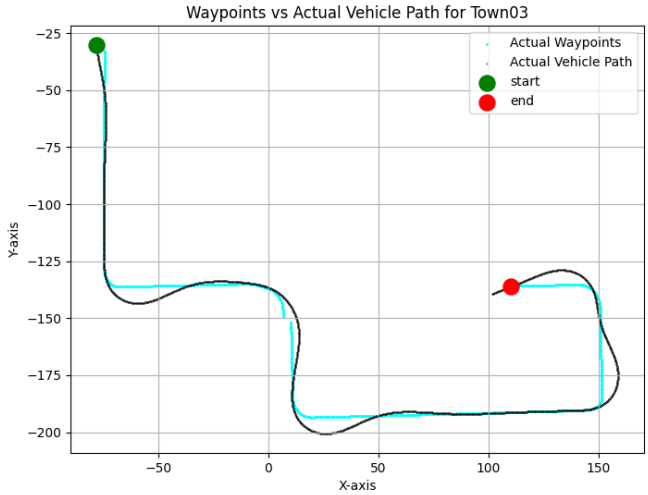


Figure 4: Plot showing the waypoints and the path followed by the vehicle

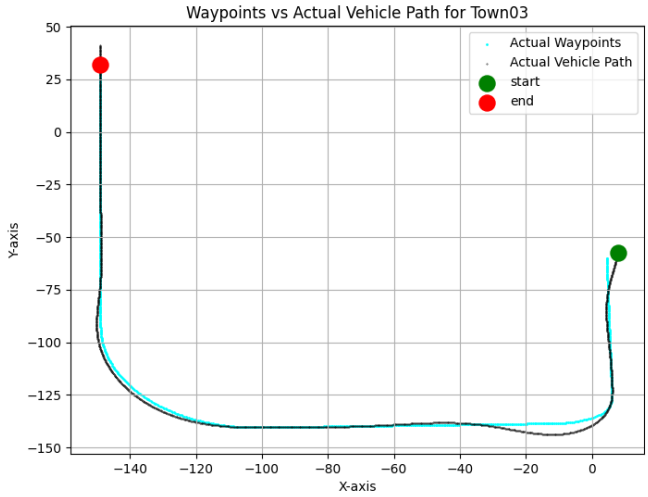


Figure 5: Plot showing the waypoints and the path followed by the vehicle

10.4 Town 4 with non-ideal parameter values

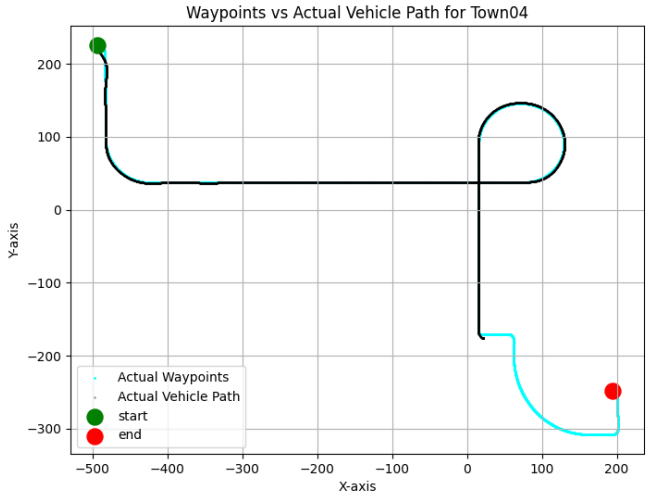


Figure 6: Plot showing the waypoints and the path followed by the vehicle

In the above case the vehicle failed to complete the designated path due to the current parameter settings hindering its ability to execute sharp turns.Adjustments to parameters such as the lookahead distance in the Pure Pursuit controller and the gains in the PID controller potentially rectifed this issue, enabling the vehicle to navigate sharp turns more effectively and follow the planned route accurately. Where the lookahead distance in our case is a function of the forward speed and the parameter K_{dd}

