Singapore Institute of Technology – DigiPen Institute of Technology Singapore
Joint Degree in Computer Science in Real-Time Interactive Simulation
Joint Degree in Computer Science in Interactive Media And Game Development

CSD4400 Capstone Project

Please complete the following form and attach it to the Capstone Report submitted.

Capstone Period: **02 APR 2024** **to** **11 APR 2025**
Assessment Trimester: **Final Trimester**
Project Type: **Industry**

Work Supervisor Details & Declaration:

Name: Harish Loganathan
Designation: Senior Software Engineer
Department: Software Infrastructure
Email Address: harish@ventitechnologies.com

Academic Supervisor Details:

Name: Elie Hosry
Designation: Principal Lecturer
Email Address: ehosry@digipen.edu

Student Particulars & Declaration:

Name of Student: Huang Wei Jhin
SIT Student ID: 2101192
DigiPen Student ID: 390009621
SIT Email Address: 2101192@sit.singaporetech.edu.sg
DigiPen Email Address: h.weijhin@digipen.edu

I hereby acknowledge that I have engaged and discussed with my **Academic Supervisor** and **Work Supervisor** on the contents of this Capstone Report (Capstone Final Report) and have sought approval to release the report to the Singapore Institute of Technology and the DigiPen Institute of Technology Singapore.

Signature 

Date: <20 Mar 2024>

END OF FORM

Singapore Institute of Technology – DigiPen Institute of Technology Singapore
Joint Degree in Computer Science in Real-Time Interactive Simulation
Joint Degree in Computer Science in Interactive Media And Game Development

Capstone Report (Capstone Final Report)
“Vehicle Dynamics Simulation”

Huang Wei Jhin
SIT Student ID: 2101192
DigiPen Student ID: 390009621

Academic Supervisor: Elie Hosry

Submitted as part of the requirement for CSD4400 Capstone Project

Table of Contents

List of Figures

List of Tables

Acknowledgments

I would like to express my sincere gratitude to everyone who supported and guided me throughout the duration of this capstone project. Without their expertise and encouragement, I would not have been able to achieve the level of accuracy attained in the development of the vehicle dynamics models.

First and foremost, I would like to thank my work supervisor, Harish Loganathan, for his continuous guidance, valuable feedback and encouragement. His expertise was crucial in shaping the direction of this project and helping me overcome many technical challenges.

I am grateful to my team lead, Viacheslav Tuzhilkov, for taking the time to review my pull requests and offering insightful advice on coding best practices and software architecture.

I am greatly appreciative of Rony Hidayatullah, Control Research Scientist, whose insights into vehicle modeling and the underlying mathematical formulations were essential in laying the foundation for this work. His guidance was also instrumental in helping me tune model parameters, ensuring the accuracy and stability of the vehicle dynamics simulations.

My sincere thanks go to Muhammad Syaiful, the Test Engineer, who conducted vehicle testing at the port and provided the essential data that enabled the calibration and validation of the vehicle dynamics models.

I am thankful to Venkataraman Natarajan, whose prior work on optimizing the VTD dynamics model offered valuable insights into its limitations and identified key areas for improvement, which helped shape the direction of this project.

I would also like to extend my gratitude to Ling Feng, the Data Engineer, for setting up the Network-Attached Storage system used to manage and provide access to the large datasets required for model development and evaluation.

Finally, I would like to thank my academic supervisor, Elie Hosry, for his guidance, support and thoughtful feedback throughout the course of this capstone project.

I also wish to express my appreciation to Venti Technologies for providing the opportunity to undertake this project as part of my internship. The experience of working on a real-world problem significantly deepened my understanding of autonomous vehicle systems and served as the foundation for this work.

Abstract

This capstone project presents the design and development of a custom vehicle dynamics model tailored to the simulation needs of autonomous vehicles (AVs) at Venti Technologies. The motivation stems from the limitations of third-party simulation tools, such as Virtual Test Drive (VTD), which often lack the fidelity required to accurately represent the real-world behavior of Venti's diesel-powered prime movers. To address this, the project is structured in two phases: a Simple Vehicle Dynamics Model using throttle, brake, and steering subsystems; and an Advanced Vehicle Dynamics Model incorporating engine torque generation, transmission dynamics, chassis motion, and gear-shifting logic.

A modular, testable architecture was developed in C++, with each subsystem independently validated against real-world data collected during on-road tests. Supporting tools were also created, including a ROS-based automation script for playback and data logging and a GUI for plotting model outputs and computing fitness metrics. The models were evaluated by comparing their outputs with actual vehicle data using quantitative metrics such as velocity, heading, and position error.

Results show that while the simple model achieves good accuracy with tuned parameters, the advanced model provides more consistent performance across unseen test cases, making it more robust and scalable. This project lays the groundwork for more accurate, efficient, and reliable AV simulation, with potential for future enhancements such as terrain modeling and expanded vehicle type support.

1. Introduction

The development of precise and robust vehicle dynamics models is critical for advancing autonomous vehicle (AV) technologies. Simulation and testing play pivotal roles in ensuring the reliability, safety and performance of AV systems. However, many existing third-party simulation tools, such as Virtual Test Drive (VTD), provide generalized vehicle dynamics that lack the fidelity required to accurately represent the unique behaviors of Venti's AVs. This limitation poses significant challenges, as realistic simulation outcomes are essential for effectively validating autonomous driving functionalities.

This capstone project aims to address these limitations by designing an in-house vehicle dynamics model tailored to the specific requirements of Venti Technologies' AVs. By doing so, this project seeks to enhance simulation accuracy and provide a reliable foundation for testing advanced autonomous functionalities. The project is driven by the motivation to close the gap between generalized simulations and real-world AV behavior, enabling the development of safer and more efficient autonomous systems.

The scope of this project includes two phases: developing a simple model with basic throttle, brake and steering models and then advancing it to a comprehensive model that incorporates detailed components such as engine dynamics, transmission systems and gear shift logic. While ambitious, the project faces some limitations such as compatibility with unique vehicle types and terrain assumptions. The current vehicle dynamics model is specifically designed for a diesel-powered prime mover with front-wheel steering and does not support other unique configurations such as rear-wheel steering systems or electric drivetrains. While this limits adaptability, the model remains sufficient for Venti Technologies' existing customers' vehicle type. Additionally, the model assumes a flat road surface, such as in the port, excluding considerations for elevation changes or slopes. Lastly, the model's accuracy depends on high-quality real-world data and inconsistencies or gaps may hinder fidelity.

This report outlines the problem, objectives, methodology, deliverables and timeline to address these challenges and achieve a high-fidelity vehicle dynamics model suitable for Venti Technologies' simulation and validation workflows.

1.1 Problem Formulation

1.1.1 Problem Statement

Autonomous vehicle (AV) testing relies heavily on simulation tools to replicate real-world driving scenarios and validate system performance. However, existing third-party simulation software, such as Virtual Test Drive (VTD), often provides generalized vehicle dynamics that fail to capture the specific behavioral characteristics of AVs. This discrepancy between simulated and real-world performance creates challenges in accurately validating and refining AV systems. The lack of precision in these simulations undermines the reliability of testing outcomes, making it essential to develop a detailed in-house vehicle dynamics model tailored to the specific requirements of Venti Technologies' AVs, as shown in figure 1.

1.1.2 Objectives and Expectations

This project aims to address the limitations of existing vehicle dynamics models by developing a robust and scalable in-house solution. The primary objective is to design a model that accurately replicates the dynamic behavior of Venti Technologies' AVs under various driving scenarios. The model must enable realistic simulations that closely align with real-world vehicle performance, thereby reducing the reliance on third-party tools and enhancing the fidelity of AV validation processes. The developed vehicle dynamics model is expected to operate efficiently in real-time to support AV system simulations. It must also be scalable and configurable to accommodate a variety of primer mover models. Seamless integration with Venti's AV system and existing simulation frameworks, such as VTD, is a crucial requirement to ensure compatibility and usability across different testing environments.

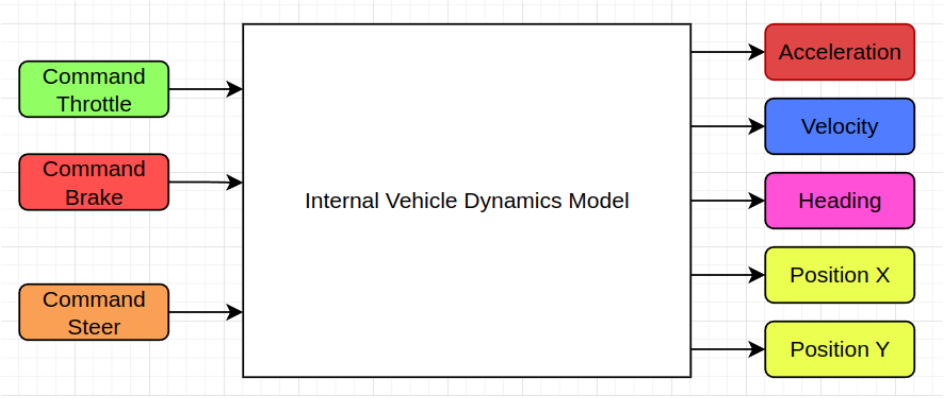


Figure 1: High level Diagram of Internal Vehicle Dynamics

1.1.3 Literature Review and Gaps

The need for precise vehicle dynamics modeling has been widely discussed in research, particularly in the context of (AV) testing. Tools like VTD are often paired with specialized software, such as veDYNA, to replace VTD's internal dynamics models, enhancing fidelity for vehicle behavior simulations (Chen, 2013). While VTD effectively creates virtual environments and supports real-time hardware integration, its generalized dynamics models lack the accuracy required to replicate intricate behaviors like gear-shifting dynamics or throttle response. This has led to co-simulation approaches using tools like Matlab/Simulink to improve simulation outcomes. Despite these advancements, gaps persist in developing unified, scalable, real-time vehicle dynamics models tailored for diverse prime mover configurations (Chen, 2013).

1.1.4 Sub-Problems

The development of an accurate and robust vehicle dynamics model for Venti Technologies' autonomous vehicles involves addressing several key sub-problems. These sub-problems reflect the challenges associated with creating a model that balances fidelity, real-time performance, scalability and integration with existing systems.

The first sub-problem is the development of a simple vehicle dynamics model. This initial phase focuses on creating a basic framework that incorporates throttle, brake and steering inputs. The simplicity of this model ensures that it can serve as a foundation for early-stage testing and benchmarking. Its design allows for a quick rollout to production, minimizing delays and enabling immediate testing within Venti Technologies' simulation environment. While straightforward in functionality, this model is critical for establishing a baseline upon which more complex dynamics can be built.

The second sub-problem involves the development of a multi-component vehicle dynamics model. Expanding upon the initial framework, this phase seeks to accurately model complex vehicle components including engine dynamics, transmission systems, gear shift behavior and refined steering control. Each component must be integrated into a cohesive system that reflects realistic behavior in various driving scenarios. This sub-problem is particularly challenging due to the interdependencies between components, which must be carefully calibrated to replicate real-world vehicle performance.

The third sub-problem is optimizing the model for real-time performance. As vehicle dynamics become more complex with the integration of advanced components, maintaining computational efficiency becomes increasingly difficult. This sub-problem addresses the need to balance the fidelity of the model with the constraints of real-time simulation. Ensuring that complex dynamics can be computed efficiently without compromising simulation speed or accuracy is essential for the practical application of the model in autonomous vehicle development.

The fourth sub-problem focuses on maintaining scalability and flexibility. Given the diverse range of customer vehicles and use cases that Venti Technologies supports, the model must be designed to accommodate various primer mover configurations. This requires a parameter-modifiable architecture that enables easy adaptation to different prime mover models and operating conditions. Scalability ensures that the model remains relevant and effective across a wide spectrum of applications, from testing individual subsystems to simulating complete vehicle behavior.

By addressing these sub-problems, the project aims to deliver a comprehensive vehicle dynamics model that bridges the gap between generalized simulation tools and the specific needs of autonomous vehicle systems. Each sub-problem plays a crucial role in achieving the overall goal of enhancing the fidelity, usability and scalability of Venti Technologies' simulation framework.

1.2 Project Introduction

1.2.1 Project Objectives

The primary objective of this project is to develop a robust and precise vehicle dynamics model that significantly enhances simulation fidelity and reliability for Venti Technologies' autonomous vehicles. To accomplish this, the project focuses on four key areas: developing a simple vehicle dynamics model, constructing a comprehensive model, optimizing the model for real-time performance and maintaining scalability and flexibility.

The first objective is to create a basic vehicle dynamics model incorporating throttle, brake and steering inputs. This foundational model will serve as the starting point for simulation and testing, enabling rapid deployment and immediate integration into Venti Technologies' existing simulation framework. Despite its simplicity, this model is designed to achieve at least 70% fidelity with the real vehicle, specifically in key metrics such as velocity, acceleration, heading and position. The basic model provides a benchmark for subsequent advancements and facilitates early-stage validation as well as a quick deployment.

Building on the basic model, the second objective is to develop a comprehensive vehicle dynamics model. This advanced iteration integrates essential components, including engine dynamics, transmission systems, chassis models and gear shift logic. By incorporating these elements, the comprehensive model aims to achieve a 90% correlation with real-world vehicle data, closely replicating the performance and behavior of Venti Technologies' autonomous vehicles. Additionally, the model must maintain real-time performance, ensuring that complex dynamics are simulated efficiently without compromising accuracy. This capability is crucial for enabling effective testing and validation in diverse driving scenarios.

The final objective focuses on optimizing scalability and flexibility. The vehicle dynamics model must accommodate the diverse requirements of various vehicle types and driving scenarios, making scalability a critical design consideration. By supporting parameter modifications, the model will adapt to different vehicle configurations and operational conditions, ensuring versatility and long-term relevance. This adaptability allows the model to address both current and future needs, serving as a valuable tool for a range of use cases.

Through these objectives, the project aims to bridge the gap between generalized simulation tools and the specific needs of Venti Technologies, delivering a vehicle dynamics model that is accurate, versatile and efficient for testing and validation purposes.

1.2.2 Project Specifications

The project specifications are designed to ensure the vehicle dynamics model achieves the necessary levels of accuracy, performance and adaptability to meet the objectives of the project. These specifications provide a clear framework for evaluating the model's effectiveness.

The first specification focuses on the fidelity of the simple vehicle dynamics model. This model serves as a foundational framework, enabling accurate simulation of basic vehicle behavior. It is expected to achieve at least 70% fidelity with the real vehicle in critical metrics such as velocity, acceleration, heading and position. By meeting this benchmark, the simple model provides a reliable starting point for testing and validation while ensuring early-stage integration into Venti Technologies' simulation environment.

Building on this foundation, the comprehensive vehicle dynamics model must achieve a higher level of fidelity. Incorporating advanced components such as engine dynamics, transmission systems, chassis models and gear shift logic, the comprehensive model is expected to demonstrate a 90% correlation with real-world vehicle data. This specification ensures that the model can accurately replicate the complex interactions between subsystems, enabling realistic simulations of Venti Technologies' autonomous vehicles in diverse driving scenarios.

In addition to accuracy, the model must deliver high performance in real-time environments. To support seamless simulation workflows, the model must process each simulation frame in under 5 milliseconds. This specification is essential for maintaining the efficiency of autonomous vehicle systems, particularly in scenarios that demand the integration of multiple subsystems and dynamic vehicle responses. Meeting this benchmark ensures that the model operates without delays, supporting timely and reliable simulation outcomes.

Flexibility and scalability are also critical to the success of the project. The model must be designed with parameter modifiability, enabling it to simulate at least three distinct vehicles. By allowing adjustments to parameters such as size, weight and powertrain characteristics, the model ensures adaptability to diverse configurations and operational conditions. This capability is essential for addressing both the current and future needs of Venti Technologies, making the model a versatile tool for testing and validation.

These specifications collectively ensure that the vehicle dynamics model balances fidelity, performance and adaptability, addressing the unique requirements of Venti Technologies' autonomous vehicle development. They establish measurable criteria for evaluating the success of the project, ensuring the model meets its intended goals.

1.3 Project Deliverables & Timeline

The deliverables for this project are categorized into four main components: a simple vehicle dynamics model, a comprehensive vehicle dynamics model, support scripts and documentation. Table 1 and Table 2 outline the project timeline across two semesters.

The first deliverable is the simple vehicle dynamics model, which serves as the foundational framework for the project. This model is implemented as a C++ library and focuses on simulating basic vehicle behaviors, including throttle, brake and steering inputs. The model is accompanied by unit tests to validate its functionality, ensuring code reliability and stability. Additionally, it is seamlessly integrated with Venti Technologies' AV system to enable immediate deployment and testing. This deliverable supports rapid testing and provides a baseline for the development of the more advanced vehicle dynamics model.

The second deliverable is the comprehensive vehicle dynamics model, which builds upon the simple model by integrating critical components such as engine dynamics, transmission systems, chassis models and gear shift logic. This advanced model is designed to achieve enhanced fidelity when compared to real-world vehicle data, enabling highly accurate simulations and validations. Furthermore, it is optimized for real-time execution, ensuring efficient and reliable testing of autonomous vehicle systems in diverse driving scenarios. This deliverable represents a significant advancement in the simulation capabilities of Venti Technologies.

To support the effective use and evaluation of the vehicle dynamics models, the project also includes support scripts. These scripts consist of Python tools for data analysis, model comparison and visualization, enabling detailed evaluation and insights into simulation results. Additionally, Bash scripts are developed to automate tasks such as ROS bag processing, reducing manual effort and ensuring consistent handling of recorded data. Lastly, parameter estimation scripts were also developed. These scripts streamline workflows and enhance the overall efficiency of the development and testing processes.

The final deliverable is documentation, which ensures that the models and tools are accessible and easy to use for all stakeholders. The documentation includes detailed technical explanations of the vehicle dynamics library and support scripts, providing clear guidance for developers and testers. In addition, user guides are created to offer comprehensive instructions for deploying and testing the models, ensuring that the deliverables can be effectively utilized and maintained.

Table 1: Planned timeline semester 1

Week	Tasks
Week 1	Initial idea discussion and research
Week 2	Development of Throttle model
Week 3	Development of Brake model
Week 4	Development of Velocity model (combine throttle and brake)
Week 5	Development of Steering model
Week 6	Combining of Steering and Velocity models
Week 7	Conduct round 1 of vehicle test to gather data
Week 8	Comparing of vehicle dynamics model and actual vehicle
Week 9	Optimising vehicle dynamics model
Week 10	Integration of simple vehicle dynamics with AV-system and VTD
Week 11	Deployment of simple vehicle dynamics
Week 12	Documentation for simple vehicle dynamics
Week 13	Research and discussion of advance vehicle dynamics
Week 14	Development of Chassis model

Table 2: Planned timeline semester 2

Week	Tasks
Week 1	Development of Steering model
Week 2	Development of Transmission model
Week 3	Combining of Steering, Chassis and Transmission models
Week 4	Development of Gear model
Week 5	Development of Engine model
Week 6	Combining of Engine and Gear models
Week 7	Integrating all the models in to advance vehicle dynamics
Week 8	Conduct round 2 of vehicle test to gather data
Week 9	Comparing of vehicle dynamics model and actual vehicle
Week 10	Optimising advance vehicle dynamics model
Week 11	Integration of advance vehicle dynamics with AV-system and VTD
Week 12	Deployment of advanced vehicle dynamics
Week 13	Documentation for advance vehicle dynamics
Week 14	-buffer

2. Literature Review

2.1 Definition and Scope

Vehicle dynamics for simulations involves modeling and replicating the physical behavior of vehicles under various operating conditions. These models aim to predict responses to inputs such as throttle, brake and steering, providing insights into a vehicle's acceleration, velocity and trajectory. In the context of autonomous vehicle (AV) development, vehicle dynamics modeling is essential for validating algorithms, testing control systems and evaluating performance without the risks and costs of real-world testing.

The scope of this review focuses on simulation-based vehicle dynamics modeling techniques used for AV systems. Specifically, it investigates subsystems such as throttle and brake dynamics, steering models, engine and transmission behavior and chassis motion. Excluded from this review are studies focused solely on environmental modeling (e.g., terrain or weather effects) and non-vehicular dynamics (e.g., pedestrian behavior or traffic flow).

2.2 General Findings in Literature

Many existing simulation tools, such as Virtual Test Drive (VTD), utilize generalized vehicle dynamics models designed for a wide range of vehicles. Studies highlight that these models prioritize versatility over accuracy, leading to significant limitations when applied to specific vehicles. (Chen, 2013) For example, generic throttle and brake models often fail to capture nonlinearities in acceleration and deceleration, resulting in discrepancies between simulated and real-world data. Similarly, steering models may overlook imperfections such as backlash or bias, which are critical for realistic AV testing.

2.3 Subsystem-Level Modeling

Research has advanced significantly in subsystem-specific modeling. Throttle and brake dynamics are often represented using Linear Parameter Varying (LPV) models or proportional-integral-derivative (PID) controllers. Steering models typically rely on kinematic or dynamic bicycle frameworks. Engine and transmission models incorporate torque generation and gear-shifting logic, simulating power delivery under varying loads. Chassis models include lateral and longitudinal dynamics, tire forces and aerodynamic drag. (Zhang, 2024) However, most studies focus on individual components, leaving a gap in integrating these subsystems into a cohesive framework.

2.3.1 Throttle and Brake Models

MathWorks demonstrated the effectiveness of LPV models in simulating throttle dynamics, noting their ability to adjust parameters dynamically based on input magnitude. (MathWorks, 2024) However, other studies acknowledged inaccuracies in low-speed scenarios due to a lack of frictional resistance modeling. (Zhang, 2014)

2.3.2 Steering Models

Research by Zhang highlighted the challenges associated with kinematic bicycle models in autonomous racing. These models, while emphasizing simplicity and computational efficiency suitable for real-time applications, struggle to predict directional changes caused by tire slips accurately. This limitation creates a gap between theoretical predictions and real-world outcomes. Additionally, while integrating tire dynamics into the kinematic model can address such issues, it significantly increases computational complexity. (Zhang, 2024)

2.3.3 Engine and Transmission Dynamics

Engine and transmission dynamics play a critical role in accurately modeling vehicle behavior, particularly in high-fidelity simulations. The interaction between the engine's output torque and the transmission ratio determines the traction force exerted on the tires, as highlighted in studies that incorporate power and torque constraints into vehicle models (Zhang, 2024). Additionally, internal combustion engine (ICE)-powered vehicles utilize gear-dependent transmission ratios to adjust the torque delivery based on vehicle velocity, with lookup tables often employed for dynamic correlation (Zhang, 2024). This comprehensive integration of engine and transmission dynamics enables the simulation of real-world behaviors such as gear transitions, essential for validating autonomous vehicle systems.

2.3.4 Chassis and Motion Dynamics

Chassis and motion dynamics are fundamental to accurately simulating vehicle behavior. Comprehensive chassis models integrate lateral and longitudinal dynamics, tire forces, road resistance and aerodynamic drag to replicate real-world behavior (Zhang, 2024). These models employ advanced equations of motion to simulate yaw motion, pitch and roll, allowing for realistic responses to steering, braking and road conditions. Tire forces are modeled using empirical approaches like the Pacejka model which captures nonlinear slip-force relationships critical for handling and stability. (Zhang, 2024) Road resistance and aerodynamic drag are incorporated to reflect varying road conditions and high-speed effects, respectively. However, these models often encounter computational challenges due to their complexity. Simpler models, such as kinematic bicycle models, enhance efficiency but reduce accuracy in scenarios involving rapid directional changes. These advancements highlight the importance of balancing fidelity and real-time performance in chassis and motion dynamics modeling.

2.4 Gaps and Limitations

The existing literature on vehicle dynamics modeling highlights key gaps that motivate this project. Most studies focus on individual subsystems, such as tire forces or transmission systems, with limited exploration of fully integrated models. Generalized models prioritize versatility but fail to capture nuanced behaviors like engine braking and steering imperfections, reducing vehicle-specific fidelity. Additionally, current models lack scalability to adapt to diverse vehicle configurations and operational conditions. A significant challenge remains the trade-off between accuracy and computational efficiency, hindering real-time performance. These limitations emphasize the need for a tailored, high-fidelity vehicle dynamics model for Venti Technologies to enhance simulation fidelity and scalability.

2.5 Summary of Findings

The literature review highlights the strengths and limitations of existing vehicle dynamics models. Generalized frameworks provide a starting point but fail to capture the intricacies of specific vehicle behaviors. By leveraging insights from the literature, this project proposes a modular, high-fidelity vehicle dynamics model that integrates throttle, brake, steering, engine, transmission and chassis dynamics. The selected approach ensures accurate simulations, scalability and compatibility with autonomous vehicle systems, addressing the identified gaps and advancing the state of vehicle dynamics modeling.

3. Implemented Design

3.1 Simple Vehicle Dynamics Model

The first version of the vehicle dynamics model is designed to simulate the basic behaviors of an autonomous vehicle. It focuses on three critical components, throttle, brake and steering, creating a modular architecture that allows for independent subsystem development while ensuring seamless integration. This modular design makes each subsystem self-contained and easy to unit test, enabling developers to verify the correctness and performance of individual components without the need for full system integration. By addressing fundamental vehicle behaviors and emphasizing testability, this model lays a solid foundation for future enhancements.

