

POLITECNICO DI TORINO

Master's Degree in Automotive Engineering

Master's Thesis



**Politecnico
di Torino**

Design and Implementation of a Path Following and Low-Level Control Module for Automated Driving Systems

Supervisor:

Prof. Aldo Sorniotti

Candidate:

Andrea Sconfienza

Academic year 2024-2025

Abstract

The automotive sector is undergoing a profound transformation, driven by the convergence of electrification, digitalization, connectivity, and automation, four forces redefining the essence of mobility. In this new paradigm, the vehicle is no longer a mere means of transportation but an intelligent system capable of perceiving, interpreting, and acting autonomously within complex environments. Among the challenges shaping this transition, autonomous driving stands out as one of the most ambitious, requiring the integration of control theory, artificial intelligence, and vehicle dynamics to translate algorithms into safe and physically coherent behavior.

This thesis presents the design, implementation, and validation of a complete *Path Following* and *Low-Level Control* module for autonomous vehicles, fully developed within the *ROS 2* framework. The system provides a modular and reproducible platform for isolating the control problem and comparing different strategies under realistic yet controlled conditions. Based on the vehicle's kinematic *bicycle model* and the formulation of tracking errors in the *Frenet frame*, three controllers were implemented: *Pure Pursuit*, characterized by geometric simplicity and smooth behavior; *Stanley*, a proportional law ensuring precision and responsiveness; and *Model Predictive Control (MPC)*, which introduces predictive optimization to anticipate curvature changes while respecting dynamic constraints.

The control chain is validated in a ROS 2-based simulation environment including trajectory generation, kinematic simulation, and data logging. Tests on paths of varying complexity, from straight sections to tight curves and curvature reversals, revealed the complementary behavior of the three controllers: Pure Pursuit provides comfort and continuity, Stanley ensures accuracy and agility, while MPC achieves the best balance between precision, stability, and adherence to physical limits.

Overall, this work bridges the gap between theoretical design and experimental validation in vehicle control, providing a scalable foundation for the integration of advanced or learning-based controllers and marking a concrete step toward intelligent, coordinated, and sustainable autonomous driving.

Table of Contents

List of Figures	VI
1 Introduction	1
1.1 Background and Motivations	1
1.1.1 Social, Global and Economic Motivations	2
1.1.2 Technological and Experimental Background	3
1.2 State of the Art	4
1.2.1 History of Autonomous Driving	4
1.2.2 Standards of Automation and Shared Architectures	5
1.2.3 Modern Applications and Industrial Cases	8
1.3 Contribution and Methodology	10
2 Architecture of Automated Driving Systems	12
2.1 Perception	13
2.1.1 Perception Sensors	14
2.1.2 Perception Pipeline	15
2.2 Localization	16
2.2.1 Localization Sensors	16
2.2.2 Localization Pipeline	17
2.3 Planning	18
2.4 Control	21
3 Path Following	24
3.1 Vehicle Model	24
3.1.1 Kinematic Bicycle Model	25
3.1.2 Frenet Reference Frame and Tracking Errors	30
3.2 Lateral Controllers	33
3.2.1 Pure Pursuit Controller	33
3.2.2 Stanley Controller	36
3.3 Longitudinal Controller	40
3.4 Model Predictive Controller	43

4	Low-Level Control Module	48
4.1	Function of the Low-Level Control	48
4.1.1	Actuator Limits and Dynamic Constraints	49
4.1.2	Application of Limits in the Controllers	51
4.2	Actuator Interface and Signal Conversion	53
5	Robot Operating System (ROS 2)	57
5.1	ROS 2 Overview	58
5.1.1	ROS 2 Architecture and Fundamentals	59
5.2	Software Implementation	63
5.2.1	Simulation Architecture	66
6	Experimental Results	69
6.1	Test Scenarios	69
6.2	Analysis of Controllers	73
6.2.1	Pure Pursuit Controller	74
6.2.2	Stanley Controller	86
6.2.3	Model Predictive Control (MPC)	96
6.3	Controllers Comparison	107
6.3.1	Lateral behaviour	107
6.3.2	Longitudinal behaviour	111
7	Conclusions and Future Developments	115
	Bibliography	118

List of Figures

1.1	SAE Levels of Automation	7
2.1	Automated Driving Systems Architecture	13
2.2	eWolf Percpetion Network	13
3.1	From Four-Wheel to Bicycle Model	26
3.2	Bicycle Kinematic Model	27
3.3	Frenet reference frame for Path following	30
3.4	Pure Pursuit control method	34
3.5	Stanley control method	37
3.6	Longitudinal view from the vehicle	40
3.7	Circumcircle of the triangle defined by three points	41
5.1	ROS 2 Architecture	60
5.2	ROS 2 eWolf Network	63
6.1	U-shaped path	71
6.2	S-shaped path	72
6.3	Pure Pursuit – U-shaped path - Vehicle trajectory and footprint . .	75
6.4	Pure Pursuit – U-shaped path - Steering angle	76
6.5	Pure Pursuit – U-shaped path - Cross-Track Error	77
6.6	Pure Pursuit – U-shaped path - Longitudinal speed	78
6.7	Pure Pursuit – U-shaped path - Longitudinal acceleration	78
6.8	Pure Pursuit – U-shaped path - Lateral acceleration	79
6.9	Pure Pursuit – S-shaped path - Vehicle trajectory and footprint . .	81
6.10	Pure Pursuit – S-shaped path - Steering angle	82
6.11	Pure Pursuit – S-shaped path - Cross-Track Error	83
6.12	Pure Pursuit – S-shaped path - Longitudinal speed	84
6.13	Pure Pursuit – S-shaped path - Longitudinal acceleration	84
6.14	Pure Pursuit – S-shaped path - Lateral acceleration	85
6.15	Stanley Controller – U-shaped path - Vehicle trajectory and footprint	88
6.16	Stanley Controller – U-shaped path - Steering angle	89

6.17	Stanley Controller – U-shaped path - Cross-Track Error	90
6.18	Stanley Controller – U-shaped path - Lateral acceleration	91
6.19	Stanley Controller – S-shaped path - Vehicle trajectory and footprint	92
6.20	Stanley Controller – S-shaped path - Steering angle	93
6.21	Stanley Controller – S-shaped path - Cross-Track Error	94
6.22	Stanley Controller – S-shaped path - Lateral acceleration	95
6.23	MPC – U-shaped path - Vehicle trajectory and footprint	97
6.24	MPC – U-shaped path - Steering Angle	98
6.25	MPC – U-shaped path - Cross-Track Error	99
6.26	MPC – U-shaped path - Longitudinal speed	100
6.27	MPC – U-shaped path - Longitudinal acceleration	100
6.28	MPC – U-shaped path - Lateral acceleration	101
6.29	MPC – S-shaped path - Trajectory	102
6.30	MPC – S-shaped path - Steering angle	103
6.31	MPC – S-shaped path - Cross-Track Error	104
6.32	MPC – S-shaped path - Longitudinal speed	105
6.33	MPC – S-shaped path - Longitudinal acceleration	105
6.34	MPC – S-shaped path - Lateral Acceleration	106
6.35	Steering angle comparison - U shaped path	107
6.36	Cross-track error comparison - U shaped path	108
6.37	Steering angle comparison - U shaped path	109
6.38	Cross-track error comparison - S shaped path	110
6.39	Longitudinal speed comparison - U shaped path	111
6.40	Longitudinal acceleration comparison - U shaped path	112
6.41	Longitudinal speed comparison - S shaped path	113
6.42	Longitudinal acceleration comparison - S shaped path	114

Chapter 1

Introduction

Autonomous driving represents one of the most fascinating and complex challenges of modern engineering. This innovation comes from the convergence of technological progress, social needs, and long-term visions for the future of mobility. Its ambition is not simply to replace the human driver, but to revolutionize road transportation, making it safer, more efficient, and environmentally sustainable. In recent years, rapid advances in sensing technologies, artificial intelligence, and vehicle control systems have transformed the concept of driving automation from a futuristic vision into a concrete and rapidly evolving reality.

This chapter outlines the framework within which this thesis is developed. It presents the motivations behind the research, retraces the main milestones in the evolution of autonomous driving, analyzes the current state of the art, with a particular focus on the most relevant industrial and experimental applications, and finally introduces the comparative methodology that constitutes the core of this project.

1.1 Background and Motivations

Modern mobility is undergoing a profound transformation: the vehicle is no longer conceived as a mere collection of mechanical components, but as a dynamic platform in which software, sensors, and actuators continuously interact with the surrounding environment. In this context, autonomous driving represents one of the most advanced frontiers of contemporary engineering, born from the convergence of heterogeneous disciplines, including mechatronics, artificial intelligence, robotics and data science, with the shared goal of progressively reducing human intervention in the driving process.

Vehicle automation does not respond to a single need, but rather to a multiplicity of interconnected factors. Social aspects concern safety and inclusiveness;

environmental ones relate to sustainability and energy efficiency; while economic motivations are tied to productivity and the transformation of mobility models. At the same time, this evolution unfolds within a rapidly changing technological landscape, where sensors, actuators, and control architectures now enable the translation of theoretical models into experimentally validated systems in the real-world.

1.1.1 Social, Global and Economic Motivations

The push toward autonomous driving arises from an interplay of social, environmental, and economic factors that converge toward a common goal: making mobility safer, more efficient, and more accessible. The first and most evident motivation is social and public-health related. According to the World Health Organization, more than 1.19 million people lose their lives on the world’s roads every year, and road crashes remain the leading cause of death among individuals aged 5 to 29 [1]. Automating the most critical phases of the decision-making process means drastically reducing human error, responsible for over 90% of accidents [2], and moving closer to the objectives of Vision Zero and the European Safe System strategy, which aim to eliminate fatalities and serious injuries by 2050 [3]. From this perspective, autonomous driving is not merely a technological breakthrough but also a public-health measure supported by a clear political and regulatory mandate.

Environmental and energy-related motivations are equally significant. The transport sector produces roughly one-fifth of global CO₂ emissions, three-quarters of which come from road transport [4]. Even as electrification advances, emissions continue to grow, and automation can directly contribute to climate efficiency: predictive control algorithms help reduce unnecessary accelerations, sudden stops, and congestion, enabling smoother driving and lower energy consumption [5]. In urban environments, where traffic congestion costs billions of euros each year in lost time and wasted fuel, autonomous fleets and cooperative traffic-management systems can improve circulation and significantly lower the overall energy footprint.

In parallel, the transition toward autonomy is driven by strong economic and industrial interests. Automated driving functions open new business models, from on-demand mobility and robotaxis to autonomous logistics and delivery platforms, representing a potential global market estimated between \$300 and \$400 billion per year by 2035 [6]. Companies are investing heavily in prototypes not only for research or brand positioning but because automation enables the collection of real driving data, the validation of algorithms in complex scenarios, and the development of the technical and organizational expertise required to compete in a high-barrier technological market.

Another driving factor is demographic and occupational in nature. The growing shortage of professional drivers, combined with the aging of the European population, is creating a structural gap in the transport and logistics sectors. Automation, especially along monotonous or nighttime routes and hub-to-hub connections, is viewed as a strategic solution to ensure service continuity and reduce dependence on human labor in repetitive or high-risk activities [7].

Finally, autonomous driving also holds an inclusive and social value: it can restore mobility to elderly or disabled individuals, mitigate congestion, and contribute to transforming cities into more livable spaces where mobility is shared, electric, and intelligent [8].

1.1.2 Technological and Experimental Background

The development of autonomous driving is now made possible by a technological ecosystem that has reached a level of maturity sufficient to enable real-world experimentation of advanced control strategies. The evolution of vehicular sensing and measurement systems has made it possible to acquire, at unprecedented frequencies and accuracies, complete data on both the dynamic state of the vehicle and the surrounding environment. LiDAR, radar, cameras, and inertial units provide a multimodal perception of the driving scenario, while differential localization systems such as GPS-RTK and high-grade IMUs allow estimation of position, velocity, and acceleration with errors below the centimeter range [9]. These measurements form the experimental foundation required to design and validate vehicle controllers that are both responsive and reliable.

In parallel, the availability of direct actuation interfaces toward the steering, throttle, and braking systems has made it possible to close the control loop with latencies below a few tens of milliseconds [10]. This allows the evaluation of vehicle behavior while respecting physical constraints such as rate, jerk, and actuator saturation. Modern experimental platforms therefore make it possible to test geometric, classical, and predictive controllers under real conditions, accurately analyzing lateral stability, longitudinal dynamics, and the effects of mechanical and dynamic limits.

On the software side, the introduction of modular robotic middleware such as ROS 2 has provided a standardized infrastructure for connecting sensors, controllers, and data-acquisition modules into a single coherent architecture [11]. This approach enables developers to implement, replace, and compare different control strategies under identical experimental conditions, ensuring traceability, synchronization, and reproducibility of tests. The interaction among perception, localization, and control subsystems thus becomes observable and measurable, providing a concrete ground for experimental validation.

Within this technological landscape lies the *eWolf* project, an experimental platform developed at the *Forschungsinstitut für Kraftfahrwesen und Fahrzeugmotoren Stuttgart* (FKFS), which integrates real sensors, dedicated computing units and a complete ROS 2 architecture [12]. This test environment, described in detail in the following chapters, represents the experimental basis on which the work of this thesis has been developed and validated. It allows a homogeneous comparison of different control strategies and the analysis of their performance through physically meaningful metrics.

In summary, the combination of high-precision sensors, accessible actuators, and modular software infrastructures has transformed vehicle-control research from a theoretical exercise into an experimental discipline. It is within this mature technological context that this thesis is positioned, with the aim of exploiting these tools to bridge the gap between modeling, implementation, and real-world validation of automated-driving control systems.

1.2 State of the Art

Autonomous driving is the outcome of a long path of research and innovation spanning more than half a century of technological evolution. From the first utopian visions of automated highways in the 1930s to today’s industrial-grade systems, the journey that has led to the development of modern *Automated Driving Systems* (ADS) represents one of the most remarkable convergences of mechanics, electronics, and artificial intelligence in the history of engineering [13]. The field has progressively evolved from isolated academic prototypes to complex, multi-sensor platforms capable of perception, decision-making, and real-time control under challenging road conditions.

To understand the current state of the technology, it is essential to retrace not only the historical milestones that have driven automation forward but also the architectural paradigms and design philosophies that now define its limits and potential. The evolution of autonomous driving can thus be analyzed through three complementary dimensions: its historical roots and pioneering experiments, the formalization of standards and levels of automation, and its industrial implementation across modern vehicle platforms.

1.2.1 History of Autonomous Driving

The history of autonomous driving does not follow a linear path but unfolds as a sequence of visionary ideas, technological advances, and inevitable setbacks. As early as the 1930s, when the automobile was consolidating its role in modern society, the first concepts of vehicles capable of moving without human intervention began to emerge. A symbolic moment came with the *Futurama* pavilion at the 1939

New York World’s Fair, where General Motors envisioned automated highways and radio-controlled traffic, a conceptual preview of what would become a concrete engineering goal decades later [14].

In the post-war years, the rapid development of electronics enabled the first infrastructure-based experimental systems. In 1953, the Radio Corporation of America (RCA) proposed a control system using transponders embedded under the road surface and receivers mounted on vehicles [15]. A few years later, in 1958, a test highway section in Nebraska implemented buried wire guidance to keep vehicles centered in their lanes [16]. However, these early approaches proved difficult to scale due to their heavy reliance on external infrastructure.

A genuine paradigm shift occurred during the 1980s, when intelligence began to migrate from the infrastructure to the vehicle itself. Thanks to advancements in electronics and sensing technologies, radar, lidar, and cameras enabled vehicles to perceive their surroundings for the first time. At the same time, the automotive industry introduced the first active-safety systems, such as ABS and ESP, while academic and military research produced the earliest robotic vehicles. In Europe, the *EUREKA PROMETHEUS* program (1987–1995), the largest research initiative of its time, saw the development of the VaMP and VITA-2 vehicles based on the Mercedes-Benz 500 SEL, capable of driving hundreds of kilometers on highways while autonomously maintaining lanes and performing overtaking maneuvers in real traffic [17]. Italy also made a notable contribution with the *ARGO* project developed at the University of Parma, which in 1998 drove over 2,000 km on Italian highways, operating autonomously for most of the journey [18].

The decisive breakthrough came in the early 2000s with competitions organized by the *Defense Advanced Research Projects Agency* (DARPA). The first Grand Challenge in 2004 revealed the technological limitations of the time, but in 2005, Stanford University’s vehicle *Stanley* successfully completed a 212-km desert route, marking a milestone in the history of robotics [19]. With the subsequent Urban Challenge in 2007, autonomous vehicles were required to navigate semi-structured urban environments, obey traffic rules, and interact with other road users. Projects such as *Boss* from Carnegie Mellon University demonstrated the maturity of a now-recognized architecture integrating perception, localization, planning, and control within a unified system [20].

These experiences marked the transition from pioneering intuition to tangible demonstration, laying the foundations for the current state of the art and bringing autonomous driving technologies to the forefront of academic and industrial research.

1.2.2 Standards of Automation and Shared Architectures

The technological evolution described in the previous sections has highlighted the need for a shared taxonomy capable of clearly defining how autonomous a vehicle

can be and what role the human driver still retains. During the early years of experimentation, terms such as *self-driving*, *autonomous*, or *driver-assist* were often used ambiguously, generating confusion both within the scientific community and among the general public. To address this lack of clarity, the *Society of Automotive Engineers* (SAE) introduced the *J3016* standard, now internationally recognized as the normative reference for classifying automation levels in road vehicles [21].

The standard defines six levels of driving automation, numbered from 0 to 5, covering the entire spectrum of vehicle autonomy: from no automation, where the driver performs all driving tasks, to full automation, where the vehicle operates in all conditions without any human intervention.

- **Level 0 – No automation:** the driver performs all driving tasks, while assistance systems may only provide warnings or momentary interventions, such as automatic emergency braking.
- **Level 1 – Driver assistance:** the vehicle can control a single aspect of driving dynamics — either longitudinal (via adaptive cruise control) or lateral (via lane-keeping assist) — but the driver must continuously monitor the environment.
- **Level 2 – Partial automation:** the system can simultaneously manage both longitudinal and lateral control, combining adaptive cruise control and lane centering. However, driver attention remains essential since the system lacks full environmental awareness.
- **Level 3 – Conditional automation:** the vehicle can monitor the environment and handle driving within a defined *Operational Design Domain* (ODD). The driver must intervene only when explicitly requested by the system.
- **Level 4 – High automation:** the vehicle can perform all driving tasks autonomously within a specific ODD (e.g., a limited urban area or a highway segment) without requiring human supervision. Outside this domain, it may request control transfer.
- **Level 5 – Full automation:** the vehicle can operate in any environment and condition without any human involvement and without the need for a steering wheel or pedals.

This classification had a decisive impact not only on regulation but also on the engineering interpretation of autonomous driving systems. For the first time, driving automation was framed as a sequence of incremental functions rather than a sudden leap from human to fully robotic driving. In other words, autonomous driving is not a product but a process, a gradual delegation of control from human

to machine, structured through successive thresholds of technical, legal, and societal maturity.

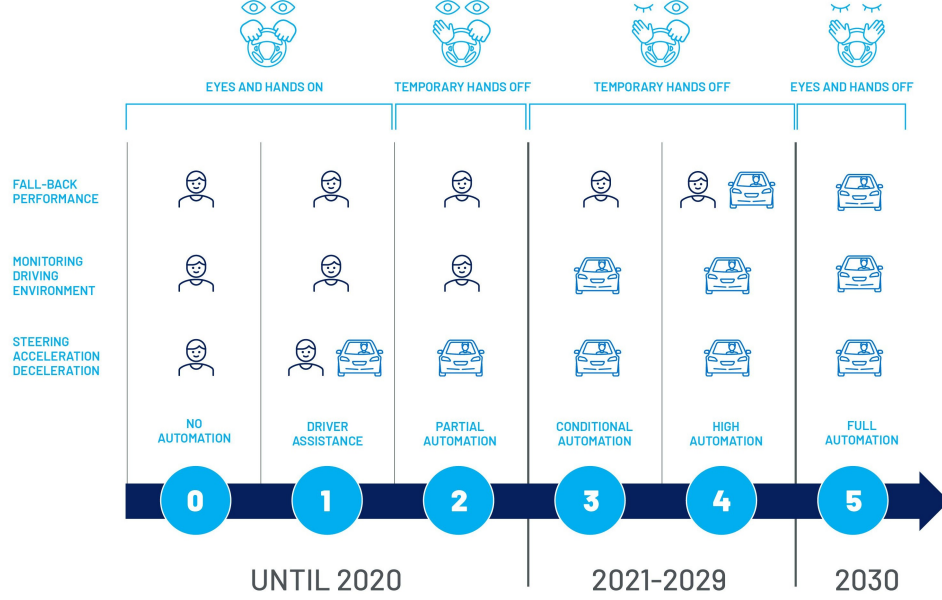


Figure 1.1: SAE Levels of Automation

The adoption of the SAE framework also encouraged both academia and industry to formalize a common functional architecture for autonomous driving systems, now universally known as the *Automated Driving System (ADS) stack*. This structure did not emerge from a single project but from decades of accumulated experience from pioneering programs, such as *PROMETHEUS* and *ARGO* to the DARPA challenges, where the complexity of urban scenarios demanded a modular and hierarchical organization of system functions [20].

The modern architecture represents the result of a conceptual evolution: from the linear *input-output* logic of early driver-assistance systems to a closed-loop ecosystem in which perception, localization, planning, and control interact continuously and dynamically [13]. Schematically, a contemporary ADS can be divided into four main macro-modules:

1. **Perception:** processes data from sensors (LiDAR, radar, cameras, GNSS) to detect the environment, obstacles, and road geometry;
2. **Localization:** estimates the vehicle's position and orientation in global or map coordinates by integrating odometry, GNSS, and state-estimation filters;

3. **Planning:** defines the optimal path and trajectory according to kinematic, dynamic, and safety constraints;
4. **Control:** converts the planned trajectory into physical commands for steering, throttle, and braking, ensuring dynamic stability and passenger comfort.

This architecture is not rigid but deeply interconnected: errors in perception propagate into localization; uncertainties in planning require more robust controllers; control delays influence the next trajectory. The system thus behaves as a distributed closed-loop network, where the accuracy of the final control action acts as the cohesive element binding the functional chain together.