10.5 Town 1 with ideal parameter values

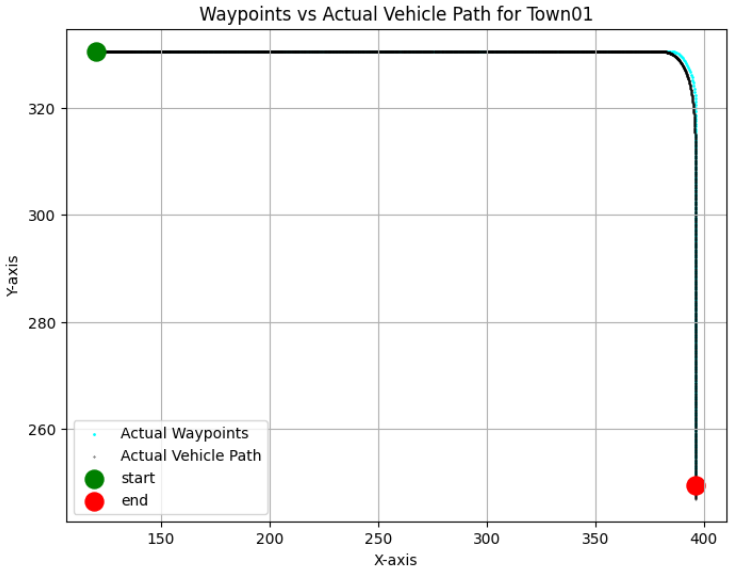


Figure 7: Plot showing the waypoints and the path followed by the vehicle

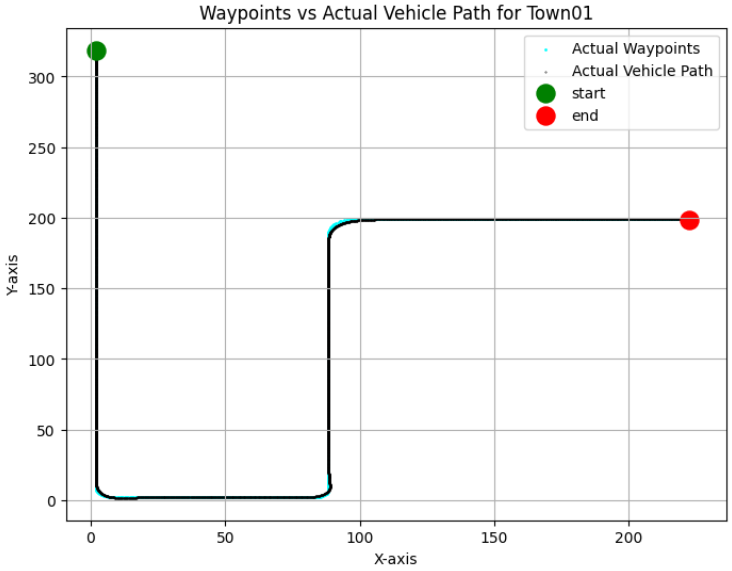


Figure 8: Plot showing the waypoints and the path followed by the vehicle

10.6 Town 2 with ideal parameter values

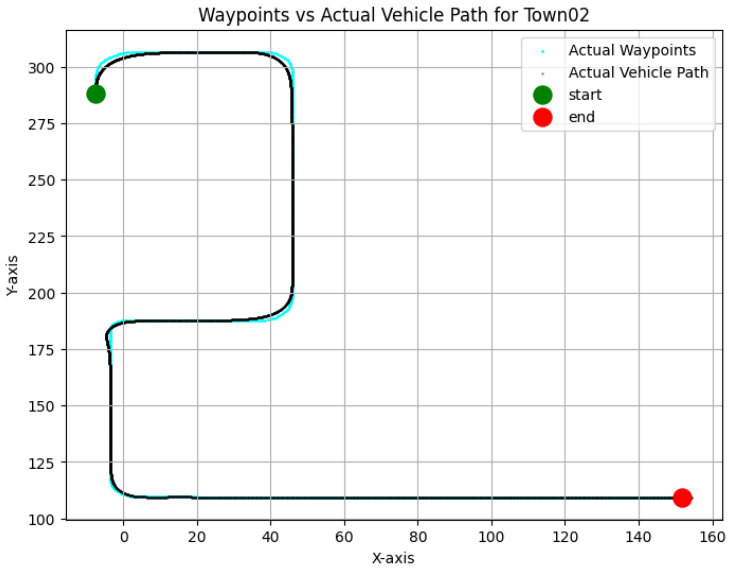


Figure 9: Plot showing the waypoints and the path followed by the vehicle

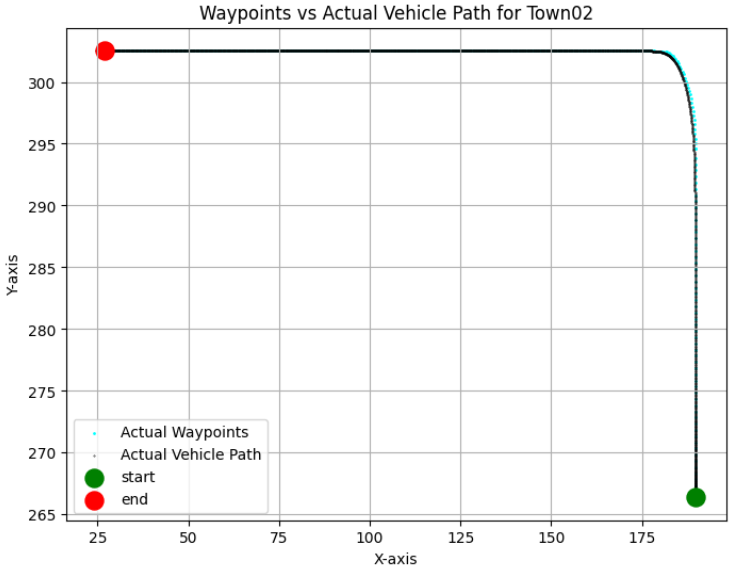


Figure 10: Plot showing the waypoints and the path followed by the vehicle

10.7 Town 3 with ideal parameter values

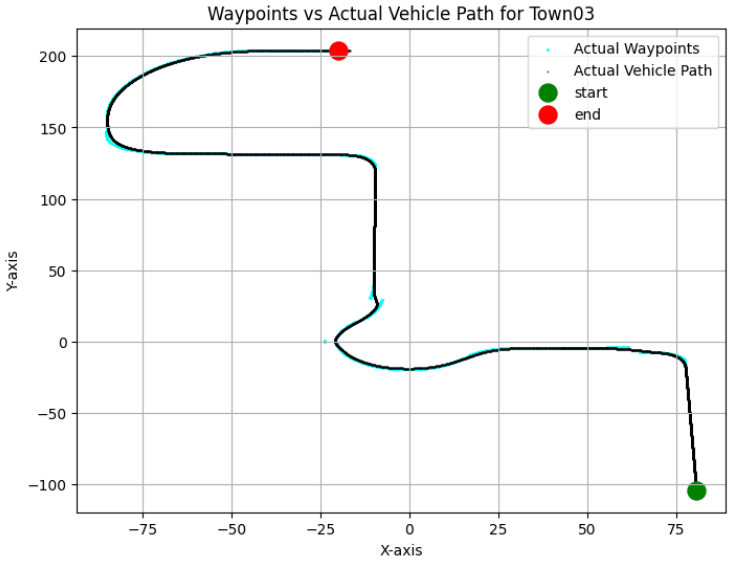


Figure 11: Plot showing the waypoints and the path followed by the vehicle

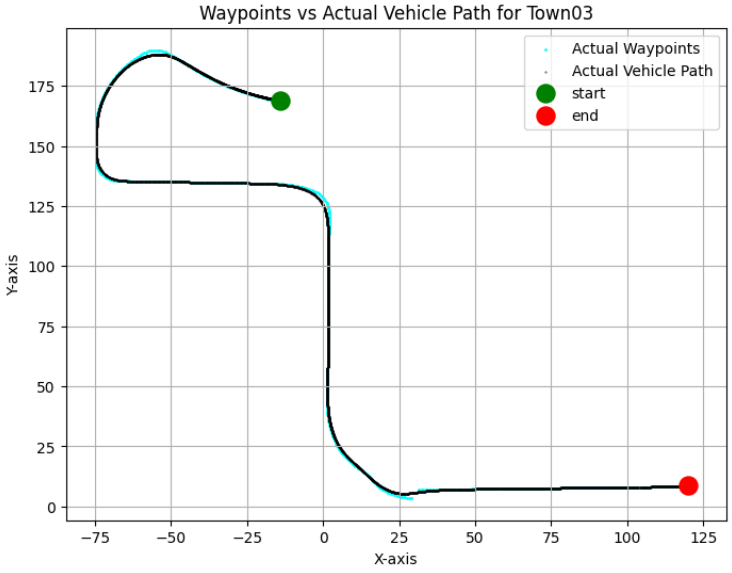


Figure 12: Plot showing the waypoints and the path followed by the vehicle

10.8 Town 4 with ideal parameter values

