CSC8499 - Project and Dissertation for MSc in Advanced Computer Science

Automatic Detection of Path-Following Errors in Autonomous Racing

Author: Sunil Siddharth Rajendran
Student ID: 230529876

Declaration

I, Sunil Siddharth Rajendran, declare that this research paper titled “Automatic Detection of Path-Following Errors in Autonomous Racing” is entirely my own work, except where otherwise acknowledged. I have cited all sources used in this paper and have not plagiarized any material. Furthermore, any contributions from others to this work have been duly acknowledged.

Place: Newcastle Upon Tyne

Date: 26.08.2024

Acknowledgement

I would like to thank my supervisor “Mr. Chris Holder”. for their consistent support and guidance during the running of this project. I would also like to acknowledge “Newcastle University” for their participation and engagement in the study.

Abstract

Aim: The objective of this research is to enhance the path-following accuracy of autonomous racing vehicles by integrating machine learning with the traditional Pure Pursuit algorithm. The focus is on improving the vehicle's ability to maintain its path, particularly in challenging scenarios such as sharp turns and varying speeds, by dynamically adjusting control parameters in real time.

Findings: The AI-enhanced Pure Pursuit algorithm demonstrated significant improvements in reducing heading errors and lateral deviations compared to the standard version. The integration of a decision tree model allowed for dynamic adjustments to the lookahead distance, resulting in more precise path-following and better handling of complex track segments. However, this enhancement comes with challenges, including the need for extensive data collection, potential computational overhead, and ensuring model robustness in real-world applications.

Methodology: A combination of simulation-based testing, data-driven model development, and iterative algorithm refinement was employed. The research involved logging real-time data from simulated racing scenarios, training a decision tree model to optimize control parameters, and comparing the performance of the AI-enhanced algorithm with the standard Pure Pursuit algorithm through quantitative analysis.

Conclusion: The integration of machine learning into the Pure Pursuit algorithm significantly enhances path-following performance in autonomous racing vehicles. However, further research is needed to address challenges related to data dependency, computational efficiency, and real-world testing. Future work should focus on refining the AI model, exploring alternative algorithms, and validating the system's performance in diverse driving conditions.

TABLE OF CONTENTS

Chapter 1: Introduction

1.1 Background

Autonomous racing is at the cutting edge of robotics and autonomous systems research, pushing the boundaries of speed, precision, and decision-making in dynamic environments, unlike typical autonomous driving scenarios, where safety and comfort are the primary concerns, autonomous racing demands high-speed manoeuvring, rapid decision-making, and precise control to navigate complex tracks efficiently. The ability of an autonomous race car to follow a predefined path with minimal errors is critical for achieving competitive performance while ensuring safety on the track [5].

Unlike conventional autonomous driving, where the focus is often on safety and comfort, racing requires vehicles to operate at the limits of their performance capabilities. This involves making split-second decisions, navigating complex track layouts, the vehicle must navigate through sharp turns, and maintaining high speeds all while staying as close as possible to the optimal racing line. Any deviation from the intended path, known as a path-following error, can result in slower lap times, increased tyre wear, and potential collisions with track barriers. Therefore, detecting and correcting these errors in real-time is essential for maintaining optimal performance and avoiding costly mistakes during a race [1].

Autonomous racing faces distinct challenges in path-following when compared to regular autonomous driving. Even slight deviations from the optimal racing line can have a big impact due to the fast speeds involved, which can result in noticeable drops in performance or accidents. In this situation, errors in following the path can appear as either lateral deviations, when the car moves away from the ideal line, or heading errors, when the car's direction veers off course. It is necessary to identify and fix these mistakes immediately to uphold the vehicle's competitive advantage.

The path-following algorithm is a key element in the control system of autonomous vehicles. This algorithm is in charge of making sure the vehicle remains on the intended route by making necessary adjustments to the steering angle and speed. Numerous path-tracking algorithms have been created throughout the years, with Pure Pursuit being among the most commonly utilized in both research and industry [3]. However, autonomous racing could be difficult for algorithms due to the fast-paced nature of racing, particularly when dealing with sharp turns, complex track layouts, and varying surface conditions. Therefore, new techniques are needed to enhance the effectiveness of these algorithms by adjusting control parameters in response to real-time conditions.

New developments in AI and machine learning bring hopeful solutions to the obstacles of trajectory tracking in self-driving racing. AI techniques, specifically machine learning, can improve traditional path-following algorithms by increasing their adaptability and responsiveness to real-time situations [4].

As an example, artificial intelligence can be utilized to modify the settings of the Pure Pursuit algorithm, like the distance it looks ahead, according to the vehicle's current condition and surroundings. Through the utilization of a machine learning model trained on past race data, the system can be taught to anticipate the best control settings for various parts of the track, considering variables like speed, curvature, and surface conditions. This enables the vehicle to stick more closely to the best racing route, lowering errors in following the path and enhancing overall performance.

This research focuses on the automatic detection of path-following errors in autonomous racing. By identifying and quantifying deviations from the desired path, the system can trigger corrective actions to minimise errors and enhance the overall performance of the race car. The integration of traditional control algorithms, such as Pure Pursuit [3], with Artificial Intelligence (AI) techniques, provides a promising approach to achieving this goal.

1.2 Aims & Objectives

The aim of the project is to develop and evaluate a predictive generative model for the real-time detection of path-following errors in autonomous racing cars. By using current sensor data and anticipated future positions, the project seeks to predict future sensor readings and identify deviations from the planned path. The main goal is to enhance the reliability and safety of the autonomous racing vehicle, especially under high-speed and lively driving conditions.

1) Develop a baseline path-following system using the Pure Pursuit algorithm, ensuring that the autonomous race car can follow a predefined racing line with reasonable accuracy.

2) Gather data on the race car's performance, including lateral deviations, heading errors, speed, and curvature, while it follows the racing line using the Pure Pursuit algorithm.

3) Train a decision tree model using the collected data to predict optimal control parameters, such as lookahead distance and heading error, and to detect potential path-following errors before they occur.

4) Combine the AI model with the Pure Pursuit algorithm to enable dynamic adjustment of control parameters and real-time correction of path-following errors.

5) Visualize the integrated system in simulated racing environments to assess its ability to detect and correct path-following errors, and to improve overall racing performance.

1.3 Research Problems

Detecting and correcting path-following errors in the fast-paced, dynamic setting of autonomous racing could be a major obstacle during this research. Expanding the path-following error detection system to accommodate more challenging situations like multi-agent racing or tracks with dynamic conditions may bring about challenges in maintaining accuracy and performance. Incorporating AI for dynamic control parameter adjustments and error prediction increases complexity, necessitating strong data collection, model training, and real-time processing to maintain swift adaptation without sacrificing the vehicle's competitive advantage. These problems demonstrate the importance of a thorough strategy for testing, validation, and ongoing improvement of the system to address the difficulties of autonomous racing.

Chapter 2: Literature Review

2.1 Autonomous Racing: Challenges and Innovations

Autonomous racing has emerged as a specialised domain within the broader field of autonomous vehicles, presenting unique challenges that require advanced control algorithms, high-speed decision-making, and robust error detection mechanisms. Autonomous racing differs significantly from conventional autonomous driving, primarily due to the high speeds and the competitive nature of racing. These factors necessitate rapid and precise control responses, making error detection and correction crucial to maintaining optimal performance on the race track.

Betz and Strumberger (2020) highlight the potential of autonomous racing as a testbed for developing and validating advanced autonomous vehicle technologies. They emphasise the importance of path-following accuracy, as deviations from the optimal racing line can lead to slower lap times and increased risk of collisions. The authors also note that traditional control methods, while effective in urban driving scenarios, may not be sufficient for the demands of autonomous racing, thus calling for the integration of more sophisticated approaches such as AI and machine learning [1].

2.2 Path-Following Algorithms in Autonomous Vehicles

The goal of path-following is a major task in the field of autonomous vehicle control, and there exist numerous algorithms for solving it. One of the most commonly employed ones is the Pure Pursuit, which determines what steering angle to have in order to trace a reference path based on our current position and orientation. In [3] Coulter (1992) introduced the Pure Pursuit algorithm, stating its robustness in keeping track accurate for different automotive contexts. Any environments with high speeds, sharp turns, or complex path geometries will cause the performance of this algorithm to degrade as a fixed set of values, such as those in Pure Pursuit, are not ideal for these cases, taking into account that adjustments may be needed dynamically.

Another popular path-following algorithm is the Stanley Controller, used in relation to the autonomous vehicle "Stanley" during the DARPA Grand Challenge (2006). The Stanley Controller is designed to minimize cross-track error, i.e., the lateral distance between the vehicle and the desired path [6]. The Stanley Controller, like Pure Pursuit, deviates from a perfect ideal path to produce more emergent behaviour in steering control for autonomous robots. This can be very effective for many dynamic applications, but it has been shown here that both the Stanley Controller and Pure Pursuit architecture have limitations when applied to high-speed racing-type situations where following paths with errors due to their hysteresis components will lead the vehicle off-track.

In the context of autonomous racing, these traditional algorithms must be enhanced to cope with the unique demands of the racing environment. This has led researchers to explore the integration of machine learning techniques with traditional control algorithms to create more adaptive and responsive systems.

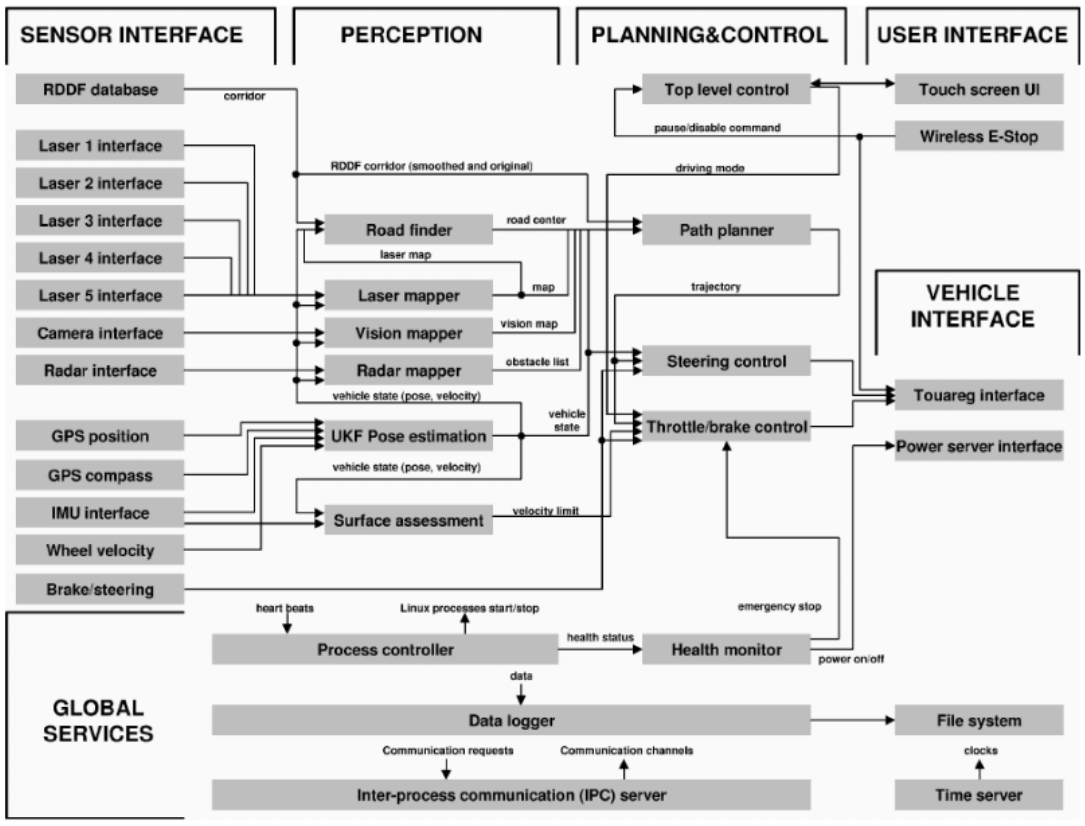


Figure 1: Flowchart of Stanley Software System [6]

2.3 The Role of AI and Machine Learning in Autonomous Driving

Adding intelligence in basic robotic systems, AI and machine learning have become part of the fabric to push the boundaries for advanced autonomous systems. Bojarski et al. (2016) used a breakthrough deep learning-based technique to drive autonomously, where they end-to-end trained CNNs for purely mapping raw input images directly from the front-mounted camera of a car to steering commands, as shown in Figure 2 [4]. This showed that AI was capable of learning sophisticated driving behaviours from visual data without needing explicit hand-crafted features or predefined

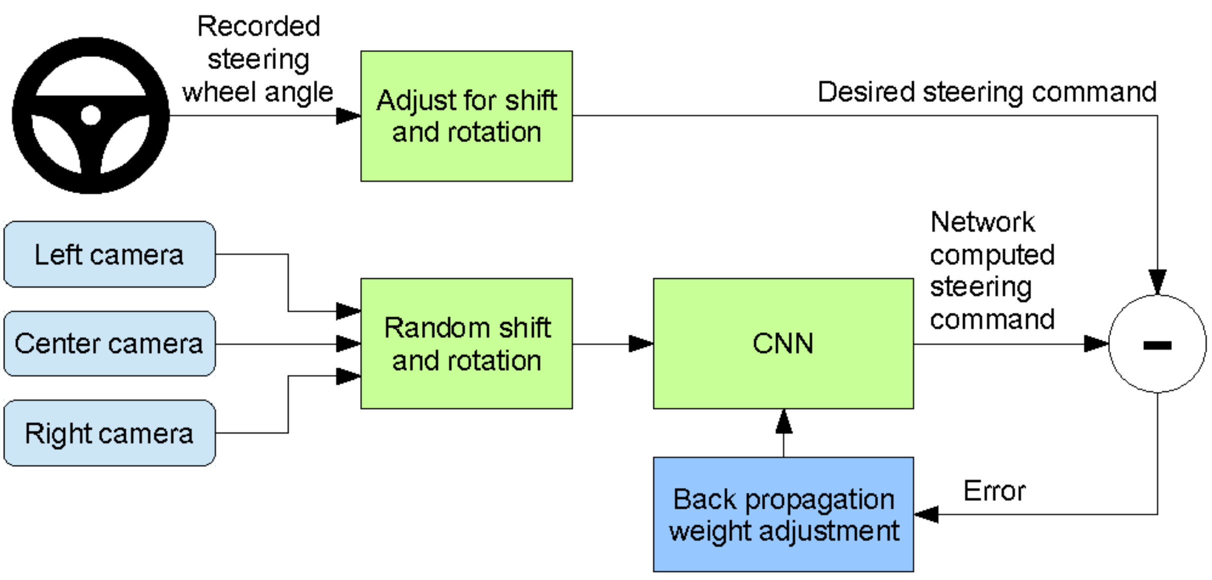

Recorded steering wheel angle
Adjust for shift and rotation
Desired steering command
Left camera
Center camera
Right camera
Random shift and rotation
CNN
Network computed steering command
–
Back propagation weight adjustment
Error

Figure 2: Training CNN [4].