The formalization of SAE levels and the convergence toward a shared functional architecture have transitioned autonomous driving from an exploratory phase to an engineering discipline. Today, the question is no longer whether to follow this structure, but how to implement it efficiently, safely, and consistently with the physical constraints of the vehicle. This represents the foundation of modern applications, described in the next section: complex systems in which choices of sensors, maps, and hardware reflect distinct engineering philosophies yet coexist within a common architectural framework.

1.2.3 Modern Applications and Industrial Cases

The most advanced industrial applications of autonomous driving demonstrate the coexistence of multiple complementary technological paradigms, all converging on a common goal: the capability of the system to generate and follow trajectories that are safe, comfortable, and physically feasible. *Tesla*, *Waymo*, and *Mercedes-Benz* represent three distinct development philosophies, each pursuing a balance between tracking accuracy, computational efficiency, and vehicle constraint compliance.

Tesla

Tesla’s approach is based on a fully vision-centric perception chain, relying on eight cameras and a three-dimensional neural architecture known as the *Occupancy Network*. This network reconstructs a real-time volumetric representation of the surrounding environment and predicts the motion and occupancy of dynamic objects [22]. Its distinctive feature lies in the direct integration of perception and decision-making within a nearly end-to-end framework: the neural model outputs a spatiotemporal representation that guides trajectory planning and provides targets directly to the kinematic controller. The control stack operates on two levels: a neural “behavioral” layer that selects maneuvers (follow, overtake, stop) and a “kinematic” layer that computes steering angle and acceleration through an adaptive model-based controller. Over-the-air (OTA) updates allow continuous

retraining of the network using real-world fleet data. Although less interpretable and highly dependent on visual data quality, this paradigm represents the frontier of the data-driven control approach, where driving policies emerge from statistical learning rather than explicit physical modeling.

Waymo

Waymo’s philosophy, in contrast, follows a model-based paradigm centered on multimodal sensor fusion combining 360° LiDAR, millimeter-wave radar, and high-resolution cameras. These sensors feed high-definition (HD) maps and probabilistic environment models continuously updated through Bayesian filtering [23]. The planning layer employs predictive algorithms that generate space–time-constrained trajectories accounting for vehicle dynamics and behavioral forecasts of surrounding agents. Downstream, a *Model Predictive Controller* (MPC) computes the optimal steering, acceleration, and braking actions over a multi-second prediction horizon, minimizing cross-track error and control variation (steering rate and jerk). This framework explicitly incorporates physical limits, such as lateral acceleration, actuator saturation, and response delays, ensuring both stability and comfort. Waymo thus represents the industrial example most aligned with the model-based and predictive control strategies adopted in this thesis, where the trajectory is not merely followed but continuously optimized with respect to context and dynamics.

Mercedes-Benz Drive Pilot

Drive Pilot is the first SAE Level 3 certified autonomous driving system approved for use in both Europe and California. It exemplifies a safety-oriented design philosophy focused on redundancy and certification under ISO 26262 (functional safety) and SOTIF (Safety of the Intended Functionality) [24]. The platform integrates LiDAR, radar, stereoscopic cameras, and GPS-RTK with proprietary HD maps supported by redundant sensor fusion. The control layer adopts predictive, comfort-oriented strategies: steering and acceleration profiles are optimized to limit lateral acceleration and jerk, ensuring smooth transitions between manual and automated modes. The entire system is structured as a dual-channel architecture: hardware and software operate in parallel to achieve fault tolerance and fail-operational behavior. In this configuration, control is not merely dynamic regulation but a functional guarantor of stability, comfort, and certifiable safety. Drive Pilot demonstrates that the maturity of autonomous driving is measured not only by decision-making capability but also by the coherence between control, safety, and regulatory compliance.

1.3 Contribution and Methodology

The work presented in this thesis arises from the need to bridge a tangible gap between theoretical research in vehicle control and its practical implementation on real experimental platforms. In recent years, both academic literature and industrial developments have shown that the success of an autonomous driving system depends not only on the effectiveness of perception or planning modules but also on the robustness of the controller that translates decisions into coherent, stable, and safe physical actions [13]. It is within this final link — where control theory meets real vehicle dynamics — that the true engineering maturity of an automated driving system can be measured.

From this perspective, the thesis focuses on the design, implementation, and experimental validation of a *Path Following* and *Low-Level Control module* dedicated to autonomous vehicles. The objective is not to propose a new abstract algorithm, but rather to develop and demonstrate a rigorous and replicable framework for comparing, under homogeneous conditions, the main families of controllers found in the state of the art — namely *Pure Pursuit*, *Stanley*, and *Model Predictive Control (MPC)* — analyzing their behavior, limitations, and stability within a unified and coherent architecture.

The experimental context of this work is the *eWolf* project, developed at the *Forschungsinstitut für Kraftfahrwesen und Fahrzeugmotoren Stuttgart (FKFS)*. The platform integrates real sensors, dedicated computation units, and the ROS 2 middleware for distributed management of perception, localization, and control modules. The experimental vehicle is equipped with a *Speedgoat* interface that enables direct communication with physical actuators, closing the control loop with latencies below a few tens of milliseconds [12]. This configuration allows the system’s behavior to be observed and measured under conditions faithfully reproducing real vehicle dynamics.

The adopted methodology constitutes the core contribution of the thesis. All controllers were implemented as independent ROS nodes sharing the same sensory inputs, kinematic model, and physical vehicle parameters. This ensures that any observed difference arises solely from the control principle itself rather than from contextual or configuration variations. Tests were performed under strictly repeatable conditions across multiple reference paths — straight segments, constant-radius curves, and S-shaped trajectories — with identical temporal and spatial sampling rates. Each session generated a high-frequency dataset describing in detail the vehicle’s state and controller actions: position, velocity, steering angle, lateral acceleration, jerk, and tracking errors. These data were recorded in real time, synchronized through ROS middleware, and subsequently analyzed in MATLAB, where objective performance and stability metrics were computed to guarantee traceability and comparability of results.

A key aspect of the methodology is its constant attention to physical realism. Each control law was implemented with realistic actuator and dynamic constraints: steering rate and jerk limits, saturation boundaries, maximum lateral acceleration, and comfort thresholds. The goal was to ensure that the resulting behavior is not only mathematically stable but also physically plausible and transferable to real-world industrial contexts. Thus, the thesis serves as a bridge between theoretical simulation and real-world experimentation, combining analytical rigor with engineering concreteness.

From a technical standpoint, the work introduces several targeted improvements to enhance the controllers' adaptability to varying dynamic conditions. The *Stanley controller* has been extended with a dynamic lookahead mechanism that adjusts the preview distance as a function of speed, improving high-speed stability and responsiveness in curves. The *Pure Pursuit controller* has been refined to reduce oscillations on tight paths, while the *MPC controller* has been enhanced with filtered rate limiting and a penalized cost function that harmonizes trajectory error, steering rate, and jerk to achieve smoother and more natural behavior. All algorithms were tested under identical input conditions and evaluated through objective performance metrics, enabling a direct comparison among geometric, classical, and predictive control models.

Beyond the technical aspects, the thesis provides a methodological contribution of general value: it defines a modular, traceable, and reproducible experimental structure that can be extended to other control algorithms or vehicle platforms simply by replacing the control node within the same ROS architecture. This approach, grounded in rigor and standardization, transforms experimentation into a systematic and repeatable practice, overcoming the fragmentation typical of academic testing and bringing university research closer to industrial validation processes.

Ultimately, the contribution of this thesis lies in its strategic position between theory and practice: it does not introduce a single revolutionary algorithm, but establishes an evaluation and comparison framework that enables the objective measurement of trajectory controller performance in realistic conditions. It represents a concrete step toward a vision of autonomous driving in which modeling, implementation, and validation are not isolated stages but integrated components of a coherent engineering process. In this sense, the thesis does not merely test controllers — it builds a common language between theory and experimentation, laying the groundwork for increasingly intelligent, reliable, and physically consistent vehicle control systems.

Chapter 2

Architecture of Automated Driving Systems

An autonomous driving system cannot be conceived as the development of a single algorithm, but rather as the coherent integration of a set of functional modules, each with specific tasks and requirements. Experience gained in both academic and industrial contexts has demonstrated that decomposing the problem into distinct functional blocks is a necessary condition for ensuring scalability, interoperability, and overall system reliability.

First, each module can be designed, tested, and optimized independently, promoting technological innovation and allowing for the replacement or improvement of individual components without redesigning the entire system. Second, a modular architecture enables flexibility: a perception algorithm can be replaced with a more advanced one, or new sensors can be introduced, without modifying the logic of the other modules [25]. Finally, modularization facilitates functional safety management: fault analysis and certification can be performed at the module level, significantly reducing overall systemic risk.

Within this structure, four main functional modules can be identified: Perception, Localization, Planning and Control. The first provides a structured representation of the surrounding environment; the second allows the vehicle to accurately determine its position in space; the third transforms environmental understanding and vehicle state into a safe, drivable trajectory; and the fourth generates the physical commands that make the vehicle's actions effective. Moreover, the presence of a robust communication infrastructure, capable of synchronizing heterogeneous data streams and ensuring compliance with real-time constraints, guarantees that all modules operate in an integrated and coordinated manner.

This chapter provides a detailed description of each block within the overall architecture, with specific references to the *eWolf* project, which serves as the case

study for this research and represents the practical environment where theoretical principles are validated experimentally.

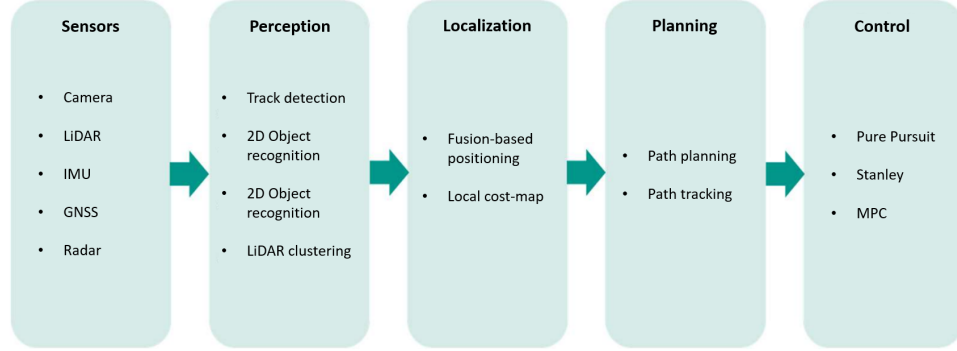


Figure 2.1: Automated Driving Systems Architecture

2.1 Perception

The Perception module represents the point of contact between the vehicle and the external world: it collects raw signals from onboard sensors, synchronizes them, and transforms them into a coherent description of the road, surrounding obstacles, and drivable space. In the *eWolf* platform, this module has been designed to be reliable, time-consistent, and easily integrable with the other subsystems, ensuring that the acquired data become structured and stable information for Localization, Planning and Control.

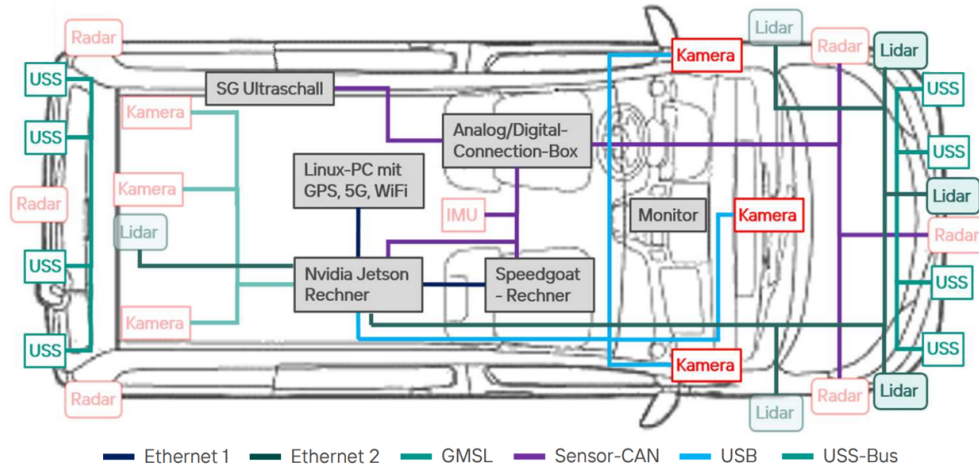


Figure 2.2: eWolf Percpetion Network

A typical perception pipeline combines multiple sensing modalities to overcome the limitations of individual sensors. Cameras provide rich semantic content, LiDAR sensors deliver precise 3D geometry and radar units contribute robustness under adverse weather or lighting conditions. Sensor fusion algorithms integrate these heterogeneous sources into a unified environmental model, improving the accuracy and reliability of object detection and free-space estimation.

The resulting representation of the environment, often structured as a *bird's-eye view* (BEV) map or occupancy grid, forms the foundation for the subsequent modules, enabling precise vehicle localization, trajectory planning, and control in complex driving scenarios.

2.1.1 Perception Sensors

In the eWolf project, environmental perception relies on a complementary set of sensors, where each technology compensates for the limitations of the others.

The front-facing cameras provide the semantic content of the scene: from the video streams, the geometry of the lane and the presence of other road users are extracted, along with their classification into vehicles, pedestrians, and cyclists. Under good lighting conditions, cameras offer rich and detailed information, ideal for lane keeping and road-context understanding. However, vision performance can degrade in cases of glare, rain, or fog; for this reason, partially overlapping views and dedicated correction algorithms are employed, executed in real time on the onboard computer.

The front LiDAR acts as the three-dimensional metric counterpart. The generated point cloud enables accurate reconstruction of road boundaries and obstacle shapes, regardless of lighting conditions, and reinforces distance estimates obtained from vision. It becomes the reference sensor when light is scarce or the scene lacks texture, such as during nighttime. However, its accuracy may decrease under heavy rain or on highly reflective or absorptive surfaces; in such conditions, more robust filtering is applied, and greater weight is given to the visual input to ensure continuity.

Alongside these two primary sensors, a 77 GHz radar planned as a future extension for the eWolf vehicle provides reliable information in adverse weather conditions and during convoy driving. It directly measures the distance and relative velocity of surrounding objects, supporting longitudinal control within the Adaptive Cruise Control (ACC) function. Finally, ultrasonic sensors (USS) complete the perception layer at very low speeds. They are particularly useful for parking maneuvers and proximity detection, where they can identify nearby obstacles not visible to the front sensors.

The integration of these sensing technologies creates a perception system capable of adapting to diverse operational contexts. All sensor data streams converge on the

dedicated onboard computer, which processes the inputs and produces high-level perceptual outputs: a regularized lane, detected objects with position and velocity, and the drivable space. These outputs serve as the main inputs for the planning and control modules.

2.1.2 Perception Pipeline

In the eWolf vehicle, the flow of information begins with the front-mounted sensors and converges on a dedicated onboard computer responsible for real-time processing. This separation between the perceptual block and the control unit ensures that the significant computational load associated with visual processing and three-dimensional reconstruction does not interfere with the deterministic logic of vehicle control.

The first step of the pipeline is temporal synchronization: each camera frame and each LiDAR scan are timestamped and aligned within a few milliseconds, preserving consistency between the two sensing modalities. Subsequently, a targeted pre-processing stage is applied to the images, including optical distortion correction, illumination normalization, and the selection of a region of interest (ROI) centered on the roadway. In parallel, the LiDAR point cloud is filtered to remove outliers and used to reinforce the geometry of the road boundaries extracted from vision and to validate depth estimations when the scene is texture-poor or partially occluded.

On these synchronized data streams, two complementary estimation chains operate. The first focuses on lane reconstruction and uses a neural network for lane detection, trained to extract lane boundaries and project them into a ground-level metric view. From this projection, a regularized centerline is obtained, providing, for each point of the local trajectory, position and orientation. This information ensures that speed profiles and steering commands remain consistent with lateral acceleration and comfort limits. The second chain deals with object detection, identifying and classifying the main dynamic agents—vehicles, pedestrians, and cyclists—while the LiDAR reinforces distance and three-dimensional shape, providing a geometric measure independent of lighting conditions. The two outputs are then combined by a multi-object tracker, which associates observations over time and produces filtered estimates of position, velocity, and heading.

At the core of the process lies multimodal fusion, which adaptively combines information from vision and LiDAR. At the feature level, road boundaries extracted from images are verified against height discontinuities in the point cloud; at the object level, visual estimates are integrated with metric measurements to stabilize distance and velocity. When one source degrades, the system dynamically rebalances the weights assigned to the different modalities, prioritizing the most reliable one at each instant: under low-light or heavy-rain conditions, LiDAR dominates, whereas on reflective or textureless surfaces, the visual channel prevails. All perception

outputs include explicit uncertainty information, allowing the planning and control modules to adapt their behavior by applying more conservative margins when confidence decreases.

The output interfaces of the perception module are designed to provide stable and easily interpretable data. The road is represented as a reference curve sampled at regular spatial intervals over a given horizon, with position, orientation, and curvature κ for each point. The entire pipeline operates at frequencies of the order of tens of hertz, with a total latency of a few hundred milliseconds—an optimal compromise between accuracy and timeliness that enables real-time updates of trajectories and control commands.

2.2 Localization

The localization module is responsible for continuously, accurately, and coherently estimating the vehicle’s pose — that is, its position, orientation, and motion quantities such as linear velocity, acceleration, and yaw rate — which describe the dynamic state of the system. It provides the common spatial reference upon which the planning and control modules are built, enabling proper alignment between trajectories and driving actions.

To remain reliable even in complex and unstructured environments, localization integrates in real time multiple heterogeneous and complementary sources, each with its own strengths and limitations. The objective is to obtain a single, consistent estimate of the vehicle state, accompanied by an explicit measure of residual uncertainty — a fundamental requirement for the stability and safety of SAE Level 2 automated driving functions.

2.2.1 Localization Sensors

The localization system of an autonomous vehicle never relies on a single measurement source; instead, it combines information from different sensors, each fulfilling a specific role. In general, three fundamental components can be identified: an absolute component, which provides a global anchor in an Earth-fixed reference frame; a relative component, which describes the vehicle’s local motion with respect to its previous pose; and an inertial component, which captures high-frequency dynamic variations and ensures continuous estimation even in the temporary absence of external references.

The *Global Navigation Satellite System (GNSS)* represents the absolute basis of localization. It provides latitude, longitude, and altitude in the WGS84 Earth reference frame, enabling the vehicle’s global position to be determined. By employing differential correction techniques such as RTK (Real-Time Kinematic) or DGPS, centimeter-level accuracy can be achieved under favorable conditions.

However, the GNSS signal is sensitive to physical occlusions—such as tunnels, underpasses, or dense urban areas—and has a relatively low update rate, typically between 1 and 10 Hz, which limits its ability to capture rapid vehicle dynamics.

Vehicle odometry is derived from the standard onboard sensors already available on the vehicle, such as wheel speed sensors and steering angle measurement. It provides an incremental estimate of displacement and orientation change between consecutive time steps, with high update frequencies and low latency. These measurements ensure local continuity between two GNSS updates and allow for fine motion tracking, but they are affected by cumulative drift due to calibration errors in wheel radius, tire slip, variations in grip, or steering bias. If uncorrected, these errors tend to accumulate over time, progressively degrading the accuracy of the estimated pose.

The *Inertial Measurement Unit (IMU)* completes the system by providing linear acceleration and angular velocity measurements at very high frequencies (hundreds of hertz). It captures rapid variations in the vehicle’s dynamics and bridges temporary gaps caused by GNSS signal loss. However, the IMU is also affected by noise and bias, which, when integrated over time, result in drift. For this reason, its data are systematically fused with those from the GNSS and odometry, allowing the respective weaknesses of each source to be mutually compensated.

Within the eWolf architecture, the localization system currently integrates GNSS and odometry, with the planned addition of the IMU to increase accuracy and robustness in critical environments such as tunnels, dense urban areas, or partially shielded spaces. This multimodal configuration ensures that the vehicle always maintains a coherent state estimate, even in the presence of temporary signal degradation or loss, preserving navigation continuity and spatial consistency for the downstream modules.

2.2.2 Localization Pipeline

The flow of localization data follows an ordered and deterministic sequence that ensures temporal, spatial, and numerical consistency among the different sensor sources. First, measurements coming from sensors with different frequencies and latencies are temporally aligned: each sample is associated with a timestamp and interpolated on a common time scale, ensuring that comparisons among the GNSS, odometry, and IMU always refer to the same physical instant. A spatial alignment then follows, where the measurements are expressed both in a global reference frame and in a local frame attached to the vehicle, which is useful for describing instantaneous motions and integrating kinematic quantities.

Once space–time consistency is ensured, the measurements are combined by an estimation filter that applies a recurrent Bayesian logic. Between two consecutive GNSS updates, the vehicle state—including position, velocity, and orientation—is

propagated forward through the kinematic model driven by odometry and IMU data. When a new satellite measurement becomes available, the estimate is corrected, reducing the accumulated error. This predict–correct mechanism, typical of Extended Kalman Filters (EKF) or more advanced Particle Filters (PF), produces a smooth and coherent vehicle trajectory, with a stable velocity estimate and a confidence indicator that quantifies the quality of the pose at each instant.

In critical scenarios—such as tunnels, urban sections with strong signal occlusion, or temporary losses of coverage—the system continues to operate in *dead reckoning* mode, using only onboard information (odometry and IMU) to maintain pose continuity. During this phase, the uncertainty associated with the estimate progressively increases, and the higher-level system is informed of the growing error. When the GNSS signal becomes available again, the filter gradually re-synchronizes, avoiding discontinuities or sudden jumps that could affect planning or generate inconsistent control commands. Similarly, if odometry becomes less reliable—for instance, due to wheel slip or sudden variations in tire grip—the filter automatically rebalances the weights assigned to each source, prioritizing the most consistent measurements.

At the output, the localization module provides a unique and continuous estimate of the vehicle’s position, orientation, and velocity, updated several times per second and accompanied by a quality index that reflects the estimated uncertainty. These data serve as a common reference for the Planning and Control modules: the planner uses the pose to generate trajectories in the correct spatial frame, while the controller relies on it to apply timely and stable corrections. The Perception module can also benefit from this information, for instance, to compensate for motion during image projection or to stabilize the LiDAR point cloud.