3.1.1 System Overview

The simple vehicle dynamics model comprises three main phases: (1) implementing the throttle and brake models using a Linear Parameter Varying (LPV) approach, (2) combining the outputs of these models to compute velocity and (3) simulating the vehicle's state (position and heading) using a kinematic bicycle model for steering. Figure 2 illustrates the high-level flow of the system, showcasing the interaction between control inputs, subsystem computations and final outputs. The model begins by processing three primary control commands: Throttle, Brake and Steering. These inputs are handled by their respective subsystems to produce key outputs such as Acceleration, Velocity, Heading, Position X and Position Y.

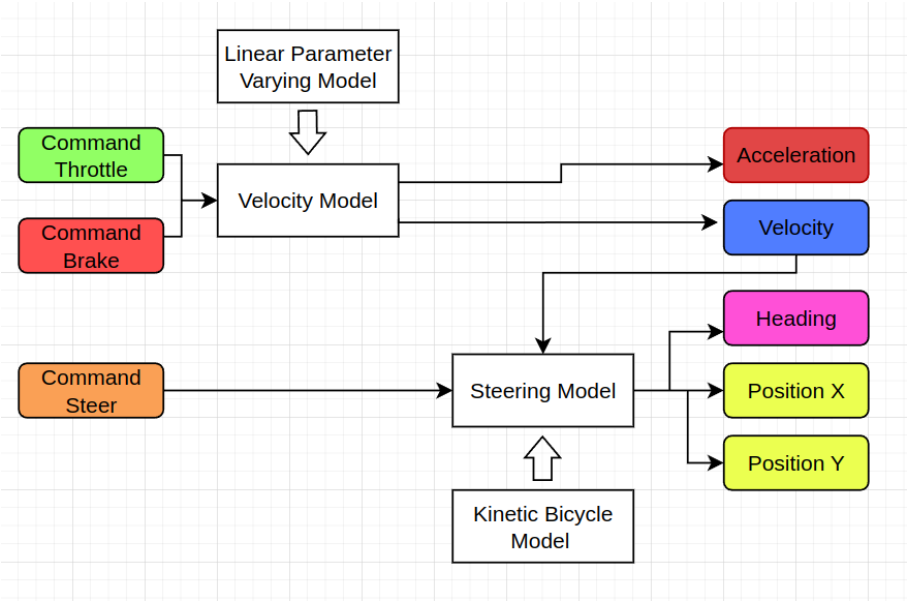


Figure 2: Simple Vehicle Dynamics Model

3.1.2 System Architecture and Class Design

The simple vehicle dynamics model is implemented using a modular, object-oriented architecture in C++, with each functional component of the system encapsulated as a separate class. Each class is designed to be self-contained, enabling independent unit testing and easy future expansion. For instance, the LPVModel class could be reused in other subsystems such as the engine or transmission models introduced in later phases of development. This modular design allows for ease of testing, extension and integration, while supporting a clear separation across subsystems.

The main system is the SimpleVehicleDynamics class, which handles the behavior of two subsystems: the VelocityModel and the SteeringModel. These components are responsible for computing the vehicle's state, including velocity, acceleration, heading and position, based on control inputs such as throttle, brake and steering angle. The SimpleVehicleDynamics class inherits from an base class, VehicleDynamics, which defines a consistent interface for both the simple and advanced dynamics models.

Control commands are encapsulated in the VehicleCommand class, which includes fields such as throttle_percent, brake_percent, steer_angle, gear_status and parking_brake. These inputs are consumed by the dynamics model to simulate the vehicle’s behavior. The resulting state of the vehicle is stored in the VehicleState class, which holds outputs such as velocity, acceleration, heading and position.

The VelocityModel class handles the calculation of vehicle velocity by integrating the outputs of the throttle and brake models. Each of these models is represented by an instance of the LPVModel class, which implements a Linear Parameter Varying (LPV) approach to simulate system response over time. The LPV model uses parameters such as static_gain, time_constant and theta (input delay) and maintains an input buffer to simulate time-lagged behavior. The VelocityModel also handles additional logic such as engine braking and gear status management such as parking brake.

Position and heading are handled by the SteeringModel class, which inherits from the KinematicBicycleModel. This model applies the kinematic bicycle formulation to compute heading and position updates based on the current steering angle and vehicle velocity. The SteeringModel also accounts for steering bias and backlash, improving the accuracy of the simulated steering response at low speeds.

Figure 2 illustrates the class architecture for the Simple Vehicle Dynamics Model. The core class, SimpleVehicleDynamics, manages two key subsystem instances: VelocityModel and SteeringModel. These components process incoming control commands and compute the vehicle's kinematic state, including velocity, acceleration, heading and position.

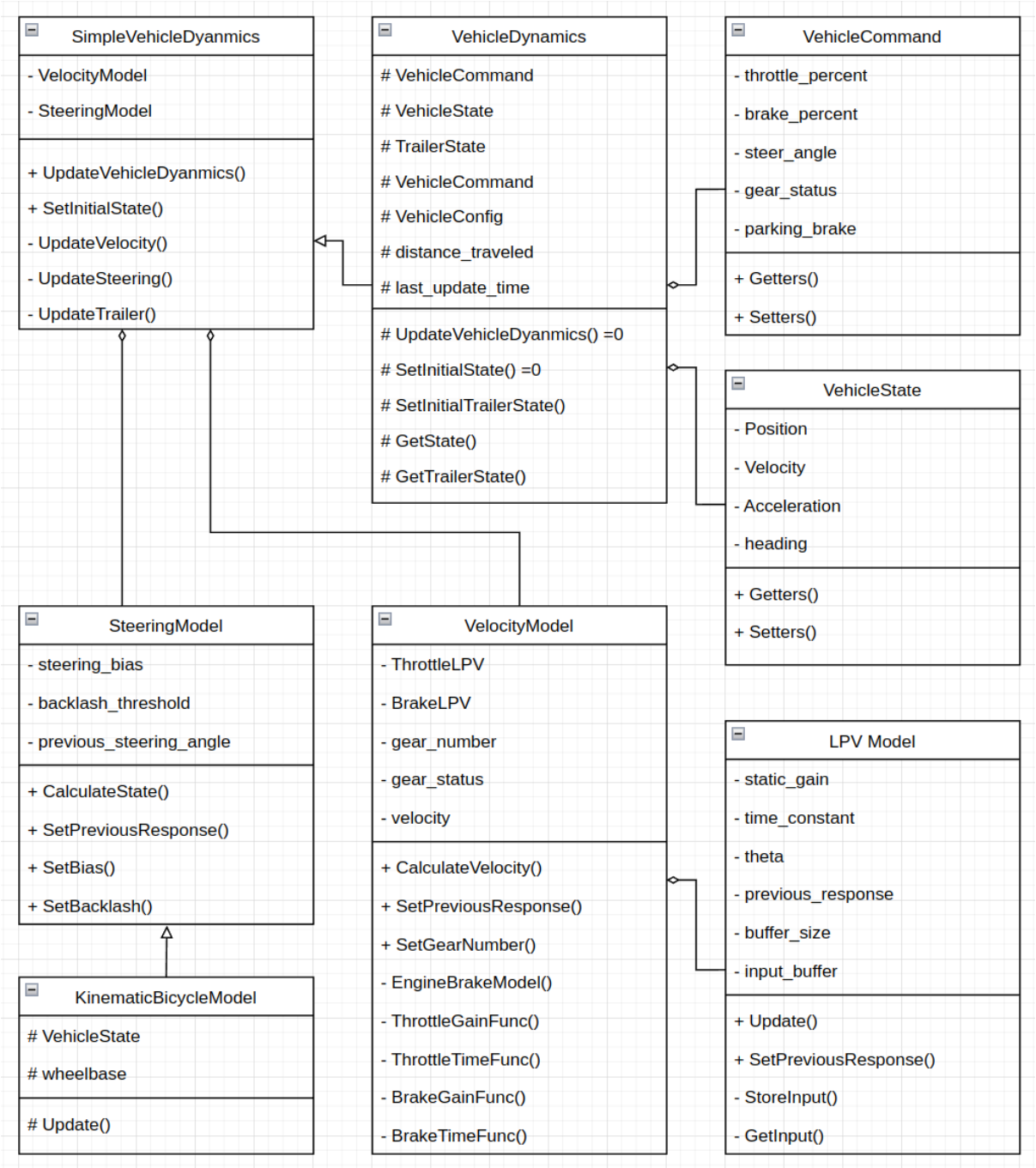


Figure 3: Simple Vehicle Dynamics Model class diagram

3.1.3 Development Plan

The development of the simple vehicle dynamics model was carried out in four distinct phases, with each phase focusing on a specific set of features. This step-by-step approach allowed for incremental development, thorough testing and easier debugging of individual components. By building and validating each subsystem independently, the design remained flexible and scalable for future improvements. In addition supporting scripts were created in Python and Bash to assist with ROS bag data comparisons, parameter tuning and visualization of results. These tools were essential for validating the performance of the model against real-world vehicle test log data.

3.1.4 Phase 1: Development of the LPV Model for Brake and Throttle

The first phase focuses on implementing the Linear Parameter Varying (LPV) model for the throttle and brake subsystems. These models take the brake or throttle percentage as input and output the corresponding velocity. This model incorporates three key parameters, static gain, time constant and delay (theta) to accurately represent real-world responsiveness.

The static gain defines the steady-state effect of the input, controlling the amplitude of the velocity output. The time constant governs the rate at which the system responds to input changes, representing how quickly the vehicle accelerates or decelerates. Finally, the theta parameter introduces a time delay, simulating the lag between the application of an input and its effect on the system's response.

To handle delay, the model uses a circular buffer input structure, where past inputs are stored and accessed based on the specified delay. The output response is calculated using the trapezoidal integration method, which ensures smooth and stable updates over time. The LPV output (velocity) at each timestep (t) is computed as:

$$y_k = \frac{2\tau - \Delta t}{2\tau + \Delta t} \cdot y_{k-1} + \frac{G \cdot \Delta t}{2\tau + \Delta t} \cdot (u_{k-\theta-1} + u_{k-\theta})$$

Where:

- y_k is the current velocity output,
- y_{k-1} is the previous velocity output,
- $u_{k-\theta}$ is the delayed control input (throttle or brake),
- τ is the time constant,
- G is the static gain,
- Δt is the simulation timestep
- θ is the delay converted to a discrete index.

This formulation balances responsiveness with smooth transitions, avoiding sudden jumps in acceleration or deceleration. It also enables tuning of vehicle responsiveness based on real-world data.

During this phase, the LPV model was implemented as a reusable class and instantiated separately for throttle and brake in the VelocityModel. Initial values for gain, time constant and delay were selected based on empirical data and later tuned to match test results. A set of unit tests was developed to validate model behavior across a variety of scenarios, including constant, step and ramp input signals.

By the end of Phase 1, the throttle and brake subsystems were functional, testable and ready to be integrated into the next phase, which combines both models to produce a unified velocity output.

3.1.5 Phase 2: Development of the Velocity Model

The velocity model combines the outputs of the throttle and brake LPV models to compute the final vehicle velocity. A rule-based selection mechanism determines which model governs the velocity based on the brake percentage. If the brake percentage exceeds 5%, the brake LPV model is used. Otherwise, the throttle LPV model determines the velocity.

In the second phase of development, the Velocity Model was created to combine the outputs from the throttle and brake LPV models and calculate the vehicle's speed. This model decides whether to use the throttle or brake based on the driver's commands and the current state of the vehicle.

A simple rule is used to choose between the two models. If the brake percentage is greater than 5%, the brake LPV model is used to reduce the vehicle's speed. If the brake is not active and the vehicle is in Drive (D) or Reverse (R), the throttle LPV model is used to increase the speed. If neither throttle nor brake is active, the model applies engine braking, which slowly reduces the speed toward an idle value. When the vehicle is in Neutral (N) or the parking brake is on, a different method is used to bring the speed to zero. This can be visualized in Figure 4:

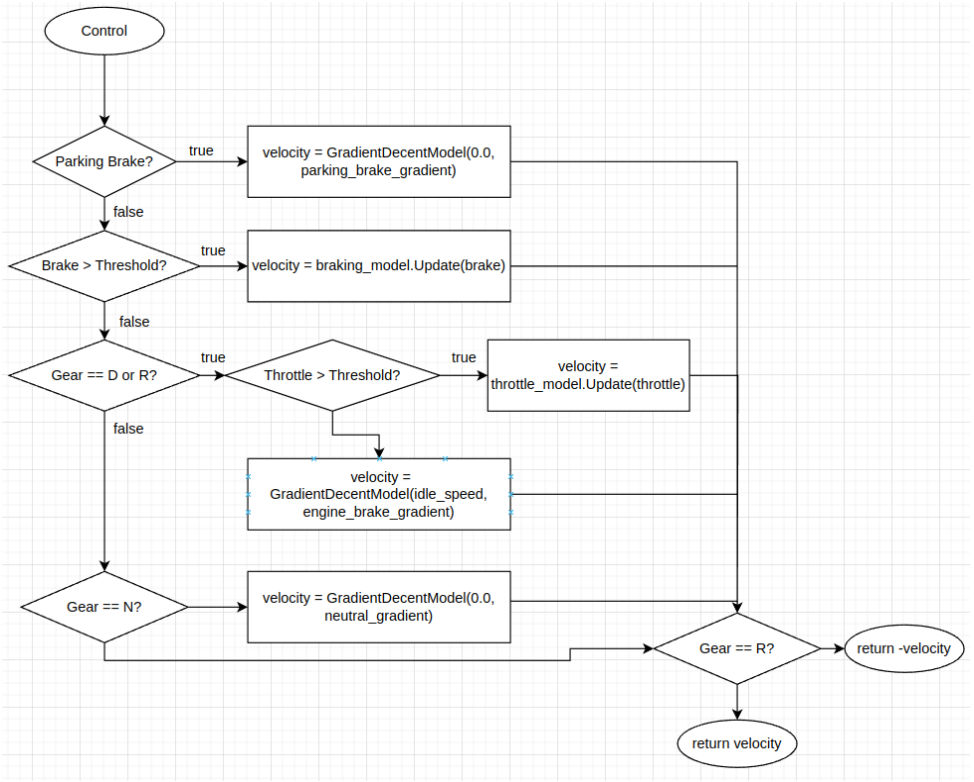


Figure 4: Flowchart showing the decision logic for velocity model

The VelocityModel class manages this logic. It contains two LPV models: one for throttle and one for brake. These models are built using the LinearParameterVaryingModel class. The behavior of each model changes based on the current gear and input values. For example, in reverse gear, the throttle model uses a different lambda function for the LPV's static gain and time constant. When the parking brake or engine braking is used, the model uses a simple gradient descent method to slowly reduce the speed to a target value over time.

This phase was tested using real data from actual vehicle runs, including acceleration, braking and slow driving. The tests showed that the model gives smooth and realistic speed outputs. Because the throttle and brake LPV models are separate, they can be optimized individually based on specific tests.

In summary, this phase added a velocity model that responds correctly to throttle and brake inputs and handles transitions like coasting and braking. It is a key part of the simple vehicle dynamics system and prepares the way for more advanced features in the next development phases.

3.1.6 Phase 3: Development of the Kinematic Bicycle Model for Steering

The third phase of the project focused on implementing a steering model using the kinematic bicycle model. This model is a common and lightweight method for simulating the motion of a vehicle during turns, especially at low to medium speeds where tire slip can be ignored. It uses basic geometry to calculate the vehicle's change in position and heading based on steering angle and velocity.

The steering model receives the current velocity and steering input (steering angle in radians) and updates the vehicle's heading and position in the X and Y directions. This model assumes a simplified bicycle structure where the two front wheels are treated as one and the two rear wheels as another, forming a single-track system. The model was implemented in the SteeringModel class, which inherits the core equations from a base KinematicBicycleModel class. This can be visualized in figure 5.

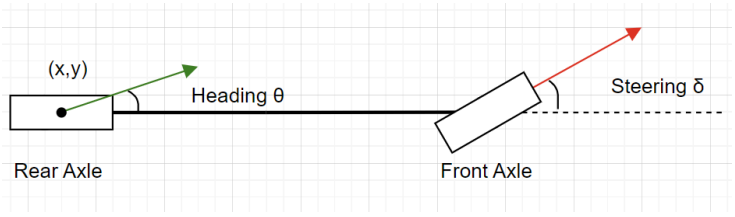


Figure 5: Kinematic Bicycle Model

The position and heading updates are based on the following equations:

$$x = v \cdot cos(\theta)$$

$$y = v \cdot sin(\theta)$$

$$\theta = \frac{v}{L} \cdot tan(\delta)$$

Where:

- x, y are the current position coordinates
- θ is the current heading (orientation)
- v is the current velocity (from the velocity model)
- δ is the steering angle
- L is the wheelbase (distance between front and rear axles)

At each timestep, the steering model updates the vehicle's position and heading using these formulas. These values are then stored in the VehicleState class.

To improve accuracy, especially at low speeds, the steering model also includes logic to handle bias and backlash in the steering angle. Bias refers to a fixed offset in the steering input, while backlash accounts for the small gap in physical steering mechanisms where input changes do not immediately affect the wheels. These corrections help to produce more realistic steering behavior, especially in scenarios such as tight cornering or zig-zag motion.

The model was tested using real vehicle data in turning scenarios, including both left and right turns. The resulting trajectories matched 70% of the expected vehicle behavior, showing smooth and continuous changes in both heading and lateral position. The use of a kinematic bicycle model allowed fast computation while maintaining good accuracy for the project's scope.

This phase completed the implementation of the steering model in the simple vehicle dynamics model. Together with the throttle and brake models, the steering model allows the system to simulate full 2D movement, making it suitable for basic autonomous vehicle behavior in flat terrain.

3.1.7 Phase 4: Development of Engine Braking for the Velocity Model

The final phase introduces engine braking to the velocity model, addressing the natural deceleration caused by friction and drivetrain resistance, which is not accounted for in the LPV throttle model. Engine braking is implemented as a constant deceleration of 0.255 m/s^2 when both throttle and brake inputs are minimal. This ensures the vehicle slows down realistically until it stabilizes at an idle speed of 0.9 m/s. If the velocity falls below the idle speed, a constant acceleration is applied to restore the velocity to its minimum stable level, simulating the vehicle's natural tendency to maintain motion.

The integration of engine braking adds realism to the velocity model by accurately representing the vehicle's behavior in coasting scenarios. This phase undergoes extensive testing to validate smooth transitions between braking, engine braking and idle speed recovery, ensuring fidelity to real-world behavior.

3.1.8 Integration and Testing

The final step for the simple vehicle dynamics model involved integrating all subsystems, throttle, brake and steering, into a single, unified module and performing functional testing to verify the correctness of the complete system. This integration step was crucial to ensure that the subsystems could work together smoothly under real-world driving scenarios.

Integration was done through the SimpleVehicleDynamics class, which serves as the central controller that connects the VelocityModel and SteeringModel components. It receives control commands from the VehicleCommand class and updates the internal VehicleState accordingly. This state includes the vehicle's current velocity, acceleration, position and heading. Each frame of simulation calls the UpdateVehicleDynamics() method, which coordinates the velocity and steering computations using the current control input and timestep.

To validate the integrated model, a series of test scenarios were created using control input data collected from actual autonomous vehicle operations at the port. This data included throttle, brake and steering commands under a variety of driving conditions such as different left and right turns, acceleration, braking and coasting.

A set of Python-based scripts was developed to run the simulation and compare the model's output with the ground truth data. These scripts extracted time-aligned signals from ROS bag files and plotted the model's output against real sensor values such as actual velocity and position. The validation tests showed that the model was able to closely follow the real vehicle behavior, with around 70% accuracy for the project's goal.

Unit tests were also written for each individual subsystem, allowing isolated testing of the LPV throttle and brake responses, velocity logic and steering model. These tests confirmed that each component returned the correct output under a variety of edge cases, including rapid input changes, gear switching and parking brake activation.

This phase marked the completion of the simple vehicle dynamics model, establishing a reliable baseline that could be extended in the advanced model phase with more detailed engine, transmission and chassis behaviors.

3.2 Advance Vehicle Dynamics Model

The Advanced Vehicle Dynamics Model expands upon the foundational design of the simple model by integrating more complex systems to simulate the behavior of a vehicle in diverse driving scenarios. This model incorporates subsystems such as the chassis, transmission, engine and gear models. These additions make the model more accurate, especially during complex driving conditions like gear shifts, torque transitions and sharp turns. Together, these subsystems provide a high-fidelity representation of vehicle behavior, enabling the simulation of realistic dynamics in terms of acceleration, velocity, heading and positional changes. The goal of this phase is to simulate more realistic vehicle responses by modeling how torque flows through the vehicle, from the engine, through the transmission, to the wheels and how it affects acceleration, velocity and vehicle movement in both longitudinal and lateral directions. Figure 6 illustrates the architectural flow of the Advanced Vehicle Dynamics Model, highlighting the interplay between various subsystems.

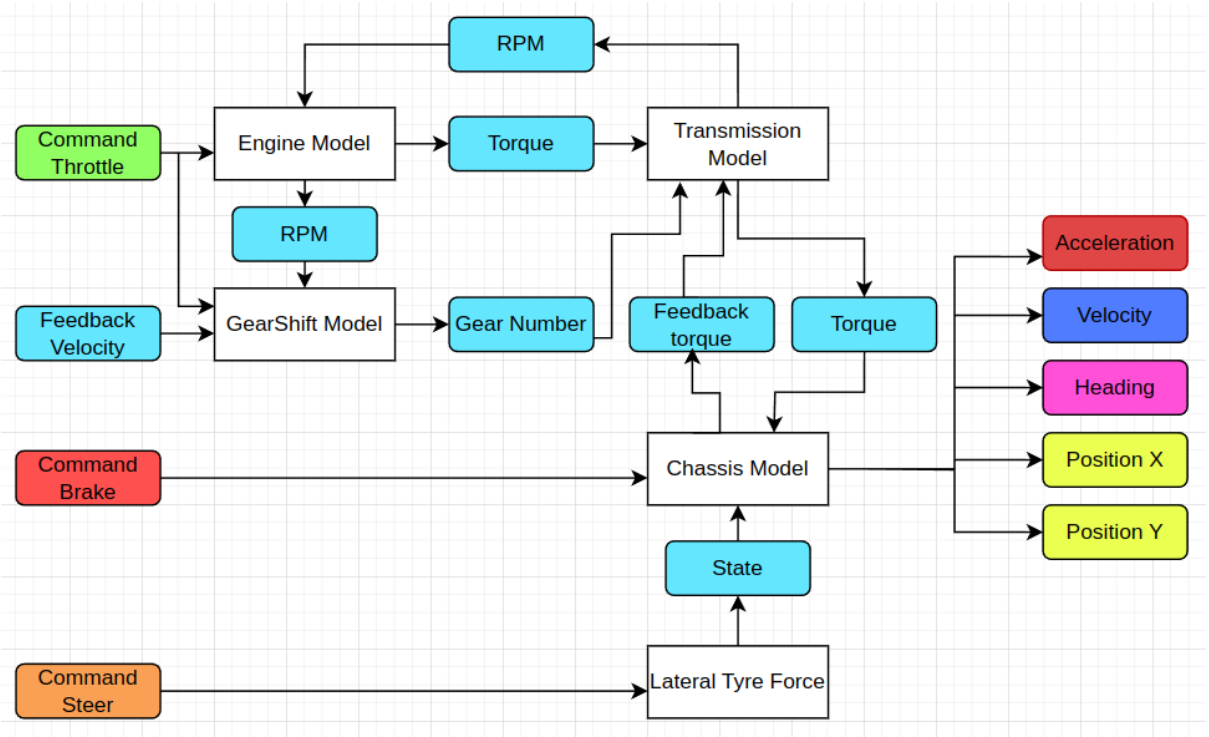


Figure 6: Advance Vehicle Dynamics Model

3.2.1 System Overview

The Advanced Vehicle Dynamics Model improves on the basic throttle-brake-steering framework by simulating how real-world vehicle components interact to affect motion. This includes modeling internal systems such as the engine, transmission, gear shifting logic and chassis dynamics. Together, these components provide a more accurate and realistic simulation of vehicle movement, especially in more complex scenarios like gear transitions and turning at higher speeds. This model also follows a modular structure, where each physical component is represented as a separate software module.

The data flow begins with throttle and brake commands. The throttle input is passed to the Engine Model, which outputs torque and RPM based on a torque map. The current RPM is used by the GearShift Model to determine the appropriate gear based on vehicle speed and shifting logic. The torque and selected gear are sent to the Transmission Model, which applies gear ratios and efficiency factors to compute the final drive torque. Meanwhile, the brake command is fed directly into the Chassis Model. The Chassis Model is the central unit that receives torque, brake force and the current vehicle state to compute new acceleration and velocity values. The Lateral Tyre Force Model estimates lateral forces based on the current speed and steering angle. This model contributes to changes in heading and position, allowing the vehicle to simulate turning behavior more accurately. Each of these values is updated every timestep and used as feedback for the next simulation cycle. This feedback loop helps the model stay consistent and stable across time, especially during rapid input changes.

This design improves realism, flexibility and testability while still keeping the simulation efficient enough to run in real time. It also lays the foundation for future extensions such as modeling suspension, terrain response or advanced tire dynamics.