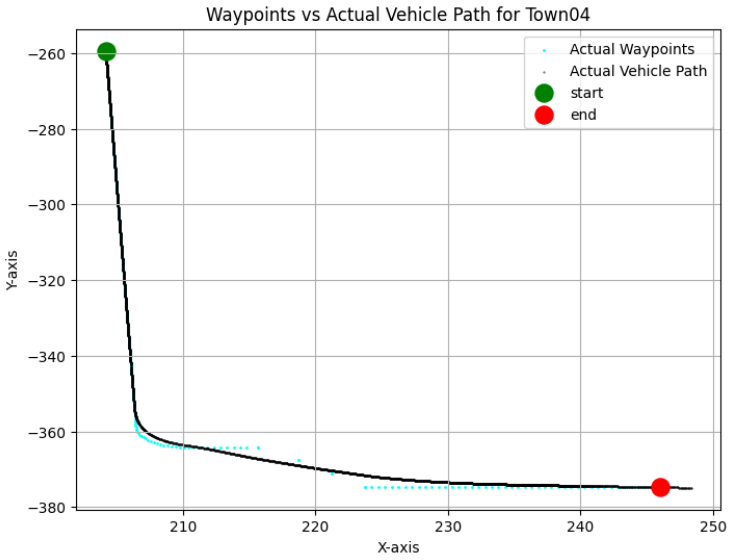


Figure 13: Plot showing the waypoints and the path followed by the vehicle

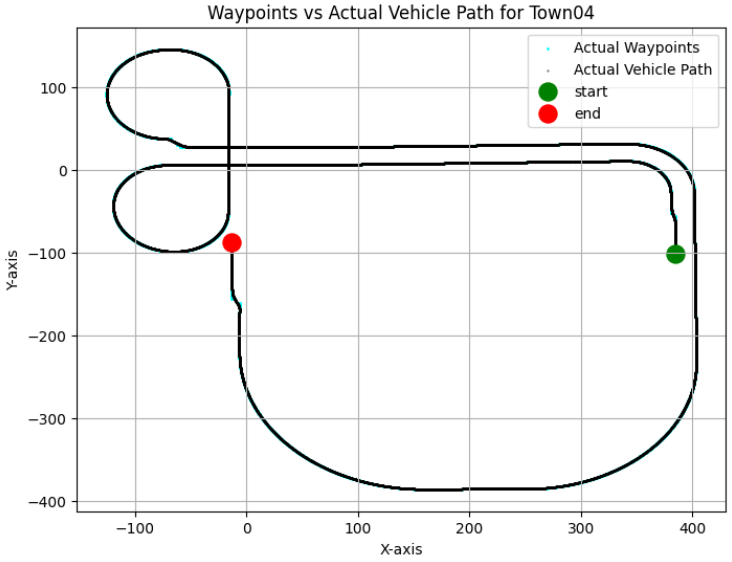


Figure 14: Plot showing the waypoints and the path followed by the vehicle

The above plots were generated using the optimal parameter values as follows:
The optimal parameter values utilized for generating the above plots are as follows:

- $K_p = 1$
- $K_d = 0.3$
- $K_i = 0.75$
- $K_{dd} = 1.4$
- Velocity V varies on turns and can reach up to 45 Km/h

Furthermore, the lookahead distance in our autonomous driving system dynamically adjusts based on the vehicle's forward velocity and the parameter $K_{dd} = 1.4$. As the vehicle approaches a turn, the decreasing velocity leads to a reduced lookahead distance, serving as an indicator for an imminent turn. This prompts the system to send appropriate steering signals, ensuring the car navigates turns at a slower, more controlled speed, thereby enhancing the precision and safety of navigation.

11 Conclusion

Waypoint Generation and Projection: The ability to generate waypoints at a specific resolution and overlay them onto the simulation environment demonstrates a crucial foundational step. The comparison between the ground truth path and the actual path followed by the vehicle offers valuable insights into the accuracy and precision of the generated waypoints.

A* Algorithm Path Generation: The successful implementation of the A* algorithm through the Global Route Planner signifies a robust planning mechanism. It's pivotal in ensuring the system's capability to plan feasible and optimal routes between arbitrary points. The accuracy of the generated paths showcases the effectiveness of this planning algorithm.