The entire pipeline is designed to operate at frequencies on the order of tens of hertz, with low overall latency and high numerical stability—fundamental requirements to guarantee constant alignment among perception, planning, and control throughout every operational cycle.

2.3 Planning

Within the context of autonomous driving systems, the planning module acts as the logical interface between perception and localization—responsible for describing the environment and the vehicle state—and the control modules, which generate the physical commands for the actuators. Its role is to translate a description of the world (road geometry, lane boundaries, obstacles, and surrounding traffic) into a coherent and feasible motion plan, specifying where the vehicle should be in the upcoming moments, with which orientation and at what speed. In other words, planning does not execute the motion itself but constructs a continuous space–time

reference that the control system will track.

In modern autonomous driving architectures, planning is not a monolithic block but a hierarchical process structured over multiple horizons, from long-term to short-term. At the highest level lies the *route planning* (or mission planning) stage, which aims to determine a global route from the current position to a specified destination, typically on an urban or interurban scale. This level operates on structured road maps or high-definition (HD) maps and produces a sequence of road segments, exits, junctions, and ramps to be followed. Conceptually, it is similar to classical navigation: it defines which roads to take, but not yet how to move dynamically along them.

On a shorter horizon operates the *behavior planning* (or decision making) layer, which selects the most appropriate tactical maneuver based on the local scene and traffic rules. This level decides, for instance, whether to stay in the current lane or initiate a lane change, whether to follow a slower vehicle or overtake it, whether to cross an intersection or yield, and whether to stop or proceed. These decisions account for the position and velocity of other vehicles, predictions of their future behavior, and normative or safety constraints. The output of the behavior planner is a driving intention that defines qualitative constraints on where the vehicle is allowed to move and the maximum admissible speed profile.

Following this phase comes what is referred to in the literature as *motion planning* or *local trajectory planning*. This is the layer that produces the trajectory actually trackable by the vehicle and is therefore directly connected to the control system. The motion planner generates, over a time horizon of a few seconds and a spatial horizon of several tens of meters ahead of the vehicle, a continuous trajectory in space and a corresponding velocity profile over time that are both kinematically and dynamically feasible. In practice, it provides a curve in the road plane—typically represented as a sequence of waypoints with position, orientation, and curvature—together with the desired velocity at each point. This trajectory must satisfy safety constraints (no predicted collisions with static or dynamic obstacles), geometric constraints (lane adherence, compliance with road boundaries and markings), and physical constraints of the vehicle (curvature limits, maximum lateral acceleration, braking capability, and admissible longitudinal jerk). The output is not merely a line to follow but a continuous space–time trajectory, with at least C^1 or C^2 continuity, designed to be comfortable, stable, and realizable by the actual actuators.

To generate this local trajectory, industrial and research systems generally employ two main families of approaches. The first is discrete planning in the configuration space: the drivable space is discretized (for instance, in a lattice or a set of admissible paths), and search algorithms are applied to evaluate alternative options, estimating costs related to safety, lane deviation, path length, and smoothness. This approach originates from classical mobile robotics and was adopted by the first

experimental autonomous vehicles and DARPA Challenge participants, where the scene was explored by generating and selecting the optimal trajectory among many candidates. The second family is continuous optimization over a finite horizon, conceptually close to Model Predictive Control (MPC). Instead of selecting from predefined trajectories, the optimal trajectory is obtained directly as the solution to a constrained optimization problem that minimizes a cost function combining deviation from the desired path, proximity to obstacles, comfort (limited jerk and acceleration), compliance with the requested speed profile, and adherence to the vehicle’s dynamic constraints. In both cases, the output of the motion planner is a trajectory already consistent with the kinematic and dynamic limits of the vehicle, ready to be tracked by the controller.

A key feature of modern planning is its operation under a *receding horizon*. This means that the local plan is not fixed once and for all but is continuously regenerated at frequencies on the order of tens of hertz, incorporating updates from perception (new obstacle detections, changing traffic, updated lane boundaries) and localization (updated vehicle pose). This continuous replanning allows the system to adapt quickly to evolving situations without generating impulsive commands: small variations in perception do not cause abrupt behavioral changes but are reflected in gradual adjustments of trajectory and speed. Moreover, when uncertainty increases—such as in tunnels, adverse weather, or occluded environments—the planner degrades gracefully: it lowers the maximum allowable speed, increases safety distances, avoids aggressive lateral maneuvers, and communicates to downstream modules that the quality of the reference has decreased. Such prudential behavior is fundamental both from a functional safety perspective and for certification purposes.

The final output of the planning module is not a direct command to the actuators but a structured reference. Typically, it consists of a time-parameterized sequence of future waypoints, each with position, orientation, local curvature, and desired velocity, and in some cases, target longitudinal acceleration and jerk limits. These references are then provided to the lateral and longitudinal control modules. At that point, the control layer closes the dynamic loop: it compares the actual vehicle state with the desired state along the trajectory and computes the physical inputs (steering angle, motor torque, braking force) while respecting actuator saturations and vehicle stability constraints. In summary, planning defines the immediate and locally feasible motion strategy, while control ensures its physical execution. The quality of the planner—in terms of geometric smoothness, respect of dynamic limits, ability to anticipate obstacles, and robustness against perceptual uncertainty—largely determines the safety, smoothness, and comfort of the overall autonomous driving behavior.

2.4 Control

The control module represents the final stage of the functional chain in an autonomous driving system. It receives the reference trajectory and velocity profile generated by the planning module and translates them into physical commands for the steering, throttle, and braking actuators. It is the link that transforms the theoretical motion plan into real vehicle dynamics, ensuring that the vehicle follows the desired path while respecting mechanical limits, tire adhesion capabilities, and passenger comfort thresholds. In a complete architecture, control forms the direct interface with the physical world—where mathematical laws and regulation strategies become tangible motion, accelerations, and forces.

In a system classified as SAE Level 2, as in the case of the eWolf platform, the control module maintains lane keeping and regulates speed and inter-vehicle distance in a smooth and predictable way, while leaving supervision to the human driver. Its organization follows a clear and modular structure: the high-level computation is implemented within ROS 2 software nodes, whereas the real-time execution of the control commands takes place on the Speedgoat unit, which acts as a deterministic interface toward the actuators. This separation ensures that the most critical operations—such as the generation and application of steering and torque commands—are handled with precise timing and minimal latency, providing fast, repeatable, and dynamically consistent responses.

From a functional perspective, vehicle control is traditionally divided into two interdependent channels: *lateral control* and *longitudinal control*. The lateral control aims to keep the vehicle aligned with the reference trajectory, minimizing the cross-track error and heading error while ensuring smooth, oscillation-free maneuvers. In eWolf, several approaches have been implemented and compared under homogeneous conditions:

- **Pure Pursuit**, based on a geometric look-ahead strategy that steers toward a target point located ahead along the trajectory;
- **Stanley controller**, which combines lateral and angular errors in a nonlinear control law, enriched with a dynamic look-ahead term to stabilize the response at higher speeds;
- **Model Predictive Control (MPC)**, which formulates steering regulation as a constrained optimization problem over a finite horizon, taking into account steering angle and rate limits, lateral acceleration, and admissible jerk.

The goal of implementing this variety of controllers is twofold: first, to experimentally assess the differences in terms of tracking accuracy, stability, and command smoothness; and second, to construct a comparative framework linking the numerical performance of each controller to the subjective driving experience—namely,

dynamic comfort and naturalness of motion.

On the longitudinal side, the adopted strategy differs from classic purely reactive PID controllers. Velocity is neither held constant nor controlled solely in proportion to the tracking error but adapted as a function of the road geometry. Starting from the reference centerline and the local curvature over a short preview horizon, the controller computes a target speed that respects a predefined lateral acceleration limit, typically between 2.4 and 3 m/s². When the road curvature increases, the target velocity gradually decreases; when the curvature diminishes, the target velocity increases smoothly. The resulting profile is continuous and smoothed through temporal filters that limit changes in acceleration (rate limiting) and jerk, preventing oscillations and unwanted spikes. In the presence of a slower preceding vehicle, the module integrates an Adaptive Cruise Control (ACC) logic: the actual vehicle speed becomes the minimum between the curvature-based limit and the safe time-gap speed. In this way, both geometric and longitudinal safety are respected, and velocity modulation remains coherent with the scene and comfort constraints—without reactive or abrupt deceleration.

Coordination between the two control channels is continuous and bidirectional. The curvature of the trajectory provides the lateral controller with a feed-forward term that anticipates cornering, reducing tracking error, while simultaneously constraining the longitudinal controller to maintain a speed compatible with tire–road adhesion and comfort limits. Conversely, when the longitudinal control reduces speed for safety or comfort reasons, the lateral channel benefits from more favorable dynamics, requiring smaller and more achievable steering efforts within mechanical limits. Both channels operate under limiters and filters that ensure physical realism: the steering angle is subject to saturation and rate limitation to contain perceived lateral jerk, while throttle and braking actions are filtered to avoid impulsive transitions. If saturation or physically implausible conditions occur, the protection logic intervenes by gradually reducing the control signals, bringing the system back into a safe operating region.

From a temporal standpoint, the control loop operates at high update frequencies—on the order of tens to hundreds of hertz—with low latency, consistent with the vehicle’s dynamic time constants. High-frequency processing is confined to the real-time Speedgoat unit, while ROS 2 modules manage reference generation and communication with other subsystems. This distributed architecture allows heavier computational loads (perception, planning, logging) to remain decoupled from time-critical operations, preserving the stability and coherence of the control system even under computational stress.

At the output level, the fundamental requirement of control is smoothness: continuous and filtered inputs must produce equally continuous and physically plausible actuator commands. Steering angles and rates, longitudinal accelerations, and jerks are always constrained within predefined thresholds so that the resulting

motion respects both the mechanical limits of the vehicle and passenger comfort boundaries.

In summary, the control module represents the point of contact between theory and reality. In eWolf, it translates the trajectory and target speed generated by the planning module into stable, clean, and safe actuator commands—built upon a well-conditioned lateral controller and a curvature-based longitudinal control strategy. Both are coordinated under a logic that prioritizes stability, predictability, and compliance with physical limits. In the following chapters, the analysis of individual controllers and their quantitative comparison will connect the numerical behavior of the system to the perceived driving quality, demonstrating how precision, smoothness, and comfort arise from a coherent balance between planning and control.

Chapter 3

Path Following

Path following represents one of the most delicate and significant phases in the control of an autonomous vehicle. After the planning system has defined the ideal trajectory to be followed, the task of this module is to translate it into concrete actions—namely, steering and acceleration commands capable of keeping the vehicle aligned with the path in a stable, smooth, and safe manner. At this stage, the planner’s decisions become actual motion, and the quality of control determines the accuracy, stability, and driving comfort of the system.

To achieve this goal, the process starts from a kinematic model that provides a simplified yet effective description of the relationship between steering commands, vehicle speed, and motion. Through this model, it is possible to quantify position and heading errors with respect to the target trajectory. On these foundations, different control algorithms are developed, which may vary in complexity and operating principle—from intuitive geometric methods such as the Pure Pursuit and the Stanley Controller to more advanced predictive approaches such as the Model Predictive Controller (MPC), capable of optimizing maneuvers while accounting for constraints and vehicle dynamics.

This chapter provides a detailed description of the theoretical and practical foundations of these control strategies, illustrating how they enable the vehicle to follow the planned trajectory with precision and stability. All the methods presented are concretely applied within the eWolf project, where the controllers have been implemented, tested, and compared under various driving conditions, with the goal of identifying the most effective solution for trajectory tracking.

3.1 Vehicle Model

The vehicle model constitutes the mathematical foundation of the path following module, as it describes the relationship between control inputs—steering angle and

acceleration—and the evolution of the vehicle state in terms of position, orientation, and velocity. Without an adequate model, the controller would be unable to predict how steering and acceleration actions influence the trajectory, making it impossible to guarantee stability and consistent adherence to the reference path. The goal is therefore to obtain a simplified yet coherent kinematic representation capable of clearly expressing how steering determines the curvature of the path and how acceleration regulates the longitudinal dynamics, while maintaining a computational load compatible with real-time execution.

The vehicle model acts as a bridge between geometry and control: on one side, it provides the basis for computing fundamental quantities such as the cross-track error and the heading error, which quantify the deviation of the vehicle with respect to the reference trajectory; on the other side, it enables the estimation of physical quantities such as curvature or lateral acceleration, directly derived from steering angle and vehicle speed. These relationships make it possible to introduce physical and comfort-related constraints into the control laws, ensuring that the generated commands are always realistic and feasible.

In the following sections, the *Kinematic Bicycle Model*—a planar, simplified yet accurate representation of vehicle motion—will be presented. Moreover, the definitions of tracking errors in the *Frenet reference frame* will be introduced, forming the common basis for all the controllers discussed in the subsequent sections.

3.1.1 Kinematic Bicycle Model

The Kinematic Bicycle Model is one of the most widely used and well-established formulations in the literature for describing the planar motion of a road vehicle[26]. It was developed to obtain a mathematical representation that is sufficiently accurate in describing the vehicle’s kinematic behavior, yet simple and computationally light enough to be implemented in real-time control systems. Its popularity stems from the fact that, while neglecting dynamic effects such as inertia, lateral forces, and tire deformation, it preserves the fundamental geometric relationship between steering angle, trajectory curvature, and vehicle yaw rotation, thus providing a coherent and interpretable framework of motion.

The basic idea of the model is to replace the vehicle’s four real wheels with two virtual ones: a front wheel capable of steering by an angle δ , and a rear wheel rigidly connected to the chassis. This single-track simplification, from which the name “bicycle” model derives, reduces the number of kinematic variables to be considered while preserving the essential features of the vehicle’s motion, allowing a faithful representation of the center of gravity (CoG) motion and orientation variation.

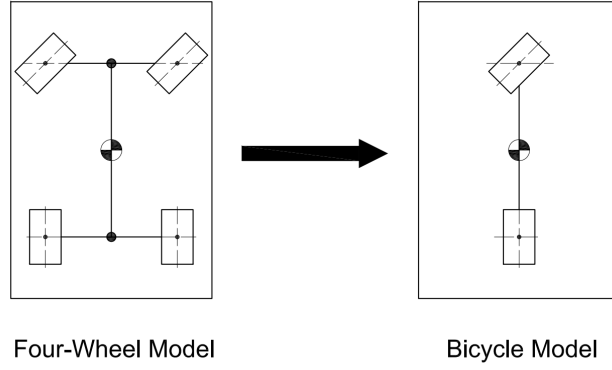


Figure 3.1: From Four-Wheel to Bicycle Model

This kinematic model relies on several key assumptions that define its validity domain:

- **Planar motion:** the vehicle is treated as a rigid body moving on a horizontal plane; roll and pitch dynamics are neglected.
- **No lateral slip:** tires are assumed to operate under pure rolling conditions, meaning that the velocity vector at each wheel contact point is perfectly aligned with the wheel plane—lateral slip angles are therefore negligible.
- **Constant geometry:** the vehicle is characterized by a constant wheelbase L , defined as the distance between the front and rear axles, and by two geometric parameters l_f and l_r , representing the distances from the CoG to the front and rear axles, such that $L = l_f + l_r$.

Under these assumptions, the vehicle trajectory can be modeled exclusively as a function of the longitudinal velocity v and the steering angle δ , without the need to include unmeasurable quantities such as lateral forces or slip angles. For this to remain valid, maneuvers and speeds must stay within moderate ranges, ensuring that dynamic effects such as lateral inertia and tire stiffness remain negligible compared to the purely kinematic terms.

The main state variables of the model are the coordinates of the center of gravity (X, Y) in the global reference frame, the yaw angle ψ —which defines the vehicle's orientation with respect to the X -axis—and the longitudinal velocity v . The control inputs are the steering angle δ and the longitudinal acceleration a . A key intermediate variable is the so-called sideslip angle β , defined as the difference between the direction of the CoG velocity vector and the vehicle's longitudinal axis. This angle accounts for the fact that, even under pure rolling, the CoG does

not move exactly along the vehicle heading, as the instantaneous center of rotation does not coincide with it.

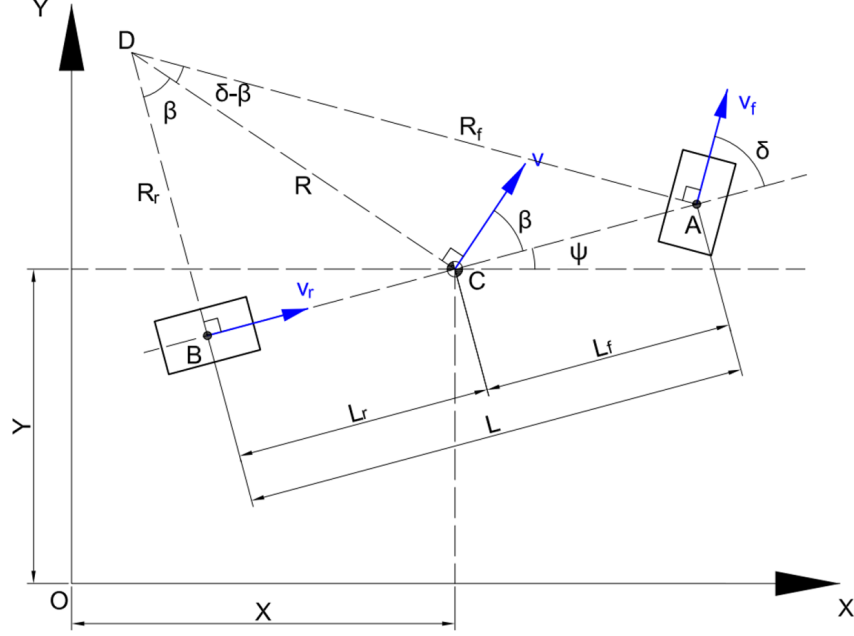


Figure 3.2: Bicycle Kinematic Model

By imposing that the velocities of the front and rear contact points are both directed toward the common center of curvature, β can be expressed as a function of the steering angle and the vehicle geometry:

$$\beta = \arctan \left(\frac{l_r}{L} \tan \delta \right) \quad (3.1)$$

showing that β increases with the steering angle and depends on the ratio between the CoG position and the axle distances.

Assuming the hypotheses described above, the continuous-time kinematic equations describing the evolution of the vehicle state can be written as:

$$\dot{X} = v \cos(\psi + \beta) \quad (3.2)$$

$$\dot{Y} = v \sin(\psi + \beta) \quad (3.3)$$

$$\dot{\psi} = \frac{v}{L} \tan \delta \cos \beta \quad (3.4)$$

$$\dot{v} = a \quad (3.5)$$

The first two equations describe the CoG translation projected along the global axes. The third equation relates the yaw rate $\dot{\psi}$ to the longitudinal speed, wheelbase, and steering angle, taking into account the sideslip angle β : higher steering or speed result in faster vehicle rotation. The last equation defines the variation of velocity as the result of the applied longitudinal acceleration.

From these equations, one can derive the fundamental relationship between steering angle and curvature κ of the CoG trajectory:

$$\kappa = \frac{1}{R} = \frac{\dot{\psi}}{v} = \frac{\tan \delta}{L \cos \beta} \quad (3.6)$$

showing that curvature is proportional to the tangent of the steering angle and inversely proportional to the wheelbase. For small steering and sideslip angles, $\cos \beta \approx 1$, yielding the simplified form:

$$\kappa \approx \frac{\tan \delta}{L} \quad (3.7)$$

The trajectory curvature is directly related to the lateral acceleration experienced by the vehicle:

$$a_y = v^2 \kappa = \frac{v^2}{L} \tan \delta \cos \beta \quad (3.8)$$

This relation shows that lateral acceleration increases quadratically with velocity and linearly with steering angle. Therefore, even with the same steering input, higher speeds lead to rapidly growing lateral forces and a higher risk of loss of grip. This connection is crucial for introducing physical and comfort constraints within the controllers, as it allows imposing direct limits on the maximum lateral acceleration and, consequently, on the allowable speed in curves.

In the low- and medium-speed conditions typical of the eWolf experimental vehicle, where δ and β remain small, the approximations $\tan \delta \approx \delta$ and $\cos \beta \approx 1$ lead to a simplified model:

$$\dot{X} = v \cos \psi \quad (3.9)$$

$$\dot{Y} = v \sin \psi \quad (3.10)$$

$$\dot{\psi} = \frac{v}{L} \tan \delta \quad (3.11)$$

$$\dot{v} = a \quad (3.12)$$

This reduced formulation, known as the *simplified kinematic model*, is the one most commonly used in real-time control applications, as it preserves the essential relationship between control inputs and vehicle state evolution while minimizing computational cost [27]. It forms the foundation for geometric controllers such as Pure Pursuit and Stanley [28, 29], and serves as the linearization basis for the MPC formulation used within eWolf.

For digital implementation, the continuous equations are discretized with integration step Δt using a forward Euler scheme:

$$X_{k+1} = X_k + v_k \cos(\psi_k + \beta_k) \Delta t \quad (3.13)$$

$$Y_{k+1} = Y_k + v_k \sin(\psi_k + \beta_k) \Delta t \quad (3.14)$$

$$\psi_{k+1} = \psi_k + \frac{v_k}{L} \tan \delta_k \cos \beta_k \Delta t \quad (3.15)$$

$$v_{k+1} = v_k + a_k \Delta t \quad (3.16)$$

where $\beta_k = \arctan\left(\frac{l_r}{L} \tan \delta_k\right)$. For a sampling frequency of 20 Hz (i.e., $\Delta t = 0.05$ s), as used in eWolf and typical of ROS 2-based systems, this discretization ensures more than adequate accuracy.

In the specific case of the Model Predictive Controller, the vehicle model is linearized around the current operating point $(\bar{x}, \bar{u})$, yielding a linear time-varying (LTV) representation with matrices A_d and B_d updated at each time step. This enables predicting future state evolution without excessive computational load, maintaining model validity even under variations in speed or steering angle.

The validity of the Kinematic Bicycle Model is limited to operating conditions in which steering angles and lateral accelerations remain moderate. In extreme maneuvers—where tires operate in the nonlinear regime and slip angles become significant—a more complex dynamic model, including lateral forces and tire stiffness, would be required. However, such models, though more accurate, demand parameters that are difficult to identify experimentally and entail a higher computational burden.