3.2.2 System Architecture and Class Design

The advanced vehicle dynamics model is also implemented using a modular and object-oriented architecture, where each major physical subsystem of the vehicle is represented by its own C++ class as seen in figure 7. This approach not only mirrors the structure of real vehicle systems, but also improves code readability, separation of responsibility and ease of testing.

At the core of this system is the AdvanceVehicleDynamics class, which extends the VehicleDynamics base class. It serves as the main controller, coordinating the update flow across four critical subsystems, EngineModel, GearShiftModel, TransmissionModel and ChassisModel. Each of these subsystems models the behavior of real-world components and contributes to the overall simulation of longitudinal and lateral motion.

Inputs such as throttle, brake and steering angle are passed into the system through the VehicleCommand class. These inputs are processed across the subsystems to generate outputs like acceleration, velocity, heading and position, which are stored in the VehicleState class. Internal state and feedback values such as engine RPM, gear number and torque are passed between modules to maintain consistency throughout the simulation cycle.

The EngineModel converts throttle input into torque using a torque map based on engine RPM and throttle percentage. The engine RPM also feeds into the GearShiftModel, which handles gear logic using a velocity map and internal buffering to avoid unnecessary gear oscillations. Torque from the engine is then passed through the TransmissionModel, which applies gear ratios and calculates final drive torque, as well as feedback torque values needed to adjust engine response. The output torque is then sent to the ChassisModel, which calculates the vehicle's acceleration and movement by combining input torque, brake force and resistance. It also handles lateral dynamics through a tire force model.

The ChassisModel contains two key internal components: XMotion for longitudinal motion and YMotion for lateral dynamics. The YMotion module utilises the LateralTyreForce class to calculate slip angle and lateral forces based on vehicle velocity and steering input. These forces influence heading and lateral velocity, enabling realistic turning behavior even at high speed.

The following diagram presents the class diagram of the advanced vehicle dynamics system. It shows the relationships between the main controller, subsystem models and supporting components.

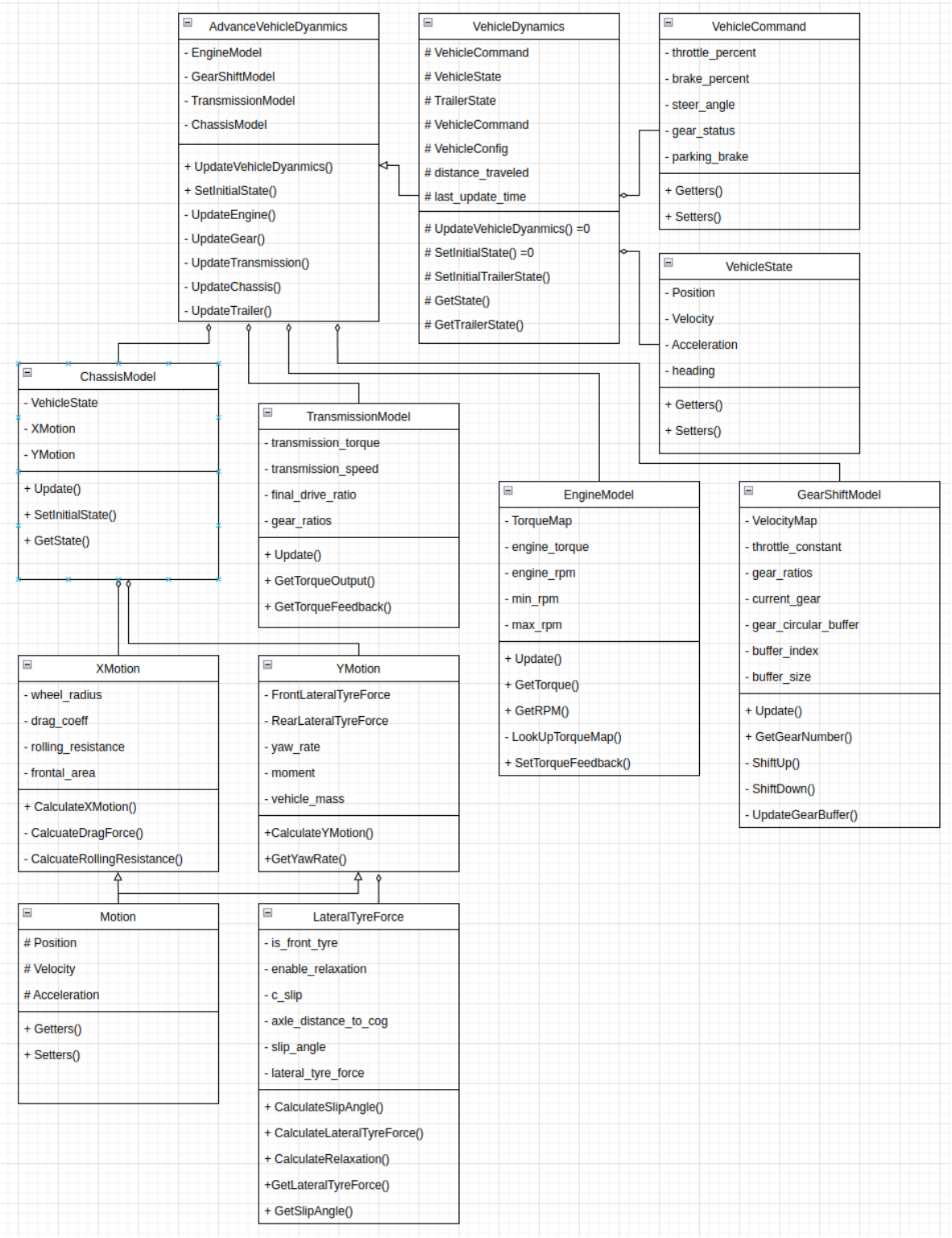


Figure 7: Advanced Vehicle Dynamics Model class diagram

3.2.3 Development Plan

The Advanced Vehicle Dynamics Model consists of four primary phases, each focusing on a critical subsystem:

1. Chassis Model: Responsible for simulating longitudinal and lateral motion, including acceleration, velocity, heading and position updates.
2. Transmission Model: Calculates torque transfer to the wheels based on current gear and final drive ratios.
3. Engine Model: Produces engine torque dynamically using throttle input and RPM via a torque map.
4. Gear Model: Determines appropriate gear shifts using a velocity-throttle mapping.

To ensure accurate implementation and prevent compounding errors, a backward development approach was adopted. In the early stages, actual values from log data (e.g., RPM, velocity, gear number) were used as inputs to test and verify each subsystem in isolation. For instance, the Chassis Model initially used ground-truth velocity from logs to calculate lateral tire forces, yaw rate, heading and position. The Engine and Transmission Models were validated using the actual gear number from log data to ensure their torque outputs aligned with expected dynamics. Only after all individual subsystems were tested and confirmed to produce consistent results was the Gear Model introduced. At this point, the system transitioned from using actual gear values to using the simulated gear output of the GearShiftModel. This step-by-step strategy ensured that validation was accurate and that any errors were isolated and resolved at the source, rather than being carried forward.

The modular design of the subsystems enabled independent testing and iteration, while the backward development strategy provided a clear path from validation to full autonomy within the simulation.

3.2.4 Phase 1: Development of the Chassis Model

The Chassis Model is the core component responsible for simulating the full-body motion of the vehicle based on physical forces. It combines longitudinal, lateral and optional vertical (Z-axis) dynamics to compute the updated vehicle state at each timestep. The model is modular and composed of three submodules (1) XMotion for longitudinal dynamics. (2) YMotion for lateral dynamics (3) ZMotion for vertical pitch/roll dynamics (present but not yet active in this phase) Each submodule inherits from a shared Motion base class and together, they update the vehicle's VehicleState, which contains position, velocity, acceleration and heading.

Table 3: Inputs and outputs of the chassis model

Input	Output
Transmission torque Command steer Command brake Current vehicle state	Acceleration Velocity (x,y,z) Heading Position (x,y,z)

The XMotion module simulates how the vehicle accelerates or slows down based on applied torque, brake force, aerodynamic drag and rolling resistance depicted in figure 8. It uses the basic form of Newton's second law $F = ma$:

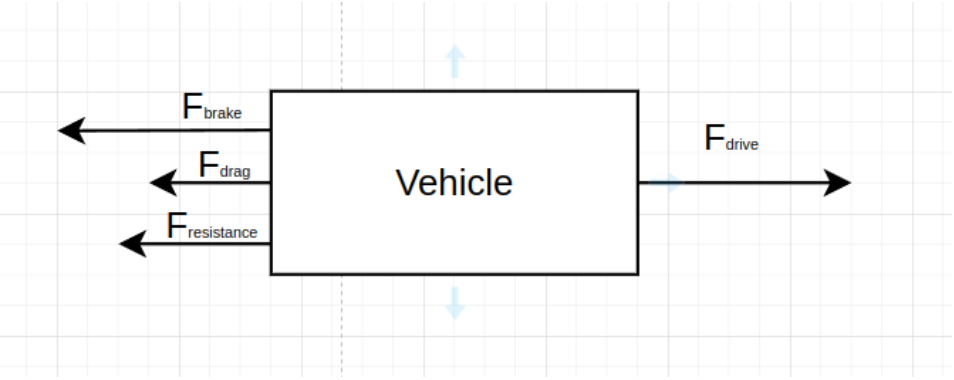


Figure 8: Free body diagram of the forces acting on the vehicle

$$F_{net} = F_{drive} - F_{brake} - F_{drag} - F_{resistance}$$

$$a_x = \frac{F_{net}}{m}$$

Where:

- F_{drive} is the forward drive force from the engine
- F_{brake} is the force front he brakes
- F_{drag} is the drag from air resistance
- $F_{resistance}$ is the rolling resistance
- m is the mass of the vehicle
- a_x is the longitudinal acceleration

Driving force is generated from engine torque after it has passed through the transmission system. It is calculated as:

$$F_{drive} = \frac{\tau}{r}$$

Where:

- τ is the transmission torque output (Nm)
- r is the wheel radius

Braking force is directly proportional to the brake input percentage. The model assumes a linear relationship:

$$F_{brake} = brake\ percent \times F_{max}$$

Where:

- F_{max} is the maximum brake force

Aerodynamic Drag Force is calculated using the standard drag equation:

$$F_{drag} = \frac{1}{2}\rho C_d A v^2$$

Where:

- ρ is the air density
- C_d is the drag coefficient
- A is the frontal area of the vehicle

- v is the longitudinal velocity

Rolling resistance models the constant resistance of tires against the ground and is approximated as:

$$F_{resistance} = C_r \cdot m \cdot g$$

Where:

- C_r is the rolling resistance coefficient
- m is the vehicle mass
- g is the gravitational acceleration

The YMotion module models side-to-side vehicle motion and rotational yaw caused by steering inputs. Lateral acceleration is calculated from tire forces:

$$a_y = \frac{F_f + F_r}{m} - \omega v_x$$

$$\omega = \frac{F_f \cdot l_f - F_r \cdot l_r}{I}$$

Where:

- F_f, F_r is the front and rear lateral tyre force
- l_f, l_r is the distance from center of gravity to front and rear axles
- I is the moment of inertia
- v_x is the longitudinal velocity
- ω is the yaw rate

The LateralTyreForce model simulates the lateral grip and slip behavior of the vehicle's tires as seen in figure 9. It plays a critical role in enabling realistic turning, cornering stability and yaw response during steering. This model is used by both the front and rear tires in the YMotion module and it feeds computed lateral forces into the yaw and lateral acceleration equations. Each tire's lateral force is calculated using a simple model:

$$F_y = -C_\alpha \cdot \alpha$$

Where:

- α is the slip angle
- C_α is the cornering stiffness
- F_y is the lateral tyre force

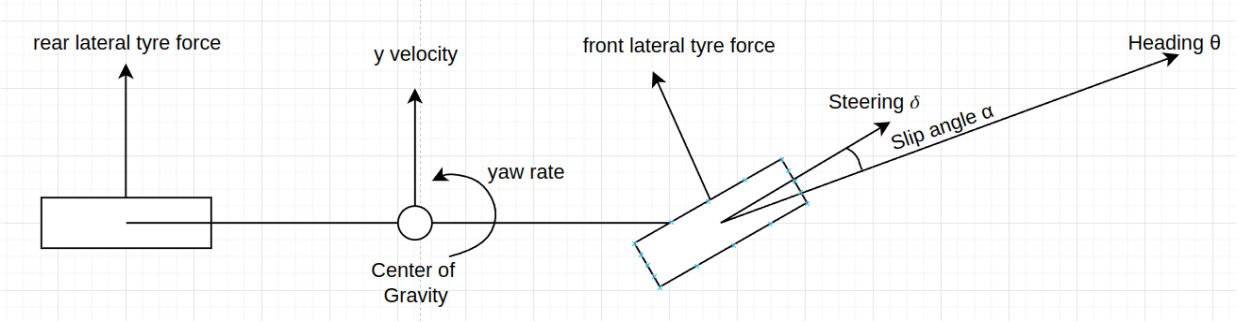


Figure 9: Lateral tyre force and slip angle

Slip angle represents the difference between the direction a wheel is pointing and the actual direction the vehicle is moving. It's the key to understanding tire behavior during cornering. This calculation helps determine how much each tire is "scrubbing" sideways during motion. Slip angle is computed for front and rear tires:

$$\alpha_f = arctan\left(\frac{v_y + l_f \cdot \omega}{v_x}\right) - \delta$$

$$\alpha_r = arctan\left(\frac{v_y - l_r \cdot \omega}{v_x}\right)$$

Where:

- δ is the steering angle
- v_x , v_y is the longitudinal and lateral velocity
- α is the slip angle

The ChassisModel Update function calls XMotion Update and YMotion Update in sequence and updates vehicle position and heading using Euler's integration method:

$$x_{t+1} = x_t + (v_x \cdot cos(\theta) + v_y \cdot sin(\theta)) \cdot \Delta t$$

$$y_{t+1} = y_t + (v_x \cdot sin(\theta) + v_y \cdot cos(\theta)) \cdot \Delta t$$

$$\theta_{t+1} = \theta_t + \omega \cdot \Delta t$$

Where:

- θ is the heading
- ω is the yaw rate
- x_t, y_t is the x and y positions

3.2.5 Phase 2: Development of the Transmission Model

The Transmission Model plays a critical role in converting engine torque into wheel torque based on the selected gear and drivetrain characteristics. In real vehicles, the transmission system amplifies or reduces torque depending on the gear ratio, enabling smoother acceleration and appropriate power delivery across various speeds. This module is essential for accurately simulating gear-dependent torque behavior and complements the engine and chassis models.

Table 4: Inputs and outputs of transmission model

Input	Output
Engine RPM Gear number Feedback torque	Transmission torque Engine speed

The transmission model uses the current gear number, provided by the GearShiftModel, to look up the corresponding gear ratio from a predefined list provided by the vehicle manufacturer. The engine torque is then scaled by this ratio to calculate the transmission torque:

$$T_{trans} = T_{engine} \times R_{gear} \times R_{final\ drive} \times L$$

Where:

- T_{trans} is the transmission torque output
- T_{engine} is the torque from the engine
- $R_{final\ drive}$ is the final drive ratio
- R_{gear} is the gear ratio of the current gear
- L is the transmission loss

In addition to computing output torque, the Transmission Model supports an inverse operation to estimate the engine RPM based on the feedback torque received from the chassis. The relationship is derived from the gear ratio and final drive ratio as follows:

$$\omega_{engine} = \omega_{wheel} \times R_{gear} \times R_{final\ drive} \times L$$

Where:

- ω_{engine} is the engine rotational speed
- ω_{wheel} is the wheel rotational speed

3.2.6 Phase 3: Development of the Engine Model

The Engine Model simulates how a vehicle's diesel engine generates torque based on throttle input. It acts as the first source of force in the vehicle dynamics model and its output feeds directly into the Transmission Model. Accurate modeling of engine torque behavior is essential for capturing the non-linear acceleration characteristics of real vehicles. The engine RPM is updated externally by the Transmission Model, which calculates it based on wheel speed and gear ratio. The current RPM is stored and used in the next update cycle to calculate torque.

Table 5: Inputs and outputs of engine model

Input	Output
Throttle command Engine RPM speed	Engine Torque

The core of the engine model is a 3D torque map as shown in figure 10, that defines engine torque values across a range of throttle percentages and engine RPMs. The map is structured with the outer map key holding the throttle percentage (0% to 100%) and each inner map contains pairs of RPM and torque values. The torque map generated from log data using a script. It estimates engine torque using velocity, gear number and engine rpm.

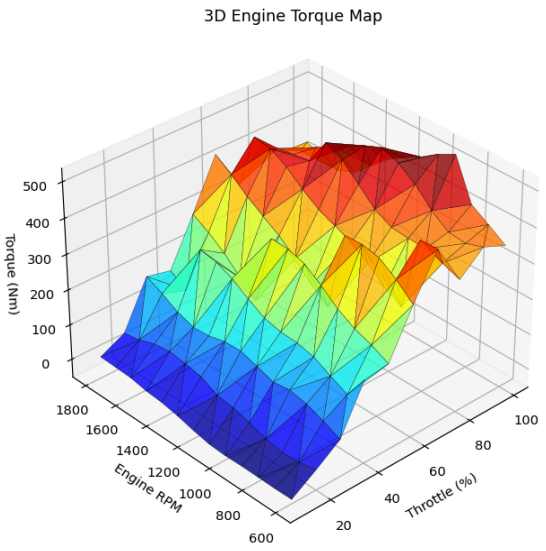


Figure 10: Generated torque map from log data

Table 6: Example of a simplified torque map

Throttle\ RPM	800	1000	1200	1400
25%	200	300	400	380
50%	250	350	450	430
75%	350	450	590	500
100%	400	480	620	550

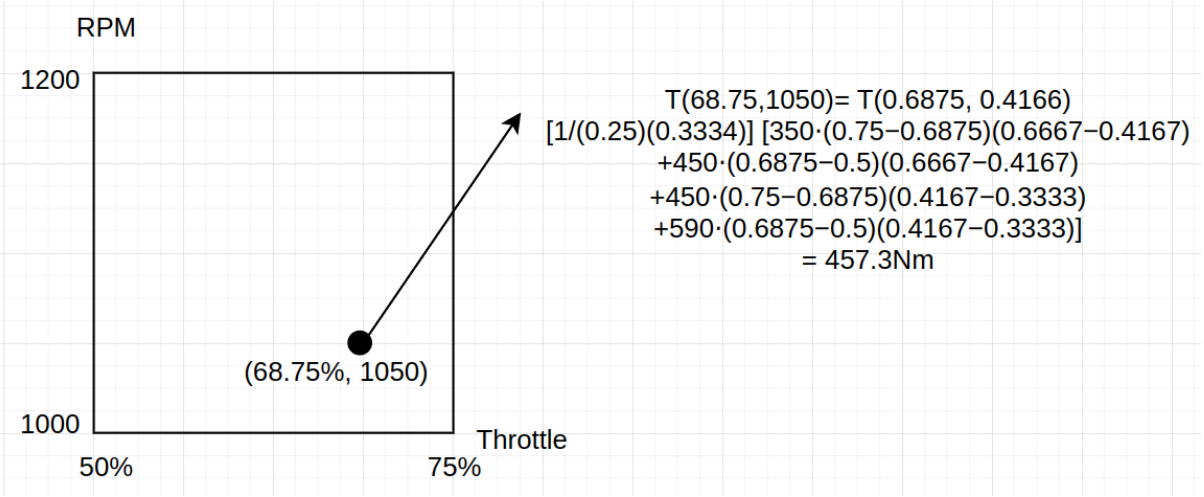


Figure 11: Visualization of Bilinear Interpolation

The EngineModel uses a bilinear interpolation method to compute torque for any combination of throttle and RPM input. This ensures torque increases smoothly with both throttle and RPM, just like in a real engine.

Given surrounding torque values:

$$T_{11} = T(Throttle_1, RPM_1)$$

$$T_{12} = T(Throttle_1, RPM_2)$$

$$T_{21} = T(Throttle_1, RPM_1)$$

$$T_{22} = T(Throttle_2, RPM_2)$$

Then the torque at a point (x,y) where x is throttle and y is RPM (normalized), is:

$$T_{(x,y)} = \frac{1}{(x_2-x_1)(y_2-y_1)} \times [T_{11}(x_2 - x)(y_2 - y) + T_{21}(x - x_1)(y_2 - y) + T_{12}(x_2 - x)(y - y_1) + T_{22}(x - x_1)(y - y_1)]$$

3.2.7 Phase 4: Development of the Gear Model

The GearShiftModel is responsible for simulating automatic gear transitions based on throttle and vehicle velocity.

Table 7: Input and output of gear shift model

Input	Output
Velocity Throttle command	Gear number (e.g., 1, 2, 3, 5, 6)

It uses a precomputed 2D lookup table velocity_map, which stores threshold velocities for each gear at different throttle levels. These velocity thresholds are derived from an array of 6 gears ratios (from vehicle manufacturer) and 5 throttle levels (0%, 25%, 50%, 75%, 100%) constants that are generated from log data using a script.

$$velocity_{(r,t)} = \frac{throttle\ constant_t}{gear_r}$$

During each update, the model selects a row in the threshold table based on the current throttle percentage. If the vehicle's velocity exceeds the threshold for the current gear, it shifts up. If the velocity drops below the threshold for the previous gear, it shifts down. Gear state is smoothed using a circular buffer that averages the last 1 second of gear values, reducing jitter from unnecessary gear shifts. This simple rule-based approach enables stable and realistic gear behavior, while remaining computationally efficient and easy to tune.

Table 8: Example of a simplified velocity map

throttle \ gear	1	2	3	4
0%	1.76	3.31	4.36	6.15
50%	1.89	3.55	4.68	6.60
100%	2.29	4.30	5.67	8.00

Throttle constants = { 6.15, 6.6,8.0 } for 0%, 50%, 100%

Gear ratios = { 3.49,1.86,1.41,1.0 } for gears 1 to 4

$$velocity\ (1,\ 50\%)\ =\ \frac{6.6}{3.49}\ =\ 1.89111$$

3.2.8 Integration and Testing

Once all individual subsystems were completed, including the engine, transmission, gear shift, chassis and lateral tire force models, they were integrated into a single AdvanceVehicleDynamics class. This integration allowed for the coordination of internal update sequences, enabling the system to simulate full vehicle behavior from raw control commands.

A step-by-step integration approach was followed to ensure both stability and accuracy. Each component was first validated in isolation using unit tests and comparisons against log data, then progressively integrated into the full pipeline. The initial testing phase relied on real vehicle log data, such as velocity, RPM and gear number, as inputs to cross-check subsystem outputs and confirm that the model's behavior matched real-world patterns. This strategy helped eliminate compounded errors by allowing each model to be tested independently. Only after a subsystem consistently produced accurate results with real data was it transitioned to accept inputs from other simulated models. For example, once the GearShiftModel was validated, its output was used in place of the logged gear number to drive the engine and transmission models. This gradual handover process ensured that integration occurred smoothly and each stage of the system maintained correctness.

To ensure high fidelity and reliability, the full system was then evaluated through simulation runs using recorded log files. These log-based tests supplied real throttle, brake and steering inputs into the model and the resulting simulated outputs such as velocity, acceleration, heading and gear were plotted and compared against ground-truth data.

3.3 Support Scripts

The development of advanced vehicle dynamics models requires not only accurate subsystem implementations but also effective tools to support the evaluation, validation and optimization of the models. To streamline workflows, improve data analysis and generate parameters based on log data, several support scripts in python and bash have been developed. These tools provide critical functionality, ranging from graphing plots to automating the comparison of simulation outputs with actual vehicle data.

3.3.1 Graph Plotting and Fitness Comparison Tool

The graph plotting tool provides a graphical user interface (GUI) for visualizing data generated by the vehicle dynamics model. This tool enables developers to analyze the output of the vehicle dynamics model intuitively, compare results and assess fitness metrics.

Key Features:

1. File Explorer GUI: Opens a file explorer. This allows users to quickly locate and open the desired .txt file generated by the comparison script as seen in figure 12.
2. Graph Visualization: The selected file is plotted directly within the GUI, displaying velocity data alongside other relevant outputs as seen in section 4 results.
3. Graph Selection: Users can toggle checkboxes to choose specific graphs to display, providing flexibility in the data visualization process.
4. Fitness Statistics: Metrics such as model fitness are prominently displayed on the top-left corner of the GUI, giving immediate feedback on the model's accuracy.

This tool simplifies data analysis, providing a visual representation of key outputs and allowing for quick iteration and debugging.

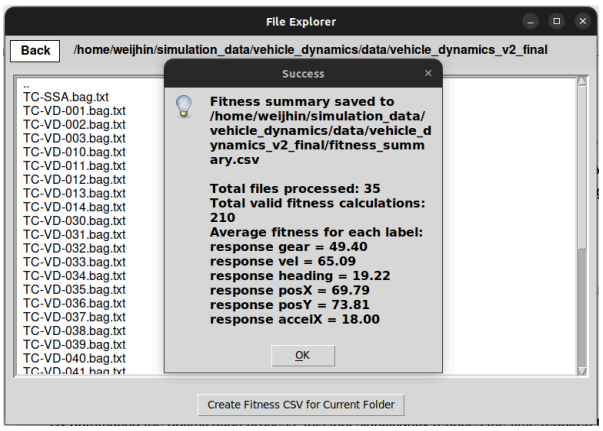


Figure 12: File Explorer GUI that opens log txt data as graph visualization

3.3.2 Automation Tool for Simulation and Data Comparison

A Python-based automation script has been developed to streamline the process of building, deploying and validating the vehicle dynamics model against actual vehicle data. This script integrates multiple steps into a seamless workflow, reducing manual effort and ensuring consistent data collection.