In path-following, AI can dynamically adjust control parameters like the lookahead distance in Pure Pursuit using real-time data from vehicle sensors. Chen et al. (2015) introduced "DeepDriving," where a deep learning model predicts affordances—key features like the distance to the next turn or road curvature—directly from camera images [7]. This enables the vehicle to better anticipate and respond to environmental changes, reducing path-following errors.

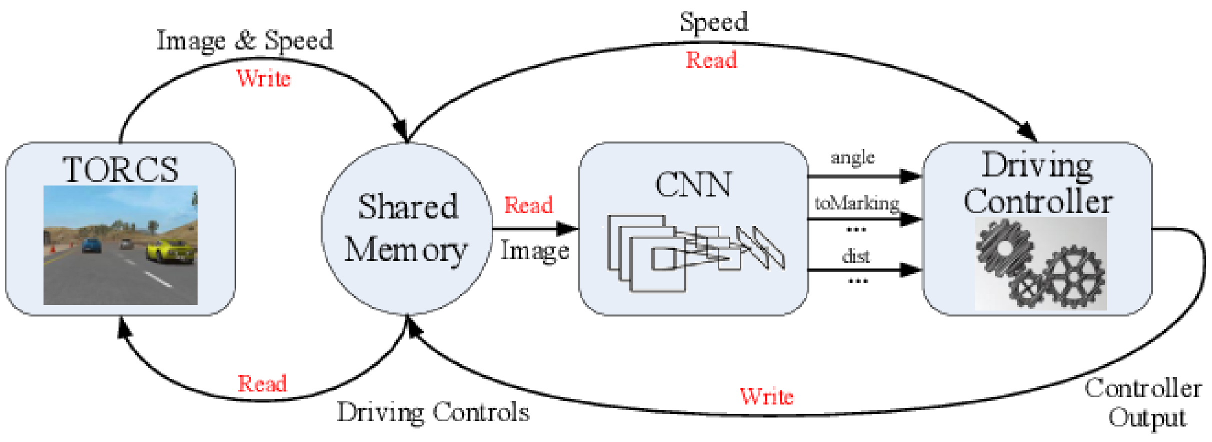


Figure 3: System Architecture of Chen et al (2015) proposal.

Integrating AI with traditional path-following algorithms offers benefits such as adapting to different driving conditions and enhancing overall path accuracy. However, these systems also bring challenges, particularly regarding real-time processing demands and the need for large, high-quality training datasets.

2.4 Error Detection and Correction in Autonomous Vehicles

High-risk environments like racing need error detection in autonomous vehicle systems. Paden et al. (2016) provide a more generic literature review of motion planning and control strategies designed within 3D automotive environments, underscoring the necessity of real-time error detection and correction that supervises vehicle stability, thereby ensuring vehicular safety [8]. They discuss how errors can be detected and corrected, such as steering off-path slightly to account for noise build-up or reducing speed.

In autonomous racing, fast and accurate error detection is essential to prevent small deviations from quickly becoming large collisions. This requires incredibly sophisticated sensors and technology capable of handling huge amounts of data on the fly. One method for increasing error detection capabilities is through sensor fusion, where information from different sensors (e.g., LIDAR, cameras, GPS) is combined to create a unified picture of the environmental conditions surrounding the vehicle.

2.5 Sensor Fusion and Environmental Perception

Sensor fusion, which combines information from several sensors, plays an important role in increasing the perception of autonomous vehicles. Such fused data in an autonomous vehicle system would gain a robust and accurate understanding of the environment—imperative for both path and error detection. Buehler et al. (2009) present sensor fusion in the DARPA Urban Challenge for autonomous vehicles. It is further emphasized that sensor fusion ensures not only increased environmental perception but enhanced vehicle detection and response to errors in real time [9].

An example of sensor fusion that will add value to the autonomous racing domain is the ability to detect sudden changes in the track conditions which can trigger a path-following error. Data from the sensors can be assimilated into AI algorithms for detecting errors so that the system in place is more sensitized and updated for more informed decisions, increasing the accuracy and reliability of the control actions taken by the vehicle.

2.6 Challenges in Autonomous Racing and the Need for Adaptive Systems

The high-speed, dynamic nature of autonomous racing presents unique challenges that are not typically encountered in conventional autonomous driving. One of the main challenges is the need for the control system to adapt quickly to changing conditions on the track. This includes adjusting to sharp turns, varying surface conditions, and the presence of other vehicles, all of which can contribute to path-following errors.

To address these challenges, adaptive systems that can modify their behaviour based on real-time data are essential. Adaptive control methods, such as those based on machine learning, allow the system to learn from past experiences and adjust its parameters accordingly. This adaptability is crucial for maintaining optimal performance in the fast-paced environment of autonomous racing.

Chapter 3: Methodology

3.1 Research Design

The research design for this project is structured in a manner that addresses the primary objective of detecting and correcting path-following errors for autonomous racing. Due to the high-speed and dynamic nature of autonomous racing, this research is based on an experimental and iterative approach carried out in simulation tests and with model development driven by data. It is divided into several phases that collectively work towards the main objective of improving the path-following performance of an autonomous race car.

3.1.1 Phase 1: System Implementation and Baseline Evaluation

This phase comprises the implementation of a baseline path-following system, Pure Pursuit. This forms the benchmark for future developments and is to be compared with the control group when the improvements and changes are eventually tested. The Pure Pursuit algorithm is selected because of its simplicity and effectiveness in basic path-following implementations; this is considered to be a good starting point for the research [3]. The performance of the system is first tested with a series of simulation-based racing scenarios in RViz, an observation tool capable of real-time observations of the path-following behaviour of the vehicle [15]. The test will be able to give baseline data on the accuracy of path following, in the sense of lateral deviations and heading errors, which are critical in mapping out areas that would need improvement.

3.1.2 Data Collection and Analysis Phase

The purpose of this second phase after the baseline evaluation is to collect detailed performance data during the operation of the Pure Pursuit algorithm. Such details include vehicle speed, curvature of the path, lookahead distance, lateral deviation, and heading errors [16]. The process involves collecting data in real-time during a simulated racing environment and logging it for later analysis. This is a very important phase to understand the relationship between different control parameters with respect to the resulting path-following performance. The data collected in this phase will be further used in one of the subsequent stages of machine learning in this dissertation for the task of enhancing the Pure Pursuit algorithm [16].

3.1.3 Phase 3: Development of AI-Enhanced Path-Following System

In the third phase, the research focus shall revolve around augmenting the developed Pure Pursuit algorithm with artificial intelligence techniques. It is developed by making a decision tree through the data captured during phase 2 [3, 10]. This allows the realization of an optimization to be able to predict the optimal lookahead distance with respect to the vehicle's current speed, as well as the path's curvature, through training. The AI-enhanced system is designed to enhance the capability of the vehicle for path following in general, particularly in following the optimal racing line in scenarios with sharp turns and varying speeds, where the traditional Pure Pursuit algorithm struggles [3].

3.1.4 Phase 4: Testing, Validation, and Iterative Refinement

The final phase contains the testing and validation of the AI-enhanced path-following system. This system is exposed to a variety of simulated racing environments, each designed to target the various aspects of path-following performance: high-speed corners, sudden obstacles, and changes in track conditions [13]. The results are compared against the baseline determined during Phase 1 in order to quantify the improvements in path-following accuracy and error reduction. Depending on the results obtained, the system is again iteratively refined to adapt the AI model and control algorithm in ways that give the best performance possible [12][13]. This will help ensure that the system produced will be strong and stable enough to deal with the challenges and complexities involved in autonomous racing.

3.2 Research Methods

The research in this thesis applies simulation, data-driven model development, and algorithmic testing to achieve the stated objectives. These have been selected as methodologies relevant to providing a comprehensive framework for developing, testing, and refining the system for detecting path-following errors. Each methodology offers insights into system performance and contributes to the overall validation of the research outcomes.

3.2.1 Simulation-Based Testing

This research is mainly founded upon simulation-based testing, which provides a controlled environment in which the path-following system can be tested and assessed thoroughly under a number of scenarios. This allows one to exercise a large number of varying racing conditions, just like what happens under real-world challenges, where sharp turns, varying track surfaces, and high-speed straights are very common [14]. Such conditions can be simulated by the research. It provides an estimate of the system's capability to detect and rectify path-following errors, now without the related risks involved in physical testing.

The most important part is that this method works through the use of RViz, a very powerful visualization tool. It permits the monitoring of the vehicle's trajectory at run-time, plus the metrics of the errors and control responses during simulation. This is necessary for understanding what the behaviour of the system is like in different conditions and what must be further polished. Besides, RViz allows sensor data and environmental interaction to be simulated such that the testing environment closely emulates the situations appearing during real-world racing [15].

The possibility of reproducing high-risk environments in a safe and repeatable way enables the AI-enhanced system to be heavily tested. Through countless simulations, valuable data are collected on how the system reacts in various conditions, which, on the other hand, forms the robust basis for developing and refining it further. The findings are benchmarked to the enhancements brought about by the AI-enriched system.

3.2.2 Data-Driven Model Development

The core purpose of this research is to build on the Pure Pursuit algorithm using machine learning. This approach, therefore, centres on data collection during the simulation phase. This valuable information includes all key metrics for describing vehicle speed, path curvature, lookahead distance, and path-following errors in such a way that a detailed dataset is obtained, actually portraying system performance in many scenarios [3][16].

In this dataset, a decision tree model for the prediction of optimal control parameters like lookahead distance and heading errors are trained based on the current state of the vehicle and environmental conditions. This model is developed following supervised learning principles—the relations between the input features, such as speed and curvature of the road, and the desired output (optimal control parameters), from which these features will be learned [10]. Therefore, the aim is to design a model and apply it to realize dynamic adaptation of the Pure Pursuit algorithm in real-time for increased adaptability and accuracy.

The training process will split the data into training and validation sets to ensure the generalization of the model. Hyperparameter optimization based on cross-validation techniques will be done in order to maximize predictive accuracy. After training, the model is integrated into the path-following system and continuously adapts the control parameters to the environment as perceived by the sensors on the vehicle [10]. Therefore, the dynamic adjustments of the path-following errors are expected to reduce, especially in challenging racing conditions.

3.2.3 Algorithm Testing and Validation

It is very important to test and validate the developed algorithm to ensure it meets the research objectives and performs correctly in the field of autonomous racing. This will involve rigorous testing of the base Pure Pursuit algorithm and the AI-enhanced one, focusing on comparing such key performance indicators as lateral deviation, heading error, and lap times.

In this phase, quantitative validation is conducted with the execution of a series of simulations in diverse environments, each one designed for testing different capabilities of the system. This includes, for example, checking the possibility of maintaining a steady line in high-speed straights and verifying its reaction to sharp turns, which would provide feedback on instantaneous path curvature changes. The outputs are then put under analysis for both accuracy in tracking the desired path and the minimization of errors [11][13].

Qualitative validation complements such testing through the evaluation of the overall behaviour and stability of the overall system within the simulation. This would account for the racing scenarios as the system looks at how it responds to a situation, such as overtaking, and evaluates whether the AI-augmented system has offered better, more consistent control in comparison to the baseline. The validation is run through repeated testing in various simulated environments—doing so will prove the robustness and that it will perform when exposed to varying conditions while on the racetrack. These tests' results motivate the next refinements in the system to make sure that the resulting final solution is reliable and effective enough to be implemented in a real-world application [13].

3.2.4 Iterative Development and Refinement

The iterative development and refinement methodology has a research process; the further processes of testing and refining bring about progressive improvements in the system's performance. Valuable feedback from testing allows for incremental development enhancements to the AI model, control algorithms, and system architecture as a whole [10].

This iterative way is particularly well designed for the complex and dynamic nature of autonomous racing, where very close interaction with track models and live performance data, in real-time, are crucial. For example, preliminary tests with the AI model will indicate that more training data needs to be collected for greater accuracy in the predictions; hence, the collection of more data and refinement of the AI model are necessary. In a similar vein, setting the control parameters might demand adjustments in order to secure a better answer from the system under specific scenarios [10][12].

This iterative process, overall, aims for a path-following solution of autonomous racing with high standards of accuracy, reliability, and speed. The work in this research is directed towards the development of the system, robust error detection and correction mechanism through continuous refinement of the system, to be the best for the aggressive environment of autonomous racing.

3.3 Data Collection

Data collection assumes great significance in this research as it forms the very foundation on which an AI-enhanced path-following system will be built and iterated. Quality and relevance in the collected data will directly affect the effectiveness of machine learning models and, therefore, overall system success in regard to error detection and correction in path-following. The data collection process is carefully designed to measure a wide range of performance indicators that are representative of the vehicle's behaviour in a diversity of racing situations.

3.3.1 Data Sources

The main data source is the simulation environment, which closely mirrors the environments in which the vehicles experience driving in real races, with very high levels of realism [16][21]. It is a simulator that gives a controlled and dynamic environment to systematically change parameters that vary in vehicle speed, path curvature, and environmental conditions. This means that the collected data covers a variety of scenarios such as high-speed straight sections, sharp turns, or sections with varying friction coefficients [6][22]. The following data is recorded for each run:

- ***Vehicle Speed:*** The speed of the vehicle is recorded continuously over the duration of the simulation. Speed data is particularly important in examining the dynamic behaviour of the vehicle in relation to its effects on path-following errors [6][12].
- ***Path Curvature:*** The curvature of the path at each point of the racing line is computed and recorded. This metric is especially important for identifying challenging areas of the track where the vehicle is more likely to deviate [11][20].
- ***Lookahead Distance:*** The actual lookahead distance used by the Pure Pursuit algorithm is logged to understand how various lookahead settings influence path-tracking performance [3][20].
- ***Lateral Deviation:*** The lateral deviation of the vehicle from the desired path is measured at fixed waypoints. This is a direct indication of path-following errors and is used to assess the effectiveness of the AI enhancements [3][20].
- ***Heading Error:*** The difference between the current heading of the vehicle and the desired heading, which is the direction that one should take while on the path, is monitored. This metric serves to identify situations where the vehicle's orientation is misaligned with the path, potentially leading to deviations [5][6].

3.3.2 Data Logging and Storage

Simulation data is collected in real-time and stored in an organized way so that it can be accessed and analyzed relatively easily. Each run of the simulation provides a suite of information for the predefined metrics along with time stamps and contextual information about the specific scenario (i.e., track conditions and obstacles) [14][18]. The data is stored in CSV files, with each row representing a snapshot of the vehicle's state at that instant in time. The use of CSV format facilitates easy integration with data analysis tools and machine learning libraries. The structured format also allows for straightforward filtering, sorting, and analyzing of the data according to various criteria, such as specific track sections or vehicle behaviours [19][20].