For the objectives of this thesis, the kinematic model thus represents an optimal compromise between physical fidelity and analytical simplicity. It accurately reproduces the planar motion of the vehicle within the expected operating range and provides all quantities required by the path-following controllers: trajectory curvature, yaw rate, and lateral acceleration. The Kinematic Bicycle Model therefore serves as the common foundation for all control strategies described in the following sections—from the geometric Pure Pursuit and Stanley laws to the predictive optimization-based MPC approach.

3.1.2 Frenet Reference Frame and Tracking Errors

In the context of path following, evaluating tracking accuracy means quantifying how far the vehicle is from the reference trajectory and how its orientation differs from the desired one. To express these errors in a consistent way, it is convenient to abandon the global coordinate system (X, Y) and describe the vehicle dynamics in a local frame attached to the reference path, known as the *Frenet reference frame* [27].

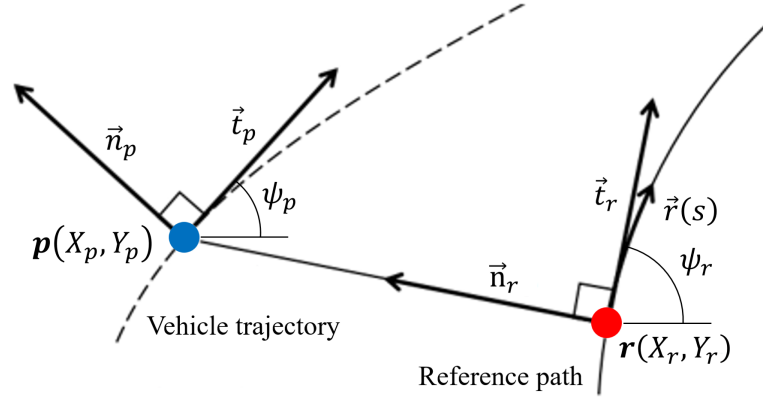


Figure 3.3: Frenet reference frame for Path following

The Frenet frame is constructed along the reference path $\gamma(s)$, parameterized by its arc length s . At each point of the path, defined by coordinates $(X_r(s), Y_r(s))$, two orthogonal unit vectors are identified: the tangent $\mathbf{t}(s)$ and the normal $\mathbf{n}(s)$, obtained by rotating the tangent by 90° counterclockwise. The angle between the tangent and the global X -axis is the path orientation $\psi_r(s)$, while the curvature $\kappa_r(s)$ describes how this angle varies along the path:

$$\kappa_r(s) = \frac{d\psi_r(s)}{ds} \quad (3.17)$$

The moving frame $\{\mathbf{t}(s), \mathbf{n}(s)\}$ thus travels along the path, adapting locally to its shape and allowing the vehicle position to be expressed relative to the trajectory rather than in absolute global coordinates.

To relate the vehicle state (X, Y, ψ, v) to the reference path, the closest point projection onto the curve is computed, i.e., the point $(X_r(s^*), Y_r(s^*))$ minimizing the distance to the vehicle. The corresponding curvilinear coordinate s^* defines the position on the path associated with the current vehicle location. This projection operation is essential, as it anchors the vehicle's kinematic quantities—position, orientation, and velocity—to a local frame moving with the reference path.

Once the projection is obtained, two fundamental quantities for lateral control are defined:

- the **cross-track error** e_y , representing the lateral distance from the vehicle to the path, measured along the normal direction at s^* ;
- the **heading error** e_ψ , defined as the angular difference between the vehicle heading and the tangent direction of the path at the same point.

These quantities are formally expressed as:

$$e_y = \begin{bmatrix} X - X_r(s^*) \\ Y - Y_r(s^*) \end{bmatrix} \cdot \begin{bmatrix} -\sin \psi_r(s^*) \\ \cos \psi_r(s^*) \end{bmatrix} \quad (3.18)$$

$$e_\psi = \text{wrap}(\psi_r(s^*) - \psi) \quad (3.19)$$

where the function $\text{wrap}(\cdot)$ ensures that the resulting angle remains within the interval $(-\pi, \pi)$, preventing numerical discontinuities. By convention, e_y is positive when the vehicle lies to the left of the path and negative otherwise.

Physically, e_y quantifies the lateral offset of the vehicle from the reference trajectory, while e_ψ measures the angular misalignment between the vehicle's heading and the tangent of the desired path. Together, they fully describe the vehicle's relative position with respect to the trajectory and form the core feedback variables for most path-following controllers.

To understand how these errors evolve over time, the Frenet equations are derived from the kinematic bicycle model [26]. Assuming that the longitudinal velocity is aligned with the vehicle's heading and denoting the sideslip angle by β and the steering angle by δ , the following relations hold:

$$\dot{s} = \frac{v \cos e_\psi}{1 - \kappa_r e_y} \quad (3.20)$$

$$\dot{e}_y = v \sin e_\psi \quad (3.21)$$

$$\dot{e}_\psi = \frac{v}{L} \tan \delta \cos \beta - \kappa_r \dot{s} \quad (3.22)$$

These equations show how vehicle dynamics interact with the path geometry. In particular, $\dot{e}_y$ depends on the heading error: if the vehicle is rotated left with respect to the path tangent ($e_\psi > 0$), the lateral velocity component increases e_y . Conversely, $\dot{e}_\psi$ depends on the steering term $(v/L) \tan \delta$ —which tends to reduce the orientation error—and on the curvature term $\kappa_r \dot{s}$, representing the path-induced yaw rate: if the path curves left while the vehicle continues straight, e_ψ increases.

For small errors and smooth trajectories, the system can be linearized into a more intuitive form:

$$\dot{s} \approx v \quad (3.23)$$

$$\dot{e}_y \approx v e_\psi \quad (3.24)$$

$$\dot{e}_\psi \approx \frac{v}{L} \tan \delta - \kappa_r v \quad (3.25)$$

This linear formulation highlights the main relationships:

- the lateral error evolves as a function of the heading misalignment;
- the angular error is directly controlled by the steering command δ ;
- the path curvature κ_r acts as a disturbance that must be compensated by the controller.

From a physical standpoint, these equations reveal that lateral control does not act directly on e_y , but on e_ψ : the controller adjusts δ to reduce e_ψ , and the gradual reorientation of the vehicle subsequently decreases the cross-track error. This principle underlies all major path-following strategies—Pure Pursuit, Stanley, and MPC—which all, either explicitly or implicitly, regulate both errors jointly.

In practical implementations, the definitions of e_y and e_ψ may vary depending on the chosen vehicle reference point. While the kinematic model typically uses the center of gravity, it is often more convenient to reference the front axle, where the steering command is applied. This translation is expressed as:

$$\begin{bmatrix} X_A \\ Y_A \end{bmatrix} = \begin{bmatrix} X \\ Y \end{bmatrix} + l_f \begin{bmatrix} \cos \psi \\ \sin \psi \end{bmatrix} \quad (3.26)$$

Using the front axle as reference improves controller responsiveness and reduces geometric lag. Similarly, in the Stanley controller, a dynamic lookahead is introduced when evaluating ψ_r : instead of computing the tangent at the closest point s^* , the path orientation is taken at a forward-shifted point $s^* + \Delta s(v)$, where Δs grows with vehicle speed. This modification mitigates sensitivity to small path oscillations, resulting in smoother and more stable control behavior.

In conclusion, the Frenet reference frame provides a clear and rigorous way to describe the vehicle's position and orientation relative to the path, allowing all lateral controllers to share a uniform formulation. The variables e_y and e_ψ form the common foundation upon which the Pure Pursuit, Stanley, and MPC controllers are built: Pure Pursuit relies mainly on the lateral offset component, Stanley explicitly uses both errors for correction, and MPC incorporates them directly into its predictive state to minimize both over time.

3.2 Lateral Controllers

Lateral control represents the core of the path-following task, as it is responsible for generating the steering command that allows the vehicle to follow the desired trajectory in a stable, accurate, and smooth manner. Among the various strategies proposed in the literature, a central role is played by *geometric model-based controllers* [27], one of the most widely adopted classes for path-following applications. These controllers are based on the vehicle’s kinematic description but define the steering law through geometric relationships between the vehicle’s position and orientation relative to the reference trajectory, without solving any optimization problem or explicitly accounting for the dynamic forces involved.

This characteristic makes geometric controllers particularly suitable for real-time implementation, thanks to their simplicity, clear physical interpretation, and robustness to variations in speed and curvature.

In the eWolf project, two representative approaches of this category were implemented and compared: the *Pure Pursuit Controller* and the *Stanley Controller*. The Pure Pursuit adopts a purely geometric approach, in which the vehicle adjusts its steering angle to intercept a lookahead point located ahead along the trajectory, ensuring a smooth and stable response. The Stanley controller, on the other hand, can be regarded as a kinematic geometric controller, as it explicitly uses the kinematic variables—namely the lateral and heading errors—to compute a proportional steering action, resulting in faster and more precise convergence.

The analysis of these two controllers provides a clear understanding of the principles underlying geometric lateral control, highlighting how different approaches, built upon the same kinematic foundations, can yield complementary behaviors. These geometric methods also provide a solid baseline for comparison with more advanced predictive control strategies, such as those presented in the following sections.

3.2.1 Pure Pursuit Controller

Among geometric controllers for path following, the *Pure Pursuit* represents one of the most popular and long-standing methods, appreciated for its conceptual simplicity and operational robustness in real driving scenarios. Its logic is based on determining, at each control step, the steering angle required for the vehicle to follow a circular arc connecting its current position with a reference point located ahead along the trajectory. This point, known as the *lookahead point*, serves as a local target that the vehicle must “pursue” to maintain proper alignment with the path [28, 27].

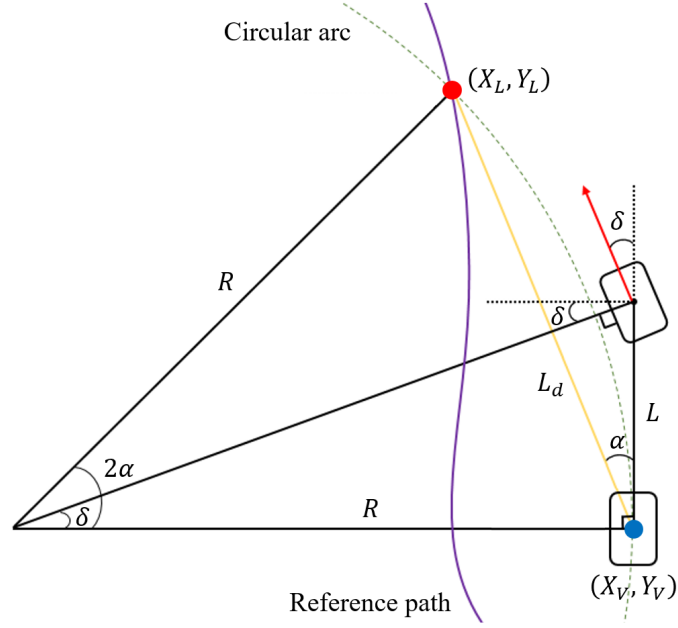


Figure 3.4: Pure Pursuit control method

Let (X, Y, ψ) denote the vehicle pose in the global frame and (X_L, Y_L) the coordinates of the lookahead point along the reference path, located at a distance L_d from the vehicle. The angle α is defined as the difference between the vehicle heading and the direction of the line connecting the vehicle to the lookahead point:

$$\alpha = \text{atan2}(Y_L - Y, X_L - X) - \psi \quad (3.27)$$

normalized in the range $(-\pi, \pi)$. Assuming that the vehicle, under a constant steering angle δ , follows a circular arc of radius R that intersects the lookahead point, a simple geometric derivation yields the curvature of the arc as:

$$\kappa = \frac{2 \sin \alpha}{L_d} \quad (3.28)$$

Since, in the kinematic bicycle model, curvature and steering angle are related by $\kappa = \tan \delta / L$, where L is the vehicle wheelbase, the classical Pure Pursuit control law follows:

$$\delta = \arctan \left(\frac{2L \sin \alpha}{L_d} \right) \quad (3.29)$$

This expression directly provides the steering angle required for the vehicle's trajectory to intersect the lookahead point. For small angular deviations, $\sin \alpha \approx \alpha$, and the law becomes linear:

$$\delta \approx \frac{2L}{L_d} \alpha \quad (3.30)$$

highlighting that the ratio $2L/L_d$ acts as a proportional gain: smaller lookahead distances L_d yield more reactive steering, while larger L_d values produce smoother but less precise responses on tight curves.

The lookahead distance is not fixed but varies dynamically with vehicle speed. In eWolf, it is defined as:

$$L_d(v) = \max(L_{d,\min}, k_L v^\gamma) \quad (3.31)$$

where $L_{d,\min}$ is the minimum allowed distance, k_L an empirical gain, and $\gamma \approx 0.83$ a sublinear exponent. This formulation ensures that, at low speeds, the vehicle uses a short lookahead distance to precisely track tight curves, while at higher speeds, the reference point moves further ahead, promoting gradual responses and reducing lateral acceleration.

Geometrically, the lookahead point is determined as the first path point satisfying two conditions: (i) it must lie in front of the vehicle with respect to its current heading, and (ii) its distance must be greater than or equal to L_d . In practice, the reference path $\mathbf{r}(s) = (X_r(s), Y_r(s))$, represented as a spline, is sampled starting from the previous valid curvilinear coordinate s_{prev} , preventing backward jumps along the trajectory. For each sampled point, the scalar product between the vehicle's heading vector and the vector connecting the vehicle position to the path point is computed; when this product is positive and the distance exceeds L_d , the point is selected as the lookahead point and s_{prev} is updated. This straightforward yet robust algorithm avoids oscillations due to abrupt target changes and minimizes computational effort by scanning the path monotonically forward.

Once the lookahead point is found and α is computed, the ideal steering command is obtained by directly applying the Pure Pursuit law above. In implementation, the function `atan2` is used instead of `atan` to correctly handle the sign of α and ensure steering direction continuity across quadrants, even under abrupt orientation changes.

Although the Pure Pursuit controller does not explicitly use the tracking errors e_y and e_ψ introduced in the previous section, it can be interpreted as indirectly acting on the lateral deviation of the lookahead point. Indeed, defining a virtual lateral error $e_y^{(\text{PP})} = L_d \sin \alpha$, the steering law can be rewritten as:

$$\delta = \arctan \left(\frac{2L e_y^{(\text{PP})}}{L_d^2} \right) \quad (3.32)$$

which corresponds to a proportional feedback on the lateral offset of the target point. This equivalence clarifies the conceptual link between Pure Pursuit and

Frenet-based controllers: both aim to reduce a geometric deviation measure, but Pure Pursuit does so by “looking ahead” by a distance L_d , while controllers such as Stanley act on the nearest projection point on the path.

An additional aspect concerns the relationship between chosen curvature and vehicle kinematics. From the bicycle model, $\dot{\psi} = (v/L) \tan \delta$ and $a_y = v^2 \kappa = (v^2/L) \tan \delta$. Combining these with the Pure Pursuit law yields:

$$a_y = \frac{2v^2}{L_d} \sin \alpha \quad (3.33)$$

showing that lateral acceleration increases quadratically with speed and decreases with lookahead distance. This demonstrates that a dynamic lookahead not only improves stability but also helps maintain lateral acceleration within predefined comfort limits.

From an implementation standpoint, the control logic used in the eWolf controller node reproduces this theoretical formulation while integrating geometric and numerical refinements for real-time ROS 2 execution. After extracting the vehicle pose and heading ψ , the dynamic lookahead distance is computed according to the above relation. The reference point is then determined by iteratively scanning the spline path with a step $\Delta s = 0.05$ m until the minimum distance condition L_d is met. The variable s_{prev} is updated at each control cycle to guarantee monotonic progression along the path.

Overall, the Pure Pursuit provides an effective and computationally lightweight solution to the path-following problem, relying solely on geometric relations between the vehicle pose and the reference trajectory. Its simplicity makes it well-suited for embedded real-time systems, while the dynamic lookahead adaptation confers remarkable flexibility. In the eWolf platform, the controller implementation also integrates steering limitation and filtering modules—described in Chapter 4—that enforce actuator and comfort constraints. This allows Pure Pursuit to operate as a stable and predictable baseline controller, capable of following the reference trajectory smoothly while maintaining dynamic consistency with the vehicle’s physical limits.

3.2.2 Stanley Controller

The *Stanley Controller* is one of the most established methods in the field of path following for autonomous vehicles, combining the simplicity of geometric controllers with the precision of error-based regulators. Unlike the Pure Pursuit, which relies on a purely geometric construction to chase a lookahead point, the Stanley controller directly aims to reduce two fundamental tracking errors defined in the Frenet frame: the *cross-track error* e_y , representing the lateral deviation of the vehicle from the reference path, and the *heading error* e_ψ , representing the

difference between the vehicle's orientation and the tangent direction of the path [29, 27].

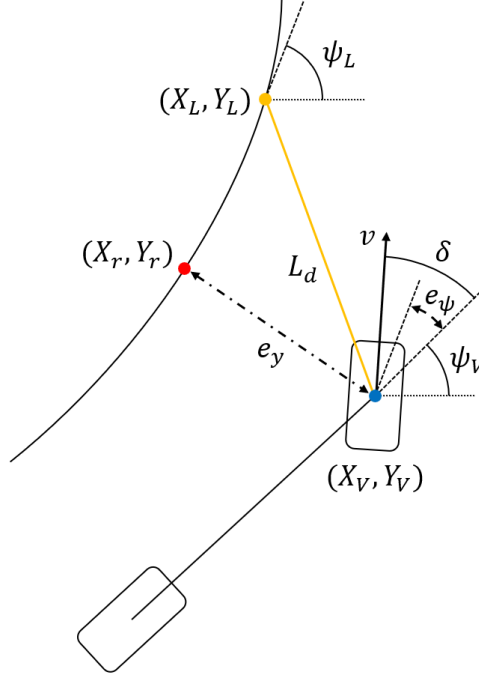


Figure 3.5: Stanley control method

The core idea is that the vehicle should both align with the path direction and converge toward it, compensating for lateral offsets. Starting from the kinematic bicycle model, the relationship between yaw rate $\dot{\psi}$ and steering angle δ is given by [26]:

$$\dot{\psi} = \frac{v}{L} \tan \delta \quad (3.34)$$

where L is the wheelbase and v is the longitudinal velocity. To align the vehicle with the path, the steering command must reduce the heading error $e_\psi = \psi_r - \psi$ while simultaneously compensating for the lateral error e_y . The control law proposed by the Stanley method combines these two effects in a single nonlinear formulation:

$$\delta = e_\psi + \arctan \left(\frac{k e_y}{v} \right) \quad (3.35)$$

The first term, e_ψ , represents the alignment correction, while the second term provides a nonlinear feedback on the lateral deviation. The gain $k > 0$ determines the lateral convergence speed: high values make the response more aggressive, while smaller ones yield smoother and more stable behavior. The velocity term v in the denominator ensures that at low speeds the lateral correction is amplified, allowing quick recovery from large deviations, whereas at high speeds the effect of e_y is attenuated, improving directional stability. The $\arctan(\cdot)$ function also acts as a natural limiter, preventing excessive steering angles in the presence of large lateral errors and thus guaranteeing stable behavior even under transient or noisy conditions.

From an operational standpoint, the control node implemented on eWolf follows this formulation while integrating several key extensions to improve stability and robustness. To compute the tracking errors consistently with the vehicle geometry, the nearest point $(X_r(s), Y_r(s))$ is first found along the spline path with respect to the vehicle's *front axle*, rather than its center of gravity, thus aligning the geometric model with the actual steering point. The search is performed iteratively along the curve with a step $\Delta s = 0.05$ m, starting from the previous curvilinear coordinate s_{prev} , and selecting the first point located ahead of the vehicle according to its heading direction. This condition is verified by ensuring that the scalar product between the vehicle heading vector $\mathbf{h} = [\cos \psi, \sin \psi]$ and the vector connecting the vehicle to the path point $\mathbf{p}_r(s) - \mathbf{p}$ is positive:

$$\langle \mathbf{h}, \mathbf{p}_r(s) - \mathbf{p} \rangle > 0 \quad (3.36)$$

Once the nearest forward point $P_n = (X_r(s_n), Y_r(s_n))$ is found, the tangent and normal unit vectors to the trajectory are computed as:

$$\mathbf{t} = \frac{1}{\|\dot{\mathbf{p}}_r(s_n)\|} \begin{bmatrix} \dot{X}_r(s_n) \\ \dot{Y}_r(s_n) \end{bmatrix}, \quad \mathbf{n} = \begin{bmatrix} -t_y \\ t_x \end{bmatrix} \quad (3.37)$$

The cross-track error e_y is then defined as the lateral distance between the vehicle's front axle position $\mathbf{p}_f = (X_f, Y_f)$ and the nearest point on the reference trajectory $\mathbf{p}_r(s_n)$:

$$e_y = (\mathbf{p}_r(s_n) - \mathbf{p}_f) \cdot \mathbf{n} \quad (3.38)$$

By convention, e_y is positive when the vehicle lies to the left of the path, in accordance with the Frenet frame convention.

The heading error e_ψ is computed as the difference between the vehicle heading ψ and the tangent orientation of the path, but with an important refinement. In

the classical Stanley formulation, ψ_r is evaluated at the nearest path point, which makes the controller sensitive to small geometric irregularities—particularly along straight segments. To mitigate this effect, the eWolf implementation introduces a *dynamic lookahead* for heading evaluation, anticipating the computation of ψ_r at a point further along the trajectory. The lookahead distance increases with speed according to:

$$L_{\text{LA}}(v) = \max(L_{\text{min}}, k_{\text{LA}}v^\gamma) \quad (3.39)$$

The reference heading ψ_r is then defined as the tangent direction at the lookahead point:

$$\psi_r = \text{atan2}(t_y, t_x) \quad (3.40)$$

and the heading error becomes:

$$e_\psi = \text{wrap}(\psi_r(s_n + L_{\text{LA}}) - \psi) \quad (3.41)$$

where $\text{wrap}(\cdot)$ confines the angle within $(-\pi, \pi)$ to prevent discontinuities. This anticipatory computation significantly reduces oscillations and noise sensitivity, resulting in smoother and more stable control without sacrificing cornering accuracy.

With e_ψ and e_y thus defined, the Stanley control law yields the ideal steering angle δ as above, to which actuator limits and filtering are later applied to ensure realistic, continuous behavior in real time.