Workflow:

1. Compiles the vehicle dynamics codebase once.
2. Launches the ROS core
3. Launches vehicle dynamics ROS node
4. Plays respective ROS bag files asynchronously in separate Docker containers.
5. The vehicle dynamics node logs data during playback for analysis.

After playback, users can execute the plot_gui.py script to visualize the results and compare model outputs with real-world vehicle data.

Command-Line Options:

- -b, --bagfile: Specify a single ROS bag file to process.
- -f, --filelist: Provide a text file listing multiple ROS bag files.
- -p, --parallel: Enable parallel processing of bag files for faster execution.

Example Command:

python3 bag_comparison.py --filelist bag_files.txt --parallel

This tool enables automation of data validation workflows, ensuring efficient testing across multiple scenarios. Depending on the number of files, this script can take hours to run, thus saving alot manual work and time.

Figure 13: Data comparison automation tool

3.3.3 Parameter Optimization Tool

A dedicated optimization program has been developed to fine-tune the parameters of the vehicle dynamics model, ensuring the best possible performance. This tool iteratively tests combinations of parameters such as static gain, time constant and delay, analyzing their impact on fitness metrics and providing the optimal number for each parameter.

Key Features:

1. Automates the process of testing parameter combinations across different scenarios.
2. Calculates fitness metrics for each parameter set, identifying the optimal configuration.
3. Outputs a summary of the best-performing parameters, simplifying the tuning process.

By automating the optimization process, this tool significantly reduces the time required to achieve a high-fidelity model.

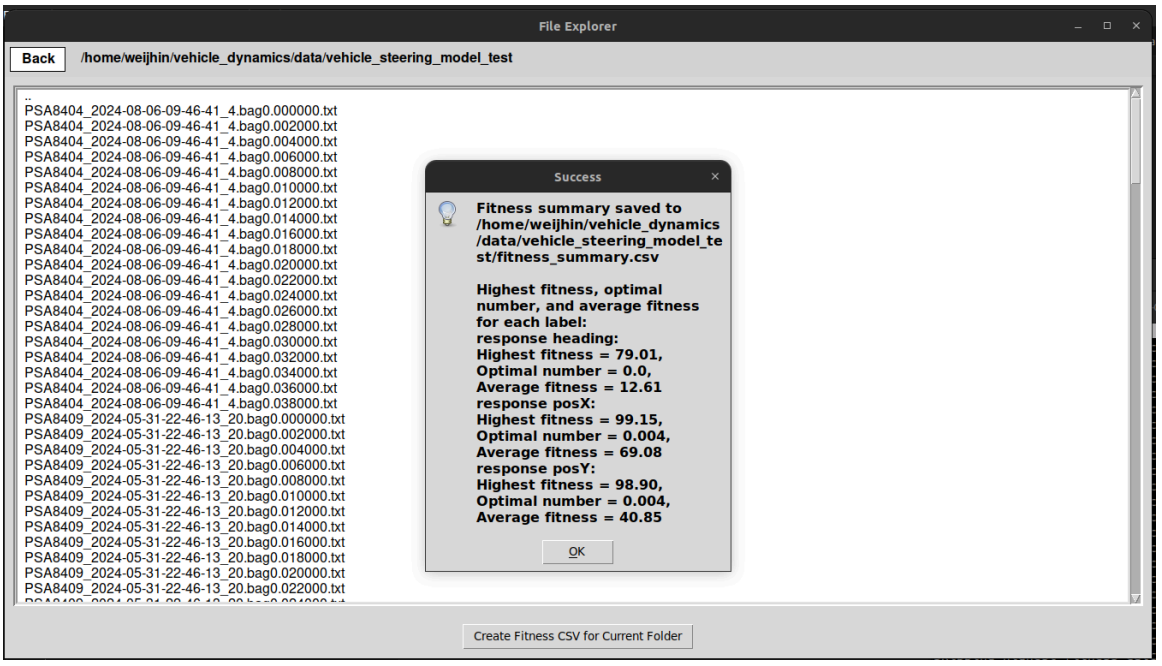


Figure 14: Output of parameter optimization tool

3.3.4 ROS Bag Splitting and Merging Tool

Managing large datasets from vehicle tests is a critical part of validating the vehicle dynamics model. In analyzing recorded ROS bags from vehicle tests, it became evident that data captured in 5-minute intervals required splitting and merging for effective processing. Manually managing this segmentation will be time-consuming and prone to error, highlighting the need for an automated solution. A specialized script has been developed to handle splitting and merging ROS bag files, making it easier to work with test data.

Key Features:

1. Splitting: Divides a large ROS bag file into smaller segments based on specified start and end times, enabling targeted analysis of specific events.
2. Merging: Combines multiple ROS bag files into a single file for comprehensive analysis.

This tool simplifies the management of test data, ensuring consistent handling and reducing manual effort during pre-processing.

3.3.5 Engine Torque Map Generation

To simulate engine behavior accurately, a torque map is generated from log data to model the relationship between throttle percentage, engine RPM and the resulting engine torque. This map is essential for replicating realistic engine dynamics in vehicle simulations.

Raw vehicle logs are parsed to extract relevant parameters, including timestamp, throttle percentage, brake percentage, gear number, vehicle velocity and engine RPM. Only data with low brake pressure (brake < 5%) are considered to avoid contamination from braking forces when estimating drive torque. Torque is estimated by reversing the vehicle dynamics equations:

1. Acceleration is computed over a buffered window (10 frames) for better stability:

   $a = \frac{v_t - v_{t-1}}{\Delta t}$

2. The net force is calculated:

   $F_{net} = m \cdot a$

3. Total resistance includes drag and rolling resistance:

   $F_{resistance} = \frac{1}{2}\rho C_d A v^2 + C_r mg$

4. Drive force is obtained by:

   $F_{drive} = F_{net} + F_{resistance}$

5. This is converted into wheel torque and finally engine torque using gear and final drive ratios.

Estimated torque values are binned by throttle percentage and engine RPM into a 3D grid. The grid is defined as:

- Throttle levels: 00% to 100% (in steps of 10%)
- RPM levels: 600 to 1800 RPM (in steps of 200)

Due to sparsity in the data, interpolation (linear and nearest-neighbor fallback) is used to ensure a complete and continuous map using python's scipy interpolate library. A Gaussian filter is applied to smooth the map and reduce noise using python's scipy ndimage library.

The final torque map is visualized in 3D surface plots.These visualizations allow verifications that the torque map follows expected real-world diesel engine behavior: flat high torque at mid-RPMs, tapering at high RPMs and lower values at low throttle inputs.

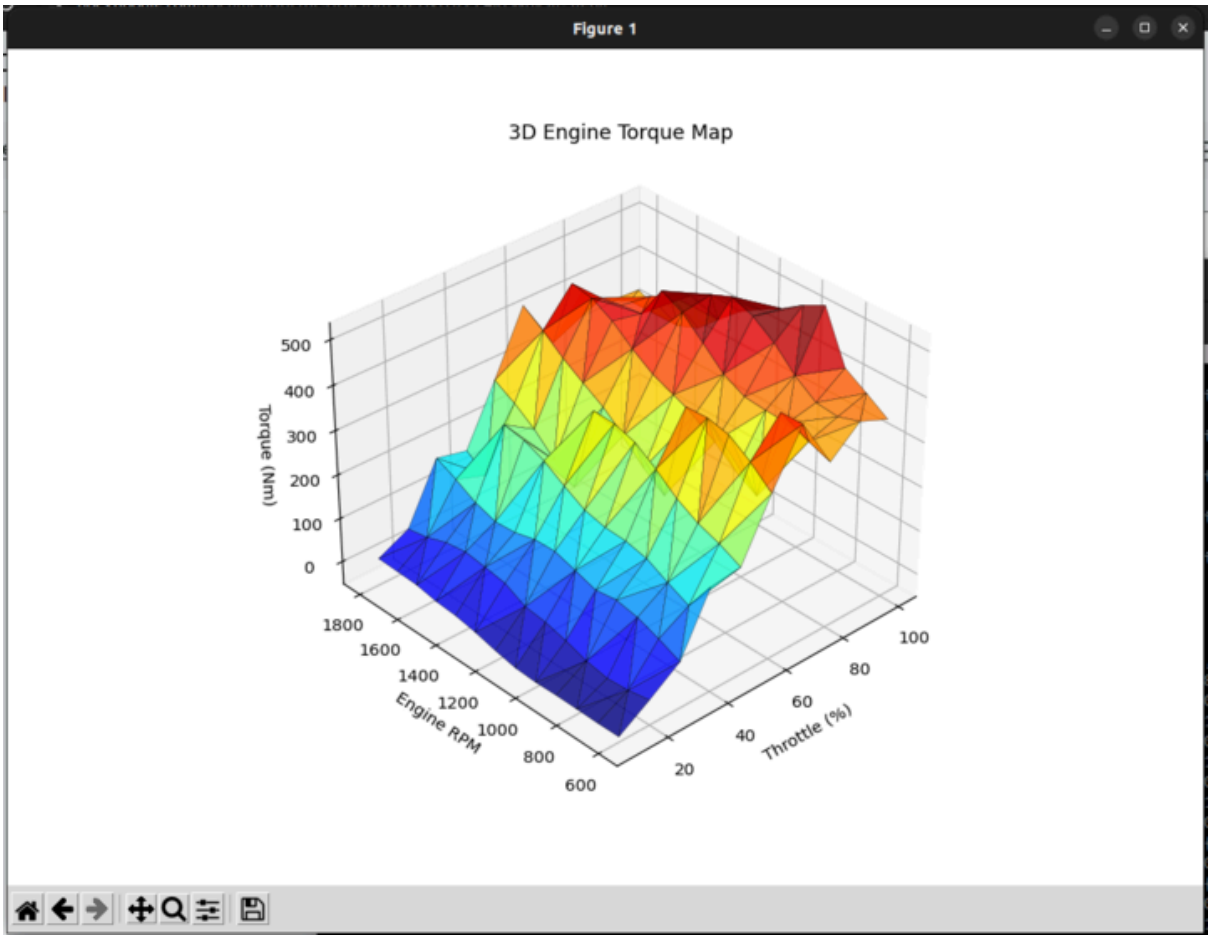


Figure 15: Generated torque map visualization

3.3.6 Gear Shift Map Throttle Constant Generation

In order to determine the optimal throttle_constant used for each gear, a Python-based script was developed to automatically generate the throttle_constant values used in the gear shift model.

The goal of this process is to optimize the throttle_constant values such that the computed current gear in the GearShiftModel matches the logged gear from real vehicle driving data as closely as possible. This ensures that the gear estimation model aligns with physical vehicle behavior.

Log files collected from actual vehicle test runs are being processed to extract throttle percentage, current gear and velocity.

The Python script performs the following steps:

1. Log Parsing: It reads a txt log file containing structured vehicle data.
2. Throttle Indexing: Converts the raw throttle percentage into one of five throttle bins: 0%, 25%, 50%, 75%, 100%.
3. Mapping Velocity Estimation: For each throttle level, it calculates a set of velocity thresholds for shifting between gears using the formula:

$$velocity_{(r,t)} = \frac{throttle\ constant_t}{gear_r}$$

4. Gear Prediction: Based on the throttle level and vehicle velocity, the script simulates the gear that would be selected by the current model.
5. Error Computation: Compares the simulated gear with the logged gear and computes the mean squared error.
6. Optimization: Uses the Nelder-Mead algorithm to iteratively update the throttle_constant values in order to minimize the error.
7. Output Generation: Produces throttle constant for each throttle percentage to be used in the gear shift model as seen in figure 15.

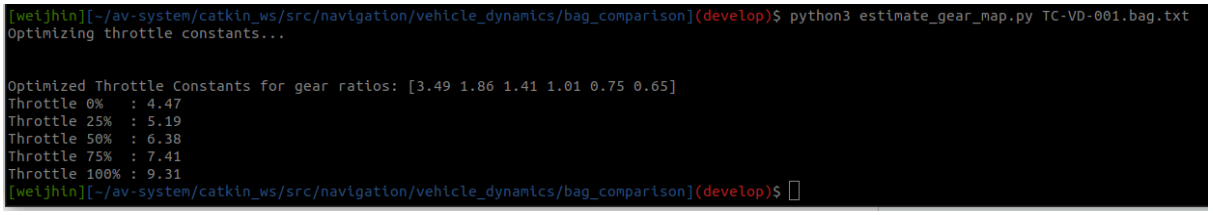


Figure 16: Output of gear shift map estimation script

3.3.7 Final Drive Ratio Estimation

This script is created to estimate the final drive ratio of a vehicle using log data collected during vehicle operation. It computes a best estimate based on engine RPM, vehicle velocity and current gear. Log files collected from actual vehicle test runs are being processed to extract engine rpm, current gear and velocity. For each log entry, the final drive ratio is calculated as:

$$final\ drive\ = \frac{engine\ rpm \times 2\Pi \times wheel\ radius}{60 \times velocity \times current\ gear\ ratio}$$

This formula converts engine RPM to wheel angular velocity and solves for the final drive ratio. To reduce the effect of low-speed noise and idle behavior, only entries with velocity > 1.0 m/s are considered valid and the vehicle has to be in Drive (D) gear status. The script prints the best estimate to the console. This value can be used in the transmission model as the final drive ratio.This script is useful for calibrating vehicle dynamics for different vehicle models.

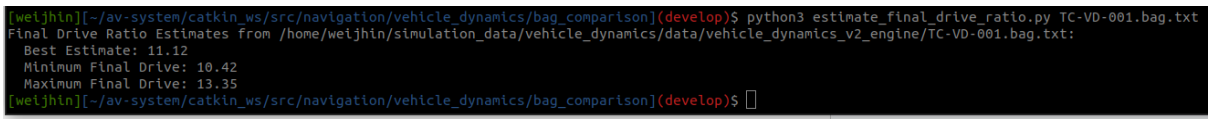


Figure 17: Output of final drive estimation script

4. Result and Analysis

4.1 Testing

To evaluate the accuracy and reliability of the developed vehicle dynamics models, a ROS-based playback and logging system was used. This approach enabled direct comparison between the model outputs and ground-truth vehicle data collected during real-world testing at the port. A ROS bag file containing recorded test scenarios was used as the input source. These bag files included control commands data such as throttle, brake and steering angle. A script was developed (described in section 3.3.2) to replay the bag in real time.The logger then provides the commands from the bag into the simulation model. As the model processes these inputs, it generates outputs including velocity, position and heading. The logger then logs both the model outputs and the actual vehicle states at each time step into a structured text file. Each entry included timestamps, command inputs, simulated outputs and the corresponding ground-truth values such as actual velocity, position and heading.

Another evaluation script (described in section 3.3.1) was then used to read the log file and generate visual plots comparing the model's output against the real data. It also computed fitness measures for key metrics such as velocity, position and heading.

This process allowed for a clear and reliable validation of the model's performance. By feeding the same real-world control inputs into the simulation, the model's response could be directly compared to actual vehicle behavior. This one-to-one correspondence ensured that any differences between simulated and real outputs were attributable solely to the model's accuracy. As a result, the testing framework enabled both quantitative evaluation through fitness metrics and qualitative validation through visual comparison of velocity, position and heading over time.

4.1.1 Test Cases

To evaluate the vehicle dynamics models under realistic driving conditions, a series of structured test cases were conducted at the port using a real autonomous vehicle. The tests were designed to isolate specific control commands and observe the resulting vehicle behavior for comparison with simulation outputs. All test scenarios and configurations, including speed limits, test names and conditions are detailed in Appendix A.

The test cases were organized into the following categories:

1. Base Runs: Straight-line driving at fixed speeds to establish reference behavior.
2. Throttle Tests: Acceleration under different throttle levels, without braking or steering, to test the vehicle's forward dynamics.
3. Brake Tests: Controlled deceleration from a known speed to evaluate braking response and stopping distance.
4. Steering Tests: Left or right turning maneuvers to validate heading updates and position accuracy.

All tests were conducted under similar dry conditions to ensure consistency and minimize environmental variables. The following preconditions and objectives were established for the test sessions:

Preconditions:

- The tests were conducted on an empty road with no other vehicles present.
- The testing surface was a dry flat road with no slopes or elevation changes.

Test Objectives:

1. To evaluate the relationship between throttle input and vehicle velocity.
2. To evaluate the relationship between brake input and vehicle velocity.
3. To evaluate the relationship between steering angle, vehicle velocity, heading and position (X, Y).

Data Collected:

- cmd_throttle – the actual throttle command issued to the vehicle.
  cmd_brake – the actual brake command issued to the vehicle.
- cmd_vel – the velocity command received.
- cmd_steer – the steering command received.
- USBCan_velocity – the velocity measured from the vehicle.
- gear_number - the current gear the vehicle is in
- heading – the orientation of the vehicle.
- position X & Y – the vehicle's global position coordinates.

4.1.2 Data Collection

To ensure consistent and reliable evaluation of the vehicle dynamics model, each test case was repeated three times under the same environmental conditions and using the same route. This repetition allowed for validation of results and reduction of noise caused by environmental or sensor fluctuations.

One run was used for parameter tuning. During this phase, specific model parameters (e.g., time constants, static gains, steering bias, slip coefficients) were adjusted to minimize error between the model's output and real-world vehicle behavior.

The remaining two runs were used for evaluation purposes, with no further adjustments made to the model. This ensured an unbiased assessment of the model's performance and its ability to generalize beyond the tuning dataset.

This approach provided a clear separation between training and testing data, which is essential for evaluating whether the model is overfitting to the tuned scenario or can perform consistently across different but similar runs. It also allowed for more confident benchmarking when comparing between the simple and advanced vehicle dynamics models.

All data collected during each test was recorded in the ROS-bags and included control commands (throttle, brake, steer), vehicle state (velocity, position, heading) and other data such as engine RPM and gear status. These logs served as the basis for performance visualization and fitness metric computation in the results and analysis phase.

4.1.3 Evaluation Metrics

To evaluate the performance of the vehicle dynamics model, a set of quantitative metrics was used to compare simulated outputs with ground-truth data collected from real-world vehicle tests. A fitness score was calculated using the normalized root mean square error (NRMSE), which quantifies how closely the model output matches actual vehicle behavior, with higher scores indicating better accuracy.

$$nrmse = sqrt(\sum(model - log)^2 / \sum(log - mean(log))^2)$$
$$fitness = (1 - nrmse) * 100$$

The following core metrics were used as primary indicators of model performance:

1. Velocity: The model's velocity output was compared against the actual velocity recorded from the vehicle. This was the most important metric for evaluating the accuracy of throttle and brake subsystems.
2. Position (X, Y): The simulated global position was compared to the ground-truth position derived from the onboard localization system. This helped assess the cumulative accuracy of velocity, heading and steering models over time.
3. Heading: Heading accuracy was measured by comparing the model's predicted yaw angle against real-world heading data. This was essential in validating the steering and lateral motion components of the steering and chassis model.

Additional metrics were also recorded to verify subsystem behaviors and detect abnormal model responses:

1. Gear Number: Used to assess the accuracy of the GearShiftModel by comparing predicted gear changes to the actual gear recorded during test runs.
2. Engine RPM: Compared simulated RPM values with logged engine data to ensure realistic modeling of torque delivery and transmission response.
3. Acceleration: The acceleration calculated by the model was compared with the data from the Inertial Measurement Unit (IMU).

4.2 Simple Vehicle Dynamics

4.2.1 Improvements Made

The simple vehicle dynamics model was evaluated against real-world data using the test procedures outlined in Section 4.1. Initial testing revealed several discrepancies between model predictions and actual vehicle responses, which led to a series of targeted improvements to enhance model fidelity.

Through early simulation results and test case analysis, the following issues were identified:

1. Brake Response

   The brake model showed excessive deceleration at low brake percentages, but insufficient braking at higher percentages. This inconsistency led to poor velocity tracking during soft stops and emergency braking.

2. Throttle Behavior at Higher Gears

   The throttle model produced unrealistic acceleration in higher gears, resulting in excessive velocity increases.

3. Steering Inaccuracy

   The steering model exhibited a slight directional bias, consistently veering slightly off-center. This was attributed to mechanical tolerances or calibration offset in the real vehicle's steering system.

Model Improvements:
Velocity Model Enhancements

- Gear-Based Tuning:

  Different throttle static gain and time constant values were applied based on the current gear number. This adjustment reflects the fact that vehicles accelerate more slowly in higher gears due to lower gear ratios and mechanical inertia.

- Brake Percentage Scaling:

  The brake model was enhanced to use dynamic static gain parameters based on brake input level. This resolved the imbalance between low and high brake percentages, improving velocity prediction accuracy during various braking intensities.

- Engine Braking Simulation:
  A decay mechanism was implemented to reduce velocity when both throttle and brake commands fall below 5%. Since the throttle model lacked a natural decay, this addition allowed the simulation to mimic engine braking behavior observed in real diesel trucks.

Steering Model Enhancements

- Backlash Threshold Implementation:
  A steering backlash threshold was added to account for the "free play" in the actual vehicle's steering mechanism. This ensures the model does not respond to small steering inputs that would have no effect in the real system.

- Steering Bias Compensation:
  A configurable steering bias term was introduced to correct for consistent drift caused by slight mechanical misalignment in the real vehicle’s steering. This adjustment improved accuracy in heading and position tracking during turning maneuvers.

These improvements significantly increased the model’s ability to replicate vehicle behavior across a range of throttle, brake and steering conditions given a simple model was being used.

4.2.2 Result Analysis

The performance of the Simple Vehicle Dynamics Model was evaluated using both trained and unseen test datasets. The key metrics assessed include heading accuracy, position X and Y, and velocity, as computed from comparison with real-world vehicle data. Three sets of tests were analyzed: trained data, test data, and test data 2.

In the trained data, the model produced relatively consistent results, achieving an average velocity fitness of 77.08 and positional accuracies of 88.71 (X) and 86.55 (Y). The average heading fitness was unfortunately -28.52. However, performance on test data, which the model had not been tuned for, showed a huge drop across all metrics. The velocity fitness dropped significantly to 47.45, and position fitness reduced to 59.70 (X) and 54.48 (Y). Heading fitness also declined sharply to -83.52, indicating the model's limited generalization capabilities when exposed to previously unseen scenarios.

Table 9 : Result of Simple vehicle dynamics

Trained Data					
Filename	Description	response heading	response posX	response pos	response vel
TC-VD-001.bag.txt	Base run	-16.08	89.42	94.23	83.75
TC-VD-011.bag.txt	Throttle	-321.31	89.45	79.00	83.08
TC-VD-030.bag.txt	Hard bake	-71.44	73.72	77.55	62.63
TC-VD-040.bag.txt	Soft brake	-56.59	92.85	92.40	75.74
TC-VD-060.bag.txt	Left turn	97.26	93.16	93.78	82.64
TC-VD-061.bag.txt	Right turn	96.44	87.90	95.75	74.13
TC-VD-070.bag.txt	Left-right turn	72.06	94.46	73.12	77.62
Overall Average		-28.52	88.71	86.55	77.08
Test Data					
Filename	Description	response heading	response posX	response pos	response vel
TC-VD-001.bag.txt	Base run	-24.32	62.36	63.86	35.47
TC-VD-011.bag.txt	Throttle	-464.22	38.22	10.94	28.07
TC-VD-030.bag.txt	Hard bake	-24.32	62.36	63.86	35.47
TC-VD-040.bag.txt	Soft brake	-20.48	50.01	46.66	39.96
TC-VD-060.bag.txt	Left turn	-19.06	77.55	82.64	62.89
TC-VD-061.bag.txt	Right turn	-13.16	49.87	30.77	67.39
TC-VD-070.bag.txt	Left-right turn	-19.06	77.55	82.64	62.89
Overall Average		-83.52	59.70	54.48	47.45
Test Data 2					
Filename	Description	response heading	response posX	response pos	response vel
TC-VD-001.bag.txt	Base run	4.80	89.92	94.36	83.80
TC-VD-011.bag.txt	Throttle	37.24	85.58	86.28	83.27
TC-VD-030.bag.txt	Hard bake	42.27	49.95	48.97	44.63
TC-VD-040.bag.txt	Soft brake	-23.10	-3.88	40.10	-241.71
TC-VD-060.bag.txt	Left turn	36.14	80.29	95.71	68.80
TC-VD-061.bag.txt	Right turn	5.97	72.93	93.55	61.39
TC-VD-070.bag.txt	Left-right turn	59.07	74.16	52.57	74.73
Overall Average		23.20	64.14	73.08	24.99

4.2.3 Comparison with Previous VTD Dynamics

To evaluate the improvement brought by the Simple Vehicle Dynamics Model, its performance was compared against the previous dynamics provided by VTD (Virtual Test Drive). The same set of test scenarios, comprising throttle, brake, and steering maneuvers, was used to assess the response of both models in terms of key metrics: heading, position X, position Y, and velocity.