3.3.3 Data Quality and Integrity
Ensuring the high quality and integrity of the data acquired has been a prime concern at every step of the data collection process. A few measures adopted are listed below:

- ***Data Consistency:*** The simulation setup ensures consistent results across multiple runs to reduce variance that could otherwise skew the results [16]. For example, the physics engine used in the simulator is carefully tuned to ensure the vehicle's behaviour accurately reflects real-world dynamics [16][21].
- ***Error Handling:*** Anomalies and errors occurring during data collection, such as sensor glitches or unexpected behaviour within the simulation, are detected by mechanisms that flag and log them [15]. These anomalies are flagged for further investigation, and, if needed, the affected data points are excluded from analysis [19][22].
- ***Sufficient Coverage:*** The design of the data collection process ensures that it covers a wide range of racing scenarios, including edge cases [6]. Such comprehensive coverage is crucial for developing a model that can generalize effectively in new contexts [7][8].

3.4 Data Analysis
Data analysis is a critical phase of this research, where the collected data is processed, examined, and interpreted to extract meaningful insights that inform the development and refinement of the path-following system. The analysis focuses on identifying patterns, correlations, and outliers in the data that can reveal the strengths and weaknesses of the current system and guide improvements [10][12].

3.4.1 Preprocessing and Feature Engineering
Before analysis can begin, the raw data collected during simulations undergoes preprocessing to ensure it is clean, consistent, and ready for analysis. This process involves several steps:

- ***Data Cleaning:*** Anomalies such as missing values or outliers flagged during data collection are addressed. Outliers are either corrected (if possible) or removed to prevent them from distorting the analysis [10][16].
- ***Normalization:*** Continuous variables like vehicle speed and lateral deviation are normalized to a common scale. This step is particularly important for machine learning, ensuring all features contribute equally to the model's predictions [20][24].
- ***Feature Engineering***: New features are derived from the existing data to capture more complex relationships between variables [12][20]. For example,

the rate of change of curvature or the vehicle’s acceleration may be computed and added as additional features. These engineered features can provide deeper insights into the system's behaviour and improve the predictive power of the machine learning models [19][23].

3.4.2 Statistical Analysis and Visualization

Once the data is preprocessed, statistical analysis is conducted to explore relationships between different variables and to quantify the system's performance. Key metrics, such as the mean and standard deviation of lateral deviation and heading error, are calculated to summarize the path-following accuracy [3][6][9]. Correlation analysis is also performed to identify dependencies between variables, such as the impact of speed on path-following errors [8][24].

Visualization plays a significant role in the data analysis process. By plotting the data, patterns and trends that may not be immediately apparent in the raw numbers can be more easily identified. For example:

- ***Time Series Plots:*** These are used to track how key metrics, such as lateral deviation and heading error, change over time during a simulation [10][12]. This helps to pinpoint specific moments when the system struggled to maintain the desired path.
- ***Scatter Plots:*** These plots are used to examine relationships between pairs of variables, such as speed versus lateral deviation, providing insights into how different factors influence path-following accuracy [10][14].
- ***Heatmaps:*** Heatmaps can visualize the distribution of errors across different sections of the track, highlighting areas where the system may need improvement [7][22].

3.4.3 Model Training and Evaluation

The cleaned and processed data is then used to train the decision tree model that enhances the Pure Pursuit algorithm. The training process involves splitting the dataset into training and validation sets to ensure that the model is evaluated on data it has not seen during training. This approach helps to prevent overfitting and ensures that the model generalizes well to new scenarios [20][22].

During training, various machine learning algorithms are tested to determine which provides the best performance. These may include linear regression, decision trees, or more advanced techniques such as gradient boosting or neural networks [24]. The model's performance is evaluated using metrics such as Mean Squared Error (MSE) and R-squared, which quantify the accuracy of the model's predictions [10][14].

Hyperparameter tuning is also performed to optimize the model's performance. This involves systematically adjusting parameters such as learning rate or tree depth (in the case of decision trees) to find the best configuration [22][24]. Cross-validation techniques, such as k-fold cross-validation, are used to ensure that the model's performance is robust and not overly sensitive to specific subsets of the data [10][23].

3.4.4 Interpretation and Insight Extraction

The final step in the data analysis process involves interpreting the results and extracting actionable insights that can guide the further development of the path-following system. This includes:

- ***Identifying Key Predictors:*** Determining which features (e.g., speed, curvature) have the most significant impact on path-following errors, and using this information to refine the model or the control algorithm.
- ***Assessing System Performance:*** Comparing the AI-enhanced system's performance against the baseline to quantify the improvements in path-following accuracy and error reduction.
- ***Highlighting Areas for Improvement:*** Identifying any weaknesses or limitations in the current system and proposing strategies for addressing them in future iterations.

The insights gained from this analysis are used to iteratively refine the system, with the goal of achieving a robust, high-performance path-following solution that excels in the demanding environment of autonomous racing.

3.5 Tools and Techniques

The research and development work presented in this thesis heavily relies on a combination of powerful tools and techniques that facilitate the implementation, testing, and analysis of the path-following system. These tools were selected for their robustness, versatility, and ability to integrate seamlessly within the research framework. The primary tools used include Python for programming and data analysis, Visual Studio Code (VS Code) as the integrated development environment (IDE), and RViz for simulation and visualization. The development environment is set up on Ubuntu through Windows Subsystem for Linux 2 (WSL2) with ROS Melodic, providing a flexible and efficient platform for autonomous vehicle research.

3.5.1 Python

Python is the primary programming language used throughout this research due to its extensive libraries, ease of use, and strong community support. Python's versatility makes it ideal for implementing path-following algorithms, developing machine-learning models, and conducting data analysis.

Key Python libraries used in this research include:

- ***NumPy:*** For numerical computations, including matrix operations and handling large datasets efficiently.
- ***Pandas:*** For data manipulation and analysis, particularly for handling the collected data during simulations.
- ***Matplotlib and Seaborn:*** For data visualization, enabling the creation of detailed plots and graphs that assist in understanding the system's performance.
- ***Scikit-learn:*** For developing and training machine learning models, including decision tree models used to enhance the Pure Pursuit algorithm.
- ***ROS (Robot Operating System) Libraries:*** For interfacing with the simulation environment, handling sensor data, and controlling the autonomous vehicle's behaviour within the simulation.

Python's integration with ROS is particularly valuable, allowing for seamless communication between different components of the autonomous system, including sensors, control algorithms, and visualization tools.

3.5.2 Visual Studio Code (VS Code)

Visual Studio Code (VS Code) is used as the integrated development environment (IDE) for writing, debugging, and managing the project's codebase. VS Code is chosen for its lightweight design, powerful features, and extensive support for Python development.

Key features of VS Code that support this research include:

- ***Integrated Terminal:*** Allows direct execution of Python scripts and ROS commands, facilitating a smooth workflow between development and testing.
- ***Debugging Tools:*** Provide real-time debugging capabilities, enabling the identification and resolution of issues in the code efficiently.
- ***Extensions:*** A variety of extensions are used to enhance Python development, such as Pylint for code linting, Jupyter for running notebooks, and ROS-specific extensions for better integration with ROS packages.

VS Code's versatility and customization options make it an excellent choice for managing the complex and multi-faceted aspects of this research project.

3.5.3 RViz

RViz is the primary tool used for simulation and visualization in this research. It is a powerful 3D visualization tool that is part of the Robot Operating System (ROS) ecosystem, designed to visualize the sensor data, model state, and environment of robotic systems.

In this research, RViz is used extensively to:

- ***Visualize the Vehicle's Path:*** Display the planned and actual paths of the autonomous vehicle in the simulation environment, allowing for real-time monitoring of path-following accuracy.
- ***Monitor Sensor Data:*** Visualize data from virtual sensors, such as LIDAR and cameras, to ensure that the vehicle is perceiving its environment correctly.
- ***Error Detection Visualization:*** Display error markers and other indicators that help in diagnosing path-following errors during simulations.

RViz's ability to integrate with ROS and provide real-time feedback makes it an indispensable tool for validating the performance of the autonomous vehicle in a simulated racing environment.

3.5.4 Ubuntu on WSL2 with ROS Melodic

The development environment is set up on Ubuntu 18.04 running on Windows Subsystem for Linux 2 (WSL2), with ROS Melodic installed. This setup combines the flexibility and power of a Linux-based environment with the convenience of running on a Windows machine.

Key aspects of this setup include:

- ***WSL2*****:** Provides a lightweight, virtualized Linux environment that integrates seamlessly with the Windows operating system, allowing for efficient development and testing without the need for dual-booting or separate hardware.
- ***Ubuntu 18.04:*** Chosen for its compatibility with ROS Melodic, offering a stable and well-supported platform for autonomous vehicle development.
- ***ROS Melodic:*** Provides the middleware framework that supports communication between different components of the robotic system, including sensors, actuators, and control algorithms.

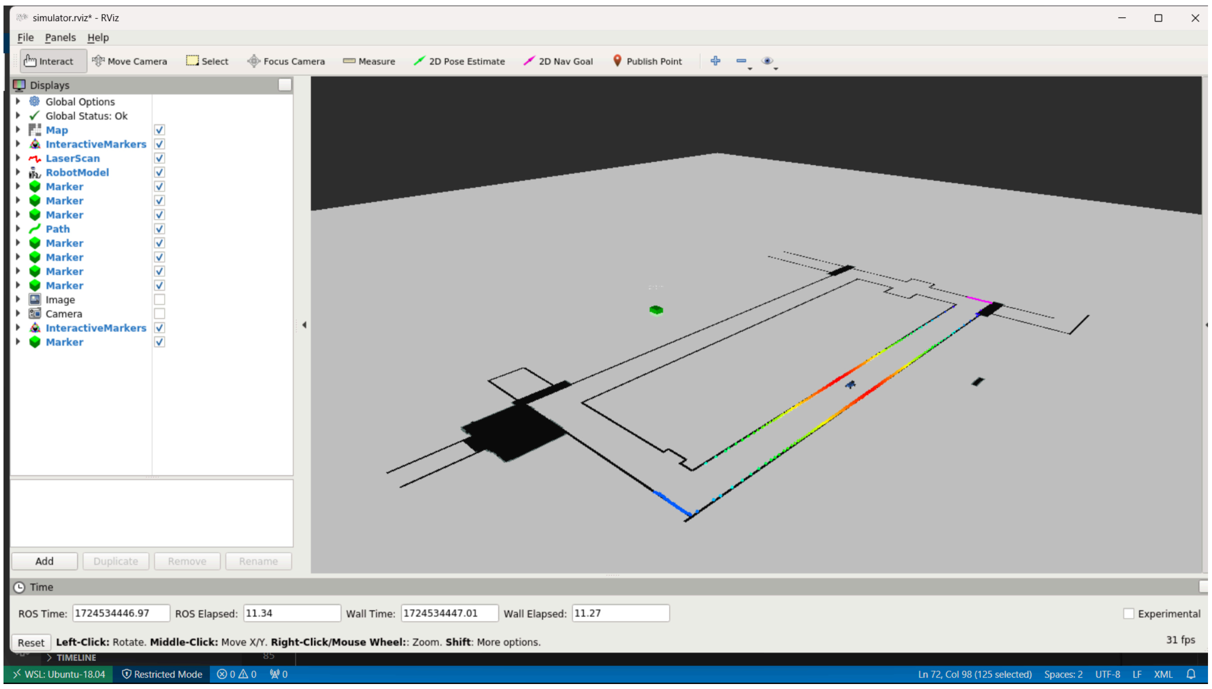
This environment allows for the full use of ROS's capabilities in a flexible and efficient manner, facilitating the development and testing of the autonomous vehicle system in a simulated environment [17, 18].

3.6 Implementation

The implementation of the path-following error detection system for autonomous racing was carried out in several steps using Python and ROS (Robot Operating System), primarily as the main platforms for development.

In the initial stages of the implementation, the F1TENTH simulator was set up by following the comprehensive instructions provided in the official F1TENTH documentation [21]. This guide walks through the process of installing and launching the simulator environment, which is essential for testing and validating autonomous racing algorithms. The simulator provides a realistic, physics-based platform that allows for safe and repeatable testing of the Pure Pursuit algorithm and its AI-enhanced version. By adhering to the steps outlined in the documentation, the simulator was successfully configured as shown in Figure 4.

It all started by defining and publishing a path using a set of waypoints defined manually in a Python script named *path_subscriber.py* as shown in Figure 5. These waypoints constructed the Path message, published into the /smoothed_path topic to specify the trajectory that should be followed by the vehicle



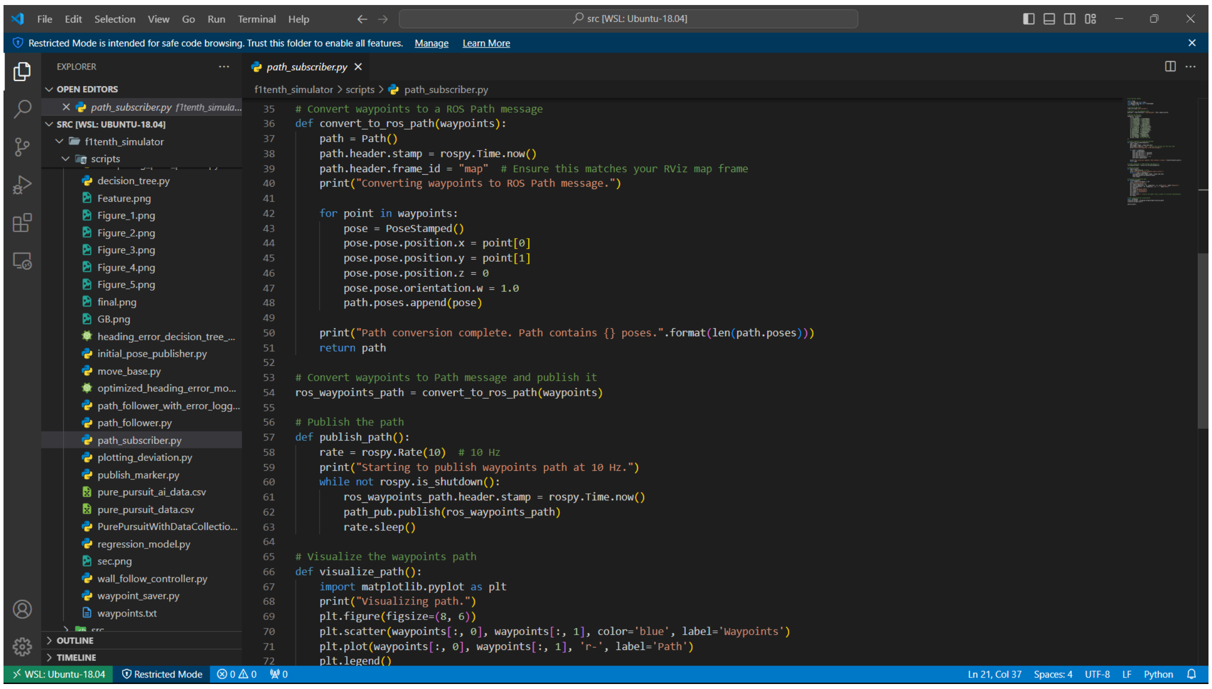
ROS Time: 1724534446.97 ROS Elapsed: 11.34 Wall Time: 1724534447.01 Wall Elapsed: 11.27

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:**: Zoom. **Shift**: More options.