From a theoretical viewpoint, the Stanley controller can be interpreted as a nonlinear feedback system derived from the linearized Frenet error dynamics:

$$\dot{e}_y = v \sin e_\psi \approx v e_\psi \quad (3.42)$$

$$\dot{e}_\psi = \frac{v}{L} \tan \delta - v \kappa_r \quad (3.43)$$

Substituting the control law into this system results in locally stabilizing feedback for small errors, which explains its effectiveness even in the presence of moderate curvature and speed variations.

In conclusion, the Stanley controller offers an elegant and computationally efficient solution to the path-following problem, balancing precision and stability through a simple yet well-founded control law. The enhanced implementation adopted in the eWolf platform—featuring a dynamic heading lookahead, numerical

filtering, and robust geometric computation—constitutes an optimized version of the original algorithm, specifically tailored for real-time execution within ROS 2 architectures, ensuring stable and realistic performance under all simulated driving conditions.

3.3 Longitudinal Controller

The longitudinal control implemented on the eWolf vehicle is designed to regulate the vehicle speed in a smooth, realistic, and context-aware manner, while guaranteeing safety, comfort, and lateral stability. The core principle is anticipatory: instead of reacting only to the instantaneous velocity error, the controller looks ahead at the geometry of the upcoming trajectory — in particular its curvature — and adjusts the target speed in advance, so that the vehicle approaches each curve while respecting the admissible lateral acceleration limits.



Figure 3.6: Longitudinal view from the vehicle

The fundamental relationship underlying this strategy is the link between the trajectory curvature κ and the lateral acceleration of the vehicle:

$$a_y = v^2 \kappa \quad (3.44)$$

Imposing that the lateral acceleration does not exceed a comfort and safety limit $a_{y,\max}$ is equivalent to constraining the maximum allowed speed in a curve:

$$v_{\text{curve}} = \sqrt{\frac{a_{y,\max}}{\kappa}} \quad (3.45)$$

This expression defines the maximum speed at which the vehicle can follow a curve of curvature κ without exceeding the prescribed lateral acceleration limit. It is therefore a direct bridge between road geometry and vehicle dynamics, and forms the basis of the preview-based longitudinal controller.

To anticipate the difficulty of the upcoming road segment, the controller estimates local curvature using three consecutive points of the planned trajectory ($\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$). The curvature is computed from the circumcircle of the triangle defined by these points:

$$\kappa = \frac{2A_{\Delta}}{\|\mathbf{p}_2 - \mathbf{p}_1\| \|\mathbf{p}_3 - \mathbf{p}_2\| \|\mathbf{p}_3 - \mathbf{p}_1\|} \quad (3.46)$$

where A_{Δ} is the signed area of the triangle, obtained from the cross product of two of its edges.

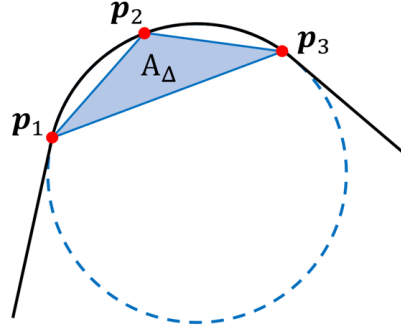


Figure 3.7: Circumcircle of the triangle defined by three points

If the three points are nearly collinear, the area tends to zero and the curvature is set to zero. This approach provides a robust estimate of the instantaneous curvature along the sampled trajectory without requiring numerical differentiation, which would be more noise-sensitive.

Starting from the current vehicle position, identified by the curvilinear coordinate s_0 , a fixed preview window of 20 m (as used in eWolf) is defined. Within this window, the controller computes:

- the maximum curvature $\kappa_{\max}$, obtained by scanning the path with a fixed step and evaluating the curvature on each triplet of points,
- the distance to the first relevant curvature increase, $d_{\text{to_curve}}$, defined as the distance from the current position to the first point where the curvature exceeds a small threshold κ_{thr} on the order of 10^{-3} m^{-1} .

These two quantities — peak curvature and distance to the start of the upcoming curve — allow the controller to determine not only how much to slow down, but also when to start braking.

Because the raw maximum curvature may be influenced by local noise or small irregularities of the reference path, an attenuation factor κ_{soft} is applied to reduce over-aggressive deceleration and produce smoother driving. The effective curvature used for speed limiting is therefore:

$$\kappa_{\text{eff}} = \kappa_{\text{soft}} \kappa_{\text{max}} \quad (3.47)$$

The corresponding curvature-based speed limit is then:

$$v_{\text{curve}} = \sqrt{\frac{a_{y,\text{max}}}{\kappa_{\text{eff}}}} \quad (3.48)$$

This speed is compared to the global reference speed provided by the higher-level planner, and the target speed v_{tgt} is selected as:

$$v_{\text{tgt}} = \min(v_{\text{global}}, v_{\text{curve}}) \quad (3.49)$$

On straight segments, where curvature is close to zero, this results in $v_{\text{tgt}} = v_{\text{global}}$. Approaching tight curves, the target speed is reduced proportionally to the geometric difficulty of the segment, ensuring that the vehicle never enters a curve at a speed incompatible with lateral grip or comfort.

Once the target velocity is determined, the controller must generate a compatible longitudinal acceleration (or deceleration) command. Two cases are considered:

Case 1 — Anticipatory braking for an upcoming curve. If a curve has been detected within the preview window and the distance to its onset $d_{\text{to_curve}}$ is sufficient, the controller computes a constant deceleration that will bring the vehicle from its current speed v to the target speed v_{tgt} exactly at the curve entry. From basic kinematics,

$$v^2 - v_{\text{tgt}}^2 = 2 a_{\text{req}} d_{\text{to_curve}} \quad (3.50)$$

which gives

$$a_{\text{req}} = \frac{v^2 - v_{\text{tgt}}^2}{2 d_{\text{to_curve}}} \quad (3.51)$$

Since this corresponds to braking, the applied command is

$$a = -\max(0, a_{\text{req}}) \quad (3.52)$$

ensuring a smooth, planned deceleration profile.

Case 2 — No relevant curve ahead (or already below the limit). If no significant curvature increase is detected in the preview window, or if the vehicle is

already travelling below v_{tgt} , the commanded acceleration is computed through a proportional tracking law:

$$a = k_v (v_{\text{tgt}} - v) \quad (3.53)$$

where k_v is tuned to provide a response that is prompt but not oscillatory. A deadband around zero is applied to avoid chattering of the throttle/brake command near steady-state. In all cases, the resulting acceleration command is then saturated to respect both comfort limits and actuator capabilities, ensuring physical realism.

This procedure is executed at every control cycle, with a typical time step of 0.05 s, updating the preview window so that it always lies ahead of the vehicle. As a result, the target speed is continuously recomputed based on the curvature the vehicle is expected to encounter over the next 20 m, and the longitudinal acceleration command is updated accordingly to maintain smoothness and compliance with dynamic limits.

The resulting behavior is natural and predictable. On straight segments, the vehicle maintains the desired cruising speed, compensating gradually for disturbances or grade changes. As a curve approaches, the preview logic detects the rise in curvature, the target speed is progressively reduced, and the controller initiates a gentle, constant deceleration that ensures the vehicle enters the curve within the allowed lateral acceleration envelope. When exiting the curve and curvature decreases again, the proportional law smoothly accelerates the vehicle back toward the global reference speed.

This approach combines the computational simplicity of a proportional regulator with the anticipative capability typically associated with more advanced strategies. Moreover, the explicit dependence on the lateral acceleration limit creates a natural coupling with the lateral controllers: the longitudinal controller enforces a speed that guarantees that steering controllers such as Pure Pursuit and Stanley operate within feasible grip and comfort bounds. In other words, although the longitudinal and lateral control loops are implemented separately, they act in synergy to keep the vehicle stable, comfortable, and well centered along the entire trajectory.

3.4 Model Predictive Controller

The Model Predictive Control (MPC) approach represents one of the most advanced and comprehensive solutions to the path-following problem, as it naturally integrates lateral and longitudinal behavior within a single optimization framework [30, 31]. Unlike geometric controllers such as Pure Pursuit or Stanley — which compute the steering command instantaneously from local kinematic relations — MPC adopts a predictive strategy: at every control step it plans a sequence of future steering and acceleration inputs that minimize the deviation of the vehicle from the desired trajectory while satisfying physical and kinematic constraints. This

predictive formulation allows the controller to anticipate curvature and speed changes, producing smoother, more coordinated, and more stable behavior.

The fundamental principle of MPC is that of a *receding horizon*. At each instant, the controller solves a finite-horizon optimal control problem that predicts the evolution of the vehicle state over a planning horizon of length N , searching for the sequence of control inputs that minimizes a cost function [31, 32]. Only the first element of the optimal input sequence is actually applied to the vehicle; the full problem is then solved again at the next control cycle using updated state measurements. In this way, the controller continuously adapts its actions to changes in the path geometry and in the operating conditions.

The MPC implemented on eWolf uses a prediction horizon of $N = 10$ steps with a sampling time $\Delta t = 0.1$ s. The control inputs are steering and longitudinal acceleration,

$$\delta = \{\delta_0, \dots, \delta_{N-1}\}, \quad a = \{a_0, \dots, a_{N-1}\},$$

and the predicted state is

$$\mathbf{x}_k = [x_k \ y_k \ \psi_k \ v_k]^\top.$$

The state propagation is described by the discrete kinematic bicycle model, which captures the geometric motion of the vehicle for moderate steering angles:

$$x_{k+1} = x_k + v_k \cos(\psi_k) \Delta t \quad (3.54)$$

$$y_{k+1} = y_k + v_k \sin(\psi_k) \Delta t \quad (3.55)$$

$$\psi_{k+1} = \psi_k + \frac{v_k}{L} \tan(\delta_k) \Delta t \quad (3.56)$$

$$v_{k+1} = \max(0, v_k + a_k \Delta t) \quad (3.57)$$

To perform prediction, the controller must compare the predicted vehicle state against a meaningful reference. On the lateral side, the MPC constructs a time-indexed reference path by mapping future time samples to future path positions. Starting from the curvilinear coordinate s_n corresponding to the nearest forward point on the reference path, a temporal grid s_k is generated by integrating a previewed velocity profile:

$$s_{k+1} = \min(s_k + \hat{v}_k \Delta t, s_{\max}) \quad (3.58)$$

where $\hat{v}_k$ is the preliminary speed profile predicted by the longitudinal preview module over approximately the next 20 m of path. This guarantees temporal-spatial consistency: the reference points $(x_r(s_k), y_r(s_k))$ are exactly those the vehicle is expected to encounter over the next $N\Delta t$ seconds.

At each step of the horizon, the MPC computes the two fundamental kinematic errors: the cross-track error, i.e. the orthogonal distance of the vehicle from the

reference path, and the heading error, i.e. the difference in orientation between the vehicle and the path tangent. Let $\mathbf{n}_r(s_k) = [-\sin \psi_r(s_k), \cos \psi_r(s_k)]$ denote the unit normal to the path and $\psi_r(s_k)$ its tangent heading. Then:

$$e_{y,k} = -\sin(\psi_r(s_k)) (x_k - x_r(s_k)) + \cos(\psi_r(s_k)) (y_k - y_r(s_k)) \quad (3.59)$$

$$e_{\psi,k} = \text{atan2}\left(\sin(\psi_r(s_k) - \psi_k), \cos(\psi_r(s_k) - \psi_k)\right) \quad (3.60)$$

On the longitudinal side, over the same s_k grid, the controller constructs two sequences: a reference speed $\{v_k^{\text{ref}}\}$ and a reference acceleration $\{a_k^{\text{ref}}\}$ derived from curvature preview along the trajectory. Curvature is estimated using three consecutive path points $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$, spaced by Δs , via the circumcircle formula:

$$\kappa = \frac{2A_{\Delta}}{\|\mathbf{p}_2 - \mathbf{p}_1\| \|\mathbf{p}_3 - \mathbf{p}_2\| \|\mathbf{p}_3 - \mathbf{p}_1\|} \quad \text{with} \quad A_{\Delta} = \frac{1}{2} |(\mathbf{p}_2 - \mathbf{p}_1) \times (\mathbf{p}_3 - \mathbf{p}_1)| \quad (3.61)$$

The admissible speed in a curve is then obtained by enforcing a lateral acceleration limit $a_{y,\text{max}}$ with $a_y = v^2 \kappa$:

$$v_{\text{curve}} = \sqrt{\frac{a_{y,\text{max}}}{\kappa_{\text{eff}}}} \quad (3.62)$$

where $\kappa_{\text{eff}} = \kappa_{\text{soft}} \kappa_{\text{max}}$ is a softened curvature that filters local spikes. The final speed target is defined as

$$v_{\text{tgt}} = \min(v_{\text{ref}}, v_{\text{curve}}) \quad (3.63)$$

If a curve is detected within the preview window, a constant-in-space braking profile is planned so that the vehicle reaches v_{curve} at the curve entry:

$$a_{\text{need}} = \frac{v_0^2 - v_{\text{curve}}^2}{2 d_{\text{plan}}} \quad (3.64)$$

$$a_k^{\text{ref}} = -\min(a_{\text{max}}, \max(0, a_{\text{need}})) \quad (3.65)$$

The steering input is further enriched with a curvature-based feed-forward term,

$$\delta_k^{\text{ff}} = \arctan(L \kappa) \quad (3.66)$$

which anticipates the steering effort required by the local curvature of the path. This term is smoothed through a moving average to avoid zig-zag behavior when κ is near zero, especially along straight segments.

All these elements enter the cost function that the MPC minimizes at each control cycle. This cost is the mathematical core of the driving objective: follow

the path accurately, while maintaining stable, comfortable, and physically feasible motion [30]. Qualitatively, the cost can be expressed as:

$$\begin{aligned}
 J = \sum_k \bigg(& w_y e_{y,k}^2 + w_\psi e_{\psi,k}^2 + w_v (v_k - v_k^{\text{ref}})^2 \\
 & + w_a a_k^2 + w_{\delta,\text{ff}} (\delta_k - \delta_k^{\text{ff}})^2 \\
 & + w_{\dot{\delta}} (\Delta\delta_k)^2 + w_{\dot{a}} (\Delta a_k)^2 \bigg) \tag{3.67}
 \end{aligned}$$

Each term has a direct physical interpretation:

- $w_y e_{y,k}^2$ and $w_\psi e_{\psi,k}^2$ penalize lateral deviation and heading misalignment. High w_y and w_ψ prioritize geometric accuracy, keeping the vehicle centered and correctly aligned with the path.
- $w_v (v_k - v_k^{\text{ref}})^2$ penalizes deviations from the curvature-aware longitudinal speed reference. This enforces comfort and grip limits by adapting speed to local curvature.
- $w_a a_k^2$ penalizes large accelerations/decelerations, limiting harsh throttle/brake actions and improving ride comfort.
- $w_{\delta,\text{ff}} (\delta_k - \delta_k^{\text{ff}})^2$ penalizes deviation from the curvature feed-forward steering. This encourages the controller to “follow the road naturally,” letting feedback act mainly for correction instead of doing all the work.
- $w_{\dot{\delta}} (\Delta\delta_k)^2$ and $w_{\dot{a}} (\Delta a_k)^2$ penalize fast changes in steering and acceleration. These terms directly enforce comfort and stability by suppressing impulsive commands and jerk-like behavior.

Overall, the cost function defines a dynamic compromise between accuracy and comfort. The lateral weights w_y and w_ψ are typically kept high to prioritize path tracking, while the rate penalties $w_{\dot{\delta}}$ and $w_{\dot{a}}$ are also significant to maintain stability and passenger comfort. The longitudinal terms are usually moderate but consistent, shaping energy usage and ensuring smooth transitions between braking and acceleration.

The resulting optimal control problem is nonlinear, due to the trigonometric kinematics and the dependence of the cost on predicted future states. However, its structure is regular and the prediction horizon is short, which makes it solvable efficiently using L-BFGS-B, a numerical optimization algorithm that directly handles bound constraints on inputs. A warm-start strategy is adopted by initializing the solver with the previous cycle’s optimal solution, ensuring fast convergence and temporal continuity, and preventing oscillations between local minima.

Once the optimal solution is obtained, the first-step commands (δ_0^*, a_0^*) are applied as controller output. Rate and jerk limiting, together with actuator saturation and filtering, are then enforced at the actuator control layer described in Chapter 4.

The resulting behavior is that of a controller capable of coordinating the longitudinal and lateral domains in a natural way. On straight segments, the cost encourages a steady steering command, eliminating residual weave and producing a clean straight-line trajectory. When approaching a curve, the longitudinal component reduces the target speed while the lateral component gradually increases steering demand, so that the vehicle enters the curve already aligned to its geometry. On exit, steering smoothly returns toward neutral and acceleration ramps back up, yielding a progressive and stable recovery to cruising speed.

In conclusion, the MPC cost encodes the desired balance between accuracy, comfort, and anticipation. Each term shapes a specific aspect of behavior: geometric precision, smooth control effort, energy regularity, and directional stability. Taken together, these effects give the controller the ability to act as an intelligent coordinator of vehicle motion, distributing effort in advance between lateral and longitudinal actions to produce driving behavior that is predictive, stable, and physically plausible. Rather than behaving as a simple regulator, MPC operates as a forward-looking decision-maker for the vehicle's motion [30, 32].

Chapter 4

Low-Level Control Module

The Low-Level Control represents the final module of the autonomous driving chain: this is where the commands computed by the path-following controllers — namely the ideal steering angles and desired accelerations — are converted into signals that can be actuated by the real vehicle. If the upper-level controllers represent the mind of the system, the low-level module is its body: it translates numbers into movements, cost functions into electric torques, and geometric trajectories into tangible rotations and forces applied to the wheels.

Unlike the preceding levels, dominated by kinematic or predictive models, the low-level stage operates in a domain where limits, delays, friction, and comfort play a decisive role. Every command must respect constraints on magnitude, rate of change, and temporal continuity, as well as lateral and longitudinal acceleration limits imposed by vehicle dynamics and human perception. Through limiters, filters, actuator models, and compensation logics, this module ensures that every decision made by the driving system is physically feasible, mechanically sustainable, and perceptually acceptable. It is here that autonomous driving ceases to be a problem of pure computation and becomes a real dynamic experience — governed by laws of stability, comfort, and smoothness.

4.1 Function of the Low-Level Control

The low-level control module acts as the direct interface between the controller logic and the vehicle actuators. While the upper layers define *where* the vehicle should go, the low-level layer determines *how* to move in order to achieve it, in a manner that is physically consistent and dynamically sustainable. Its primary function is to ensure that the steering angles and accelerations computed by the path-following algorithms are converted into real, continuous, and safe actuator commands, while respecting the mechanical, electrical, and perceptual limits of the

system.

At this stage, theory meets physics: each signal must be modulated, limited, and filtered to avoid abrupt variations that could compromise stability or ride comfort. Through saturation functions, rate limiters, and temporal filters, the module guarantees that the vehicle’s dynamic response remains smooth, controllable, and coherent with physical constraints, without sacrificing control accuracy.

The low-level control therefore does not merely execute commands, but interprets them — harmonizing the controller’s intent with the vehicle’s actual response. The following sections present the theoretical principles that define actuator limits and dynamic constraints, followed by the methods adopted to integrate these limitations into the controllers developed within this work.

4.1.1 Actuator Limits and Dynamic Constraints

The problem of low-level control is not merely to determine the amplitude of a command, but to define *how* that command should evolve over time so that it remains compatible with the vehicle’s physics and with the passenger’s perception of comfort. Before a signal reaches the electric steering system, the motor, or the braking actuator, it is filtered and limited according to a set of rules that constrain its amplitude, rate of variation, and temporal continuity. These constraints — *saturation*, *deadband*, *rate limiting*, *jerk limiting*, and *comfort and safety limits* — form the grammar through which a theoretical signal is translated into a physically feasible maneuver.

Saturation The first and most immediate constraint is the *saturation* of the command amplitude. Every actuator, whether it is the steering system, the traction motor, or the braking unit, has a maximum range or torque limit beyond which it cannot physically operate. For steering, this limit typically corresponds to a maximum wheel rotation of about $\pm 30^\circ$, beyond which the mechanical geometry of the system cannot deform without structural risk. Similarly, in the longitudinal domain, limits are imposed on acceleration and deceleration, beyond which tire adhesion is lost or thermal stress becomes critical. Mathematically, saturation is expressed as $|u| \leq u_{\max}$, yet its meaning is more profound: it represents the frontier between what is controllable and what is forbidden by the laws of physics.

When applied improperly, saturation can cause *integrator windup* in controllers with integral action. If the command is clipped while the integrator continues to accumulate error, the system may respond with delay or sudden overshoot once the limit is released. To prevent this, every saturation block should be complemented by an *anti-windup* mechanism, which projects or unloads the integral term within the admissible region, thus preserving stability and local linearity.

Deadband A second essential feature for stability is the *deadband*, an insensitivity region around zero where small-amplitude commands are deliberately ignored. In steering control, a deadband on the order of a few milliradians (approximately 0.005 rad, or 0.3°) helps eliminate micro-oscillations caused by numerical noise, backlash, or structural vibrations. Physically, it corresponds to the static friction threshold and the mechanical play intrinsic to the actuator; perceptually, it filters out micro-movements that a human driver would neither perceive nor attempt to correct. This “dead zone” helps stabilize vehicle dynamics along straight segments and reduces mechanical fatigue, imitating the natural behavior of an experienced driver who reacts only when the deviation exceeds a perceptible threshold.

Rate limiting Beyond amplitude, physics also constrains the speed at which a command can change over time. No real actuator can modify its position or torque instantaneously: electric motors have finite time constants, mechanical systems possess inertia and friction, and the steering system has a limited bandwidth. For steering, the maximum angular rate typically ranges between 0.5 and 1 rad/s (equivalent to 60–120°/s), corresponding to a few hertz of dynamic bandwidth. Applying a *rate limit* therefore forces the control signal to evolve at a pace consistent with actuator dynamics and chassis response. From a control perspective, this constraint mitigates discontinuities and prevents oscillatory behavior; from a perceptual standpoint, it avoids excessively sharp maneuvers, ensuring a smoother and more predictable response.