The VTD dynamics showed significant deviations from real-world vehicle behavior, particularly in heading and position metrics. The average heading error was -184.20, indicating major inaccuracies in directional response. Positioning errors were also high, with -0.29 in X and -22.12 in Y. While some velocity results were acceptable (average 34.34), others were severely underestimated or even negative, such as the throttle test which resulted in a velocity error of -70.04.

In comparison, the Simple Vehicle Dynamics Model (as presented in earlier subsections) consistently produced more accurate velocity and positional data across both trained and test datasets. Though heading results varied depending on tuning, the model showed a noticeable reduction in variance compared to VTD.

This comparison highlights the inadequacy of VTD's default vehicle dynamics for use in high-fidelity autonomous vehicle simulation. The Simple Vehicle Dynamics Model, despite being simpler and more modular, delivers superior alignment with real-world vehicle behavior, providing a more reliable foundation for simulation-based development and testing.

Tabel 10: Result of previous VTD Dynamics

Filename	Description	response heading	response posX	response posY	response vel
TC-VD-001.bag.txt	Base run	-897.19	71.75	84.54	64.00
TC-VD-011.bag.txt	Throttle	-128.83	-89.87	-89.96	-70.04
TC-VD-030.bag.txt	Hard bake	-149.81	85.08	66.62	50.85
TC-VD-040.bag.txt	Soft brake	-316.17	-187.74	-237.33	9.40
TC-VD-060.bag.txt	Left turn	35.92	-35.26	-123.98	72.92
TC-VD-061.bag.txt	Right turn	84.60	61.69	67.25	52.15
TC-VD-070.bag.txt	Left-right turn	82.05	92.30	77.99	61.11
Overall Average		-184.2039590439	-0.2937901878	-22.12489151	34.3419981

4.3 Advanced Vehicle Dynamics

4.3.1 Result Analysis

The Advanced Vehicle Dynamics Model was also evaluated using both trained and unseen test data to assess its generalization capabilities and accuracy. The evaluation focused on four key performance metrics: heading, position X, position Y, and velocity. Results were compared across three categories: trained data, test data, and a second batch of test data.

For the trained dataset, the model demonstrated strong performance with an overall average fitness of 33.30 in heading, 85.54 in position X, 88.16 in position Y, and 83.46 in velocity. These results indicate that the model was able to learn and replicate vehicle dynamics behavior effectively during development and tuning.

On the test dataset, the model maintained good accuracy with an average heading fitness of 20.51 and velocity fitness of 76.45, showing strong generalization without requiring re-tuning. The performance in position metrics also remained consistent, with 84.50 and 84.98 for position X and Y respectively.

The second batch of test data, with the same scenarios, further confirmed the robustness of the model. While the overall heading fitness slightly dropped to 22.35, the velocity prediction remained relatively accurate at 74.99, and the position fitness metrics were still within acceptable ranges.

These results demonstrate that the Advanced Vehicle Dynamics Model achieves high consistency across different datasets. Although peak performance on the trained data is slightly better, the model performs reliably even with unseen inputs, confirming its robustness and reduced reliance on dataset-specific tuning.

Table 11: Results of Advanced Vehicle Dynamics

Trained data					
Filename	Description	response heading	response posX	response posY	response vel
TC-VD-001.bag.txt	Base run	29.71	88.43	92.27	84.50
TC-VD-011.bag.txt	Throttle	45.73	93.88	94.42	89.90
TC-VD-030.bag.txt	Hard bake	5.75	79.59	82.26	79.14
TC-VD-040.bag.txt	Soft brake	23.03	89.63	88.89	83.20
TC-VD-060.bag.txt	Left turn	78.87	72.99	91.99	85.11
TC-VD-061.bag.txt	Right turn	13.74	81.99	94.26	81.86
TC-VD-070.bag.txt	Left-right turn	36.24	92.27	73.03	83.48
Overall Average		33.30	85.54	88.16	83.88
Test Data					
Filename	Description	response heading	response posX	response posY	response vel
TC-VD-001.bag.txt	Base run	4.80	89.92	94.36	83.80
TC-VD-011.bag.txt	Throttle	37.24	85.58	86.28	83.27
TC-VD-030.bag.txt	Hard bake	5.39	79.90	80.47	71.49
TC-VD-040.bag.txt	Soft brake	7.96	78.10	78.02	72.54
TC-VD-060.bag.txt	Left turn	9.96	88.57	85.59	81.91
TC-VD-061.bag.txt	Right turn	47.31	86.09	85.94	74.23
TC-VD-070.bag.txt	Left-right turn	30.89	83.36	84.16	67.94
Overall Average		20.51	84.50	84.98	76.45
Test Data2					
Filename	Description	response heading	response posX	response posY	response vel
TC-VD-001.bag.txt	Base run	1.70	85.96	87.21	81.55
TC-VD-011.bag.txt	Throttle	49.85	77.34	74.85	74.95
TC-VD-030.bag.txt	Hard bake	7.96	78.10	78.02	72.54
TC-VD-040.bag.txt	Soft brake	22.80	88.57	85.59	82.35
TC-VD-060.bag.txt	Left turn	24.99	80.07	84.04	83.12
TC-VD-061.bag.txt	Right turn	26.82	80.17	84.98	79.57
TC-VD-070.bag.txt	Left-right turn	22.34	58.91	88.76	60.39
Overall Average		22.35	78.45	83.35	76.35

4.3.2 Comparison with Simple Dynamics

The Advanced Vehicle Dynamics Model was compared against the Simple Vehicle Dynamics Model using the same test cases and evaluation metrics. On the trained dataset, both models performed similarly, even with the Simple Model achieving slightly better results in some scenarios. This is expected, as the Simple Model was directly tuned using the training data, allowing it to closely fit specific patterns. However, overall the Advanced model performs better.

The real advantage of the Advanced Model is seen in the evaluation on unseen data. While the Simple Model showed a significant drop in performance, especially in heading accuracy and velocity tracking, the Advanced Model maintained consistent and reliable results. For example, on Test Data 2, the Advanced Model achieved much higher accuracy across all metrics, including heading, position, and velocity, compared to the Simple Model, which struggled due to its reliance on tuned parameters.

This demonstrates that while both models are capable of fitting known data, the Advanced Model generalizes significantly better. Its physics-based subsystems such as the engine, chassis, and gear-shifting models allow it to respond more realistically to different scenarios without the need for returning, making it a more robust and scalable solution.

4.4 Conclusion

The VTD dynamics showed large deviations in key metrics such as heading and position. It lacked tunability and failed to replicate real vehicle behavior in various scenarios, making it unsuitable for high-fidelity simulations.

The simple vehicle dynamics model demonstrated strong performance on known data but required careful parameter tuning to achieve accuracy. It lacked flexibility across varying conditions and did not adapt well to new input data.

In contrast, the advanced model, though initially more complex but only slightly more accurate on the tuned dataset, proved to be more robust. Its reliance on physical quantities and a data-driven torque map allowed it to generalize better to new test scenarios. Given more time for optimization and refinement, the advanced model has strong potential to surpass the simple model in both accuracy and realism.

5. Project management

The vehicle dynamics capstone project was managed using a structured and phased approach over two academic semesters. The development was divided into 2 clear stages with defined deliverables. Task planning and tracking were carried out using Jira, which was instrumental in managing priorities, deadlines, and sprint planning. Tasks were categorized by type (e.g., research, development, testing), assigned priority levels, and tracked weekly to ensure consistent progress. Each item logged in Jira allowed updates, time tracking and clear visibility into the project's path.

The project began with the Simple Vehicle Dynamics Model, including throttle, brake and steering components built using a Linear Parameter Varying (LPV) model and a kinematic bicycle model. This phase also involved creating testing utilities such as the graph plotting tool and log comparison scripts. In Semester 2, the focus shifted to the Advanced Vehicle Dynamics Model, which introduced subsystems for chassis motion, engine and transmission simulation, lateral tire forces and gear shifting. Each subsystem was validated individually using unit tests and real-world data, then integrated progressively to avoid compounded errors. The following table highlights each task, type, priority, time spent and timeline.

Table 12: Project management table

Task	Type	Priority	Time Spent	Timeline
Vehicle Dynamics Models	Research	high	6 hours	Sem 1 Week 1
Planning of Simple Vehicle Dynamics	Documentation	high	4 hours	Sem 1 Week 1
Linear Varying Parameter Model (LPV)	C++ Development	high	8 hours	Sem 1 Week 2
Unit test for LPV model	C++ Testing	low	7 hours	Sem 1 Week 2
Velocity Model (Combine throttle and brake LPV)	C++ Development	high	8 hours	Sem 1 Week 3
Unit test for velocity model	C++ Testing	low	4 hours	Sem 1 Week 3

Log data comparison tool	Python Development	mid	16 hours	Sem 1 Week 3
Graph plotting and fitness calculator tool	Python Development	mid	16 hours	Sem 1 Week 4
Velocity comparison with log data	Testing	high	5 hours	Sem 1 Week 4
Tuning of LPV parameters	Refinement	mid	6 hours	Sem 1 Week 4
Kinematic bicycle model (KB)	C++ Development	high	6 hours	Sem 1 Week 5
Unit test for KB model	C++ Testing	low	4 hours	Sem 1 Week 5
Steering model	C++ Development	high	8 hours	Sem 1 Week 6
Heading comparison with log data	Testing	mid	3 hours	Sem 1 Week 6
Position comparison with log data	Testing	mid	3 hours	Sem 1 Week 6
Vehicle steering	Research	low	6 hours	Sem 1 Week 7
Modification to Steering Model	C++ Development	low	4 hours	Sem 1 Week 7
Simple Vehicle Dynamics Model	C++ Development	high	10 hours	Sem 1 Week 8
Compasion with log data	Testing	high	8 hours	Sem 1 Week 8
Parameter Optimization Tool	Python Development	mid	10 hours	Sem 1 Week 9
Tuning of simple vehicle dynamics model	Refinement	mid	10 hours	Sem 1 Week 10

Simple vehicle dynamics results	Documentation	mid	6 hours	Sem 1 Week 11
Deployment to VTD simulation	Integration	high	14 hours	Sem 1 Week 12
Deployment to Simple Vehicle simulator	Integration	low	8 hours	Sem 2 Week 1
Vehicle Config loader class	C++ Development	low	8 hours	Sem 2 Week 1
Vehicle command and state class	C++ Development	low	2 hours	Sem 2 Week 2
Vehicle dynamics base class	C++ Development	mid	5 hours	Sem 2 Week 2
Advanced vehicle dynamics models	Research	mid	12 hours	Sem 2 Week 3
Planning of advanced vehicle dynamics	Documentation	mid	8 hours	Sem 2 Week 3
Lateral tyre force model	C++ Development	mid	8 hours	Sem 2 Week 4
Unit test for lateral tyre force	C++ Testing	low	4 hours	Sem 2 Week 4
Motion base class	C++ Development	mid	2 hour	Sem 2 Week 5
X motion	C++ Development	mid	8 hours	Sem 2 Week 5
Y motion	C++ Development	mid	10 hours	Sem 2 Week 5
Chassis model	C++ Development	mid	8 hours	Sem 2 Week 5
Unit test for chassis model	C++ Testing	low	4 hours	Sem 2 Week 5

Heading compassion with log data	Testing	mid	3 hours	Sem 2 Week 6
Position compassion with log data	Testing	mid	4 hours	Sem 2 Week 6
Tuning of lateral tyre force model	Refinement	low	5 hours	Sem 2 Week 6
Vehicle gear shifting	Research	low	4 hours	Sem 2 Week 7
Gear shift model	C++ Development	mid	7 hours	Sem 2 Week 7
Unit test for gear shift model	C++ testing	low	4 hours	Sem 2 Week 7
Gear number comparison with log data	Testing	mid	4 hours	Sem 2 Week 8
Script to estimate gear shift map	Python Development	low	7 hours	Sem 2 Week 8
Tuning of gear shift model	Refinement	low	4 hours	Sem 2 Week 8
Diesel engine and transmission	Research	mid	5 hours	Sem 2 Week 9
Engine model	C++ Development	mid	8 hours	Sem 2 Week 9
Transmission model	C++ Development	mid	6 hours	Sem 2 Week 9
Script to generate torque map	Python Development	low	12 hours	Sem 2 Week 10
Script to estimate final drive ratio	Python Development	low	6 hours	Sem 2 Week 10

Velocity Comparison with log data	Testing	mid	5 hours	Sem 2 Week 10
Tuning of Engine model with torque map	Refinement	low	8 hours	Sem 2 Week 10
Advanced Vehicle dynamics model	C++ Development	mid	10 hours	Sem 2 Week 10
Tuning of advanced vehicle dynamics model	Refinement	low	12 hours	Sem 2 Week 11
Advanced vehicle dynamics results	Documentation	low	6 hours	Sem 2 Week 11
Deployment to VTD simulation	Integration	mid	2 hour	Sem 2 Week 11
Deployment to Simple Vehicle simulator	Integration	low	2 hour	Sem 2 Week 11

6. Conclusion

This capstone project focuses on developing an advanced vehicle dynamics model to address the limitations of existing generalized simulation tools, such as VTD, which fail to accurately replicate the nuanced behavior of autonomous vehicles (AVs). The project delivers a comprehensive solution tailored to Venti Technologies' AV systems, with a focus on enhancing simulation fidelity, scalability and real-time performance.

The development is structured into two key stages: the Simple Vehicle Dynamics Model and the Advanced Vehicle Dynamics Model. The simple model establishes a foundation by incorporating throttle, brake and steering dynamics using a Linear Parameter Varying (LPV) framework and a kinematic bicycle model. The advanced model builds upon this foundation by integrating more complex subsystems, including a chassis model for vehicle motion dynamics, an engine model for torque generation, a transmission model for power delivery and a gear-shifting model. These components collectively simulate intricate interactions between subsystems, enabling realistic outputs for velocity, position and heading.

Supporting tools were developed to optimize the workflow, including a GUI-based graph plotting tool for visualizing velocity outputs, a simulation automation script for validating the model against real-world vehicle data, an optimization tool for fine-tuning parameters and a ROS bag splitting and merging script to manage large datasets effectively. These tools streamline the evaluation, comparison and refinement processes, ensuring efficient and accurate model development.

The resulting vehicle dynamics model will provide a modular, scalable and high-fidelity framework capable of simulating real-world behavior in diverse scenarios. By achieving a strong correlation with actual vehicle performance and aligning with the requirements of Venti Technologies' AV systems, this project lays the groundwork for future advancements in AV simulation and testing.

To further enhance the accuracy and flexibility of the vehicle dynamics model, several improvements are proposed for future work. First, expanding the model to support different vehicle types such as rear-steered systems and electric drivetrains would significantly broaden its applicability across more vehicle types. Currently, the model is limited to a diesel-powered, front-steered prime mover configuration. Incorporating terrain modeling is also recommended. By accounting for factors such as slope and elevation, the model can better simulate real-world road conditions, improving fidelity in scenarios involving gradients and uneven surfaces. These enhancements would make the model more accurate, adaptable and robust for broader autonomous vehicle applications.

7. References

Chen, K. (2013). A review of the application of VTD simulation software in intelligent driving. ResearchGate.
https://www.researchgate.net/publication/377501269_A_Review_of_The_Application_of_VTD_Simulation_Software_in_Intelligent_Driving

MathWorks. (2024). *LPV Model of Engine Throttle*. MathWorks. https://www.mathworks.com/help/control/ug/lpv-model-of-engine-throttle.html

Zhang, S. (2014, November). *LPV Modeling and Mixed Constrained H2 H∞ Control of an Electronic Throttle*. IEEE Xplore.
https://ieeexplore.ieee.org/document/6954436

Zhang, T. (2024, March). *A Survey of Vehicle Dynamics Modeling Methods for Autonomous Racing*. IEEE Xplore.
https://ieeexplore.ieee.org/document/10384847

8 Knowledge and Training Requirements

The section lists all the knowledge, skillsets and certifications both from the degree programme and beyond that was necessary for the successful completion of the capstone projects.

8.1 Applicable Knowledge from the Degree Programme

The prerequisite knowledge and skill sets from the degree programme that was necessary for the capstone projects are as follows:

No.	Module(s)	Knowledge(s) Applied
1	CSD1170: High-Level Programming 2	Provided a strong base in object-oriented programming using C++, which was critical for implementing model components and class-based architecture.
2	CSD1130: Game Implementation Techniques	Introduced the concept of delta time, which was essential for updating model states accurately, especially when handling variable time steps during simulation.
3	CSD1240: Linear Algebra and Geometry	Provided the mathematical foundation for transforming vehicle position and orientation between different coordinate frames, which is crucial in simulation and result comparison. Applied interpolation techniques in the engine torque map and throttle/rpm modeling to estimate values between discrete data points.
4	CSD2181: Data Structures	Supported the use of efficient data handling methods such as circular buffers, queues and input storage for LPV models.
5	CSD2300: Motion Dynamics and Lab	Offered foundational understanding of physics and vehicle kinematics such F=ma, which was directly applied to modeling acceleration, drag and lateral tyre forces.
6	CSD2400–3450: Software Engineering Projects	Reinforced skills in software development life cycle, project planning, modular development and version control, used throughout the capstone.
7	CSD2180: Operating Systems	Provided insights into thread management and real-time data processing, ensuring smooth execution of ros bag playback and logging scripts in parallel
8	CSD2200: Calculus and Analytic Geometry 2	Enabled modeling of motion by applying integration techniques to compute vehicle position and heading over time from velocity and yaw rate data.
9	CSD3185: Machine Learning	Reinforced the importance of data integrity by applying best practices in splitting tuning and testing data. This separation helped ensure unbiased model evaluation and fair performance assessment.
10	CSD3240: Probability and Statistics	Aided in data analysis, especially for calculating fitness metrics like NRMSE and interpreting test results.
11	CSD3156: Mobile and Cloud Computing	Introduced the concept of containerization, which was used to run ROS bag comparisons efficiently using Docker containers in parallel.

8.2 Additional Knowledge, Skill Sets, or Certifications Required

The following are the additional requirements and knowledge that were required for for the capstone projects:

No.	Additional Requirement(s)	Knowledge(s) Applied
1	ROS (Robot Operating System)	Required for integrating the vehicle dynamics model into a real-time simulation framework, handling topics, services and nodes for communication.
2	Vehicle Dynamics Domain Knowledge	Studied independently to understand real-world behavior of vehicles including torque, gear logic, chassis dynamics and tyre forces.
3	Python and Bash Scripting	Used to develop automation and data processing tools such as log parsing, fitness evaluation and GUI-based visualization.
4	Creating Docker & Docker Compose files	Used to containerize and run the simulation environment, including automated playback of ROS bag files in parallel for efficient testing.
5	Google Test (gtest)	Used to write and run unit tests for each component of the vehicle dynamics model to ensure functionality and correctness.
6	CMake & Bazel Build Systems	Used to configure and build the C++ project, especially for managing dependencies in ROS-based development.

Appendix A: Vehicle Test Cases For Data Collection

Base Case Test

No.	Description	Test steps	Notes	Start - End
TC-VD-001	25km/h Straight base run	1. APM starts stationary and straight 2. APM goes straight along **main** road 3. APM accelerates to is max speed limit (**25km/h**) 4. APM comes to a stop	• Try to keep vehicle going in a straight line	**Main Road** near TD **TD01 1 - TD01 30**
TC-VD-002	15km/h Straight base run	1. APM starts stationary and straight 2. APM goes straight along **travelling** lane 3. APM accelerates to is max speed limit (**15km/h**) 4. APM comes to a stop	• Try to keep vehicle going in a straight line	**Main Road** near TD **TD01 1 - TD01 30**
TC-VD-003	10km/h Straight rain run	1. APM starts stationary and straight 2. Publish `simulate_rain_level perc_msgs::RainLevel 'rain_level: 3, is_wet_ground: 0'` (cap speed at 10km/h) 3. APM goes straight along any straight road 4. APM accelerates to is max speed limit (**10km/h**) 5. APM comes to a stop	• Try to keep vehicle going in a straight line	**Main Road** near TD TD01 1 - TD01 30

Throttle Case Test

No.	Description	Test steps	Notes	Start - End
TC-VD-010	full throttle run (manual mode)	1. APM starts stationary and straight 2. APM goes straight along any straight road 3. APM accelerates to its max speed <= **40kph** 4. APM coast to idle speed (**no brake, no throttle**) 5. APM brake after 5 seconds travelling at idle speed 6. APM comes to a stop	• Try to keep vehicle going in a straight line	Main Road near TD Start: **TD01 1** End: **TD01 30**
TC-VD-011	full throttle run (manual mode)	1. APM starts stationary and straight 2. APM goes straight along any straight road 3. APM accelerates to **25 km/h** 4. APM coast to idle speed (**no brake, no throttle**) 5. APM brake after 5 seconds travelling at idle speed 6. APM comes to a stop	• Try to keep vehicle going in a straight line	Main Road near TD Start: **TD01 1** End: **TD01 30**
TC-VD-012	full throttle run (manual mode)	1. APM starts stationary and straight 2. APM goes straight along any straight road 3. APM accelerates to **20 km/h** 4. APM coast to idle speed (**no brake, no throttle**) 5. APM brake after 5 seconds travelling at idle speed 6. APM comes to a stop	• Try to keep vehicle going in a straight line	Main Road near TD Start: **TD01 1** End: **TD01 30**
TC-VD-013	full throttle run (manual mode)	1. APM starts stationary and straight 2. APM goes straight along any straight road 3. APM accelerates to **15 km/h** 4. APM coast to idle speed (**no brake, no throttle**) 5. APM brake after 5 seconds travelling at idle speed 6. APM comes to a stop	• Try to keep vehicle going in a straight line	Main Road near TD Start: **TD01 1** End: **TD01 30**
TC-VD-014	full throttle run (manual mode)	1. APM starts stationary and straight 2. APM goes straight along any straight road 3. APM accelerates to **10 km/h** 4. APM coast to idle speed (**no brake, no throttle**) 5. APM brake after 5 seconds travelling at idle speed 6. APM comes to a stop	• Try to keep vehicle going in a straight line	Main Road near TD Start: **TD01 1** End: **TD01 30**

Brake Case Test

No.	Description	Test steps	Notes	
TC-VD-030	25km/h Emergency Stop	1. APM starts stationary and straight 2. APM goes straight along any straight road 3. APM accelerates to **25km/h** then 4. Press emergency stop button 5. APM comes to a stop	• Try to keep vehicle going in a straight line	Main Road near TD Start: **TD01 1** End: **TD01 30**
TC-VD-031	20km/h Emergency Stop	1. APM starts stationary and straight 2. APM goes straight along any straight road 3. APM accelerates to **20km/h** then 4. Press emergency stop button 5. APM comes to a stop	• Try to keep vehicle going in a straight line	Main Road near TD Start: **TD01 1** End: **TD01 30**
TC-VD-032	15km/h Emergency Stop	1. APM starts stationary and straight 2. APM goes straight along any straight road 3. APM accelerates to **15km/h** then 4. Press emergency stop button 5. APM comes to a stop	• Try to keep vehicle going in a straight line	Main Road near TD Start: **TD01 1** End: **TD01 30**
TC-VD-033	10km/h Emergency Stop	1. APM starts stationary and straight 2. APM goes straight along any straight road 3. APM accelerates to **10km/h** then 4. Press emergency stop button 5. APM comes to a stop	• Try to keep vehicle going in a straight line	Main Road near TD Start: **TD01 1** End: **TD01 30**
TC-VD-034	5km/h Emergency Stop	1. APM starts stationary and straight 2. APM goes straight along any straight road 3. APM accelerates to **5km/h** then 4. Press emergency stop button 5. APM comes to a stop	• Try to keep vehicle going in a straight line	Main Road near TD Start: **TD01 1** End: **TD01 30**

Brake Case Test

No.	Description	Test steps	Notes	
TC-VD-035	25km/h **Hard Stop**	1. APM starts stationary and straight 2. APM goes straight along any straight road 3. APM accelerates to **25km/h** then 4. Publish `simulate_arbitrator_stop std_msgs/Int8 'data: 3'` 5. APM comes to a stop	• Try to keep vehicle going in a straight line	Main Road near TD Start: **TD01 1** End: **TD01 30**
TC-VD-036	20km/h Hard Stop	1. APM starts stationary and straight 2. APM goes straight along any straight road 3. APM accelerates to **20km/h** then 4. Publish `simulate_arbitrator_stop std_msgs/Int8 'data: 3'` 5. APM comes to a stop	• Try to keep vehicle going in a straight line	Main Road near TD Start: **TD01 1** End: **TD01 30**
TC-VD-037	15km/h Hard Stop	1. APM starts stationary and straight 2. APM goes straight along any straight road 3. APM accelerates to **15km/h** then 4. Publish `simulate_arbitrator_stop std_msgs/Int8 'data: 3'` 5. APM comes to a stop	• Try to keep vehicle going in a straight line	Main Road near TD Start: **TD01 1** End: **TD01 30**
TC-VD-038	10km/h Hard Stop	1. APM starts stationary and straight 2. APM goes straight along any straight road 3. APM accelerates to **10km/h** then 4. Publish `simulate_arbitrator_stop std_msgs/Int8 'data: 3'` 5. APM comes to a stop	• Try to keep vehicle going in a straight line	Main Road near TD Start: **TD01 1** End: **TD01 30**
TC-VD-039	5km/h Hard Stop	1. APM starts stationary and straight 2. APM goes straight along any straight road 3. APM accelerates to **5km/h** then 4. Publish `simulate_arbitrator_stop std_msgs/Int8 'data: 3'` 5. APM comes to a stop	• Try to keep vehicle going in a straight line	Main Road near TD Start: **TD01 1** End: **TD01 30**