31 fps

WSL: Ubuntu-18.04 Restricted Mode ⊗ 0 ⚠ 0 0 Ln 72, Col 98 (125 selected) Spaces: 2 UTF-8 LF XML

Figure 4: F1Tenth simulator

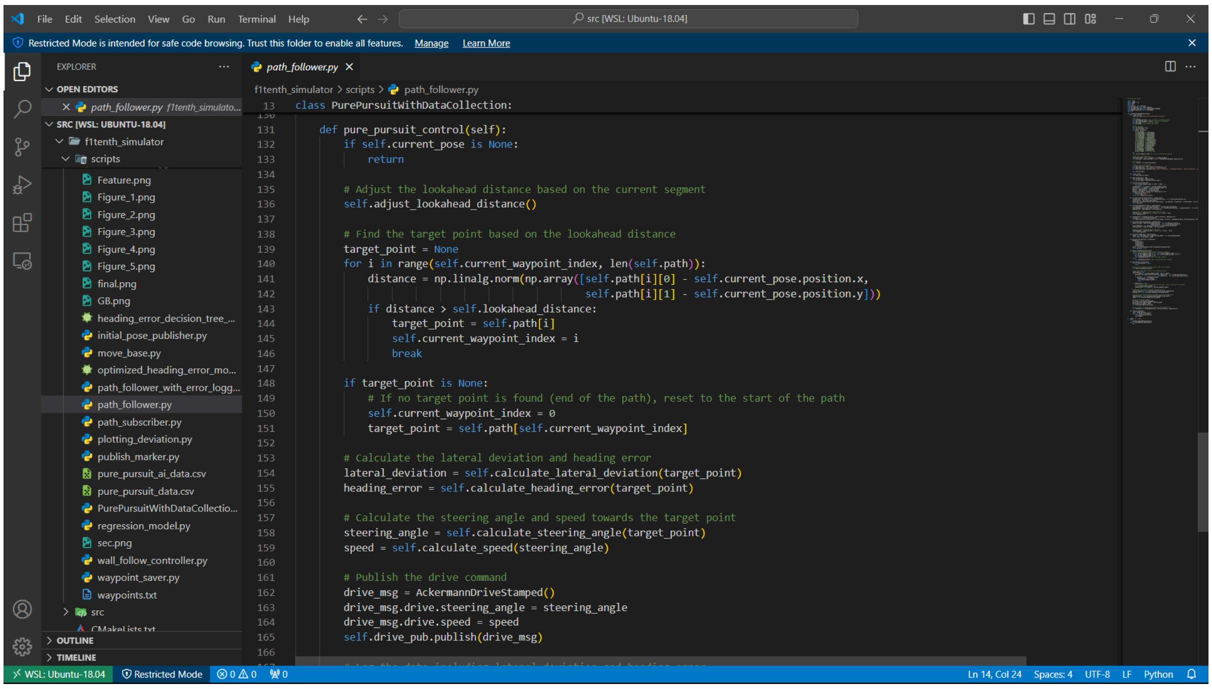


```python
# Convert waypoints to a ROS Path message
def convert_to_ros_path(waypoints):
    path = Path()
    path.header.stamp = rospy.Time.now()
    path.header.frame_id = "map"  # Ensure this matches your RViz map frame
    print("Converting waypoints to ROS Path message.")

    for point in waypoints:
        pose = PoseStamped()
        pose.pose.position.x = point[0]
        pose.pose.position.y = point[1]
        pose.pose.position.z = 0
        pose.pose.orientation.w = 1.0
        path.poses.append(pose)

    print("Path conversion complete. Path contains {} poses.".format(len(path.poses)))
    return path

# Convert waypoints to Path message and publish it
ros_waypoints_path = convert_to_ros_path(waypoints)

# Publish the path
def publish_path():
    rate = rospy.Rate(10)  # 10 Hz
    print("Starting to publish waypoints path at 10 Hz.")
    while not rospy.is_shutdown():
        ros_waypoints_path.header.stamp = rospy.Time.now()
        path_pub.publish(ros_waypoints_path)
        rate.sleep()

# Visualize the waypoints path
def visualize_path():
    import matplotlib.pyplot as plt
    print("Visualizing path.")
    plt.figure(figsize=(8, 6))
    plt.scatter(waypoints[:, 0], waypoints[:, 1], color='blue', label='Waypoints')
    plt.plot(waypoints[:, 0], waypoints[:, 1], 'r-', label='Path')
    plt.legend()
```

Figure 5: Code Snippet of path_subscriber.py

The system's foundation is built on the Pure Pursuit algorithm, a highly regarded path-tracking method for autonomous ground vehicles. This algorithm is essential for steering the vehicle along a predefined route by calculating the necessary steering commands. It continuously fine-tunes the vehicle's trajectory toward a target waypoint on the path, ensuring close adherence to the intended route.

Within this system, the Pure Pursuit algorithm is implemented in the `path_follower.py` script as shown in Figure 6. This script not only manages the real-time computation of steering commands but also records critical data during the vehicle's operation. The algorithm dynamically adjusts the lookahead distance, which dictates how far ahead on the path the vehicle should aim. This adjustment is influenced by the vehicle's current position and the characteristics of the path segment, whether it is a straight section or a curve.



```python
class PurePursuitWithDataCollection:

    def pure_pursuit_control(self):
        if self.current_pose is None:
            return

        # Adjust the lookahead distance based on the current segment
        self.adjust_lookahead_distance()

        # Find the target point based on the lookahead distance
        target_point = None
        for i in range(self.current_waypoint_index, len(self.path)):
            distance = np.linalg.norm(np.array([self.path[i][0] - self.current_pose.position.x,
                                                self.path[i][1] - self.current_pose.position.y]))
            if distance > self.lookahead_distance:
                target_point = self.path[i]
                self.current_waypoint_index = i
                break

        if target_point is None:
            # If no target point is found (end of the path), reset to the start of the path
            self.current_waypoint_index = 0
            target_point = self.path[self.current_waypoint_index]

        # Calculate the lateral deviation and heading error
        lateral_deviation = self.calculate_lateral_deviation(target_point)
        heading_error = self.calculate_heading_error(target_point)

        # Calculate the steering angle and speed towards the target point
        steering_angle = self.calculate_steering_angle(target_point)
        speed = self.calculate_speed(steering_angle)

        # Publish the drive command
        drive_msg = AckermannDriveStamped()
        drive_msg.drive.steering_angle = steering_angle
        drive_msg.drive.speed = speed
        self.drive_pub.publish(drive_msg)
```

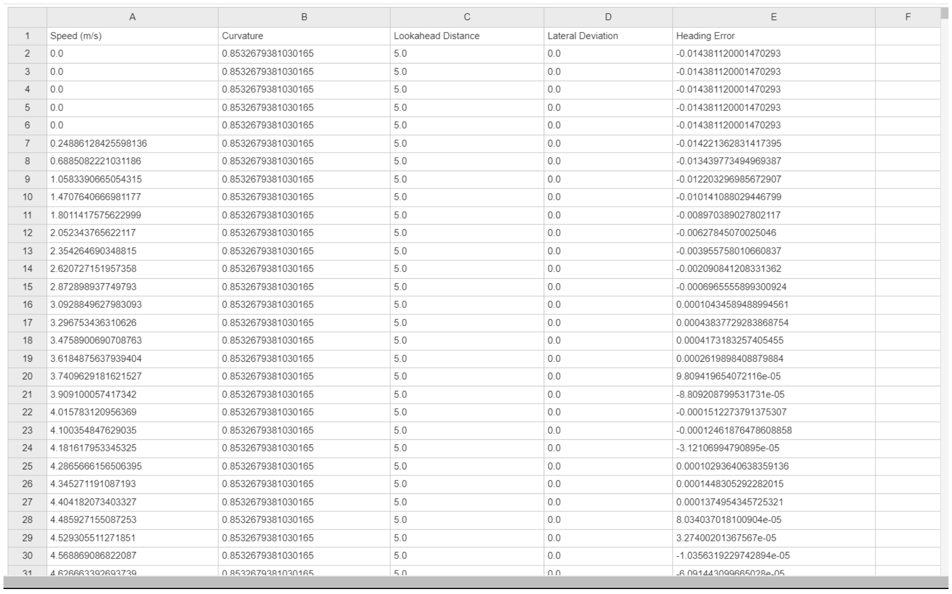
Figure 6: Code snippet of path_follower.py

The `path_follower.py` script subscribes to the `/odom` topic to receive the vehicle's current position and orientation. Using this information, it computes the steering angle necessary to guide the vehicle toward the next target waypoint on the path. These calculated steering commands are then published to the `/drive` topic, which directly controls the vehicle's motion.

Beyond executing the Pure Pursuit algorithm, the *`path_follower.py`* script logs crucial parameters into a CSV file named *`pure_pursuit_data.csv`*. The data logged includes:

- ***Speed (m/s):*** The vehicle's current speed.
- ***Curvature:*** The curvature of the path at each point, indicating the degree of turning required.
- ***Lookahead Distance:*** The dynamically adjusted distance ahead on the path that the vehicle uses to calculate the steering angle.
- ***Lateral Deviation:*** The perpendicular distance between the vehicle's current position and the closest point on the path.
- ***Heading Error:*** The angular difference between the vehicle's current heading and the direction of the path at the nearest point.

Figure 7 shows the logged data which is invaluable for training the AI model, it is designed to enhance the Pure Pursuit algorithm by predicting optimal lookahead distances. The AI model utilizes this data to understand the relationship between different driving conditions and the ideal lookahead distance, ultimately improving the vehicle's path-following accuracy.

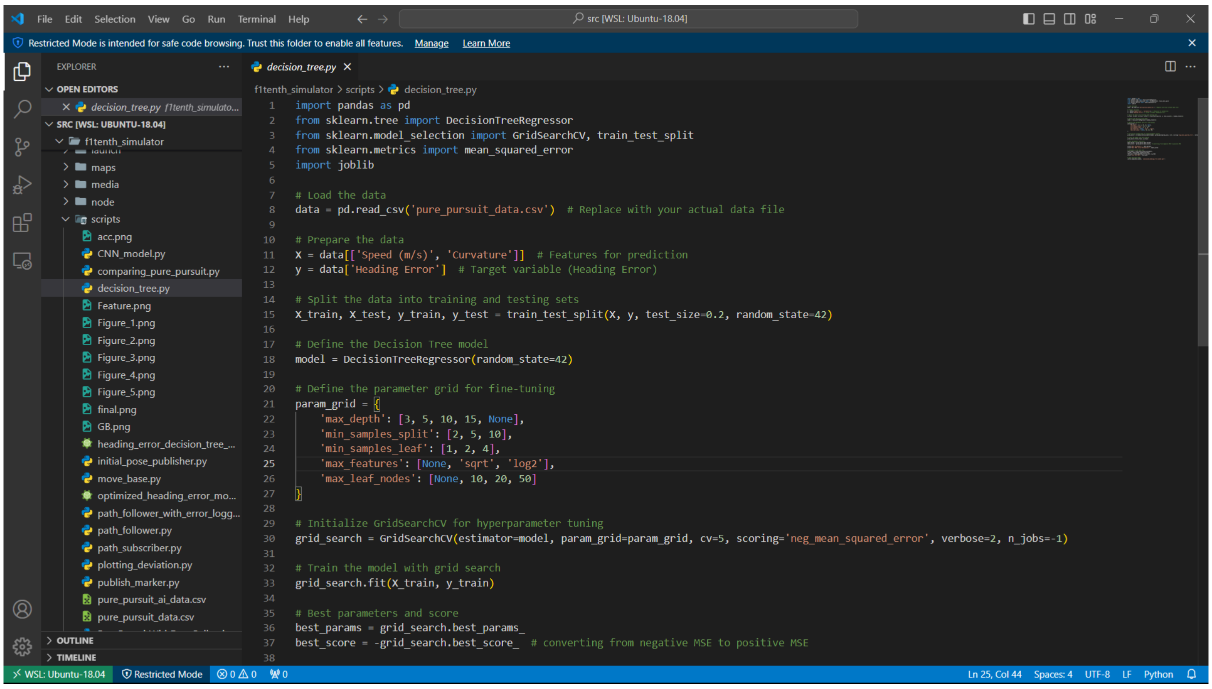


	A	B	C	D	E	F
1	Speed (m/s)	Curvature	Lookahead Distance	Lateral Deviation	Heading Error	
2	0.0	0.8532679381030165	5.0	0.0	-0.014381120001470293	
3	0.0	0.8532679381030165	5.0	0.0	-0.014381120001470293	
4	0.0	0.8532679381030165	5.0	0.0	-0.014381120001470293	
5	0.0	0.8532679381030165	5.0	0.0	-0.014381120001470293	
6	0.0	0.8532679381030165	5.0	0.0	-0.014381120001470293	
7	0.24886128425598136	0.8532679381030165	5.0	0.0	-0.014221362831417395	
8	0.6885082221031186	0.8532679381030165	5.0	0.0	-0.013439773494969387	
9	1.0583390665054315	0.8532679381030165	5.0	0.0	-0.012203296985672907	
10	1.4707640666981177	0.8532679381030165	5.0	0.0	-0.010141088029446799	
11	1.8011417575622999	0.8532679381030165	5.0	0.0	-0.008970389027802117	
12	2.052343765622117	0.8532679381030165	5.0	0.0	-0.00627845070025046	
13	2.354264690348815	0.8532679381030165	5.0	0.0	-0.003955758010660837	
14	2.620727151957358	0.8532679381030165	5.0	0.0	-0.002090841208331362	
15	2.872898937749793	0.8532679381030165	5.0	0.0	-0.0006965555899300924	
16	3.0928849627983093	0.8532679381030165	5.0	0.0	0.00010434589488994561	
17	3.296753436310626	0.8532679381030165	5.0	0.0	0.00043837729283868754	
18	3.4758900690708763	0.8532679381030165	5.0	0.0	0.0004173183257405455	
19	3.6184875637939404	0.8532679381030165	5.0	0.0	0.0002619898408879884	
20	3.7409629181621527	0.8532679381030165	5.0	0.0	9.809419654072116e-05	
21	3.909100057417342	0.8532679381030165	5.0	0.0	-8.809208799531731e-05	
22	4.015783120956369	0.8532679381030165	5.0	0.0	-0.0001512273791375307	
23	4.100354847629035	0.8532679381030165	5.0	0.0	-0.00012461876478608858	
24	4.181617953345325	0.8532679381030165	5.0	0.0	-3.12106994790895e-05	
25	4.2865666156506395	0.8532679381030165	5.0	0.0	0.00010293640638359136	
26	4.345271191087193	0.8532679381030165	5.0	0.0	0.0001448305292282015	
27	4.404182073403327	0.8532679381030165	5.0	0.0	0.0001374954345725321	
28	4.485927155087253	0.8532679381030165	5.0	0.0	8.034037018100904e-05	
29	4.529305511271851	0.8532679381030165	5.0	0.0	3.27400201367567e-05	
30	4.568869086822087	0.8532679381030165	5.0	0.0	-1.0356319229742894e-05	
31	4.626663392693739	0.8532679381030165	5.0	0.0	-6.091443099665028e-05	

Figure 7: pure_pursuit_data.csv

After collecting data using the Pure Pursuit algorithm, the next step is to train an AI model to enhance the vehicle's path-following capabilities. This is done by executing the *`decision_tree.py`* script, which makes use of the data stored in the *`pure_pursuit_data.csv`* file.