Jerk limiting Even when the rate of change of the command respects the *rate limit*, it may still exhibit abrupt changes in slope. In such cases, *jerk limiting* is applied to control how quickly the command rate itself varies over time. The *jerk* is the second derivative of the command with respect to time, and limiting it guarantees continuity not only in amplitude but also in velocity variation. This constraint is directly related to comfort: the human body is much more sensitive to sudden changes in acceleration than to constant acceleration itself. Experimental studies indicate that lateral jerks above 0.4–0.5 m/s³ are perceived as uncomfortable, while longitudinal jerks exceeding 2–3 m/s³ generate a noticeable jolt sensation. Limiting jerk is therefore essential to achieve a natural and comfortable autonomous driving experience.

Comfort and safety limits In addition to mechanical actuator constraints, there are limits dictated by human comfort and vehicle safety. Laterally, comfortable accelerations typically remain below 1 m/s², with a tolerable threshold up to 1.5 m/s²; beyond 2.5–3 m/s², tire adhesion and stability are compromised. Longitudinally, international standards — such as ISO 15622 for *Adaptive Cruise Control* systems — recommend maintaining accelerations within ± 2 m/s² and jerks

below 3 m/s^3 . These values are not arbitrary: they stem from physiological studies showing that the human body tolerates a constant force far better than an abrupt change in force.

In the *eWolf* vehicle, these principles are implemented by limiting the maximum deceleration to 3 m/s^2 and filtering longitudinal jerk through a dual low-pass filter with time constants between 0.1 and 0.2 s, ensuring smooth and progressive transitions between acceleration and braking phases. The comfort and safety limits, both lateral and longitudinal, follow the same philosophy: rigid constraints to respect physics, and dynamic filters to respect perception.

The coherent integration of all these constraints — saturation to prevent impossible demands, deadband to suppress excessive noise, rate and jerk limits to guarantee continuity, and comfort limits to ensure harmonic motion — produces an overall behavior comparable to that of a skilled human driver. The vehicle does not become slower, but smarter: it modulates force, rhythm, and progressiveness according to context, maintaining a refined balance between precision and smoothness. It is in this fusion of physics, dynamics, and perception that the *Low-Level Control* fulfills its most profound function: transforming a numerical command into a natural, stable, and believable action.

4.1.2 Application of Limits in the Controllers

The constraints and filters described in the previous section are directly applied within the path-following controllers implemented in this thesis, although their integration varies according to the controller structure and control philosophy. In geometric controllers such as *Pure Pursuit* and *Stanley*, limits are applied as rigid constraints downstream of the command computation, acting as a physical and perceptual protection mechanism. In the *Model Predictive Controller* (MPC), by contrast, these limits are embedded within the mathematical formulation of the optimization problem itself, becoming part of the language through which the controller reasons and plans its actions. In other words, in geometric controllers, limits are enforced *after* the command has been computed, acting as corrective filters to ensure physical feasibility; in the MPC, on the other hand, these same limitations are taken into account during command generation, since they are directly incorporated into the cost function and the optimization constraints, becoming structural elements of the control problem.

In the *Pure Pursuit* controller, the ideal steering angle is computed solely from the geometry of the reference trajectory and the chosen lookahead distance. Since this value does not account for actuator dynamics or comfort constraints, the low-level module subsequently processes it through a chain of filters and limiters — saturation, deadband, rate and jerk limits, followed by an exponential smoothing filter. Each stage of this chain plays a specific role: saturation prevents physically

infeasible commands, the deadband suppresses micro-oscillations and numerical noise, rate and jerk limits ensure temporal continuity, and the final filter harmonizes the signal, making it smooth and coherent with the system's update frequency. In this configuration, the controller itself is unaware of the constraints: it applies them passively, yet the low-level stage produces a response that remains realistic, smooth, and dynamically stable.

The *Stanley* controller follows the same principle, but its higher sensitivity to the yaw error makes it more prone to impulsive or oscillatory steering actions, particularly in the presence of abrupt curvature transitions. Here again, the chain of limiters and filters acts as a damping mechanism that guarantees progressive and coherent responses, maintaining comfort and stability even in high-curvature maneuvers. The underlying philosophy remains unchanged: the controller proposes, but the low-level module decides how much of that proposal can be physically realized by the vehicle.

In the *Model Predictive Controller*, however, constraints are no longer an external corrective mechanism, but an intrinsic component of the decision-making process. The MPC directly incorporates command amplitude, rate, and jerk limits within its predictive model, formulating them as *hard constraints* that delimit the optimization domain, such that:

$$|\delta_k| \leq \delta_{\max}, \quad |\dot{\delta}_k| \leq \dot{\delta}_{\max}, \quad |\ddot{\delta}_k| \leq \ddot{\delta}_{\max}.$$

These constraints are inviolable and ensure that candidate solutions always comply with the physical capabilities of the system, filtering out unrealizable control trajectories *a priori*.

Alongside the rigid bounds, the MPC also employs *soft constraints*, introduced as penalty terms within the cost function (analyzed in detail in the previous chapter). The penalties associated with the soft constraints allow for a controlled relaxation of less critical limits — such as maximum lateral acceleration or command variation — when this is necessary to preserve problem feasibility or overall stability. In practice, the hard constraints represent the absolute physical boundaries of the system, while the soft constraints introduce an optimized elasticity margin, enabling the controller to achieve a dynamic compromise among accuracy, comfort, and safety.

In this formulation, the MPC does not merely react to the constraints; it reasons *with* them, embedding the vehicle dynamics and actuator capabilities directly into its optimization process. As a result, the computed command is inherently consistent with both physical laws and comfort limits, requiring only minimal post-processing — typically a deadband and a light low-pass filter — to guarantee numerical smoothness and temporal consistency with the actual control frequency.

4.2 Actuator Interface and Signal Conversion

Once the steering and acceleration commands have been limited, filtered, and harmonized by the low-level control module, they must be converted into physical quantities that the actuators can interpret: the effective steering angle at the front wheels, the drive torque at the driven axle, or the braking force. This is the stage closest to the physical domain, where mathematics becomes engineering: numerical signals produced by the controller are transformed into mechanical actions through electric motors, reduction gears, sensors, power electronics, and feedback loops. The low-level control acts as an interpreter between software and physics, ensuring that every digital command translates into coherent, smooth, and safe vehicle behavior.

The steering command generated by the controller, denoted as δ_{cmd} , represents an ideal wheel angle computed from the vehicle's kinematic model and the desired trajectory. The real steering system, however, is an electromechanical assembly composed of an electric assist motor, a rack-and-pinion mechanism, a mechanical reduction stage, position and torque sensors, and an internal control unit (the Electric Power Steering ECU, EPS) [33, 34]. The dynamics of this system are not instantaneous: the steering actuator can be approximated as a first-order system that tracks the reference δ_{cmd} with a time constant τ_δ on the order of a few tenths of a second. To first order, its dynamics can be expressed as:

$$\frac{d\delta_{\text{real}}}{dt} = \frac{1}{\tau_\delta} (\delta_{\text{cmd}} - \delta_{\text{real}})$$

where δ_{real} is the actual steering angle achieved at the wheels. In other words, the physical steering system attempts to follow the desired command but with finite bandwidth rather than instantaneously. As the requested motion approaches rate or jerk limits, the response is further smoothed by the low-level control, which prevents abrupt transitions and preserves a continuous, natural steering behavior.

The steering ratio defines the relationship between steering wheel rotation and wheel angle. A ratio of 15:1, for instance, means that a rotation of 15° at the steering wheel results in a 1° rotation at the road wheels. In steer-by-wire systems, where the steering wheel is mechanically decoupled from the road wheels, the low-level module no longer transmits a target angle but directly issues a torque or current request to the EPS motor, which closes a local position-control loop. In that case, feedback from position and torque sensors enables a high-frequency inner loop that regulates the actuator so that δ_{real} converges to δ_{cmd} with minimal steady-state error and controlled dynamic response [35].

This inner EPS loop typically operates at high frequency (200–500 Hz) and ensures accurate steering tracking. At a higher level, the low-level control runs at lower frequencies (50–100 Hz), compares the desired command with the measured

response, and enforces the kinematic, dynamic, and comfort constraints described previously. The result is a clear hierarchy: the path-following controller computes the ideal steering angle, the low-level module enforces smoothness and physical limits, and the EPS loop physically realizes the command on the steering column and rack.

On the longitudinal side, the controller output can be expressed either as a desired longitudinal acceleration $a_{x,\text{cmd}}$ or directly as a target drive or brake torque T_{cmd} . The low-level module converts the requested acceleration into the wheel torque required to produce it, under quasi-static assumptions, according to:

$$T_{\text{cmd}} = \frac{m a_{x,\text{cmd}} r}{\eta i_g i_0}$$

where m is the vehicle mass, r is the effective rolling radius, i_g and i_0 are the gear and final drive ratios, and η is the overall drivetrain efficiency. This relation explicitly bridges the kinematics desired by the controller with the physical effort required by the powertrain: for the same requested acceleration, a heavier vehicle or a taller gear ratio demands a higher torque.

The computed signal is sent to the traction motor or braking system controller, which modulates current, voltage, or hydraulic/electrohydraulic pressure to generate the required longitudinal force at the tire-road interface. The low-level control applies slew-rate limits (maximum torque variation per unit time) and longitudinal jerk limits to prevent torsional shocks and driveline vibration. Excessive torque steps would be perceived as a sudden surge or harsh braking event, reducing comfort and component durability [36].

In more advanced implementations, the low-level module incorporates an estimation of available traction. The tire-road friction coefficient μ can be estimated from wheel slip, or from discrepancies between predicted and measured accelerations, allowing the controller to impose a dynamic limit on transmissible torque and thus on maximum acceleration without wheel spin. When loss of traction is detected, the requested torque T_{cmd} is proactively reduced, implementing a predictive anti-slip strategy [37]. In autonomous vehicles, this function is crucial because it represents the interface between numerical control logic and physical interaction with the road.

After generating the physical commands (target steering angle and drive/brake torque), they must reach the actuators with appropriate timing and integrity. In the *eWolf* system, communication between ROS 2 nodes and the real-time Speedgoat unit occurs via Ethernet using the UDP protocol, at frequencies around 50–100 Hz. The Speedgoat receives packets containing steering and longitudinal commands, validates them through checksum and timestamp verification, and converts them into analog or digital signals compatible with the actuators. Each packet includes key operating variables — requested steering angle, drive torque, or deceleration —

together with cycle identifiers and safety flags, ensuring temporal coherence and data integrity along the control chain.

Temporal synchronization is critical. Delays of just a few tens of milliseconds between computation and actuation can introduce phase lag, residual oscillations, or delayed response, especially for fast-reacting controllers. To mitigate these effects, the ROS–Speedgoat communication employs minimal buffering and clock synchronization, maintaining point-to-point latency below 10 ms — consistent with automotive prototypes of steer-by-wire and drive-by-wire systems [38, 39]. This temporal alignment is also important from a sampling standpoint: the high-level controller operates in discrete time (e.g., 50 Hz), the actuators behave continuously, and the EPS inner loop runs at much higher frequency. Preserving coherence across these time scales avoids dynamic aliasing between discrete commands and continuous actuator responses.

Once received and validated, the commands are applied to the physical actuators. In the lateral channel, δ_{cmd} becomes a position reference for the EPS motor, which generates the torque required to move the rack to the desired δ_{real} , closing a position/torque feedback loop. In the longitudinal channel, T_{cmd} is converted into motor current (for propulsion) or hydraulic pressure (for braking), producing the longitudinal tire force required to achieve the desired acceleration or deceleration.

This entire chain — from computation to signal and from signal to motion — forms the bridge between model, electronics, and mechanics. The controller defines the intent; the low-level control enforces physical feasibility and comfort; the interface converts it into current, pressure, and torque; and the actuators deliver the resulting forces to the ground. All these layers must be consistently calibrated: a rate limit that is too conservative slows response, while an overly permissive one may excite structural vibration; a gain mismatch between δ_{cmd} and δ_{real} may introduce systematic tracking errors.

The cycle does not end with actuation: every action produces measurable feedback. Position, speed, and torque sensors within the EPS and powertrain continuously report real values to the controller. These data, published on ROS topics (e.g., `/steering_feedback`, `/odom`, `/wheel_speed`), are used to update the vehicle state estimate and close the feedback loop. This enables compensation for friction, load variations, and external disturbances, ensuring consistent performance over time. Functionally, the low-level module thus acts as a true closed-loop dynamic system rather than a one-way signal translator.

Accurate calibration between the control and physical domains is also essential. The transfer curves $\delta_{\text{cmd}} \rightarrow \delta_{\text{real}}$ and $a_{x,\text{cmd}} \rightarrow T_{\text{cmd}}$ should remain linear, unbiased, and symmetric. Scaling or offset errors can cause steady-state lateral or longitudinal drift and tracking inaccuracies. During testing, the *eWolf* system measures these relationships in real time and updates calibration factors in the Speedgoat unit, maintaining mean deviations below $\pm 0.5^\circ$ on steering and $\pm 0.05 \text{ m/s}^2$ on

acceleration.

From a functional safety perspective, the Speedgoat implements watchdog timers and hardware safety limits. If valid packets are not received within a defined timeout interval, the commands are automatically driven to neutral (zero torque, centered steering), returning the vehicle to a safe state. This ensures that even in the event of communication loss or software malfunction, the system remains controllable.

The actuator interface is the point where autonomous driving becomes tangible: where a logical decision turns into a physical motion, and where passenger comfort depends on the precision of that translation. It is in this transition — from computation to action and back — that the low-level control fulfills its fundamental role: ensuring that physics speaks the language of logic.

Chapter 5

Robot Operating System (ROS 2)

The software foundation of this thesis project is built upon the *Robot Operating System 2* (ROS 2), the reference framework for the development and integration of complex robotic and vehicular systems. ROS 2 provides a modular and scalable infrastructure for communication among sensors, actuators, and distributed computational modules, offering a standardized environment in which independent components can cooperate and exchange information in real time.

In the context of autonomous driving, ROS 2 serves as the backbone of the entire system: all modules—from perception to control—operate as *nodes* that exchange information through a decentralized network of *topics*, *services*, and *actions*. This distributed architecture allows individual functions to be isolated, reused in different scenarios, and validated independently, while maintaining global coherence and synchronization across the entire control loop.

This chapter examines two complementary aspects of ROS 2. The first part introduces the theoretical and architectural principles that form its foundation, highlighting the multilayer structure and communication mechanisms that have driven its widespread adoption in autonomous systems. The second part describes the specific software implementation adopted in the *eWolf* project, which employs ROS 2 as an integration platform connecting the perception, localization, planning, and control modules.

Finally, the chapter presents the simulation architecture developed within this thesis, designed to virtually reproduce the eWolf vehicle control chain. This simulation environment enables systematic testing and comparison of the different *Path Following* and *Low-Level Control* strategies under realistic and repeatable conditions.

5.1 ROS 2 Overview

The *Robot Operating System* (ROS) has become the de facto software framework for robotics and autonomous systems. It is an open-source middleware designed to provide a common and modular infrastructure capable of orchestrating communication among sensors, actuators, and distributed computational modules [40]. Its widespread adoption stems from its ability to integrate heterogeneous components into a unified architecture, greatly simplifying the development and experimentation of complex systems such as autonomous vehicles.

The ROS project originated in 2007 at Stanford University as part of the *STAIR* (Stanford Artificial Intelligence Robot) program and was later developed and distributed by the Willow Garage laboratory, a key player in American robotics research. Its original goal was to create a shared and standardized software platform that could separate perception and control logic from hardware-specific details, allowing research groups to collaborate and compare results using a common language.

The success of ROS arises from several fundamental design principles that define its philosophy and effectiveness:

- **Distributed peer-to-peer architecture:** ROS is structured as a decentralized system, where each software component (called a *node*) runs as an independent process that communicates with others over a logical network, without a central server. This ensures scalability and fault tolerance, since each node can be started, stopped, or relocated without compromising the entire system. In vehicular contexts, this allows sensor management, path planning, and control to be handled by separate, self-contained modules.
- **Modularity and reusability:** Functionalities are organized into autonomous modules called *packages*, containing source code, configurations, message definitions, and launch files. Each package can be replaced or reused across projects, enabling incremental system development and fostering collaboration and interoperability among researchers and industrial developers.
- **Multi-language and interoperability:** ROS natively supports multiple programming languages—primarily C++ and Python—through the *rclcpp* and *rclpy* client libraries. Communication between nodes is language-agnostic, allowing high-performance algorithms to be written in C++ and prototyping or testing components in Python within the same distributed system.
- **Comprehensive tool ecosystem:** ROS includes a wide suite of tools for debugging, visualization, data recording, and analysis. Tools such as *rviz* and *rqt* enable real-time graphical visualization of sensors and system states, while

rosbag allows logging and playback of message streams. Additional utilities make it possible to inspect topics, tune parameters, and monitor the node network. This complete development environment has turned ROS from a simple framework into a true software ecosystem for robotics.

- **Open-source philosophy and global community:** ROS is distributed under an open-source license and maintained by a worldwide community of universities, research institutes, and companies. Its open nature has led to the creation of thousands of ready-to-use packages, forming a shared repository of tools and knowledge for robotics development.

The first generation, known as *ROS 1*, served for more than a decade as the reference platform for academic research. However, its architecture—originally designed for laboratory environments—proved limiting for real-world and safety-critical applications, such as industrial robotics and autonomous driving [41]. The main limitations of ROS 1 included the presence of a central coordination node (*roscore*) for network management, communication based on a proprietary protocol (TCPROS), the absence of real-time guarantees, and poor compatibility with embedded systems.

To overcome these constraints, the second generation—*ROS 2*—was introduced in 2017 as a complete redesign of the framework, built on principles of reliability, scalability, and multi-platform compatibility [42]. The most significant innovation was the adoption of the *Data Distribution Service* (DDS) as its communication middleware. DDS is an established industrial standard widely used in aerospace and automotive domains [43], implementing the *publish/subscribe* paradigm in a fully distributed fashion. Thanks to DDS, ROS 2 nodes can automatically discover each other on the network, communicate without a central server, and configure their data exchange policies through customizable *Quality of Service* (QoS) parameters. This allows balancing efficiency and reliability depending on the criticality of the data flow: control signals can be transmitted with strict delivery and latency guarantees, while high-frequency sensor streams can prioritize speed at the expense of absolute reliability.

These features make ROS 2 suitable not only for research but also for industrial prototyping and real-world validation of autonomous systems. The version used in this thesis, *ROS 2 Foxy Fitzroy*, is a stable *Long-Term Support* (LTS) distribution, widely adopted in research and development environments, and particularly suited for complex experimental setups such as the *eWolf* platform.

5.1.1 ROS 2 Architecture and Fundamentals

The architecture of ROS 2 can be interpreted as a distributed graph of software components that cooperate to operate a complex robotic or vehicular system. The

central concept of this ecosystem is the *ROS graph*, which represents the set of active nodes and the communication links between them. Unlike monolithic systems, in which all functions of a robot or vehicle are integrated into a single executable, ROS 2 adopts an explicitly modular structure: each function is assigned to a distinct entity called a *node*, and the overall system emerges from the cooperation of multiple nodes exchanging information through standardized communication channels.

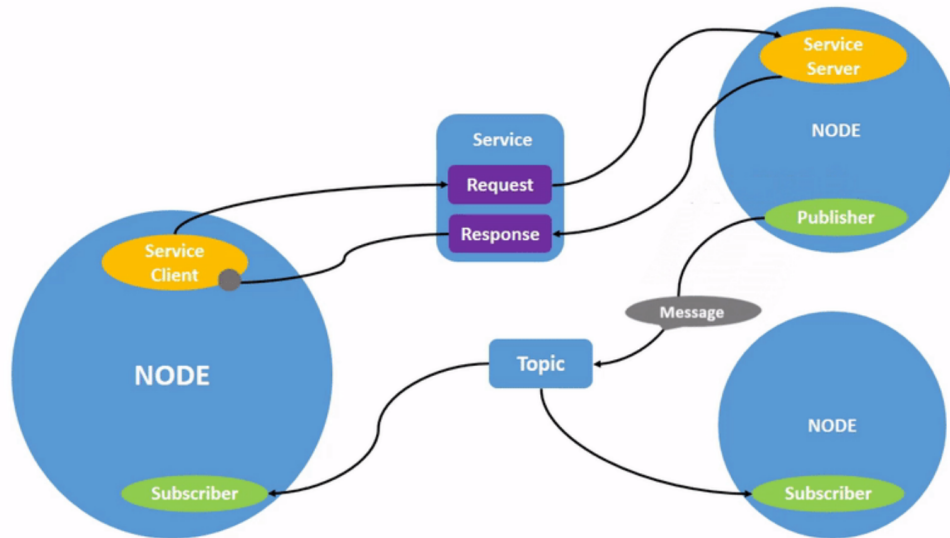


Figure 5.1: ROS 2 Architecture

Node A *node* is the fundamental execution unit in ROS 2. From a practical standpoint, a node is a process running on the operating system that performs a specific task within the overall architecture. A node may, for instance, read the vehicle odometry, compute the steering command, or generate the reference trajectory to follow. The fundamental idea is that a node should focus on a single, well-defined responsibility. This design choice has two direct benefits: first, it simplifies development and debugging, since each node has a clear purpose and can be tested independently; second, it allows parts of the system to be replaced or upgraded without rewriting the entire application, simply by substituting a node that publishes and subscribes to the same data types.

Topic For nodes to cooperate, ROS 2 provides several communication mechanisms, the most widely used being the *topic*-based exchange. A topic is a named data channel (e.g., `/odometry`, `/steering_angle`, `/reference_path`) associated with a specific message type. A node can publish messages to a topic, and one or more

nodes can subscribe to the same topic to receive them. This scheme is known as the *publish/subscribe* model. Architecturally, it introduces an important consequence: the publisher and subscriber have no direct dependency on one another. They only need to agree on the message type and topic name. For example, a kinematic simulator may publish the vehicle state (position, orientation, and velocity) on the topic `/odometry`, while the controller reads that same topic to compute steering commands, and another node may simply subscribe to it for data logging.

Communication through topics is asynchronous and continuous, making it ideal for data streams that evolve over time, such as sensor readings, state estimates, or control commands. Each topic carries strongly typed messages: message types are formally defined (e.g., `nav_msgs/Odometry` for odometry, `sensor_msgs/Imu` for IMU data, `geometry_msgs/Twist` for velocity commands) and can be either standard, from the ROS message library, or custom-defined. This strict typing ensures interoperability among nodes developed by different teams or in different programming languages.