Controller Implementation for Path Following: The successful implementation of controllers to guide the vehicle along the ground truth path from start to end signifies a critical milestone. The controllers' ability to steer and control the vehicle's speed along the planned trajectory demonstrates the system's capability to execute planned routes accurately.

Overall System Performance: The combined success in waypoint generation, path planning through the A* algorithm, and accurate path following with controllers presents a promising foundation for the autonomous driving system. The seamless integration of these components indicates a cohesive system architecture, capable of translating planned routes into actionable vehicle behavior within the simulation environment.

In this project, we have meticulously implemented the PID controller and the Pure Pursuit controller from scratch, tailoring them specifically to our requirements in the Carla simulator. These custom implementations allowed for granular control and optimization, aligning precisely with the dynamics of our simulated environment. Additionally, for route planning, we utilized the A* algorithm, implemented via the Networkx library. The graph structure for the A* algorithm was generated using the detailed layouts of towns from the Carla simulator, enabling accurate and efficient pathfinding in complex urban environments

12 Working in the Carla Simulator

12.1 Visualization of Path Generation Using A* Algorithm

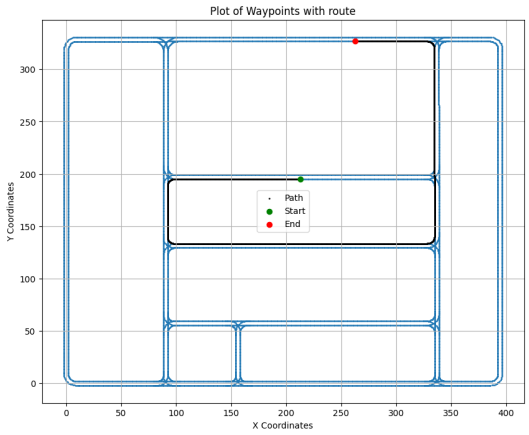


Figure 15: Illustration of the pathfinding process using the A* algorithm, highlighting extracted waypoints and the generated path.

12.2 Visualization of Vehicle in the Carla Simulator

Figure 16: Illustration of the Vehicle in the Carla Simulator at a straight line

Figure 17: Illustration of the Vehicle in the Carla Simulator at a turn

13 Future Work and Scope of Improvement

- **Enhanced Perception:** Improve sensor fusion techniques or use more advanced sensors to enhance perception capabilities and robustness.
- **Adaptive Algorithms:** Implement more adaptive or machine learning-based algorithms to handle dynamic environments and unpredictable scenarios.
- **Behavioral Planning:** Integrate more complex decision-making systems that account for higher-level behavioral planning, like handling complex intersections or traffic scenarios.
- **Simulation and Testing:** Conduct extensive simulations and real-world tests to refine algorithms, control strategies, and overall system performance under various conditions.

References

[1] Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). *A Formal Basis for the Heuristic Determination of Minimum Cost Paths*. IEEE Transactions on Systems Science and Cybernetics, 4(2), 100-107.

[2] LaValle, S. M. (2006). *Planning Algorithms*. Cambridge University Press.

[3] Åström, K. J., & Hägglund, T. (2006). *Advanced PID Control*. ISA-The Instrumentation, Systems, and Automation Society.

[4] O'Dwyer, A. (2009). *Handbook of PI and PID Controller Tuning Rules*. Imperial College Press.

[5] Coulter, R. C. (1992). *Implementation of the Pure Pursuit Path Tracking Algorithm*. Technical Report CMU-RI-TR-92-01, Carnegie Mellon University.

[6] Snider, J. M. (2009). *Automatic Steering Methods for Autonomous Automobile Path Tracking*. Robotics Institute, Carnegie Mellon University.

[7] Rajamani, R. (2011). *Vehicle Dynamics and Control*. Springer.

[8] Dosovitskiy, A., et al. (2017). *CARLA: An Open Urban Driving Simulator*. In Proceedings of the 1st Annual Conference on Robot Learning.

[9] Schwarz, J., & Behnke, S. (2016). *Simulating the Sensing and Processing Pipeline of Autonomous Vehicles*. In 2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC).

[10] Feng, D., Haase-Schütz, C., Rosenbaum, L., Hertlein, H., Glaeser, C., Timm, F., Wiesbeck, W., & Dietmayer, K. (2020). *Deep Multi-Modal Object Detection and Semantic Segmentation for Autonomous Driving: Datasets, Methods, and Challenges*. IEEE Transactions on Intelligent Transportation Systems.