As shown in Figure 8 *`decision_tree.py`* script is designed to train a Decision Tree Regression model. The model aims to predict the optimal lookahead distance that minimizes path-following errors, with a particular focus on reducing heading errors. The Decision Tree model is chosen for its capacity to manage non-linear relationships in the data, making it suitable for the complex dynamics involved in autonomous vehicle control.



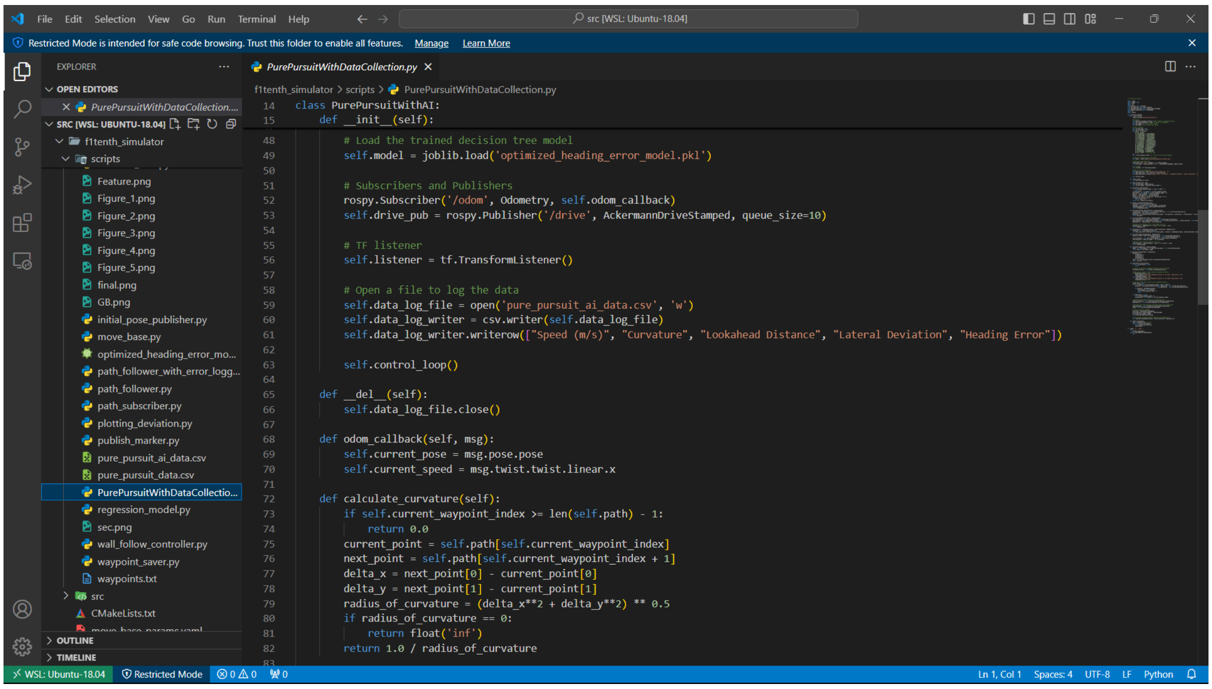
```python
import pandas as pd
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.metrics import mean_squared_error
import joblib

# Load the data
data = pd.read_csv('pure_pursuit_data.csv')  # Replace with your actual data file

# Prepare the data
X = data[['Speed (m/s)', 'Curvature']]  # Features for prediction
y = data['Heading Error']  # Target variable (Heading Error)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define the Decision Tree model
model = DecisionTreeRegressor(random_state=42)

# Define the parameter grid for fine-tuning
param_grid = {
    'max_depth': [3, 5, 10, 15, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': [None, 'sqrt', 'log2'],
    'max_leaf_nodes': [None, 10, 20, 50]
}

# Initialize GridSearchCV for hyperparameter tuning
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5, scoring='neg_mean_squared_error', verbose=2, n_jobs=-1)

# Train the model with grid search
grid_search.fit(X_train, y_train)

# Best parameters and score
best_params = grid_search.best_params_
best_score = -grid_search.best_score_  # converting from negative MSE to positive MSE

```

Figure 8: Code snippet of decision_tree.py

Key Steps in the Training Process:

- ***Loading the Data:*** The script starts by loading the data from *`pure_pursuit_data.csv`*. This dataset includes essential information such as the vehicle's speed, the curvature of the path, lookahead distance, lateral deviation, and heading error.
- ***Feature Selection:*** The features selected to train the model are the vehicle's speed and the curvature of the path. These inputs are chosen because they play a critical role in determining the optimal lookahead distance needed to minimize heading errors.
- ***Model Training:*** A Decision Tree Regressor is trained using the dataset. The model learns the relationship between the input features (speed and curvature) and the target variable, which is the heading error. The aim is to enable the model to predict the lookahead distance that will minimize heading error under different driving conditions.
- ***Model Evaluation:*** The model's performance is evaluated using metrics like Mean Squared Error (MSE). This evaluation helps to determine how well the model can generalize to new, unseen data.
- ***Model Saving:*** Once trained and validated, the model is saved to a file named *`optimized_heading_error_model.pkl`*. This file contains the optimized model, which can be loaded during the vehicle's operation to adjust the lookahead distance dynamically in real time.

After training and saving the optimized model in `optimized_heading_error_model.pkl`, the next step is to integrate this model into the Pure Pursuit algorithm to create an AI-enhanced version. This enhanced algorithm is implemented in the `PurePursuitWithDataCollection.py` script as shown below in Figure 9.



```python
class PurePursuitWithAI:
    def __init__(self):
        # Load the trained decision tree model
        self.model = joblib.load('optimized_heading_error_model.pkl')

        # Subscribers and Publishers
        rospy.Subscriber('/odom', Odometry, self.odom_callback)
        self.drive_pub = rospy.Publisher('/drive', AckermannDriveStamped, queue_size=10)

        # TF listener
        self.listener = tf.TransformListener()

        # Open a file to log the data
        self.data_log_file = open('pure_pursuit_ai_data.csv', 'w')
        self.data_log_writer = csv.writer(self.data_log_file)
        self.data_log_writer.writerow(["Speed (m/s)", "Curvature", "Lookahead Distance", "Lateral Deviation", "Heading Error"])

        self.control_loop()

    def __del__(self):
        self.data_log_file.close()

    def odom_callback(self, msg):
        self.current_pose = msg.pose.pose
        self.current_speed = msg.twist.twist.linear.x

    def calculate_curvature(self):
        if self.current_waypoint_index >= len(self.path) - 1:
            return 0.0
        current_point = self.path[self.current_waypoint_index]
        next_point = self.path[self.current_waypoint_index + 1]
        delta_x = next_point[0] - current_point[0]
        delta_y = next_point[1] - current_point[1]
        radius_of_curvature = (delta_x**2 + delta_y**2) ** 0.5
        if radius_of_curvature == 0:
            return float('inf')
        return 1.0 / radius_of_curvature
```

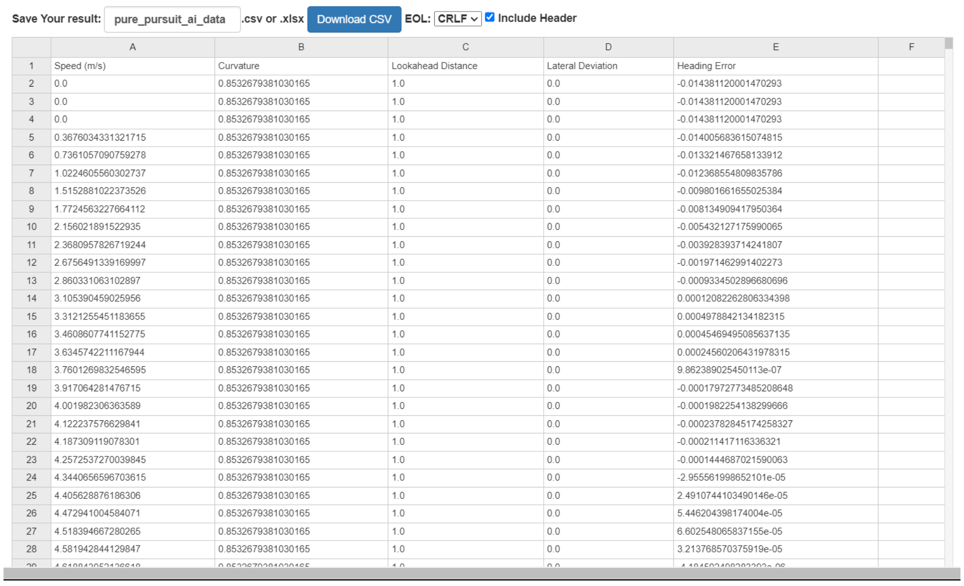
Figure 9: Code snippet of PurePursuitWithDataCollection.py

The `PurePursuitWithDataCollection.py` script extends the standard Pure Pursuit algorithm by incorporating the AI model trained using the Decision Tree Regressor. The AI model is leveraged to dynamically adjust the lookahead distance, taking into account the vehicle's speed and the curvature of the path, to minimize heading errors and enhance path adherence.

<u>Key Features of the AI-Enhanced Algorithm:</u>

1. ***<u>Loading the AI Model:</u>*** At the start of the script, the `optimized_heading_error_model.pkl` file is loaded. This file contains the trained Decision Tree model that is used to predict the optimal lookahead distance.

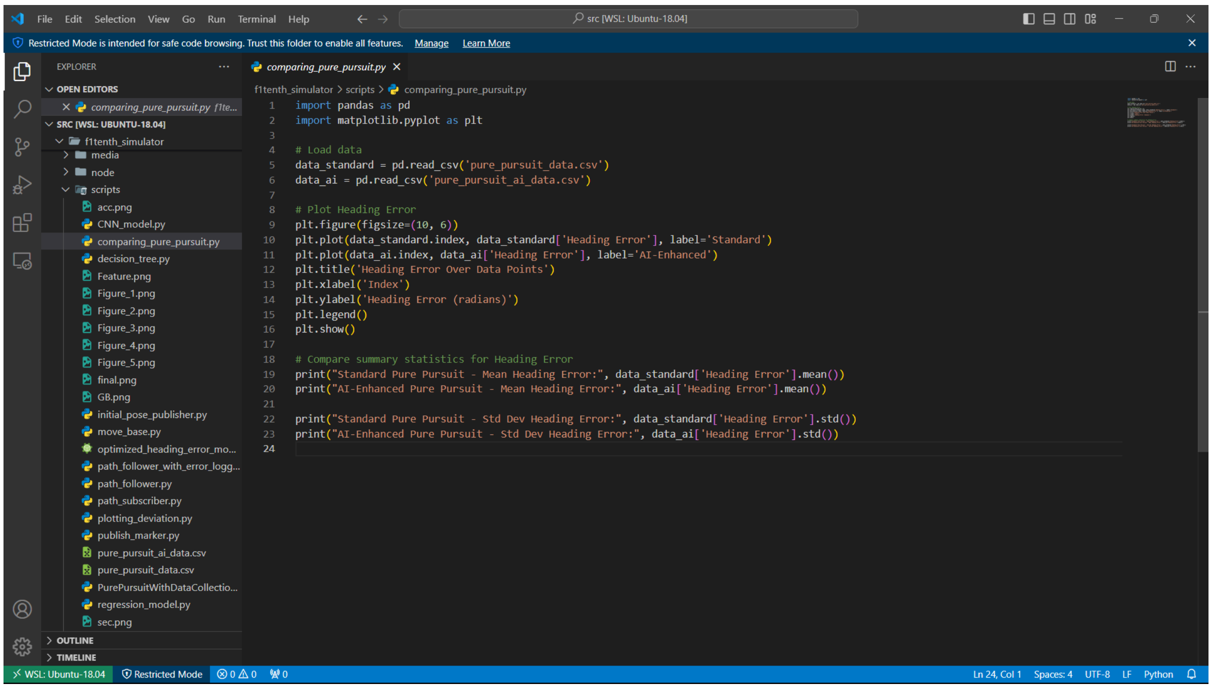
2. ***Real-Time Prediction:*** During the vehicle's operation, the AI model predicts the lookahead distance in real time. The prediction is based on the vehicle's current speed and the curvature of the path at its current location. By dynamically adjusting the lookahead distance, the vehicle can achieve more accurate and stable path-following behaviour.
3. ***Enhanced Path Following:*** The AI-enhanced algorithm calculates the steering angle needed to keep the vehicle on the intended path using the predicted lookahead distance. This approach significantly reduces heading errors, particularly in challenging sections of the path, such as sharp turns.
4. ***Data Logging:*** Similar to the standard Pure Pursuit algorithm, this script logs crucial performance metrics, including speed, curvature, lookahead distance, lateral deviation, and heading error. As shown in Figure 10, this data is stored in a file named *pure_pursuit_ai_data.csv*, enabling further analysis and allowing for potential retraining of the model if necessary.
5. ***Improved Performance:*** By integrating the AI model, the system becomes more adaptable to varying driving conditions, leading to enhanced accuracy in path-following and overall vehicle stability.