Service Some operations, however, require explicit request/response interactions rather than continuous data flow. For such cases, ROS 2 provides *services*. A service represents a synchronous communication between two nodes: a client sends a request to a server, which processes it and returns a response. This mechanism is similar to a remote function call. Unlike topics, which deliver data continuously and unidirectionally, services are designed for discrete operations such as resetting odometry, querying the status of a sensor, loading a configuration, or retrieving a computed value on demand. The synchronous nature of services means that the client waits for the server's reply before proceeding, making them suitable for command and query operations rather than streaming data.

Action A third communication mechanism extends the service concept to long-running tasks: *actions*. Actions define an interaction between a client and a server in which the client sends a *goal*, the server works toward achieving it, and during execution may provide intermediate *feedback* until the final *result* is returned. The client can also cancel the ongoing action if necessary. This paradigm suits operations that take time and must be monitored externally, such as following a trajectory to a target pose, executing a complex maneuver, or performing a motion plan. Compared with topics, actions are task-oriented rather than data-oriented; compared with services, they introduce duration, feedback, and cancelation semantics.

Parameter Another fundamental element of the ROS 2 architecture is node configuration through *parameters*. Each node can be associated with a set of internal variables that define its operational behavior. Parameters may represent tuning constants (e.g., control gains, maximum steering limits, target velocity),

configuration options (e.g., topic names or operation modes), or physical system constraints. They can be loaded at startup from configuration files—typically in YAML format—or modified at runtime using ROS 2 command-line tools or dynamic reconfiguration utilities. This feature is particularly valuable in vehicle control: the same controller node can run with different setups (for example, more restrictive acceleration limits or more aggressive gains) without altering its source code.

All these communication primitives—nodes, topics, services, actions, and parameters—operate within a fully decentralized communication infrastructure. Unlike ROS 1, which relied on a central *master* node to register publishers and subscribers, ROS 2 leverages the *Data Distribution Service* (DDS) middleware to handle message routing and discovery [43]. DDS is a proven industrial publish/subscribe protocol widely used in aerospace and automotive domains. In ROS 2, it is responsible for *discovery*: the automatic detection of nodes that publish or subscribe to the same topics, and negotiation of communication parameters. When a node begins publishing on a topic, it does not register with a central server; instead, it announces its presence on the ROS 2 network, and all interested subscribers automatically establish communication. This discovery process is dynamic: nodes can start or stop at any time, and the network topology adapts accordingly.

A key feature of DDS—and therefore of ROS 2—is its fine-grained control over communication quality through *Quality of Service* (QoS) policies [42]. Each publisher–subscriber connection can specify requirements such as reliability level, maximum latency, queue depth, message persistence, or delivery history. This is particularly relevant in autonomous systems, where not all data streams have the same importance. Steering and acceleration commands must be reliable and up to date, while high-bandwidth LiDAR data can tolerate occasional loss to maintain frequency. ROS 2 therefore models communication not as a single uniform channel but as a network of links, each with its own temporal and reliability guarantees tailored to its role in the control hierarchy.

The final structural component of ROS 2 is its shared spatial and temporal reference system. In mobile robots and autonomous vehicles, it is essential to describe the position and orientation of all components—vehicle, sensors, and world—in a consistent way. ROS 2 provides this mechanism through the `tf2` transformation library [44]. This library maintains a graph of reference frames (e.g., `map`, `odom`, `base_link`, `imu_link`) and allows each node to publish transformations between them. Other nodes can query this graph to determine, at any given time, the position of the vehicle in the global map, the location of a sensor relative to the vehicle center, or to transform a point detected by a camera into world coordinates. In essence, `tf2` provides the geometric coherence necessary for consistent interpretation of the data exchanged through ROS topics.

Combining all these building blocks reveals how an architecture originally designed for general robotics perfectly suits autonomous driving platforms. Perception, localization, planning, and control can each be represented as independent nodes sharing structured data over configurable communication channels—maintaining clear separation of responsibilities while ensuring a continuous, coherent, and deterministic data flow throughout the system.

5.2 Software Implementation

The software architecture of *eWolf* was developed in ROS 2 Foxy with the objective of obtaining a complete and continuous data-processing chain, from environment perception to the generation of steering and acceleration commands. The entire system is organized as a set of ROS 2 packages that cooperate by publishing and subscribing to standardized topics. This organization follows the principle of functional separation typical of autonomous vehicles: each block has a clearly defined responsibility (sensing, interpretation, localization, planning, control), and the blocks communicate through formally structured messages. In this way, the control pipeline can be tested, replaced, or extended without having to reconstruct the entire system.

The data flow can be described in five main stages: sensor acquisition, pre-processing and normalization of raw data, vehicle localization, reference trajectory extraction, and motion control. The following sections walk through these stages and highlight the main packages that make up the *eWolf* architecture.

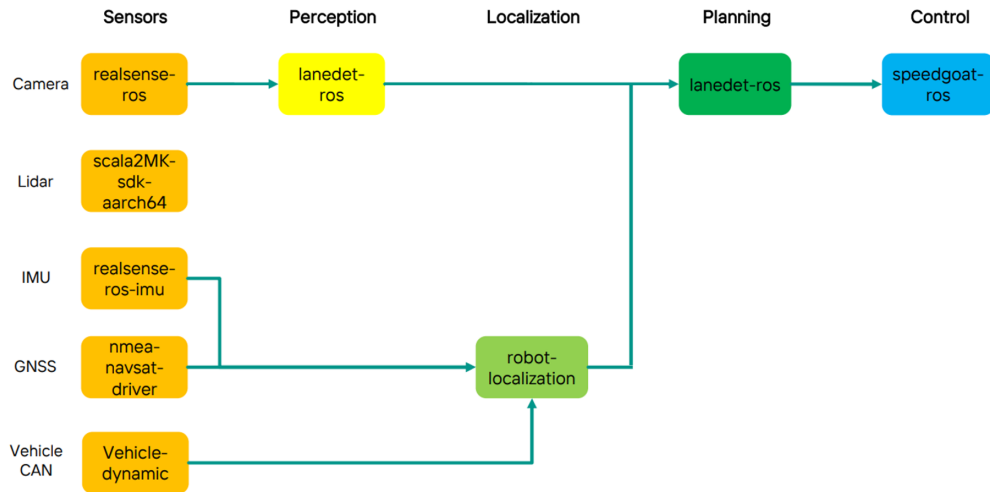


Figure 5.2: ROS 2 *eWolf* Network

Sensor Acquisition In the first stage, sensor acquisition is handled by the packages responsible for interfacing with the onboard hardware and publishing its data into the ROS 2 graph. The `realsense_ros` package provides access to the Intel RealSense camera mounted on the vehicle: it publishes RGB images, depth information, and inertial measurements from the IMU (via the `realsense_ros_imu` components) associated with the camera frame. In parallel, the `nmea_navsat_driver` package reads NMEA sentences from the GNSS receiver and converts them into standard ROS messages containing latitude, longitude, and altitude. These two data streams — visual and inertial measurements on one side, GNSS measurements on the other — represent the vehicle’s primary observation of itself and of the surrounding environment. At this stage, the information is still raw: the IMU provides linear accelerations and angular velocities expressed in the IMU sensor frame, the GNSS provides absolute geographic coordinates, and the camera provides only pixel intensities and depth.

Pre-processing and Frame Alignment For this data to be usable by the rest of the system, it must be transformed into a coherent and consistent representation. This is the role of the second stage, dedicated to pre-processing and frame alignment.

The `imu_transform_package` node transforms the IMU data into the desired reference frame, correcting axis conventions, orientations, and mounting offsets, so that inertial measurements become directly comparable to the other sources.

With respect to position, the `gps_odometry_package` node acquires GNSS data and converts latitude/longitude measurements into local metric coordinates. In practice, the node constructs a GPS-derived odometry estimate and publishes, on the topic `/odometry/gps`, a pose estimate of the vehicle expressed in a local reference frame. This node also publishes the relevant TF transformations between the GNSS frame(s) and the vehicle frame, allowing the rest of the system to interpret where the vehicle is located in space.

State Estimation and Localization After sensor normalization, the next step is to produce a unified estimate of the vehicle state, i.e., its position, orientation, and velocity. In *eWolf*, this function is handled by the `robot_localization` package, which fuses multiple data sources (IMU, GPS odometry, and optionally vehicle wheel odometry) using an Extended Kalman Filter (EKF). The output is a filtered, time-consistent estimate of the vehicle pose and velocity. This estimate is both published as a standard ROS odometry message and also injected into the TF tree, continuously updating the spatial relations between frames such as `map`, `odom`, and `base_link`. This step is fundamental for control, since it provides a single, smooth, and globally consistent description of the vehicle’s state that will later be used by both the local planner and the motion controller.

Lane Perception and Reference Path Generation Once the vehicle knows *where it is*, it must determine *where it should go*. This is achieved through road perception and reference path generation.

The `lanedet_ros` package performs lane detection from the forward-facing camera images. The node runs a lane detection neural network, identifies the left and right lane boundaries, and projects these curves into a bird’s-eye-view representation. The output is not just an annotated image: it is a parametric geometric description of the lane in front of the vehicle, including lateral offset with respect to the lane center, local curvature, and short-horizon corridor geometry.

Starting from this lane description, the `lane_boundary_transformer` package acts as a bridge between perception and local planning. This node takes the lane boundaries detected by the neural network, transforms them into the vehicle reference frame, and re-expresses them in the appropriate motion frame using TF2 transforms. From this geometric representation, the node computes the drivable centerline of the lane, i.e., a locally feasible path representing the lane center. This reference path is then filtered, interpolated, and finally published as a `nav_msgs/Path` message on a topic such as `/reference_path`. At this point the system has produced a target path that the vehicle should follow — precisely the information required by the path-following controller.

Control and Command Generation The final stage of the pipeline is motion control, i.e., the generation of actuator commands.

This functionality is handled by the `speedgoat_package`, which acts as the vehicle control node. The controller reads two categories of input: (i) the current vehicle state, as provided by the filtered odometry (pose, orientation, and velocity), and (ii) the reference trajectory published by the lane transformation stage. Based on these inputs, it computes the steering command and the longitudinal action (acceleration or deceleration request).

The resulting commands are published on dedicated topics (e.g., `/steering_angle` and `/acceleration`) and, in parallel, are transmitted via UDP to the Speedgoat real-time platform. The Speedgoat serves as a bridge toward the physical vehicle actuators or toward a hardware-in-the-loop (HIL) bench. As a result, the controller outputs are immediately suitable either for execution on the physical platform or for faithful closed-loop testing in a controlled environment.

The final result is a closed and coherent control chain. All stages described above communicate through standard ROS 2 interfaces: topics for continuous data streams, parameters for node configuration, and TF for spatial alignment. This preserves full modularity of the system. Each block can be tested in isolation, replaced with an alternative implementation, or executed on different hardware without breaking the rest of the pipeline, because the communication interfaces remain stable.

This dataflow architecture, which corresponds to the vehicle’s software diagram, is therefore not only a logical description of the system, but also the foundation that enables both real-vehicle execution and controlled simulation. It is precisely this continuity of interfaces that makes the *eWolf* architecture suitable both for algorithmic research and for vehicle-in-the-loop integration.

5.2.1 Simulation Architecture

To design and systematically validate the control algorithms, a simulation environment entirely based on ROS 2 was developed. It was conceived to reproduce the complete control chain of an autonomous vehicle under realistic yet fully controllable conditions. The Path Following problem was therefore isolated from the rest of the autonomous driving stack, allowing the focus to remain exclusively on the controller logic, on the algorithmic differences among control strategies, and on the impact of the vehicle’s physical limitations, while avoiding any distortion introduced by external modules.

The main objective of this architecture was to create a modular, scalable, and fully parameterizable structure, in which each node performs a specific function within the control loop while maintaining a clear data flow and logical separation between processing stages. The simulation revolves around four main nodes: *Reference Path*, *Simulator*, *Speedgoat*, and *Plotter*.

Reference Path Node This node is responsible for generating and publishing the reference trajectory that the vehicle must follow, in the form of a `nav_msgs/Path` message on the topic `/reference_path`. Upon startup, the node subscribes to the `/odometry` topic, reads the initial pose and orientation of the vehicle, and uses them as the world reference origin. From this initial pose, it generates controlled driving scenarios such as a U-turn and an S-curve with two different radii. These trajectories are built from straight segments connected with curved sections generated by concatenating constant-curvature arcs. The resulting path consists of an ordered sequence of `PoseStamped` messages, each with position and orientation coherent with the desired curvature profile. In essence, this node acts as a local trajectory planner: it provides the controller with a smooth, continuous, and kinematically feasible reference defined in the `odom` frame.

Simulator Node This node implements the vehicle model and therefore represents the kinematic simulator of the system. It is based on the simplified bicycle model [rajamani2011vehicle], integrated in real time. From a ROS perspective, the simulator receives as inputs the same commands that, in the real world, would be sent to the actuators: steering angle and longitudinal acceleration. These commands arrive through the topics `/steering_angle` (in degrees) and `/acceleration`

(in m/s^2). At each simulation step (with a timestep of 0.05 s), the node integrates the vehicle's kinematic equations, updating position (x, y) , orientation (yaw), and longitudinal velocity v according to the time step, the wheelbase, and the received steering command. The result of this integration is the current simulated vehicle state, published as a `nav_msgs/Odometry` message on the topic `/odom`. In summary, the *Simulator Node* replaces the physical vehicle, acting as the source of the vehicle's state feedback in the loop.

Speedgoat Node At the core of the simulation lies the *Speedgoat Node*, which implements both the Path Following and Low-Level Control modules. It computes the steering and acceleration commands based on the current vehicle state and the reference trajectory. The node subscribes to two key topics: `/odometry`, from which it obtains the simulated pose and velocity, and `/reference_path`, which provides the target path generated by the planner. Once the reference path is received, it is processed and regularized: point coordinates are filtered, uniformly resampled, and converted into a continuous and differentiable spline representation, ensuring a smooth and numerically stable trajectory. At each new odometry update, the node evaluates the vehicle's current position relative to this trajectory and computes the control actions required to minimize the tracking error.

As described throughout this thesis, the node integrates multiple tracking controllers—*Pure Pursuit*, *Stanley*, *PID*, and *MPC*—which can be dynamically selected through a ROS parameter. This unified implementation allows consistent management of both lateral and longitudinal dynamics, representing the most advanced and realistic configuration of the system. Regardless of the selected controller, the *Speedgoat Node* produces two primary outputs: the steering angle and the acceleration command, published respectively on the topics `/steering_angle` and `/acceleration`. In parallel, the node publishes a wide range of diagnostic variables—steering angle, cross-track error, lateral acceleration, velocity, acceleration—on dedicated topics, facilitating real-time monitoring and later analysis of each controller's behavior. Within the closed simulation loop, the *Speedgoat Node* acts as the central control unit: it receives state and reference data, computes control commands, and sends them to the kinematic simulator. It is also designed to transmit commands via UDP to the real Speedgoat hardware, ensuring seamless continuity toward future hardware-in-the-loop (HIL) validation.

Plotter Node Completing the simulation chain is the *Plotter Node*, which fulfills two essential roles. First, it provides real-time visualization of the simulation, displaying the comparison between the actual and reference trajectories, the evolution of steering commands, lateral acceleration, and tracking errors. This visualization enables immediate assessment of controller stability, detection of oscillations, and identification of undesired behaviors. Second, it performs data logging: at

the end of each run, all recorded signals are automatically saved into an Excel file containing, for each timestamp and path position, the control inputs, vehicle dynamic responses, tracking errors, and other relevant kinematic quantities. This enables detailed offline analysis, which was performed in MATLAB, allowing a direct comparison of controller performance under identical trajectory, initial, and vehicle parameter conditions.

In conclusion, this simulation infrastructure forms the methodological foundation of the results presented in the following chapter. It ensures that the performance of different controllers is compared under equivalent conditions, providing an objective assessment of the effectiveness of each control strategy and preparing the ground for future integration and validation on the physical *eWolf* vehicle.

Chapter 6

Experimental Results

This chapter presents the experimental results obtained from the implementation and validation of the vehicle control algorithms — namely the *Pure Pursuit*, *Stanley*, and *Model Predictive Control* models — within the ROS 2 architecture described in the previous chapters. After defining the test scenarios and reference paths, the results achieved by each controller are analyzed in detail, with particular focus on lateral and longitudinal performance indicators, compliance with comfort and safety limits, and the robustness of maneuvers under different conditions.

Subsequently, a comparative analysis among the different controllers is conducted, highlighting their respective strengths and weaknesses with respect to the selected trajectories. This comparison makes it possible to objectively evaluate the performance of each approach, emphasizing the different trade-offs between tracking accuracy, dynamic stability, and steering smoothness.

The objective of this chapter is therefore to provide a comprehensive and rigorous overview of the effectiveness of the developed and tested control system, demonstrating the extent to which it meets the initial goals of the thesis and identifying possible directions for future improvements and applications.

6.1 Test Scenarios

For the experimental validation of the controllers, it was necessary to design reference paths capable of reproducing different driving situations, in order to analyze the vehicle's behavior under heterogeneous yet controllable and repeatable conditions. To this end, two representative tracks were selected: a **U-shaped path**, useful for evaluating controller performance in the presence of tight curvature and direction reversals, and an **S-shaped path**, designed to assess the controller's ability to react to alternating variations of curvature and changes in the sign of the lateral error. These scenarios make it possible to virtually reproduce typical

real-driving maneuvers while maintaining a high degree of test repeatability.

Both paths were implemented in dedicated ROS 2 nodes that autonomously generate the reference trajectory and publish it on the `/reference_path` topic. The trajectories are procedurally generated within the node, starting from the vehicle's initial pose acquired through the `/odometry` topic. This configuration ensures that each simulation starts with perfect alignment between the vehicle's reference frame and that of the path, avoiding initial discontinuities or geometric misalignments.

The generation of path points occurs with a constant spatial step of 0.05 m, ensuring a uniform discretization and continuous curvature along the entire track. The path is published periodically at a frequency of 1 Hz, keeping the trajectory constantly updated during the simulation. Although the geometric shape of the path remains unchanged, this cyclic publication guarantees that the trajectory remains synchronized with the vehicle's current pose, thereby contributing to the overall stability of the testing environment.

For both paths, the graphical representation follows the same convention: the red line indicates the central trajectory or reference path, while the black lines represent the lane boundaries, placed at a distance of 1.5 m on each side, for a total lane width of 3 m. The left black line is dashed, recalling the separation line between adjacent lanes, whereas the right line is solid, representing the outer edge of the roadway, as in real road markings. This choice allows for a more realistic visualization of the lane geometry and makes it immediately possible to verify whether the vehicle remains within the lane boundaries. Figures 6.1 and 6.2 thus represent a balanced and realistic test set, suitable for a comparative evaluation of the developed *Path Following* controllers.

U-shaped Path

The U-shaped path was designed to reproduce a U-turn maneuver or the traversal of a tight roundabout, typical of urban contexts where the vehicle must handle very narrow curves and significant heading changes within limited space. This track therefore emulates low-speed driving conditions, in which the control system must maintain stability and accuracy while performing large steering angles and wide orientation variations.

From a geometric standpoint, the track consists of three sections:

- an entry straight section for reaching and stabilizing the desired speed;
- a left-hand curve of 135° with a radius of 8 m;
- and an exit straight section for realignment and acceleration.

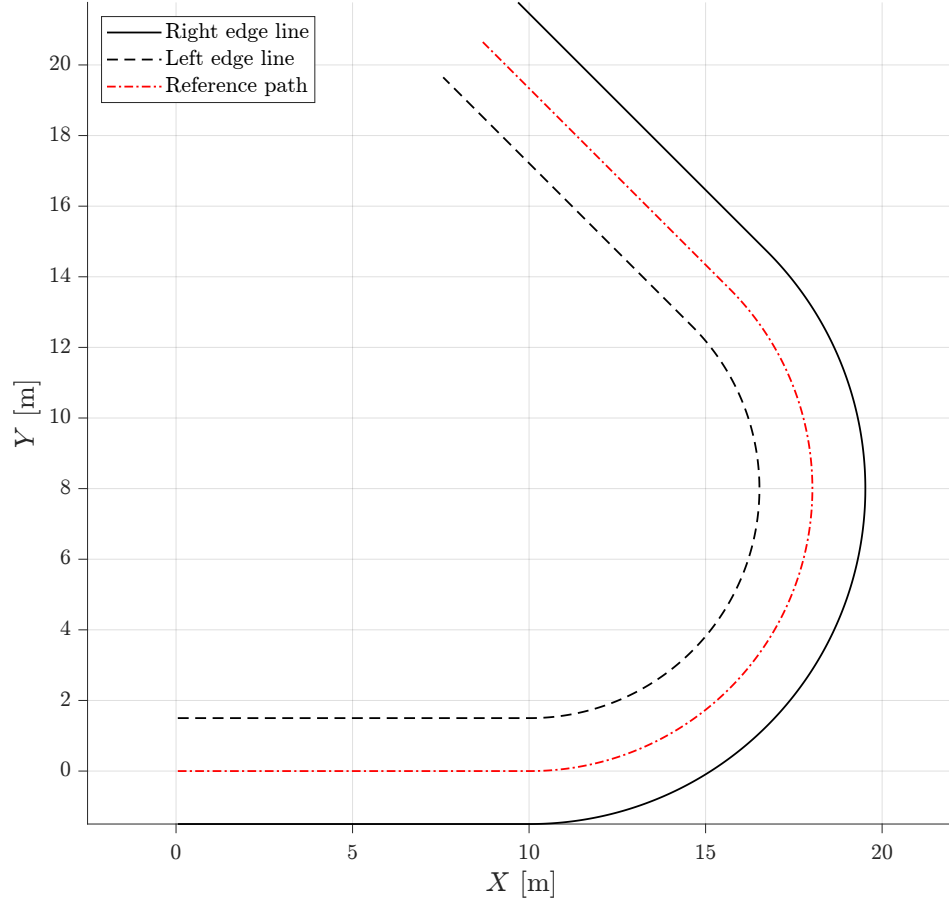


Figure 6.1: U-shaped path

The curvature of the arc is constant, generating a smooth and kinematically continuous trajectory, with an overall rotation simulating a complete inversion of direction within a 3 m-wide lane. The compact configuration and high curvature make this path particularly suitable for stressing the vehicle’s lateral dynamics.