Brake Case Test

No.	Description	Test steps	Notes	
TC-VD-040	25km/h **Soft Stop**	1. APM starts stationary and straight 2. APM goes straight along any straight road 3. APM accelerates to **25km/h** then 4. Publish `simulate_arbitrator_stop std_msgs/Int8 'data: 2'` 5. APM comes to a stop	• Try to keep vehicle going in a straight line	Main Road near TD Start: **TD01 1** End: **TD01 30**
TC-VD-041	20km/h Soft Stop	1. APM starts stationary and straight 2. APM goes straight along any straight road 3. APM accelerates to **20km/h** then 4. Publish `simulate_arbitrator_stop std_msgs/Int8 'data: 2'` 5. APM comes to a stop	• Try to keep vehicle going in a straight line	Main Road near TD Start: **TD01 1** End: **TD01 30**
TC-VD-042	15km/h Soft Stop	1. APM starts stationary and straight 2. APM goes straight along any straight road 3. APM accelerates to **15km/h** then 4. Publish `simulate_arbitrator_stop std_msgs/Int8 'data: 2'` 5. APM comes to a stop	• Try to keep vehicle going in a straight line	Main Road near TD Start: **TD01 1** End: **TD01 30**
TC-VD-043	10km/h Soft Stop	1. APM starts stationary and straight 2. APM goes straight along any straight road 3. APM accelerates to **10km/h** then 4. Publish `simulate_arbitrator_stop std_msgs/Int8 'data: 2'` 5. APM comes to a stop	• Try to keep vehicle going in a straight line	Main Road near TD Start: **TD01 1** End: **TD01 30**
TC-VD-044	5km/h Soft Stop	1. APM starts stationary and straight 2. APM goes straight along any straight road 3. APM accelerates to **5km/h** then 4. Publish `simulate_arbitrator_stop std_msgs/Int8 'data: 2'` 5. APM comes to a stop	• Try to keep vehicle going in a straight line	Main Road near TD Start: **TD01 1** End: **TD01 30**

Steering Case Test

No.	Description	Test steps	Notes	Start -End
TC-VD-060	4 Left turns	1. APM starts stationary 2. APM turns left around a block 4 times 3. APM comes to a stop		Start: **TH01L11 36** End: **TH01L11 08**
TC-VD-061	4 Right turns	1. APM starts stationary, end point behind start point 2. APM turns right around a block 4 times 3. APM comes to a stop		Start: **TG07L?? 31** End: **TG07L?? 04**
TC-VD-062	Across junction left	1. APM starts stationary 2. APM goes across junction on the left lane 3. APM comes to a stop		Start: Between TJ03 30 and TJ04 30 **Travelling lane** End: **TK03L11 08**
TC-VD-063	Across junction right	1. APM starts stationary 2. APM goes across junction on the right lane 3. APM comes to a stop		Start: Between TH03 30 and TH04 30 **Travelling lane** End: **TJ04L00 08**

Steering Case Test

No.	Description	Test steps	Notes	Start -End
TC-VD-064	Left turn to small slot left lane	1. APM starts stationary 2. APM Left turn to small slot left lane 3. APM comes to a stop		Start: **Main road** Near TH02 1 End: **TH03L11 6**
TC-VD-065	Left turn to small slot right lane	1. APM starts stationary 2. APM Left turn to small slot left lane 3. APM comes to a stop		Start: **Main road** Near TH02 1 End: **TH04L00 6**
TC-VD-066	Right turn to small slot left lane	1. APM starts stationary 2. APM Left turn to small slot left lane 3. APM comes to a stop		Start: **Main road** Near TH05 1 End: **TH03L11 6**
TC-VD-067	Right turn to small slot right lane	1. APM starts stationary 2. APM Left turn to small slot left lane 3. APM comes to a stop		Start: **Main road** Near TH05 1 End: **TH04L00 6**

Steering Case Test

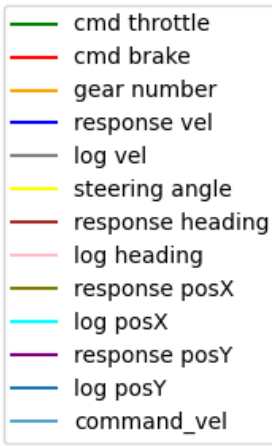
No.	Description	Test steps	Notes	Start -End
TC-VD-068	Right lane change	1. APM starts stationary 2. APM Right lane change 3. APM comes to a stop		Start: **TH02L11 2** End: **TH03L00 34**
TC-VD-069	Left lane change	1. APM starts stationary 2. APM Left lane change 3. APM comes to a stop		Start: **TH03L00 3** End: **TJ02L11 34**
TC-VD-070	Left right turns	1. APM starts stationary 2. APM right lane change 3. APM turn right 4. APM turn left 5. APM left lane change 6. APM comes to a stop		Start: **TH02L11 3** End: **TJ03L11 34**

Appendix B: Results of Simple Vehicle Dynamics

The results of the simple vehicle dynamics model were validated through vehicle tests using specifically designed test cases to compare the simulation with real-world vehicle behavior. These test cases were meticulously crafted to isolate individual factors, including throttle-velocity relationships, brake-throttle interactions and steering-position dynamics. The model demonstrated consistent performance in replicating these factors, achieving 70% fidelity in key metrics such as velocity, acceleration and heading.

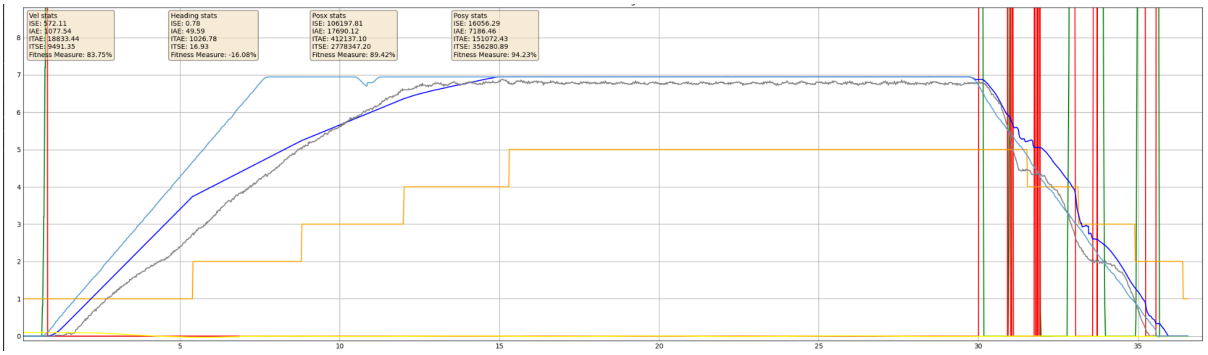
This legend provides a comprehensive guide to the variables utilized in the simple vehicle dynamics model test.

- Green: Command throttle input given to the model.
- Red: Command brake input.
- Orange: Gear number indicating the current transmission state.
- Blue: Response velocity from the simulation model.
- Gray: Logged velocity data from vehicle for validation.
- Yellow: Steering angle input to the model.
- Dark Red: Response heading output from the model.
- Pink: Logged heading data for validation.
- Teal: Response X-coordinate position.
- Light Cyan: Logged X-coordinate position.
- Purple: Response Y-coordinate position.
- Navy Blue: Logged Y-coordinate position.
- Light Blue: Command velocity input for trajectory adjustments.

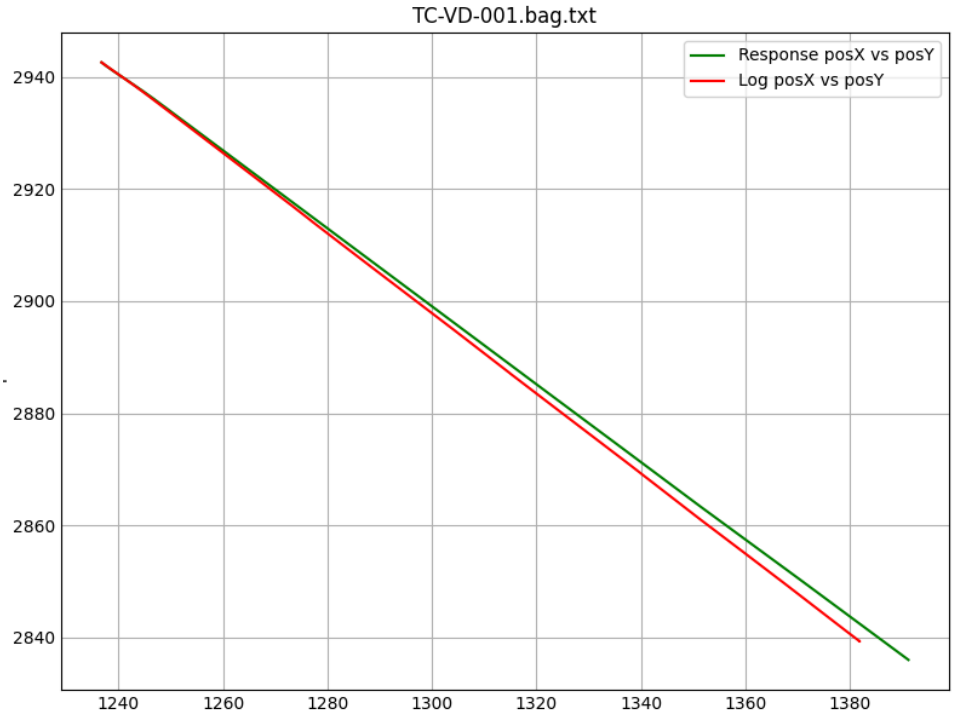


Test case Description: Vehicle goes straight at 25 km/h

Velocity Graph:



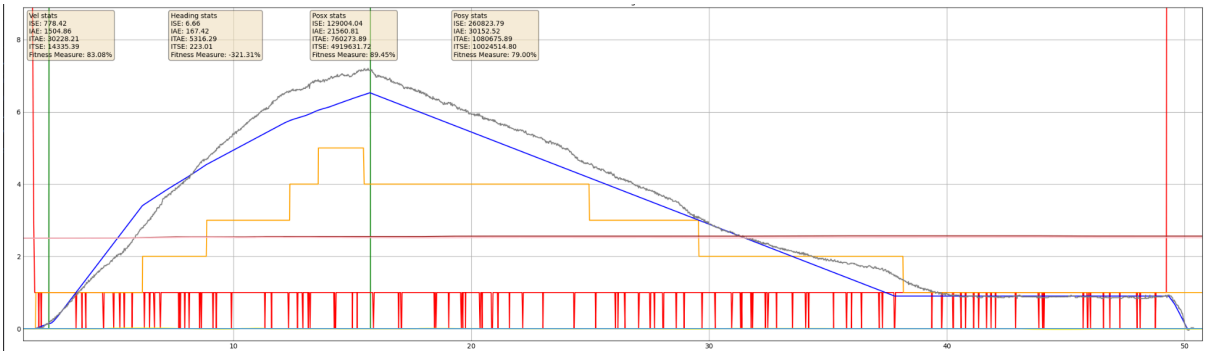
Position Graph:



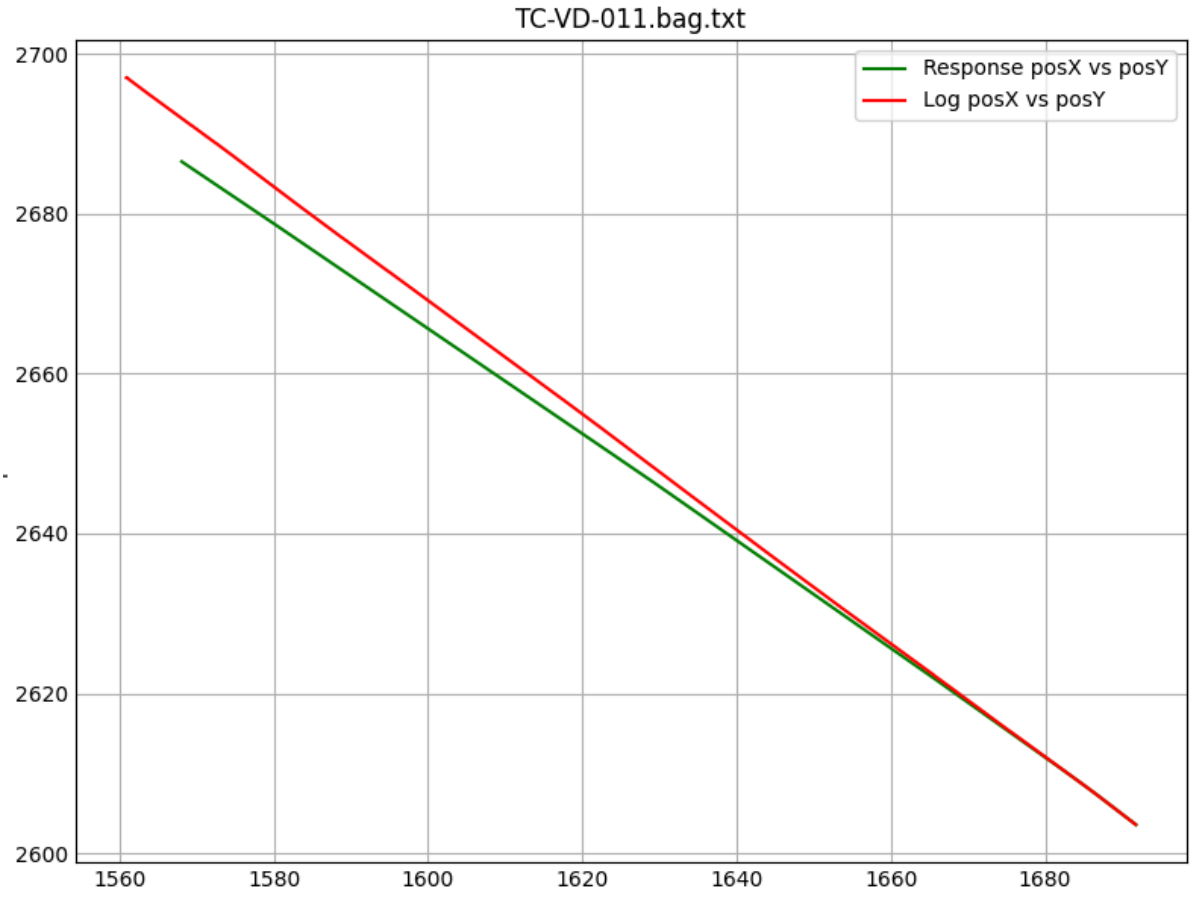
Graph Fitness:
Velocity: 83.75
Heading: -16.08
Position X: 89.42
Position Y: 94.23

Test case Description: Vehicle goes full throttle run in manual mode

Velocity Graph:



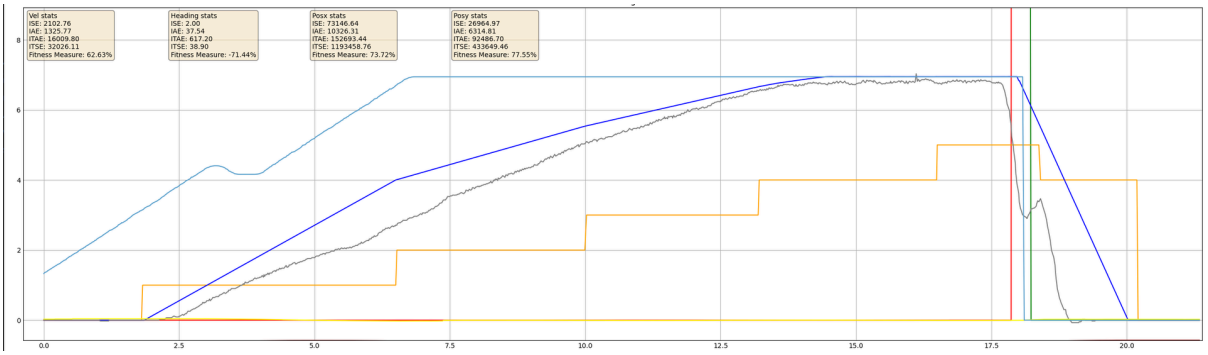
Position Graph:



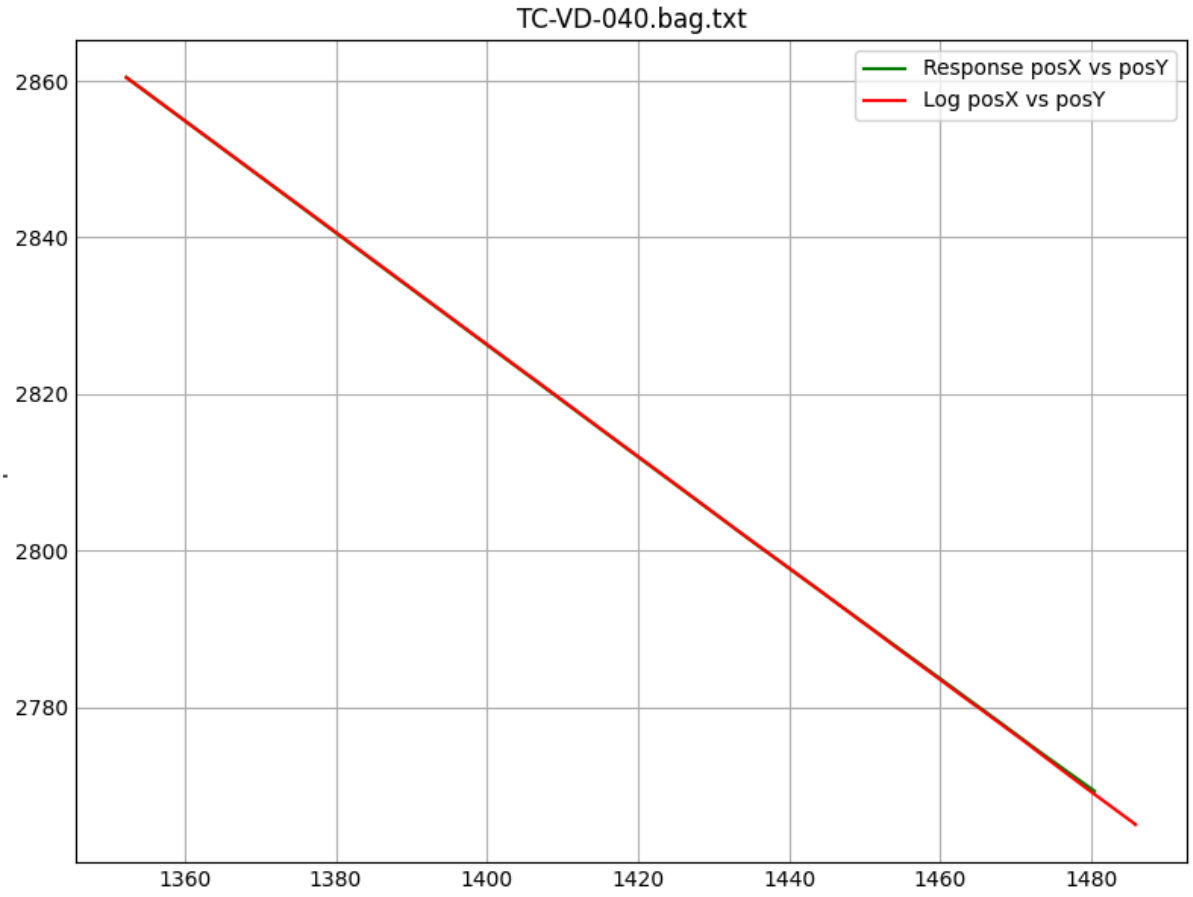
Graph Fitness:
Velocity: 83.08
Heading: -321.31
Position X: 89.45
Position Y: 79.00

Test case Description: Vehicle does a emergency Stop at 25 km/h

Velocity Graph:



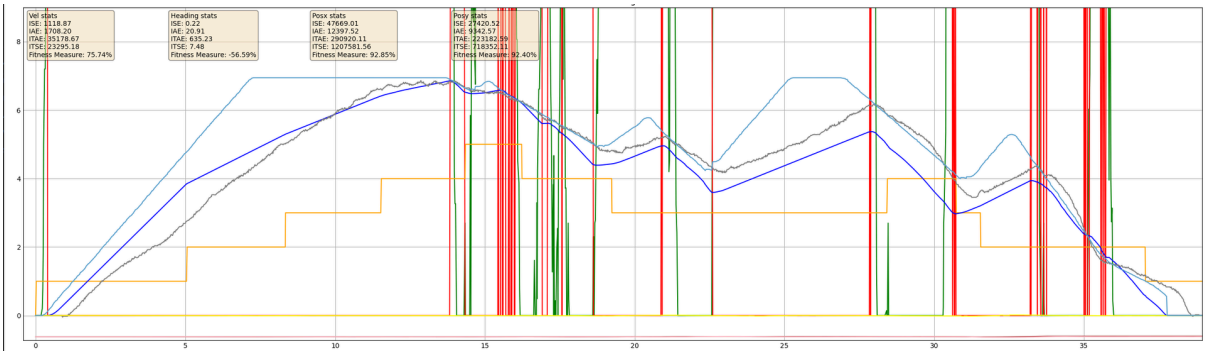
Position Graph:



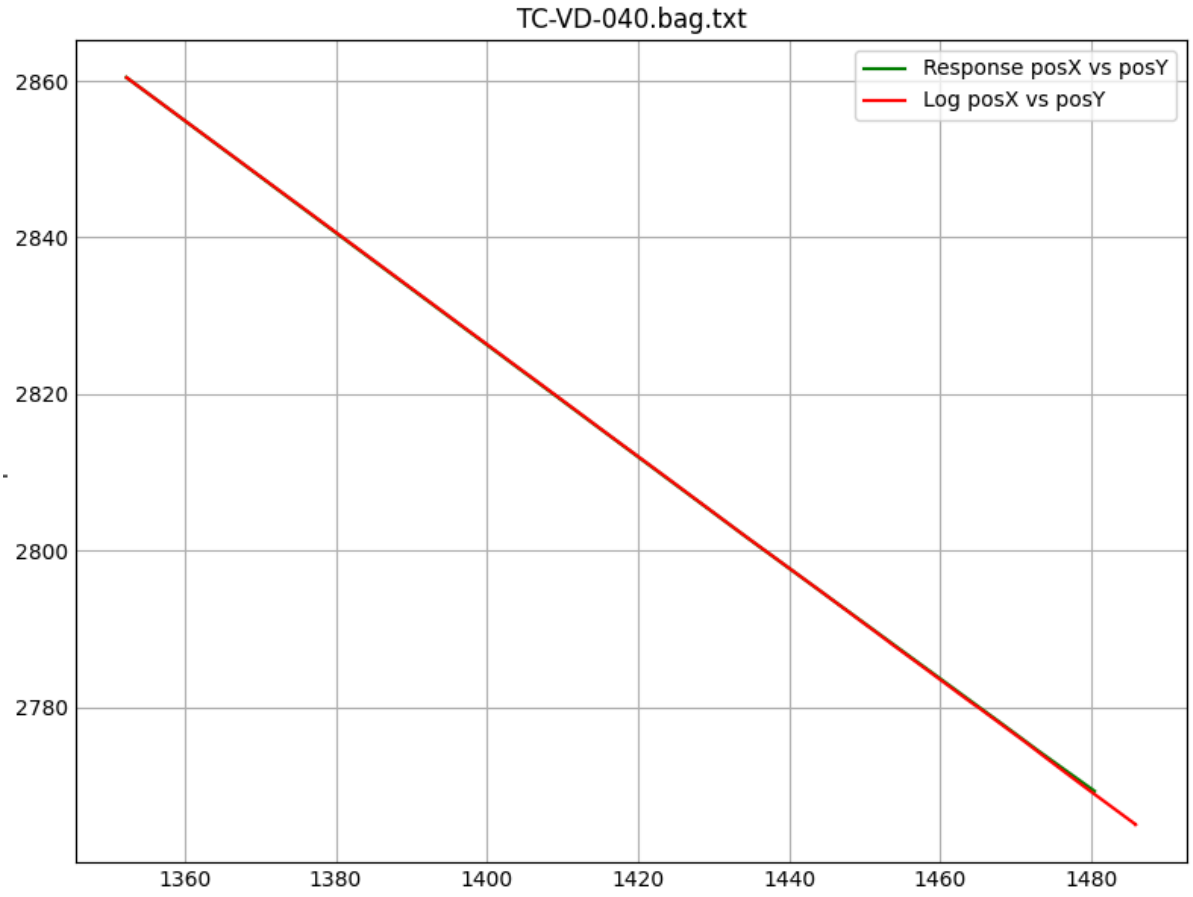
Graph Fitness:
Velocity: 62.63
Heading: -71.44
Position X: 73.72
Position Y: 77.55

Test case Description: Vehicle does a soft Stop at 25 km/h

Velocity Graph:



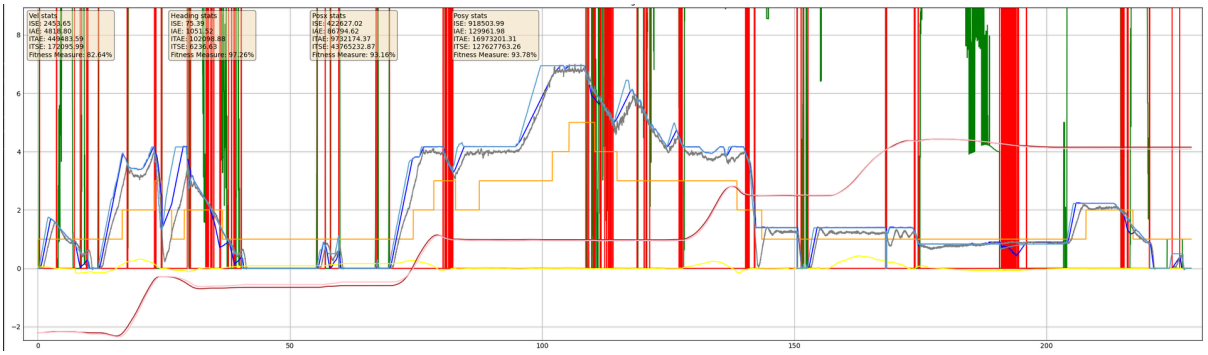
Position Graph:



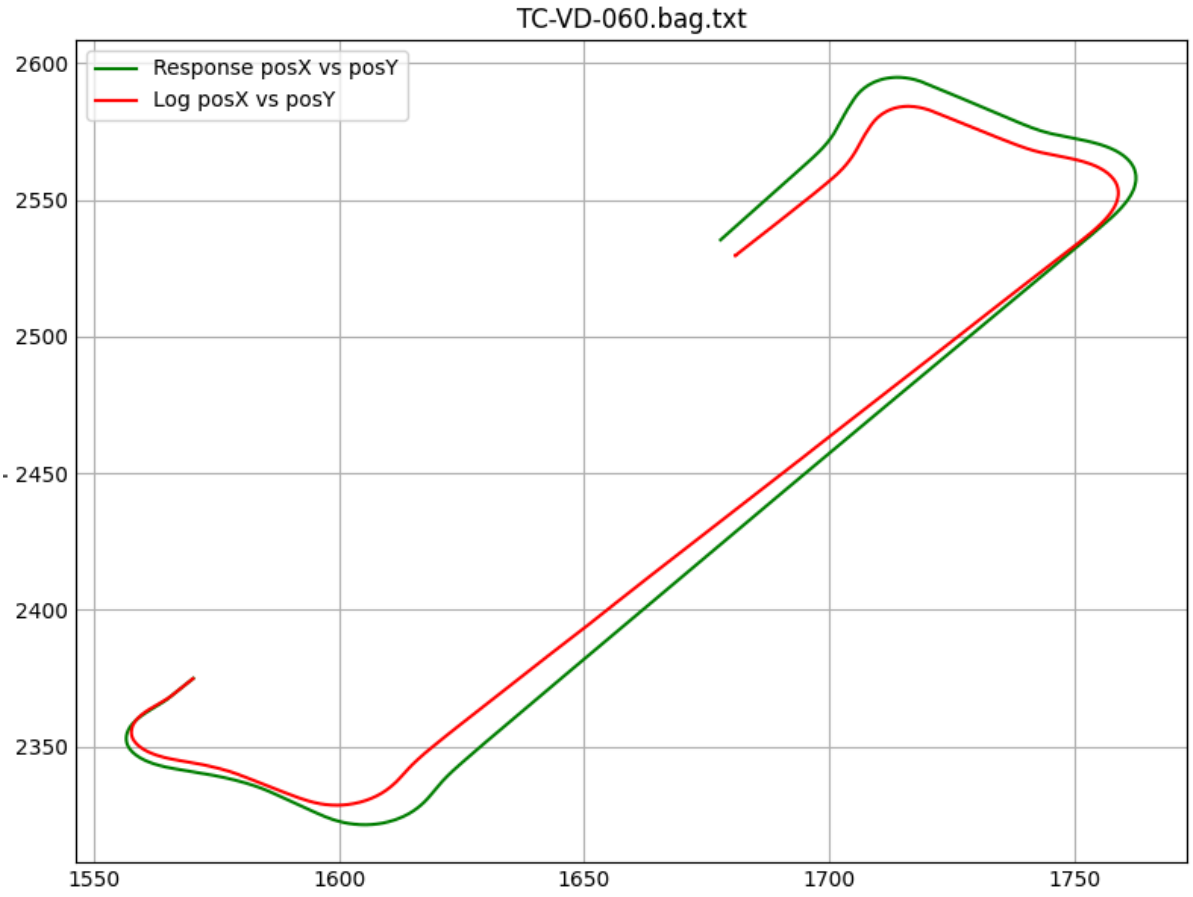
Graph Fitness:
Velocity: 75.74
Heading: -56.59
Position X: 92.85
Position Y: 92.40

Test case Description: Vehicle makes 4 Left turns

Velocity Graph:



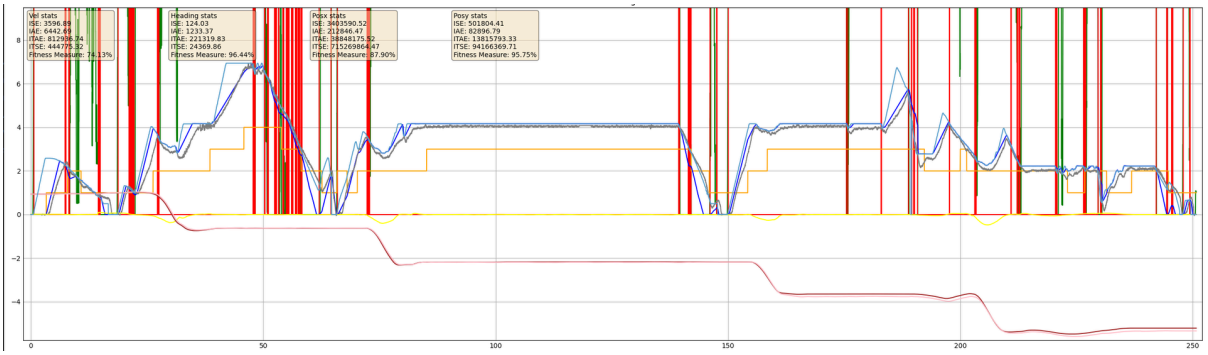
Position Graph:



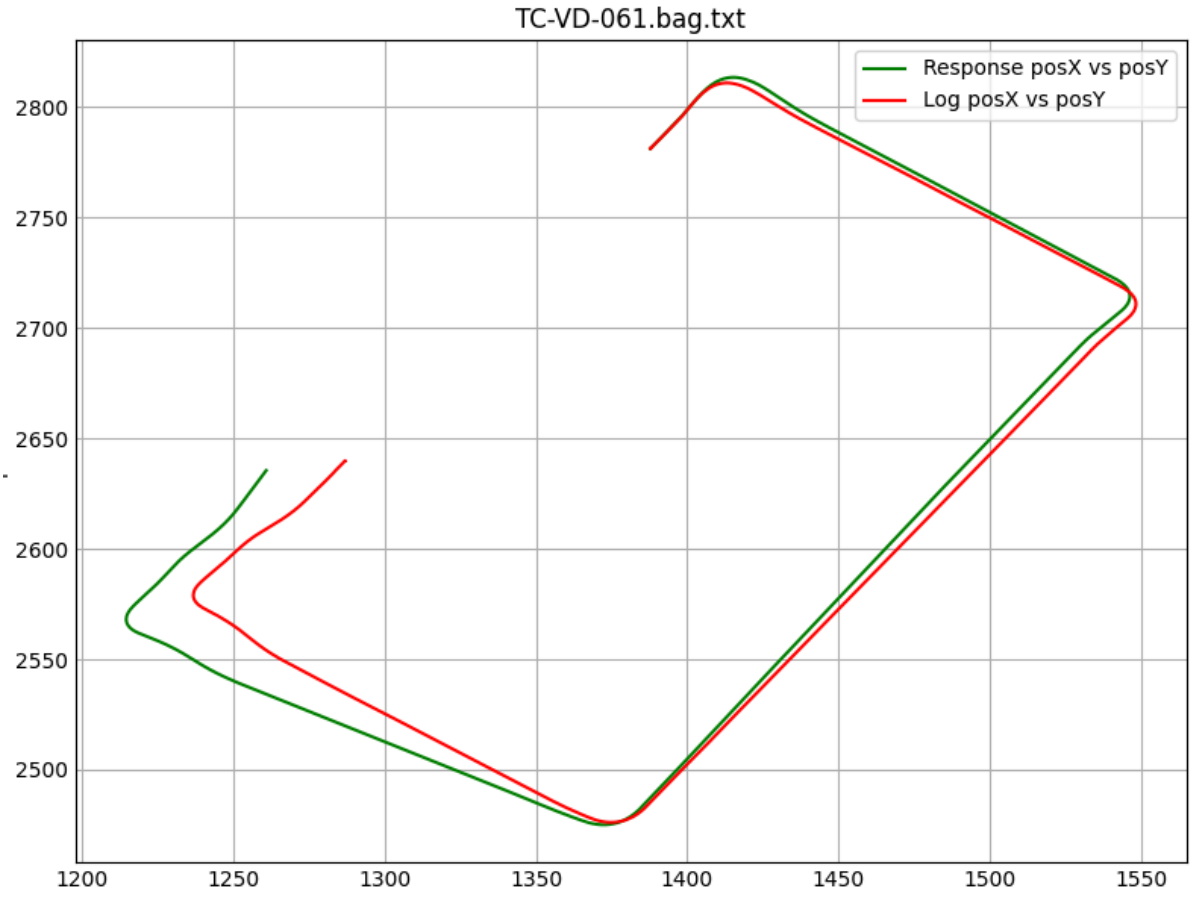
Graph Fitness:
Velocity: 82.64
Heading: 97.26
Position X: 93.16
Position Y: 93.78

Test case Description: Vehicle makes 4 Right turns

Velocity Graph:



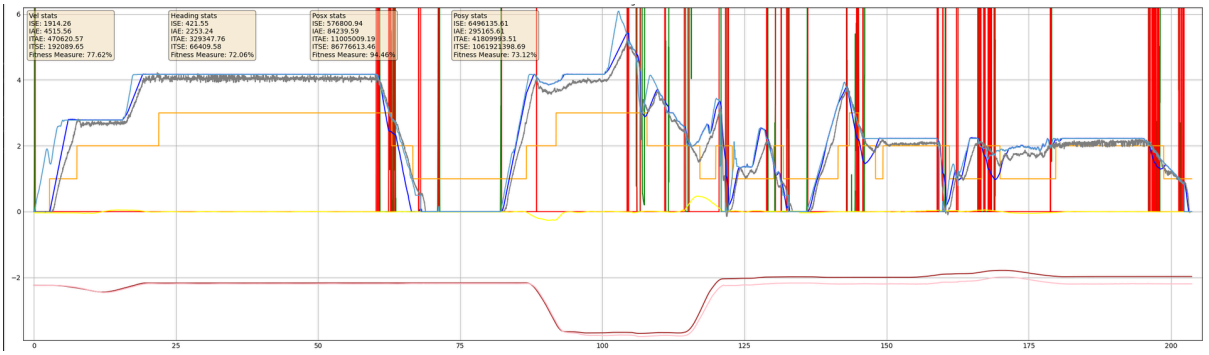
Position Graph:



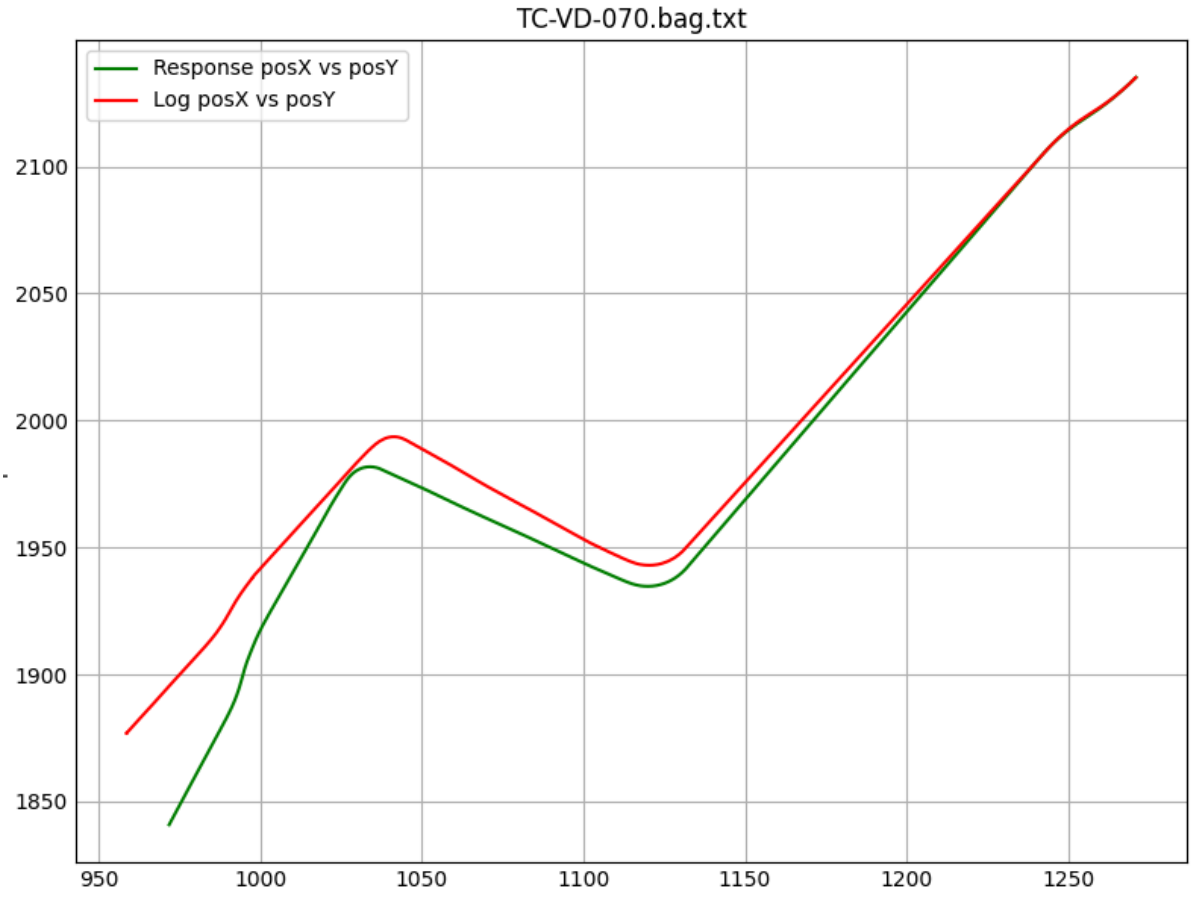
Graph Fitness:
Velocity: 74.13
Heading: 96.44
Position X: 87.90
Position Y: 95.75

Test case Description: Vehicle makes Left turn then right turn

Velocity Graph:



Position Graph:



Graph Fitness:
Velocity: 77.62
Heading: 72.06
Position X: 94.46
Position Y: 73.12

Appendix C: Results of Advanced Vehicle Dynamics

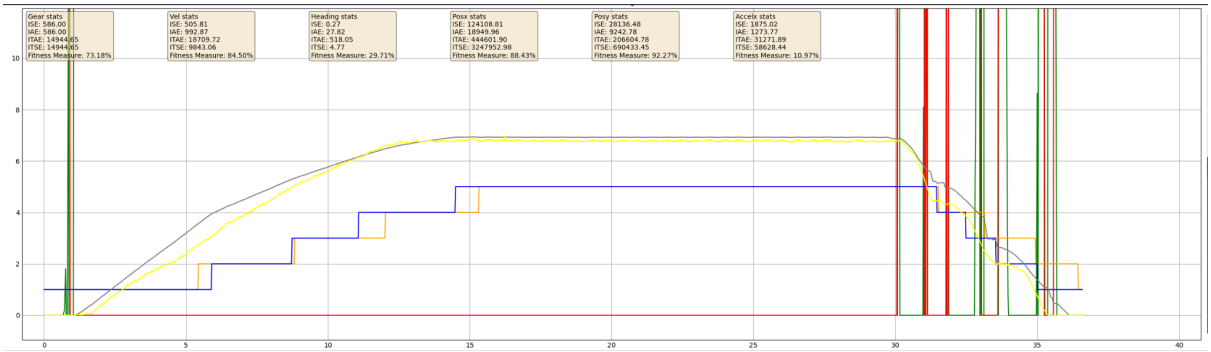
The results of the advanced vehicle dynamics model were validated through vehicle tests similar to simple vehicle dynamics.

This legend provides a comprehensive guide to the variables utilized in the advanced vehicle dynamics model test.

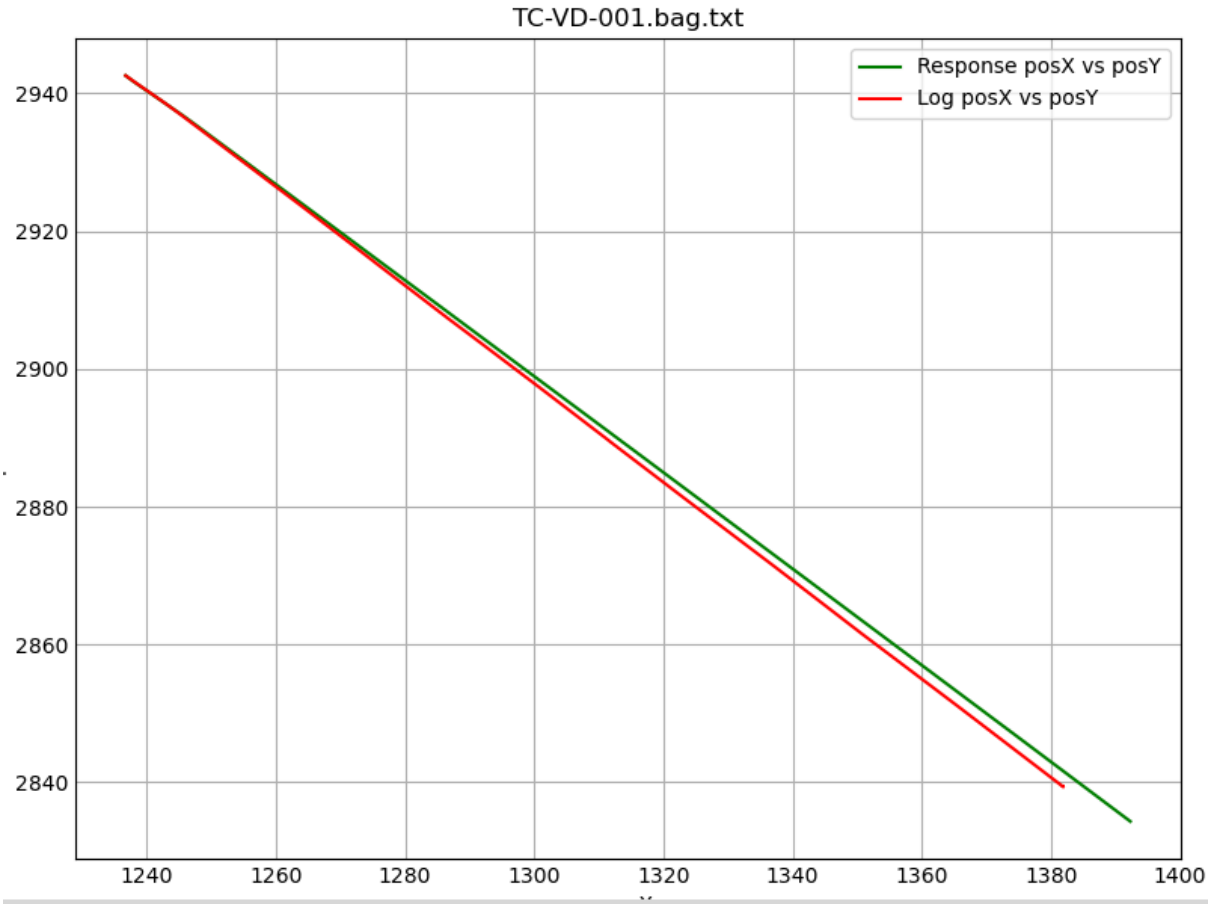
- Green: Command throttle input given to the model.
- Red: Command brake input.
- Orange: Gear number indicating the current transmission state.
- Blue: Response Gear number
- Gray: Response velocity from the simulation model.
- Yellow: Logged velocity data from vehicle for validation.
- Dark Red: Command velocity input for trajectory adjustments.
- Pink: Steering angle input to the model.
- Dark Orange: Response heading output from the model.
- Teal: Logged heading data for validation.
- Dark Blue: Response X-coordinate position.
- Dark Pink: Logged X-coordinate position.
- Orange: Response Y-coordinate position.
- Light Green: Logged Y-coordinate position.

Test case Description: Vehicle goes straight at 25 km/h

Velocity Graph:



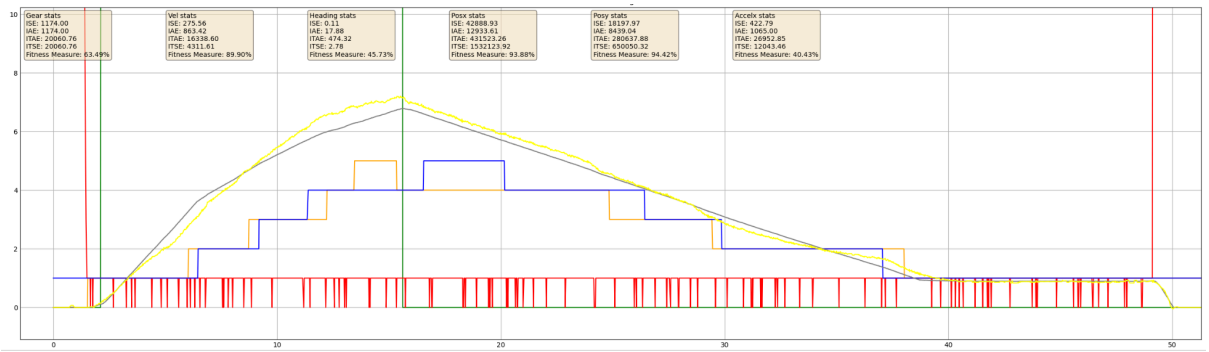
Position Graph:



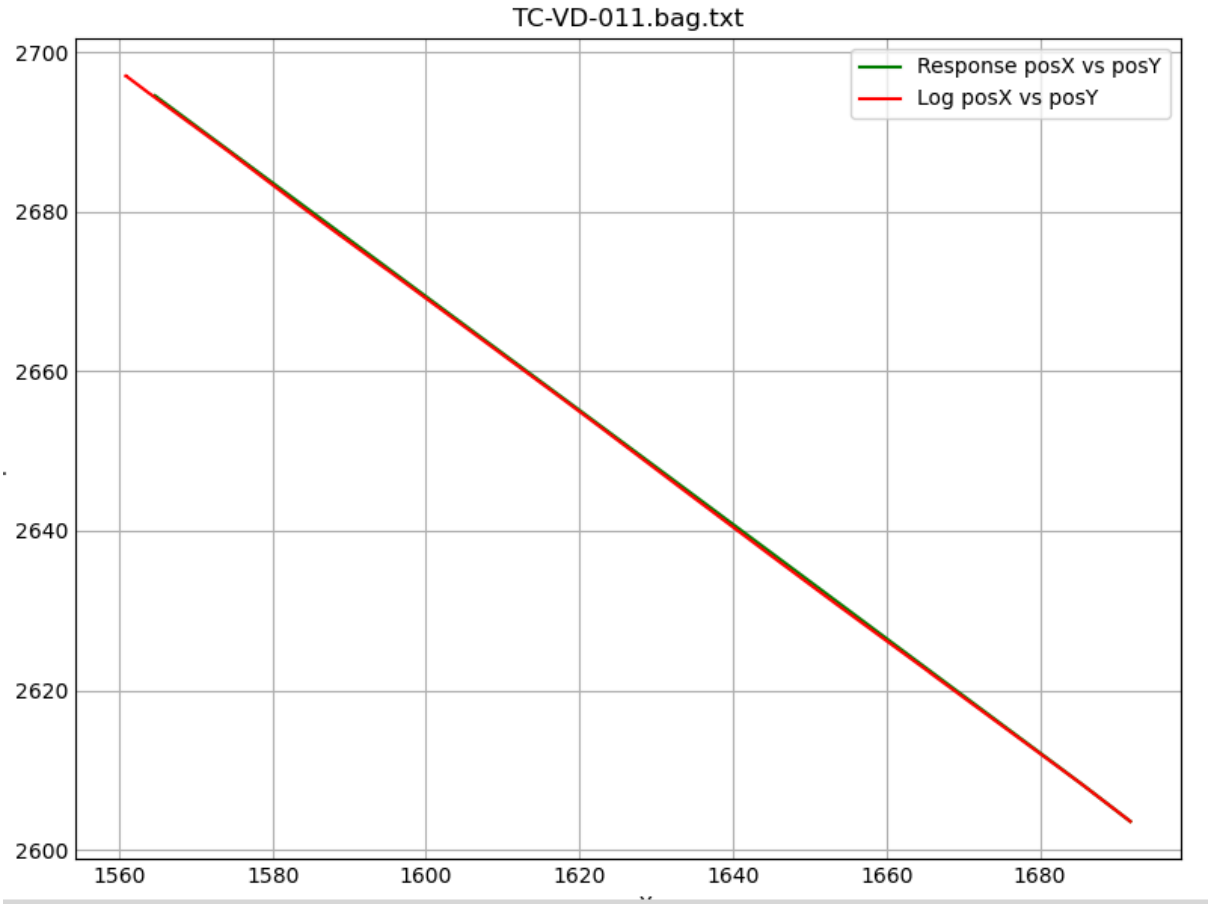
Graph Fitness:
Velocity: 84.55
Heading: 29.71
Position X: 88.43
Position Y: 92.27
Gear: 73.18