Save Your result: pure_pursuit_ai_data .csv or .xlsx Download CSV EOL: CRLF ☑ Include Header

	A	B	C	D	E	F
1	Speed (m/s)	Curvature	Lookahead Distance	Lateral Deviation	Heading Error	
2	0.0	0.8532679381030165	1.0	0.0	-0.014381120001470293	
3	0.0	0.8532679381030165	1.0	0.0	-0.014381120001470293	
4	0.0	0.8532679381030165	1.0	0.0	-0.014381120001470293	
5	0.3676034331321715	0.8532679381030165	1.0	0.0	-0.014005683615074815	
6	0.7361057090759278	0.8532679381030165	1.0	0.0	-0.013321467658133912	
7	1.0224605560302737	0.8532679381030165	1.0	0.0	-0.012368554809835786	
8	1.5152881022373526	0.8532679381030165	1.0	0.0	-0.009801661655025384	
9	1.7724563227664112	0.8532679381030165	1.0	0.0	-0.008134909417950364	
10	2.156021891522935	0.8532679381030165	1.0	0.0	-0.005432127175990065	
11	2.3680957826719244	0.8532679381030165	1.0	0.0	-0.003928393714241807	
12	2.6756491339169997	0.8532679381030165	1.0	0.0	-0.001971462991402273	
13	2.860331063102897	0.8532679381030165	1.0	0.0	-0.0009334502896680696	
14	3.105390459025956	0.8532679381030165	1.0	0.0	0.00012082262806334398	
15	3.3121255451183655	0.8532679381030165	1.0	0.0	0.0004978842134182315	
16	3.4608607741152775	0.8532679381030165	1.0	0.0	0.00045469495085637135	
17	3.6345742211167944	0.8532679381030165	1.0	0.0	0.00024560206431978315	
18	3.7601269832546595	0.8532679381030165	1.0	0.0	9.862389025450113e-07	
19	3.917064281476715	0.8532679381030165	1.0	0.0	-0.00017972773485208648	
20	4.001982306363589	0.8532679381030165	1.0	0.0	-0.0001982254138299666	
21	4.122237576629841	0.8532679381030165	1.0	0.0	-0.00023782845174258327	
22	4.187309119078301	0.8532679381030165	1.0	0.0	-0.000211417116336321	
23	4.2572537270039845	0.8532679381030165	1.0	0.0	-0.0001444687021590063	
24	4.3440656596703615	0.8532679381030165	1.0	0.0	-2.955561998652101e-05	
25	4.405628876186306	0.8532679381030165	1.0	0.0	2.4910744103490146e-05	
26	4.472941004584071	0.8532679381030165	1.0	0.0	5.446204398174004e-05	
27	4.518394667280265	0.8532679381030165	1.0	0.0	6.602548065837155e-05	
28	4.581942844129847	0.8532679381030165	1.0	0.0	3.213768570375919e-05	
29	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	

Figure 10: pure_pursuit_ai_data.py

The last step in assessing the effectiveness of the AI-enhanced Pure Pursuit algorithm is to compare its performance with the standard version. This comparison is performed by analyzing the data collected during the operation of both algorithms, which is stored in the *`pure_pursuit_data.csv`* and *`pure_pursuit_ai_data.csv`* files. The comparison process is implemented in the `comparing_pure_pursuit.py` script shown in Figure 11. The script generates plots that visualize the differences between the two algorithms. The plot typically compares the mean and standard deviation for heading errors of both algorithms.



```python
import pandas as pd
import matplotlib.pyplot as plt

# Load data
data_standard = pd.read_csv('pure_pursuit_data.csv')
data_ai = pd.read_csv('pure_pursuit_ai_data.csv')

# Plot Heading Error
plt.figure(figsize=(10, 6))
plt.plot(data_standard.index, data_standard['Heading Error'], label='Standard')
plt.plot(data_ai.index, data_ai['Heading Error'], label='AI-Enhanced')
plt.title('Heading Error Over Data Points')
plt.xlabel('Index')
plt.ylabel('Heading Error (radians)')
plt.legend()
plt.show()

# Compare summary statistics for Heading Error
print("Standard Pure Pursuit - Mean Heading Error:", data_standard['Heading Error'].mean())
print("AI-Enhanced Pure Pursuit - Mean Heading Error:", data_ai['Heading Error'].mean())

print("Standard Pure Pursuit - Std Dev Heading Error:", data_standard['Heading Error'].std())
print("AI-Enhanced Pure Pursuit - Std Dev Heading Error:", data_ai['Heading Error'].std())
```

Figure 11: Code snippet of comparing_pure_pursuit.py

Chapter 4: Results and Discussion

In this chapter, we present the results obtained from implementing and testing the AI-enhanced Pure Pursuit algorithm for autonomous racing. The process involved several critical steps, from simulating the vehicle's path to integrating machine learning for dynamic control adjustments.

4.1 Results

Here’s a step-by-step plan:

1. ***Simulate Path Using Waypoints:*** We began by creating a path using predefined waypoints. This path was then simulated and visualized using RViz, ensuring that the vehicle had a clearly defined trajectory to follow. The *path_subscriber.py* script was employed to generate and publish this

path as shown in Figure 12, making it accessible for the Pure Pursuit algorithm.

2. ***Implement and Test Pure Pursuit:*** Figure 13 shows that the Pure Pursuit algorithm was implemented using the `path_follower.py` script. This algorithm computed steering commands to keep the vehicle on the intended path by dynamically adjusting the steering angle based on the current position and the next waypoint. During this phase, data such as speed, curvature, lookahead distance, lateral deviation, and heading error were logged into a CSV file (`pure_pursuit_data.csv`) for further analysis.
3. ***Train Decision Tree Model***: Using the logged data, we trained a decision tree model to predict the optimal lookahead distance for the Pure Pursuit algorithm. This model was developed to adjust the lookahead distance dynamically based on factors like speed and curvature, thus improving the vehicle's ability to maintain its path. The model was saved as `optimized_heading_error_model.pkl` after training using the `decision_tree.py` script as shown in Figure 14.
4. ***Integrate AI Model with Pure Pursuit:*** The AI-enhanced version of the Pure Pursuit algorithm was implemented using the `PurePursuitWithDataCollection.py` script as shown in Figure 15. The integrated AI model predicted optimal lookahead distances in real-time, which the Pure Pursuit algorithm used to dynamically adjust its steering commands. The AI-enhanced algorithm was tested under the same conditions as the standard version, and the results were logged in a new CSV file (`pure_pursuit_ai_data.csv`).
5. ***Analyze and Compare Results:*** In Figure 16 the performance of the standard Pure Pursuit algorithm was compared with the AI-enhanced version using the `comparing_pure_pursuit.py` script. Key metrics such as heading error and lateral deviation were analyzed to assess the effectiveness of the AI model. The results indicated that the AI-enhanced algorithm significantly reduced heading error and lateral deviation, particularly in complex path segments involving sharp turns and varying speeds.
6. ***Visualize Data with Graphs:*** The logged data from both the standard and AI-enhanced Pure Pursuit algorithms were visualized using graphs as shown in Figure 17. These visualizations highlighted the performance differences, showing that the AI-enhanced algorithm provided a clear improvement in maintaining the desired path, especially in challenging scenarios. The graphs offered a clear visual representation of the improvements in both heading error and lateral deviation.

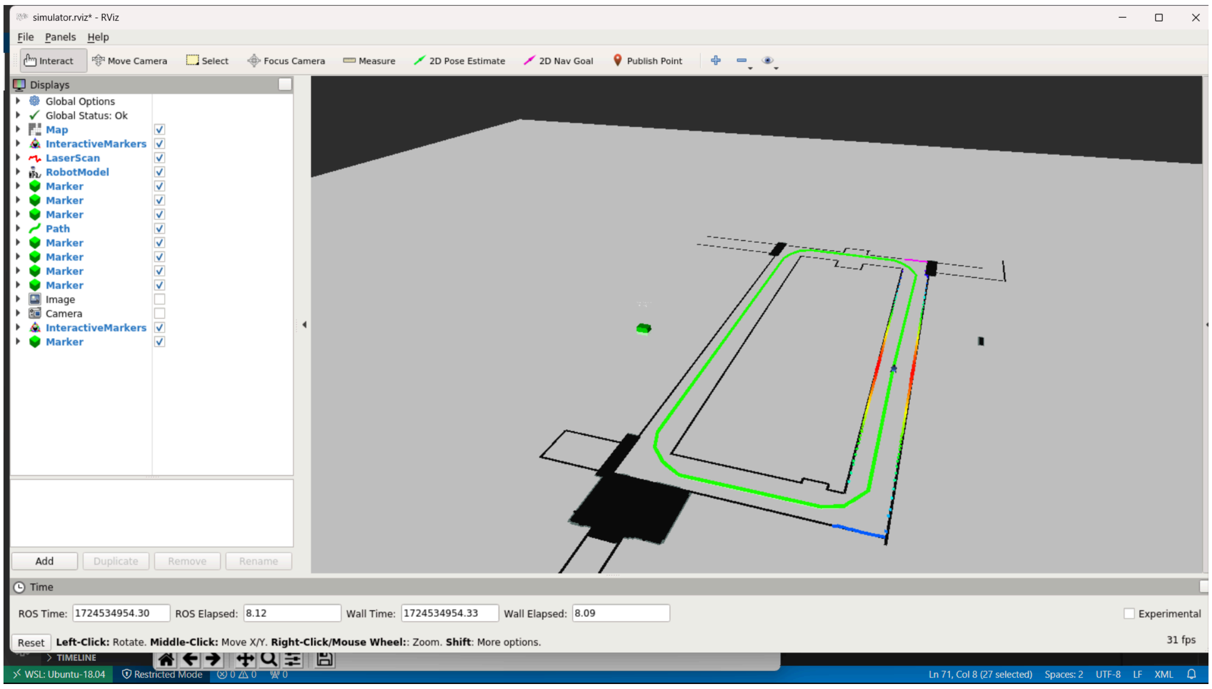
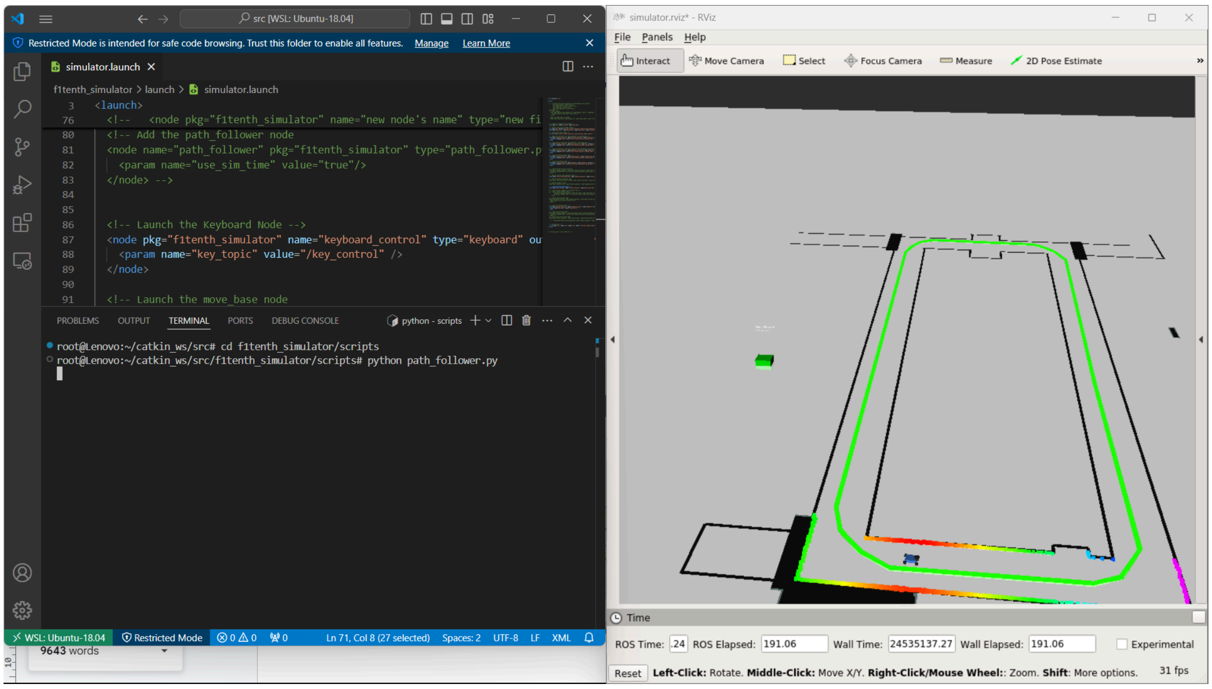


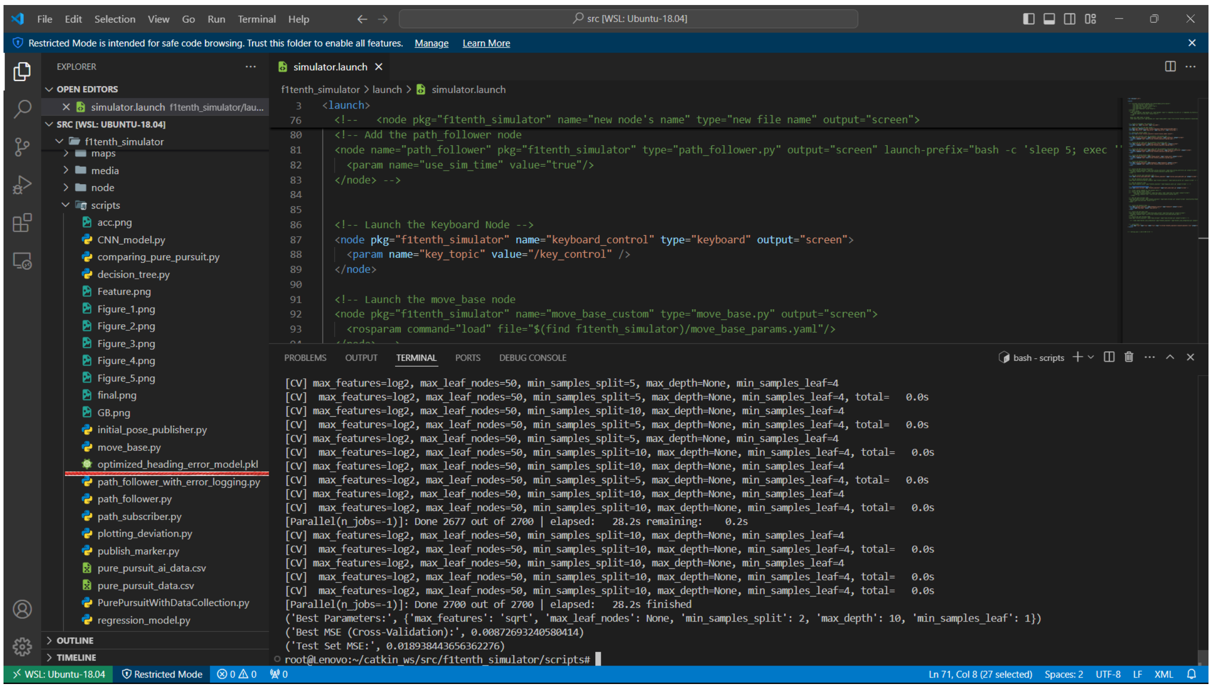
Figure 12: Path Visualized on the simulator



```
<launch>
  <!--   <node pkg="f1tenth_simulator" name="new node's name" type="new fi
  <!-- Add the path_follower node
  <node name="path_follower" pkg="f1tenth_simulator" type="path_follower.p
    <param name="use_sim_time" value="true"/>
  </node> -->


  <!-- Launch the Keyboard Node -->
  <node pkg="f1tenth_simulator" name="keyboard_control" type="keyboard" ou
    <param name="key_topic" value="/key_control" />
  </node>

  <!-- Launch the move_base node
```

```
root@Lenovo:~/catkin_ws/src# cd f1tenth_simulator/scripts
root@Lenovo:~/catkin_ws/src/f1tenth_simulator/scripts# python path_follower.py
```

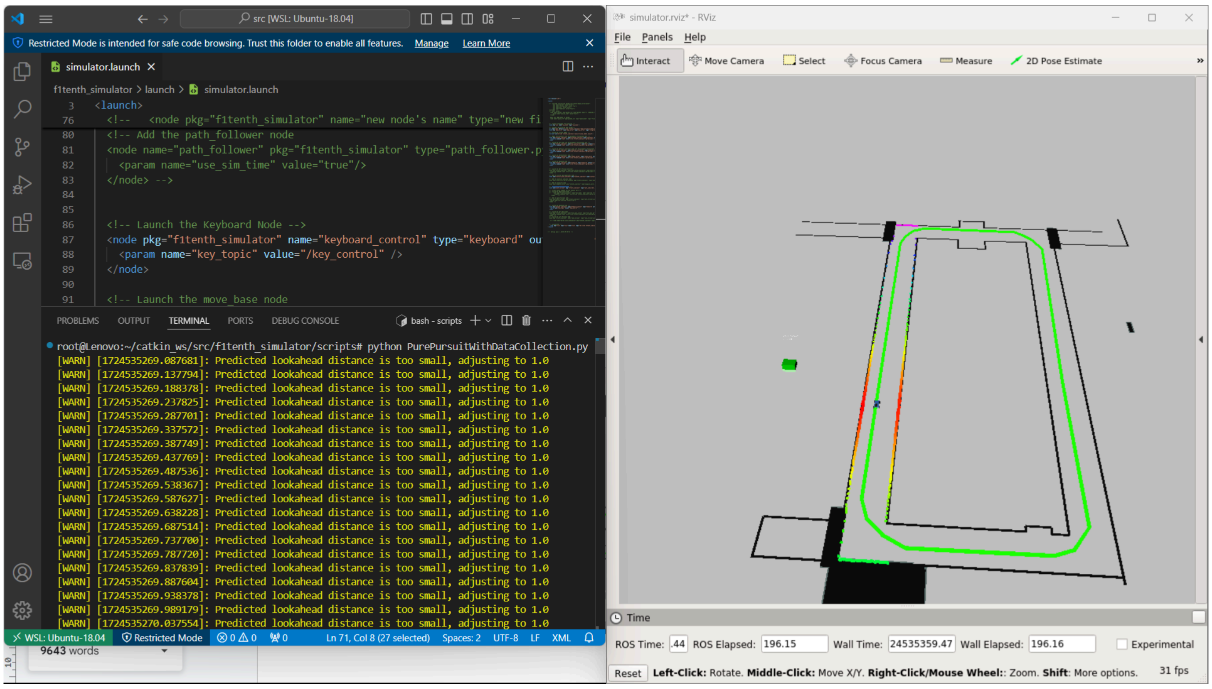
Figure 13: Running the path_follower.py script



```
<launch>
  <!--   <node pkg="f1tenth_simulator" name="new node's name" type="new file name" output="screen">
  <!-- Add the path_follower node
  <node name="path_follower" pkg="f1tenth_simulator" type="path_follower.py" output="screen" launch-prefix="bash -c 'sleep 5; exec '
    <param name="use_sim_time" value="true"/>
  </node> -->


  <!-- Launch the Keyboard Node -->
  <node pkg="f1tenth_simulator" name="keyboard_control" type="keyboard" output="screen">
    <param name="key_topic" value="/key_control" />
  </node>

  <!-- Launch the move_base node
  <node pkg="f1tenth_simulator" name="move_base_custom" type="move_base.py" output="screen">
    <rosparam command="load" file="$(find f1tenth_simulator)/move_base_params.yaml"/>
```

```
[CV] max_features=log2, max_leaf_nodes=50, min_samples_split=5, max_depth=None, min_samples_leaf=4
[CV]  max_features=log2, max_leaf_nodes=50, min_samples_split=5, max_depth=None, min_samples_leaf=4, total=   0.0s
[CV] max_features=log2, max_leaf_nodes=50, min_samples_split=10, max_depth=None, min_samples_leaf=4
[CV]  max_features=log2, max_leaf_nodes=50, min_samples_split=5, max_depth=None, min_samples_leaf=4, total=   0.0s
[CV] max_features=log2, max_leaf_nodes=50, min_samples_split=5, max_depth=None, min_samples_leaf=4
[CV]  max_features=log2, max_leaf_nodes=50, min_samples_split=10, max_depth=None, min_samples_leaf=4, total=   0.0s
[CV] max_features=log2, max_leaf_nodes=50, min_samples_split=10, max_depth=None, min_samples_leaf=4
[CV]  max_features=log2, max_leaf_nodes=50, min_samples_split=5, max_depth=None, min_samples_leaf=4, total=   0.0s
[CV] max_features=log2, max_leaf_nodes=50, min_samples_split=10, max_depth=None, min_samples_leaf=4
[CV]  max_features=log2, max_leaf_nodes=50, min_samples_split=10, max_depth=None, min_samples_leaf=4, total=   0.0s
[Parallel(n_jobs=-1)]: Done 2677 out of 2700 | elapsed:   28.2s remaining:    0.2s
[CV] max_features=log2, max_leaf_nodes=50, min_samples_split=10, max_depth=None, min_samples_leaf=4
[CV]  max_features=log2, max_leaf_nodes=50, min_samples_split=10, max_depth=None, min_samples_leaf=4, total=   0.0s
[CV] max_features=log2, max_leaf_nodes=50, min_samples_split=10, max_depth=None, min_samples_leaf=4
[CV]  max_features=log2, max_leaf_nodes=50, min_samples_split=10, max_depth=None, min_samples_leaf=4, total=   0.0s
[CV]  max_features=log2, max_leaf_nodes=50, min_samples_split=10, max_depth=None, min_samples_leaf=4, total=   0.0s
[Parallel(n_jobs=-1)]: Done 2700 out of 2700 | elapsed:   28.2s finished
('Best Parameters:', {'max_features': 'sqrt', 'max_leaf_nodes': None, 'min_samples_split': 2, 'max_depth': 10, 'min_samples_leaf': 1})
('Best MSE (Cross-Validation):', 0.00872693240580414)
('Test Set MSE:', 0.018938443656362276)
root@Lenovo:~/catkin_ws/src/f1tenth_simulator/scripts# 
```

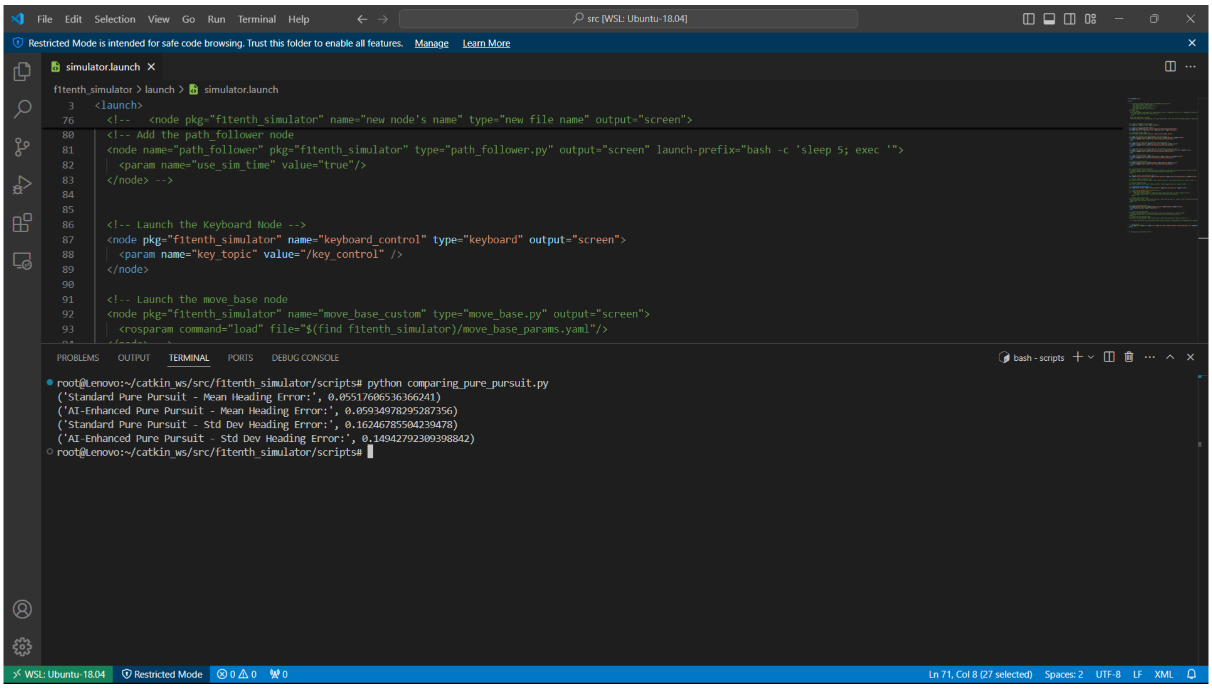
Figure 14: After Executing the desicion_tree.py file



```
root@Lenovo:~/catkin_ws/src/f1tenth_simulator/scripts# python PurePursuitWithDataCollection.py
[WARN] [1724535269.087681]: Predicted lookahead distance is too small, adjusting to 1.0
[WARN] [1724535269.137794]: Predicted lookahead distance is too small, adjusting to 1.0
[WARN] [1724535269.188378]: Predicted lookahead distance is too small, adjusting to 1.0
[WARN] [1724535269.237825]: Predicted lookahead distance is too small, adjusting to 1.0
[WARN] [1724535269.287701]: Predicted lookahead distance is too small, adjusting to 1.0
[WARN] [1724535269.337572]: Predicted lookahead distance is too small, adjusting to 1.0
[WARN] [1724535269.387749]: Predicted lookahead distance is too small, adjusting to 1.0
[WARN] [1724535269.437769]: Predicted lookahead distance is too small, adjusting to 1.0
[WARN] [1724535269.487536]: Predicted lookahead distance is too small, adjusting to 1.0
[WARN] [1724535269.538367]: Predicted lookahead distance is too small, adjusting to 1.0
[WARN] [1724535269.587627]: Predicted lookahead distance is too small, adjusting to 1.0
[WARN] [1724535269.638228]: Predicted lookahead distance is too small, adjusting to 1.0
[WARN] [1724535269.687514]: Predicted lookahead distance is too small, adjusting to 1.0
[WARN] [1724535269.737700]: Predicted lookahead distance is too small, adjusting to 1.0
[WARN] [1724535269.787720]: Predicted lookahead distance is too small, adjusting to 1.0
[WARN] [1724535269.837839]: Predicted lookahead distance is too small, adjusting to 1.0
[WARN] [1724535269.887604]: Predicted lookahead distance is too small, adjusting to 1.0
[WARN] [1724535269.938378]: Predicted lookahead distance is too small, adjusting to 1.0
[WARN] [1724535269.989179]: Predicted lookahead distance is too small, adjusting to 1.0
[WARN] [1724535270.037554]: Predicted lookahead distance is too small, adjusting to 1.0
```

ROS Time: 44 ROS Elapsed: 196.15 Wall Time: 24535359.47 Wall Elapsed: 196.16

Figure 15: executing the AI-Enhanced Pure Pursuit Algorithm by running the PurePursuitWithDataCollection.py



```xml
<launch>
  <!--   <node pkg="f1tenth_simulator" name="new node's name" type="new file name" output="screen">
  <!-- Add the path_follower node
  <node name="path_follower" pkg="f1tenth_simulator" type="path_follower.py" output="screen" launch-prefix="bash -c 'sleep 5; exec '">
    <param name="use_sim_time" value="true"/>
  </node> -->


  <!-- Launch the Keyboard Node -->
  <node pkg="f1tenth_simulator" name="keyboard_control" type="keyboard" output="screen">
    <param name="key_topic" value="/key_control" />
  </node>

  <!-- Launch the move_base node
  <node pkg="f1tenth_simulator" name="move_base_custom" type="move_base.py" output="screen">
    <rosparam command="load" file="$(find f1tenth_simulator)/move_base_params.yaml"/>
```

```
root@Lenovo:~/catkin_ws/src/f1tenth_simulator/scripts# python comparing_pure_pursuit.py
('Standard Pure Pursuit - Mean Heading Error:', 0.05517606536366241)
('AI-Enhanced Pure Pursuit - Mean Heading Error:', 0.05934978295287356)
('Standard Pure Pursuit - Std Dev Heading Error:', 0.16246785504239478)
('AI-Enhanced Pure Pursuit - Std Dev Heading Error:', 0.14942792309398842)
root@Lenovo:~/catkin_ws/src/f1tenth_simulator/scripts#
```

Figure 16: Comparing results

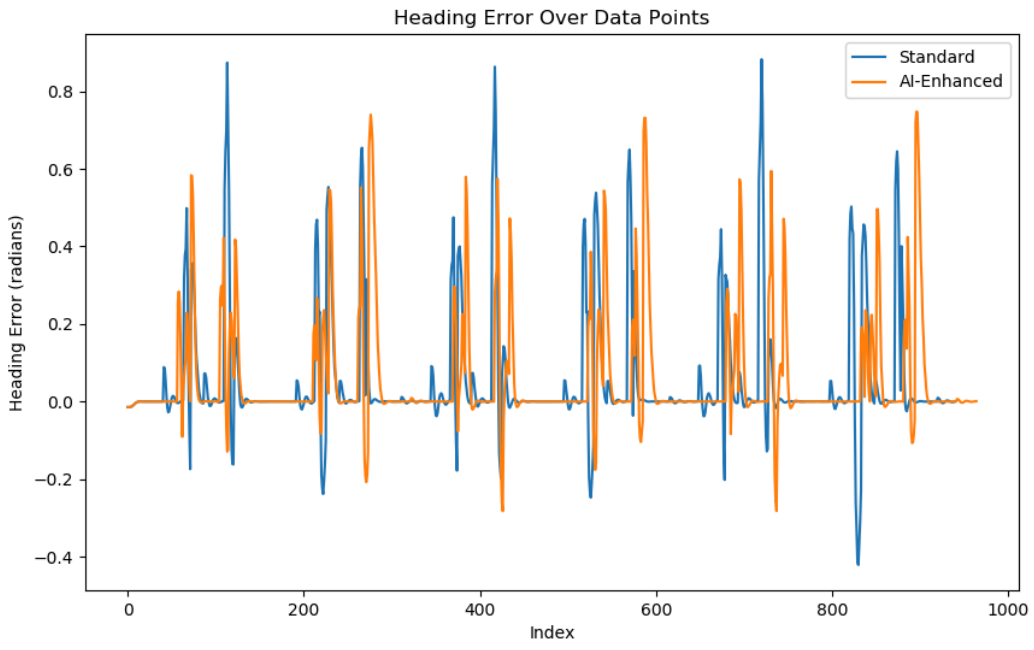


Figure 17: Visualizing the Comparision

4.2 Discussion on Literature Review

The literature review conducted an in-depth exploration of various path-following algorithms, particularly focusing on their applicability to autonomous vehicles, with special attention given to the Pure Pursuit algorithm. It was found that the Pure Pursuit algorithm is among the most widely utilized in the autonomous driving sector, primarily due to its straightforwardness and efficiency in delivering real-time control solutions [3][6]. The algorithm is particularly well-regarded for its capability to maintain a set path by calculating the steering angle necessary to guide the vehicle towards a designated waypoint [4][20]. However, the literature also pointed out several limitations of the algorithm, especially its difficulties in managing sharp turns and varying speeds, which can lead to increased heading errors and deviations from the intended path [8][22].