Test case Description: Vehicle goes full throttle run in manual mode

Velocity Graph:



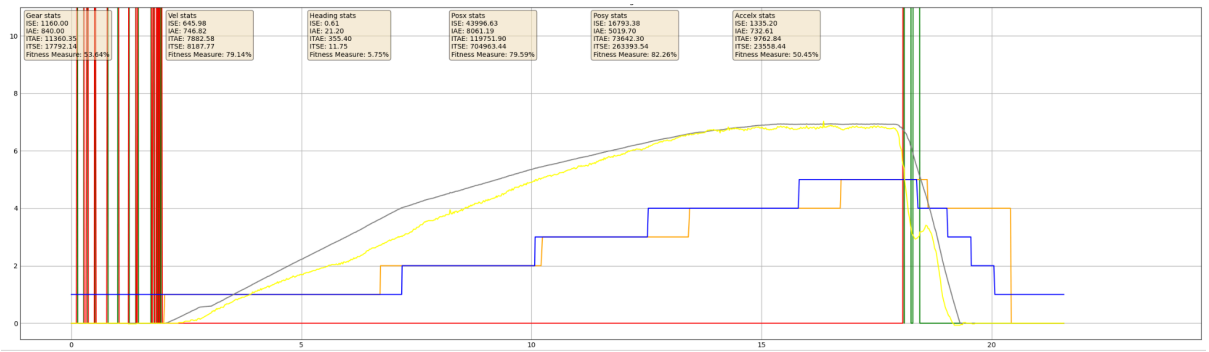
Position Graph:



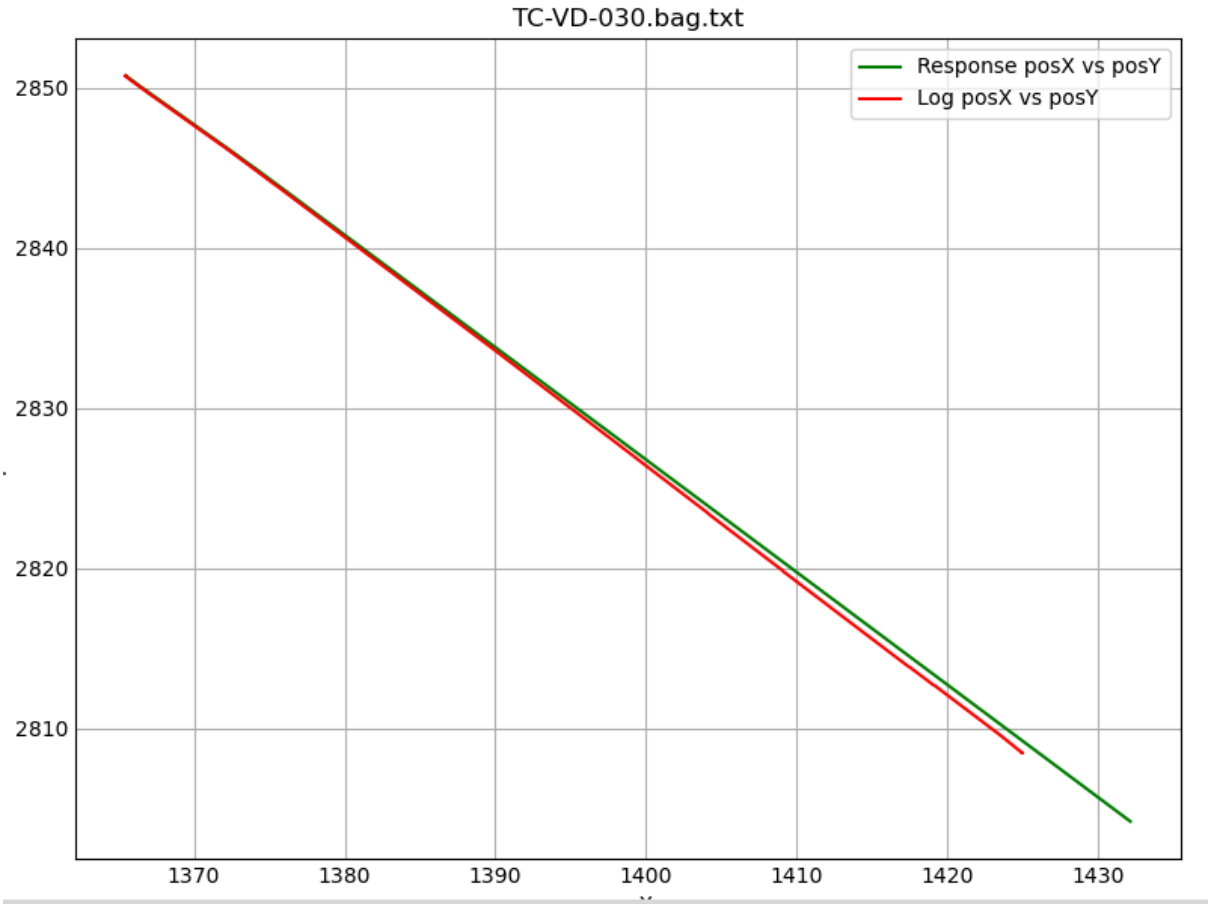
Graph Fitness:
Velocity: 83.90
Heading: 45.73
Position X: 93.88
Position Y: 94.42
Gear: 63.49

Test case Description: Vehicle does a emergency Stop at 25 km/h

Velocity Graph:



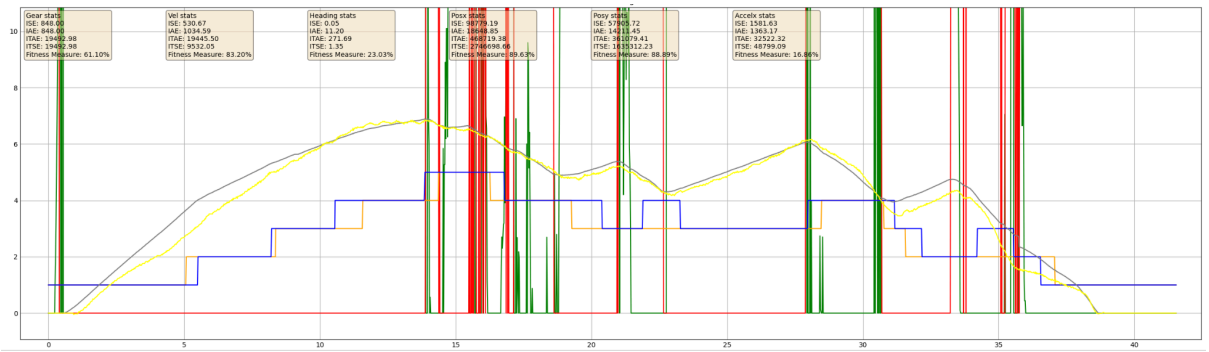
Position Graph:



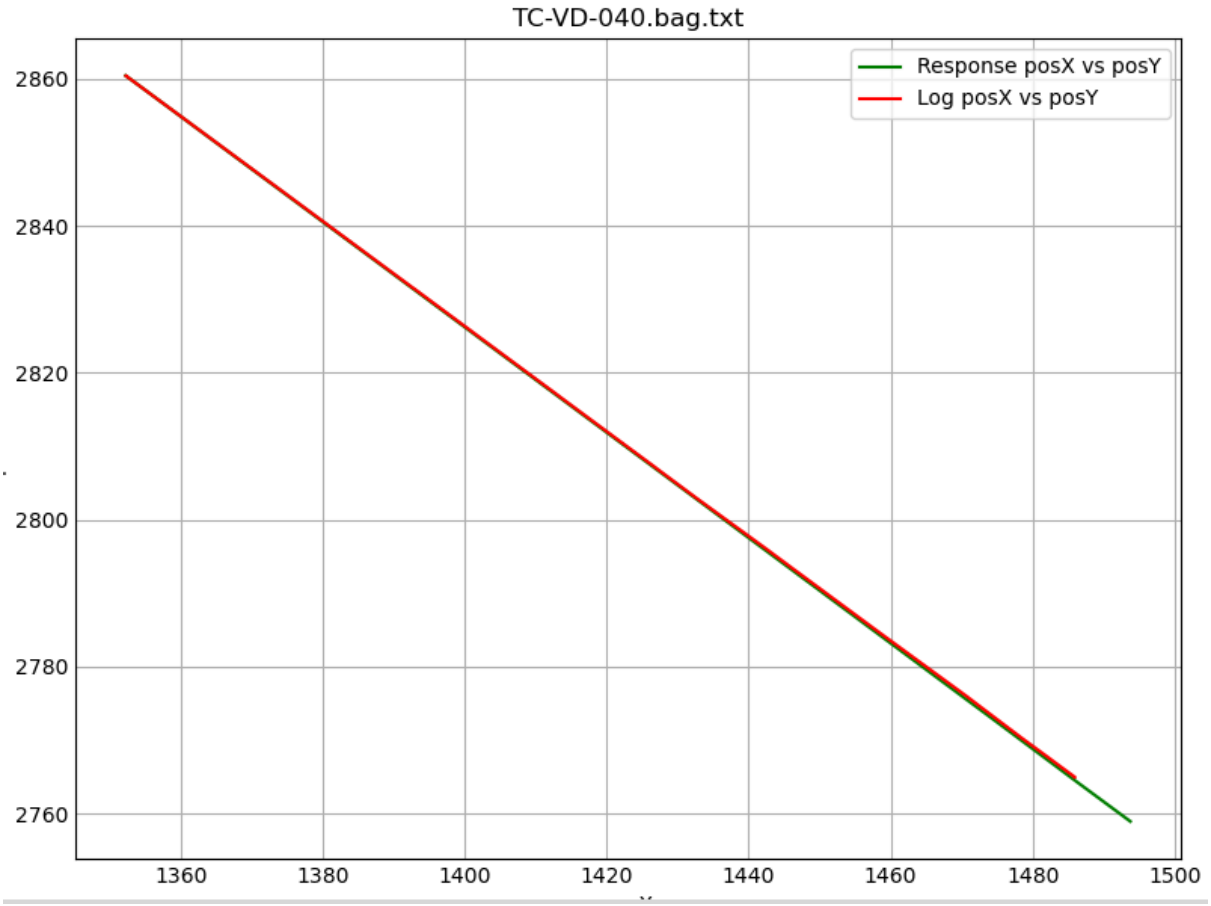
Graph Fitness:
Velocity: 79.14
Heading: 5.75
Position X: 79.59
Position Y: 82.26
Gear: 52.64

Test case Description: Vehicle does a soft Stop at 25 km/h

Velocity Graph:



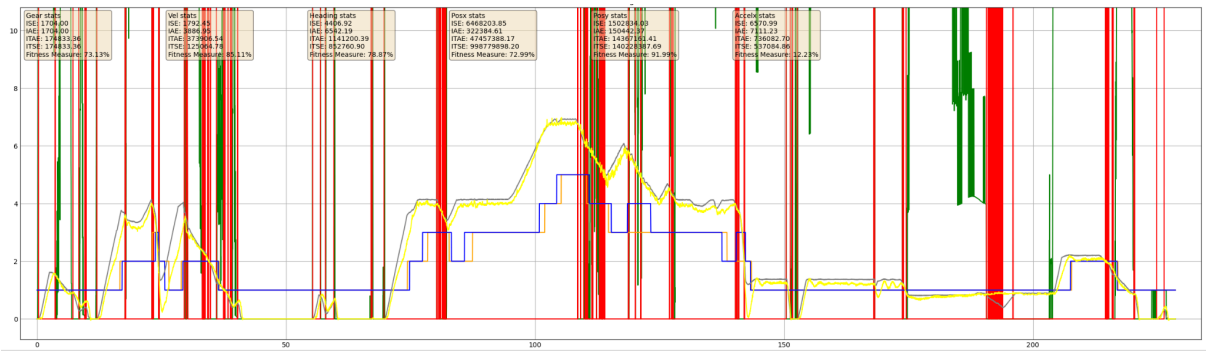
Position Graph:



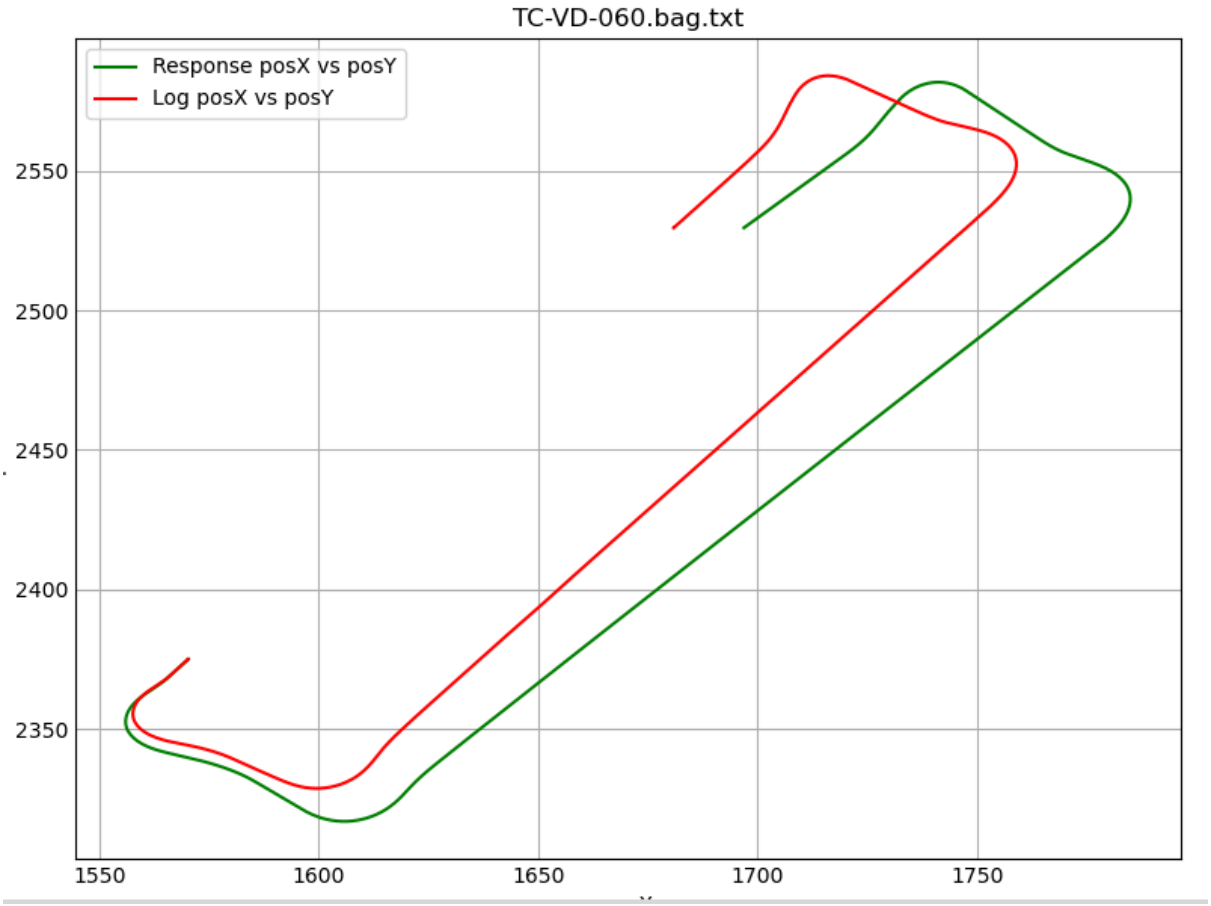
Graph Fitness:
Velocity: 83.20
Heading: 23.03
Position X: 89.63
Position Y: 88.89
Gear: 61.10

Test case Description: Vehicle makes 4 Left turns

Velocity Graph:



Position Graph:

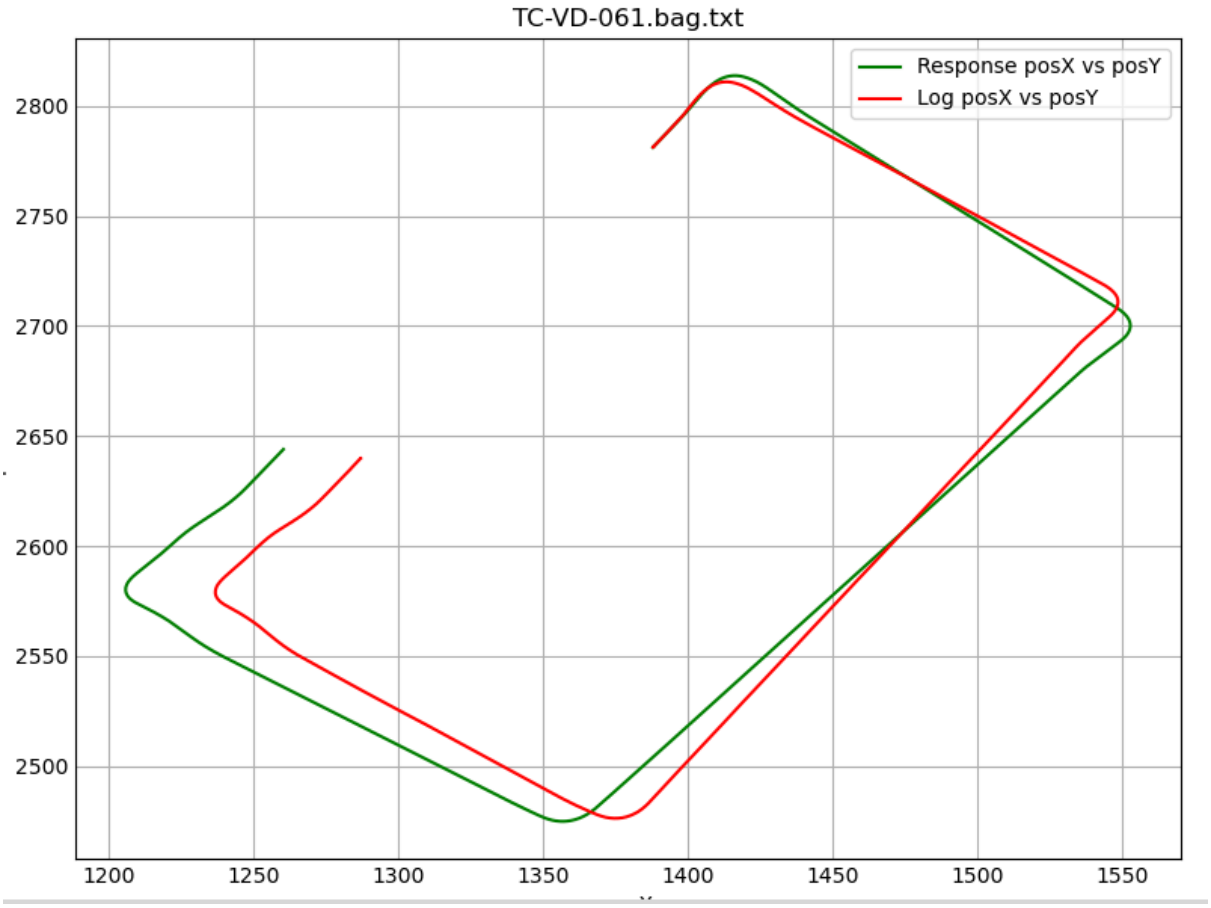


Graph Fitness:
Velocity: 85.11
Heading: 78.87
Position X: 72.99
Position Y: 91.99
Gear: 73.13

Test case Description: Vehicle makes 4 Right turns

Velocity Graph:

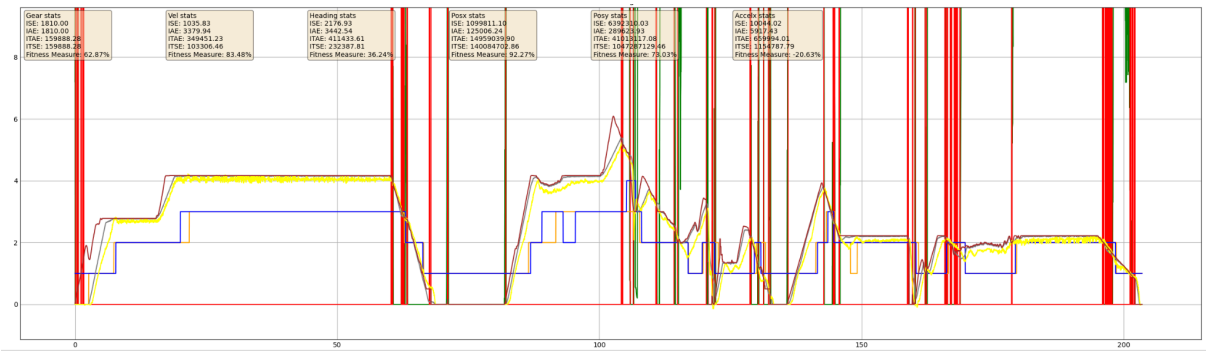
Position Graph:



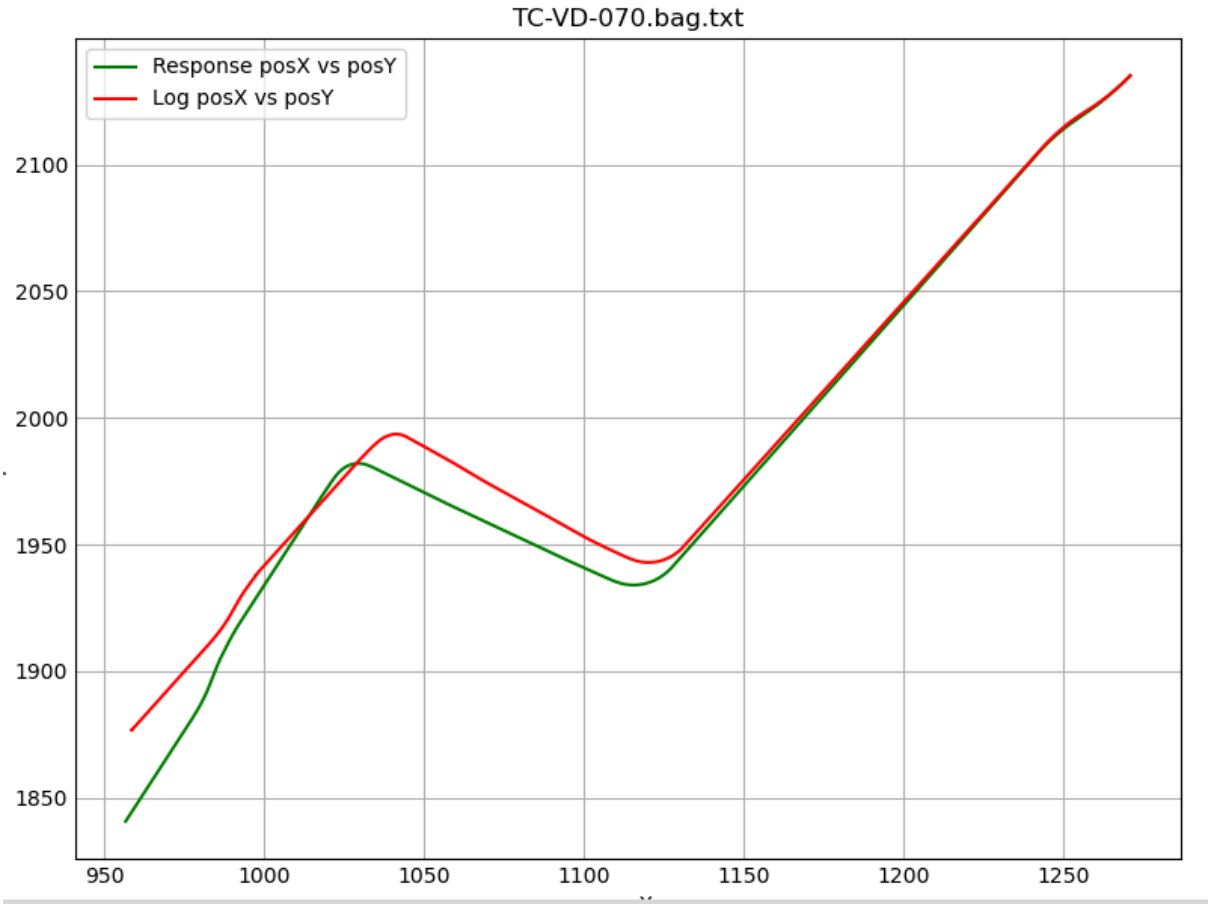
Graph Fitness:
Velocity: 81.86
Heading: 13.74
Position X: 81.99
Position Y: 94.26
Gear: 58.26

Test case Description: Vehicle makes Left turn then right turn

Velocity Graph:



Position Graph:



Graph Fitness:
Velocity: 83.48
Heading: 36.24
Position X: 92.27
Position Y: 73.03
Gear: 62.87

END OF REPORT