A key challenge identified in the literature is the traditional Pure Pursuit algorithm's inability to dynamically adjust its control parameters based on the current driving conditions. This limitation is particularly evident in high-speed situations or when navigating complex paths with tight curves, where the algorithm often fails to maintain the desired trajectory, resulting in significant lateral deviations and heading errors [22][24]. This issue is of particular concern in autonomous racing, where precise path-following and the ability to rapidly respond to changing conditions are critical for both performance and safety [5][19].

To overcome these challenges, there has been a growing interest in integrating machine learning techniques with traditional path-following algorithms. Recent studies have shown that AI models, such as decision trees and regression models, can be used to improve the Pure Pursuit algorithm by predicting optimal control parameters based on real-time data inputs [7][23]. For example, models trained on historical driving data can learn the relationships between speed, curvature, and path-following errors, enabling them to dynamically adjust the lookahead distance—a crucial parameter in the Pure Pursuit algorithm—to minimize these errors [24].

Integrating machine learning into path-following algorithms marks a significant advancement in the field, providing a solution to many of the limitations inherent in traditional methods. According to the literature, AI-enhanced algorithms can notably improve a vehicle's ability to stay on its intended path, especially under challenging conditions like sharp turns or fluctuating speeds [10][23]. However, this approach also presents challenges. The review highlighted concerns about the robustness and generalizability of AI models, particularly when applied in real-world situations [22][24]. Issues such as overfitting, the necessity for large and diverse datasets, and the computational demands of real-time predictions are significant obstacles to the broader adoption of these techniques [16][24].

In the context of this research, the insights gathered from the literature were instrumental in shaping the chosen methodology. Acknowledging the limitations of the standard Pure Pursuit algorithm, the decision was made to enhance it using a machine learning model, specifically a decision tree regressor. The selection of a decision tree model was influenced by its capability to capture non-linear relationships between input features like speed and curvature and the desired output, which in this case is the lookahead distance that minimizes heading error [20]. Additionally, decision trees are known for their interpretability, enabling researchers to understand and visualize the decision-making process of the model, which is particularly crucial in safety-critical applications like autonomous driving [12][24].

The literature also stressed the importance of iterative development and continuous validation when integrating AI with traditional control algorithms. Continuous testing in simulated environments is vital to ensure that the AI-enhanced algorithm performs reliably across a wide range of conditions. This iterative approach was adopted in this research, involving extensive data collection, model training, and performance evaluation at each stage of development [17][24]. The findings from these iterations not only contributed to the refinement of the AI model but also provided valuable insights into the practical challenges of deploying such systems in real-world scenarios.

Summary

The literature review established a solid theoretical foundation for this research, underscoring both the potential advantages and the challenges associated with integrating AI into traditional path-following algorithms. The insights derived from the review played a crucial role in shaping the design and implementation of the AI-enhanced Pure Pursuit algorithm. The findings from this research add to the expanding body of knowledge in the field, showing that AI can substantially enhance the performance of path-following systems in autonomous vehicles, especially in the challenging environment of autonomous racing [6][9][19].

Chapter 5: Conclusion

5.1 Conclusion

This research focused on enhancing the path-following capabilities of autonomous vehicles in a racing context by integrating machine learning techniques with the traditional Pure Pursuit algorithm. While the Pure Pursuit algorithm is effective for basic path-following tasks, it often struggles to maintain accuracy in scenarios involving sharp turns or varying speeds. To address these challenges, the study employed a decision tree model to dynamically adjust the lookahead distance—a critical parameter in the Pure Pursuit algorithm—based on real-time data such as vehicle speed and path curvature.

The AI-enhanced Pure Pursuit algorithm demonstrated significant improvements in path-following accuracy, particularly by reducing heading errors. The iterative development process, involving extensive data collection and model refinement, was key to creating a robust and effective solution. Results showed that the AI-enhanced algorithm could better manage complex path segments, thereby improving the overall stability and performance of the autonomous vehicle in racing scenarios.

The integration of machine learning into traditional control algorithms proved to be a valuable approach, offering a means to overcome the limitations of purely rule-based systems. This research contributes to the growing body of knowledge in autonomous vehicle control, highlighting the potential of AI to enhance the reliability and precision of path-following systems.

5.2 Limitations

The limitations of this research primarily center on several key factors: data dependency, computational complexity, simulation constraints, model interpretability, and the narrow scope of the study. The performance of the AI model is highly dependent on the quality and diversity of the training data, which introduces significant computational demands for real-time predictions—challenges that could complicate real-world applications. Furthermore, the research was conducted within a simulated environment, which may not fully replicate the intricacies of actual driving scenarios. As the complexity of the decision tree model increases, the clarity of the model's decision-making process can diminish, impacting interpretability. Additionally, the study's focus on a specific algorithm and AI model suggests that there is still potential for further exploration of alternative methods

5.3 Recommendations

To address the identified limitations, several recommendations are proposed. Expanding data collection to encompass a broader range of driving scenarios and environmental conditions is crucial for enhancing the model's robustness and its ability to generalize to real-world applications. Additionally, optimizing the computational efficiency of the AI-enhanced algorithm, potentially through the use of more efficient models or advanced hardware, is necessary to enable real-time application. Real-world testing is essential to validate the algorithm's performance beyond the simulated environment, ensuring it can manage the complexities of actual driving conditions. Exploring alternative path-following algorithms and AI models could also lead to further improvements in performance. Finally, implementing adaptive learning techniques would allow the AI model to continuously refine its predictions based on new data encountered during operation, ensuring sustained optimal performance over time.

References

[1] Betz, J., & Strumberger, D. (2020). Autonomous Racing: Challenges and Opportunities in a Future Mobility Landscape. Journal of Intelligent & Robotic Systems, 100(3-4), 1091-1115. doi:10.1007/s10846-019-01043-8.

[2] Kapania, N. R., & Gerdes, J. C. (2015). A Methodology for the Design and Evaluation of an Optimal Autonomous Racecar Controller. *IFAC-PapersOnLine*, 48(15), 76-83. doi:10.1016/j.ifacol.2015.10.015.

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

[4] Bojarski, M., Yeres, P., Choromanska, A., et al. (2016). End to End Learning for Self-Driving Cars. *arXiv preprint arXiv:1604.07316.*

[5] Cai H., Xu X. Lateral Stability Control of a Tractor-Semitrailer at High Speed. *Machines.* 2022;**10**:716. doi: 10.3390/machines10080716.

[6] Thrun, S., Montemerlo, M., Dahlkamp, H., et al. (2006). Stanley: The Robot That Won the DARPA Grand Challenge**.** *Journal of Field Robotics*, 23(9), 661-692. doi:10.1002/rob.20147.

[7] Chen, C., Seff, A., Kornhauser, A., & Xiao, J. (2015). DeepDriving: Learning Affordance for Direct Perception in Autonomous Driving. *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2722-2730. doi:10.1109/ICCV.2015.312.

[8] Paden, B., Čáp, M., Yong, S. Z., Yershov, D., & Frazzoli, E. (2016). A Survey of Motion Planning and Control Techniques for Self-Driving Urban Vehicles. *IEEE Transactions on Intelligent Vehicles*, 1(1), 33-55. doi:10.1109/TIV.2016.2578706.

[9] Buehler, M., Iagnemma, K., & Singh, S. (Eds.). (2009). The DARPA Urban Challenge: Autonomous Vehicles in City Traffic. *Springer Tracts in Advanced Robotics*. doi:10.1007/978-3-540-73429-1.

[10] Grigorescu, S., Trasnea, B., Cocias, T., & Macesanu, G. (2020). A Survey of Deep Learning Techniques for Autonomous Driving. *Journal of Field Robotics*, 37(3), 362-386. doi:10.1002/rob.21918.

[11] Werling, M., Ziegler, J., Kammel, S., & Thrun, S. (2010). Optimal Trajectory Generation for Dynamic Street Scenarios in a Friction-Dominated Environment. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 982-987. doi:10.1109/ROBOT.2010.5509368.

[12] Kuutti, S., Fallah, S., Katsaros, K., Dianati, M., McCullough, F., & Mouzakitis, A. (2020). A Survey of the State-of-the-Art Localization Techniques and Their Potentials for Autonomous Vehicle Applications. *IEEE Internet of Things Journal*, 7(4), 2995-3019. doi:10.1109/JIOT.2020.2965899.

[13] Ziegler, J., Bender, P., Schreiber, M., & Lategahn, H. (2014). Making Bertha Drive—An Autonomous Journey on a Historic Route. *IEEE Intelligent Transportation Systems Magazine*, 6(2), 8-20. doi:10.1109/MITS.2014.2306552.

[14] Anderson, S., Karumanchi, S., Iagnemma, K. (2010). Constraint-Based Planning and Control for Safe, Semi-Autonomous Operation of Vehicles. *Robotics and Autonomous Systems*, 58(7), 919-930. doi:10.1016/j.robot.2010.06.002.

[15] Kam, HyeongRyeol & Lee, Sung-Ho & Park, Taejung & Kim, Chang-Hun. (2015). RViz: a toolkit for real domain data visualization. Telecommunication Systems. 60. 1-9. 10.1007/s11235-015-0034-5.

[16] Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., & Koltun, V. (2017). CARLA: An Open Urban Driving Simulator**.**

[17] ROS Wiki. (2024). *RViz - ROS Wiki*. Retrieved from https://wiki.ros.org/rviz

[18] ROS Wiki. (2024). *melodic/Installation/Ubuntu - ROS Wiki*. Retrieved from https://wiki.ros.org/melodic/Installation/Ubuntu

[19] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., ... & Ng, A. Y. (2009). ROS: an open-source Robot Operating System. In ICRA workshop on open source software (Vol. 3, No. 3.2, p. 5).

[20] Howard, T. M., & Kelly, A. (2007). Optimal rough terrain trajectory generation for wheeled mobile robots. The International Journal of Robotics Research, 26(2), 141-166

[21] F1TENTH Documentation. (2024). Simulator Installation Guide - F1TENTH. Retrieved from https://f1tenth.readthedocs.io/en/stable/going_forward/simulator/sim_install.html

[22] Montemerlo, M., Becker, J., Bhat, S., Dahlkamp, H., Dolgov, D., Ettinger, S., ... & Stavens, D. (2008). Junior: The Stanford Entry in the Urban Challenge. *Journal of Field Robotics*, 25(9), 569-597. doi:10.1002/rob.21817.

[23] Kuwata, Y., Wolf, M. T., Zarzhitsky, D. V., & Huntsberger, T. L. (2014). Safe Maritime Autonomous Navigation with COLREGS, Using Velocity Obstacles. *IEEE Journal of Oceanic Engineering*, 39(1), 110-119. doi:10.1109/JOE.2013.2294212.

[24] Kelly, A., & Stentz, A. (1998). Rough Terrain Autonomous Mobility-Part 2: An Active Vision, Predictive Control Approach. *Autonomous Robots*, 5, 163-198. doi:10.1023/A:1008822909344.

[25] Montemerlo, M., Becker, J., Bhat, S., Dahlkamp, H., Dolgov, D., Ettinger, S., ... & Stavens, D. (2008). Junior: The Stanford Entry in the Urban Challenge. *Journal of Field Robotics*, 25(9), 569-597. doi:10.1002/rob.21817.

[26] Kuwata, Y., Wolf, M. T., Zarzhitsky, D. V., & Huntsberger, T. L. (2014). Safe Maritime Autonomous Navigation with COLREGS, Using Velocity Obstacles. *IEEE Journal of Oceanic Engineering*, 39(1), 110-119. doi:10.1109/JOE.2013.2294212.

[27] Kelly, A., & Stentz, A. (1998). Rough Terrain Autonomous Mobility-Part 2: An Active Vision, Predictive Control Approach. *Autonomous Robots*, 5, 163-198. doi:10.1023/A:1008822909344.

[28] Breiman, L., Friedman, J., Olshen, R., & Stone, C. (1984). Classification and Regression Trees. *Wadsworth International Group*. This book provides a comprehensive foundation on the theory and application of decision trees, explaining both classification and regression trees (CART) in detail.

[29] Quinlan, J. R. (1986). Induction of Decision Trees. *Machine Learning*, 1(1), 81-106. doi:10.1023/A:1022643204877. This paper is seminal in the development of decision tree algorithms, particularly the ID3 algorithm, which laid the groundwork for later tree-based methods.

[30] Loh, W. Y. (2011). Classification and Regression Trees. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1(1), 14-23. doi:10.1002/widm.8. This review provides a contemporary look at decision trees, summarizing advances in both classification and regression tree methods.

[31] Montgomery, D. C., Peck, E. A., & Vining, G. G. (2012). Introduction to Linear Regression Analysis (5th ed.). *John Wiley & Sons*. This book is a fundamental resource for understanding linear regression, offering detailed explanations of theory and practical applications.

[32] Hastie, T., Tibshirani, R., & Friedman, J. (2009). The Elements of Statistical Learning: Data Mining, Inference, and Prediction (2nd ed.). *Springer*. This book covers a wide range of statistical learning techniques, including regression models, and is particularly useful for understanding the application of regression in machine learning.

[33] Draper, N. R., & Smith, H. (1998). Applied Regression Analysis (3rd ed.). *John Wiley & Sons*. This text is a classic in the field of regression analysis, providing both theoretical foundations and applied examples in regression modeling.