From a control perspective, the U-shaped curve serves as a targeted test for evaluating both lateral and longitudinal controller performance under high-curvature conditions. The lateral analysis makes it possible to verify the controller’s ability to follow tight trajectories with continuous steering without oscillations or exit-phase delays, while the longitudinal analysis allows observation of speed modulation as a function of the curve’s small radius. The controller must indeed reduce the velocity to keep the lateral acceleration within comfort and safety limits, coordinating the interaction between the two control domains.

S-shaped Path

The S-shaped path was designed to simulate a sequence of curves with alternating curvature sign, typical of hilly roads or slip-road connectors, where the vehicle must adapt to alternating curvature while maintaining stability and smoothness. It represents a more fluid and realistic driving scenario compared to the U-shaped curve, as it combines wide, progressive sections with narrower, more dynamic ones.

Geometrically, the track consists of four sections:

- an entry straight section for reaching the target speed;
- a left-hand curve of 90° with a radius of 30 m;
- a right-hand curve of 90° with a radius of 20 m;
- and an exit straight section to observe steering realignment.

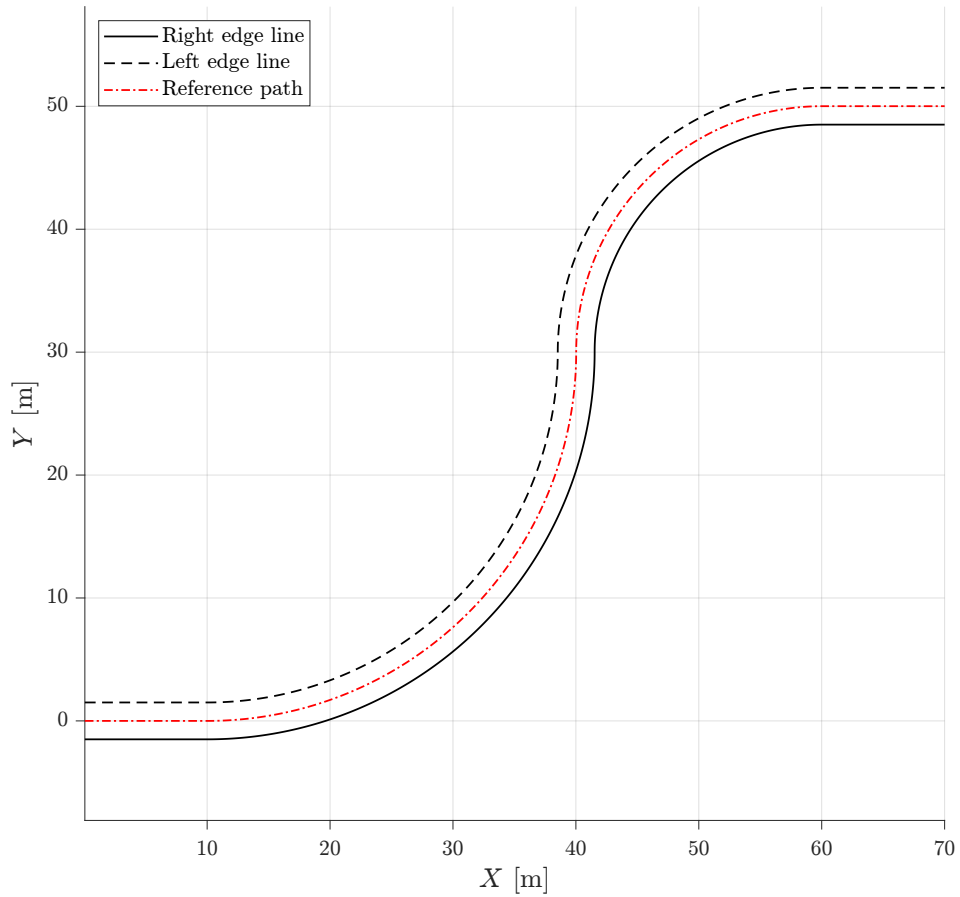


Figure 6.2: S-shaped path

The transition between the two curves is continuous, with no curvature discontinuities, ensuring a smooth and physically consistent trajectory. The difference in radius creates a geometric contrast that makes it possible to observe the vehicle's response to different curvatures in sequence, simulating a realistic sinuous motion.

From a control perspective, the S-shaped path provides a complete evaluation of the interaction between the lateral and longitudinal control domains. In the lateral domain, it allows for analysis of steering smoothness and stability during direction changes, while in the longitudinal domain it enables verification of speed regulation as a function of local curvature: in the first, wider curve, the vehicle can maintain a higher velocity, whereas in the second, tighter one, it must slow down to respect lateral acceleration limits and ensure comfort. This track therefore highlights the dynamic coordination between steering and velocity, showing how effectively the system adapts to the geometric variations of the path.

6.2 Analysis of Controllers

After defining the simulation environment and the reference paths, this section presents the experimental analysis of the controllers developed for the *Path Following* and *Low-Level Control* tasks. Each controller was tested under the same scenarios previously described, in order to ensure homogeneous and repeatable comparison conditions. Moreover, before proceeding with the specific analysis of each controller, it is useful to introduce the general parameters that define the vehicle configuration and the operating conditions common to all simulations, ensuring maximum consistency across all tests.

Table 6.1: General simulation parameters common to all controllers.

Vehicle geometry		
Wheelbase	2.9	m
Max steering angle	30.0	°
Comfort limits		
Reference speed	9.0	m/s
Max acceleration	2.0	m/s ²
Max deceleration	3.0	m/s ²
Max lateral acceleration	2.5	m/s ²
Actuators limits		
Steering rate	600	°/s
Steer deadband	0.3	°

The simulation results, represented through the plots recorded by the plotting node, allow direct observation of the controllers' dynamic response and enable a comparison of their performance in terms of lateral error, steering angle, velocity, and lateral acceleration. For each controller, the main plots are presented and the most relevant behaviors are discussed, highlighting both the strengths and the critical aspects observed during the simulations.

6.2.1 Pure Pursuit Controller

The *Pure Pursuit* controller represents a geometric approach to the lateral control problem, based on constructing a circular arc that connects the current vehicle position with a reference point located ahead along the desired trajectory. The distance between the vehicle and this point, known as the *lookahead distance*, is the key parameter determining the responsiveness and stability of the controller.

In the implemented version, the lookahead distance varies dynamically as a function of the vehicle's longitudinal speed, according to the following relationship:

$$L_d = \max(L_{\min}, k_L v^\gamma)$$

Unlike traditional *Pure Pursuit* formulations, where the gain k_L is adapted as a function of velocity to increase the lookahead distance at higher speeds, in this implementation the gain is kept constant while the modulation is introduced through the exponent γ . From a mathematical standpoint, varying γ allows for a continuous tuning of how L_d grows with velocity: for values of $\gamma < 1$, the relationship is sub-linear, resulting in smoother and more stable behavior; for $\gamma = 1$, the relationship is linear, and the lookahead distance grows proportionally with velocity; for $\gamma > 1$, the growth becomes super-linear, leading to a more aggressive increase of the preview distance and a potential reduction in steering responsiveness.

In this specific case, $\gamma = 0.83$ was selected, providing a gradual yet sufficient increase of the lookahead distance with velocity. This design choice improves trajectory smoothness and overall control stability, particularly during transitions between low- and high-speed segments.

The parameters used in the simulations are reported in Table 6.2.

Table 6.2: Pure Pursuit controller parameters.

Parameter	Symbole	Value	Unit
Minimum lookahead distance	$L_{\min}$	1.0	m
Lookahead gain	k_L	1.0	–
Velocity exponent	γ	0.83	–

This formulation allows the lookahead distance to increase progressively with speed, improving tracking smoothness and overall stability at higher velocities, while maintaining adequate reactivity in low-speed or high-curvature sections. The resulting behavior provides a balanced trade-off between tracking accuracy and steering comfort, making the controller particularly suitable for smooth driving scenarios and moderate curvature trajectories.

U-shaped path

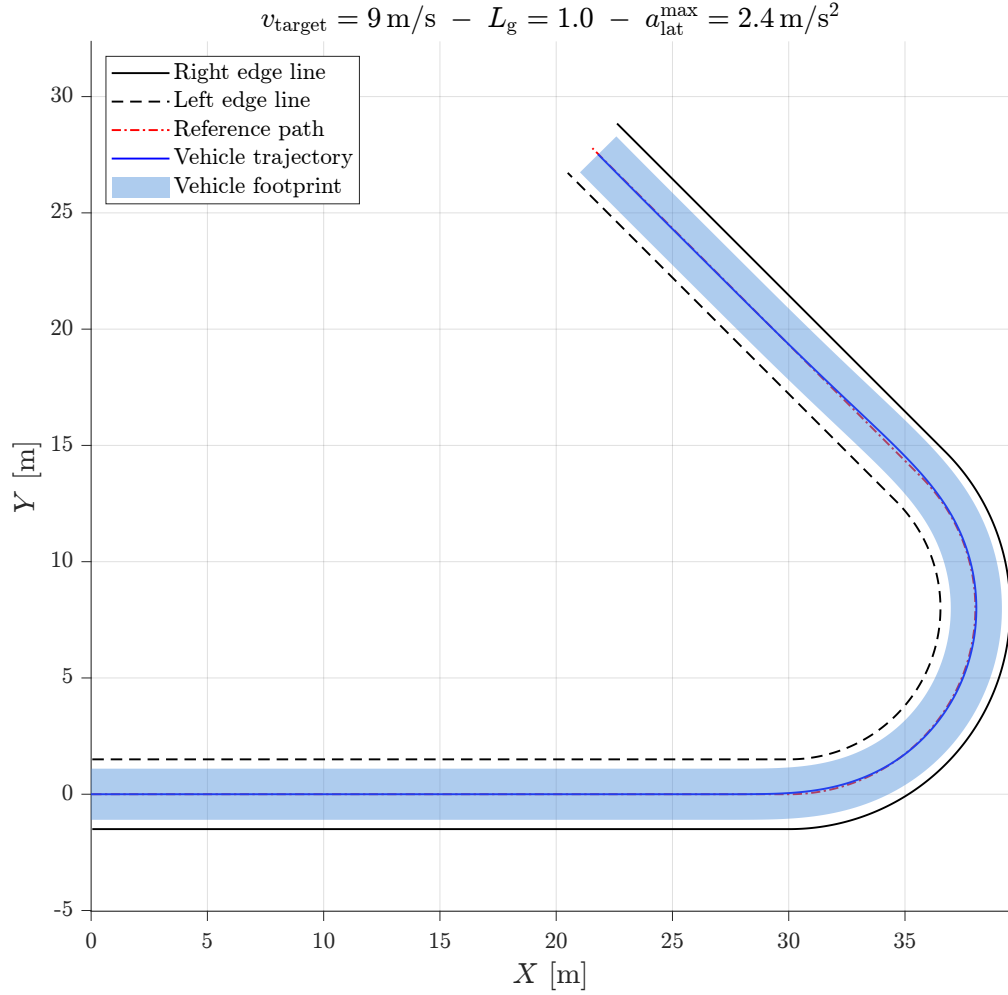


Figure 6.3: Pure Pursuit – U-shaped path - Vehicle trajectory and footprint

Trajectory The vehicle trajectory adheres closely to the reference path throughout the entire maneuver, and the footprint remains within the lane boundaries. This indicates that both the steering control and the velocity modulation respect

the vehicle's physical and dynamic limits. Nonetheless, small lateral deviations can be observed at the entry and exit of the curve. The vehicle tends to cut slightly inside the lane at the beginning of the turn, as the controller's lookahead point anticipates the curvature to maintain stability. At the same time, the steering rate and jerk limits constrain the system's responsiveness, preventing abrupt variations but introducing a slight delay at the start of the turn. In the exit phase, the same mechanism acts in the opposite direction: the steering angle is released gradually, and the output filter smooths the return towards the lane centerline, resulting in a small residual offset that disappears after a few meters. Overall, these small deviations represent the natural consequence of the dynamic constraints imposed on the system and illustrate an optimal trade-off between geometric precision and smooth vehicle behavior, as also confirmed by the following plots.

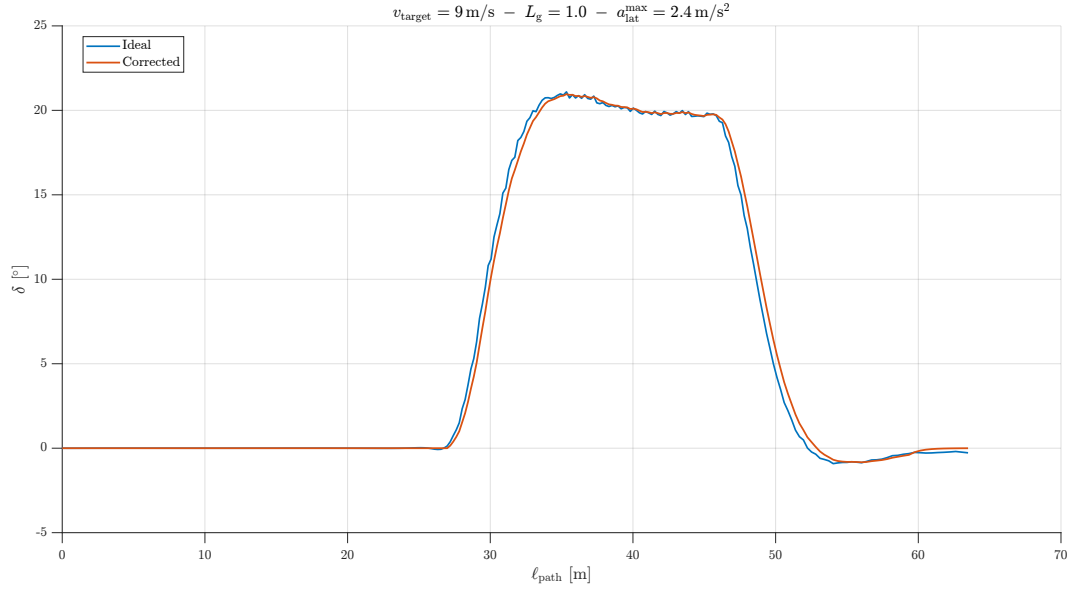


Figure 6.4: Pure Pursuit – U-shaped path - Steering angle

Steering Angle The steering angle plot highlights two distinct signals: the ideal command computed by the controller and the effective command applied to the vehicle after the dynamic limits are enforced. The corrected profile appears smoother and more regular, indicating that the constraints on steering rate and jerk, together with the exponential output filter, effectively dampen abrupt variations and prevent unwanted oscillations. The maximum amplitude remains consistent with the vehicle geometry and the path curvature, without reaching saturation. A slight phase difference can also be observed between the ideal and actual steering commands: the real steering slightly anticipates the curve entry, due to the lookahead behavior,

and delays marginally in the exit phase because of filtering and rate limits. The steering profile thus confirms the correct implementation of both kinematic and dynamic constraints, ensuring a progressive, stable, and comfort-compliant steering response.

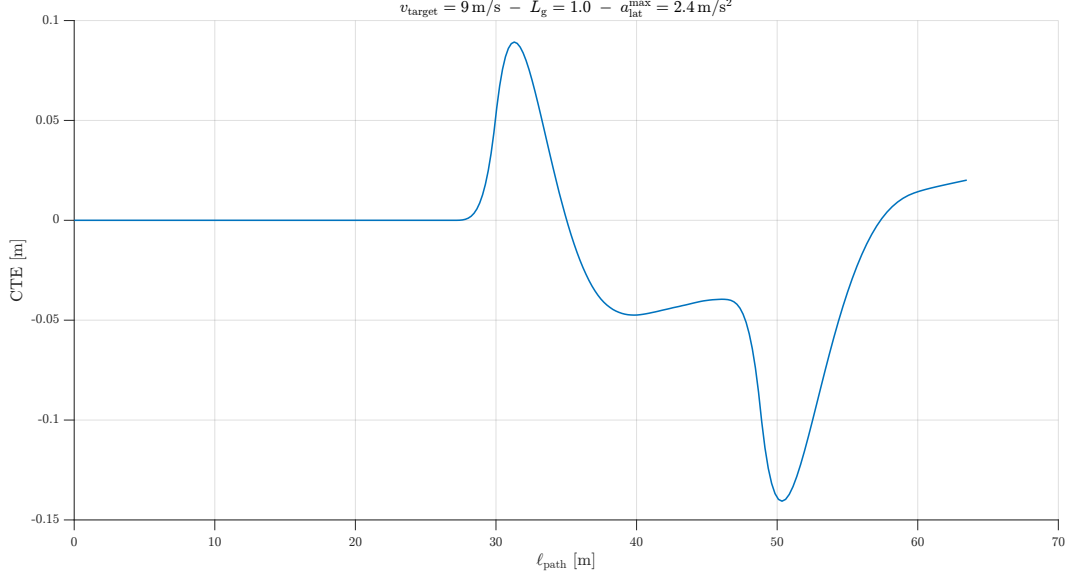


Figure 6.5: Pure Pursuit – U-shaped path - Cross-Track Error

Cross-Track Error The Cross-Track Error remains very low throughout the entire maneuver, with oscillations of only a few centimeters. The error slightly increases—up to around 10 cm—at the curve entry, when the vehicle generates the necessary lateral component for turning, then changes sign during the curve and stabilizes below 5 cm. At the exit, after a small overshoot of about 15 cm, a residual positive offset appears due to the gradual steering release and the deadband, which delay the complete return of the command by a short moment. The symmetry of the deviations and their rapid attenuation confirm a stable and well-balanced interaction between the anticipatory effect of the lookahead and the corrective action of the controller.

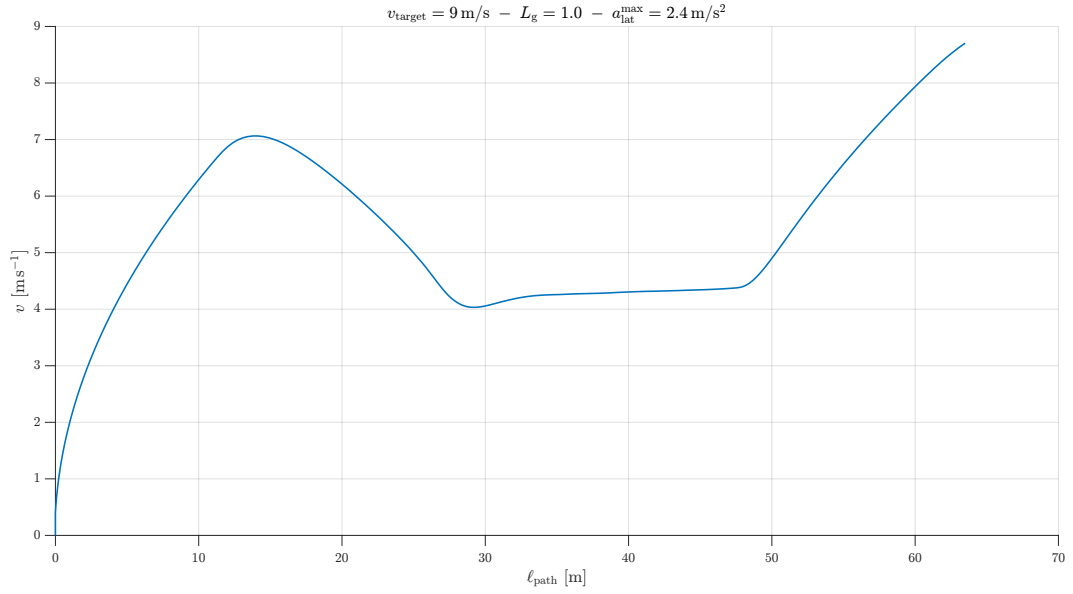


Figure 6.6: Pure Pursuit – U-shaped path - Longitudinal speed

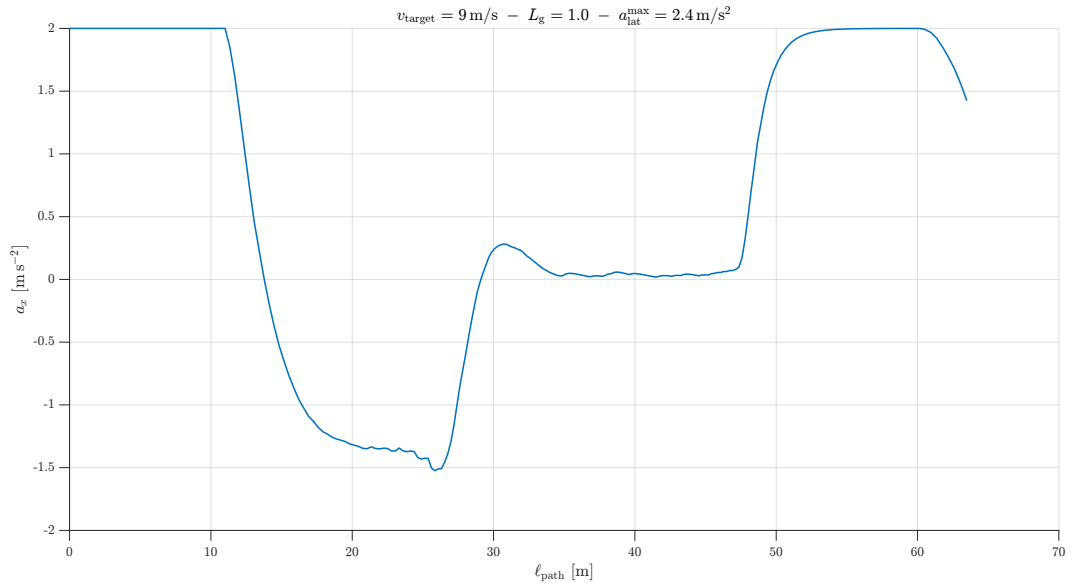


Figure 6.7: Pure Pursuit – U-shaped path - Longitudinal acceleration

Longitudinal speed and acceleration At the beginning of the path, the vehicle accelerates smoothly from standstill up to approximately 7 m/s within the first 12 m. In this phase, the controller maintains the target speed of 9 m/s, since the longitudinal preview window (set to 20 m) does not yet include any

critical curvature. After this point, as the preview window starts encompassing the upcoming U-turn (around 30 m), the controller reacts in advance by modulating the acceleration. The plot shows a progressive reduction of acceleration, gradually transitioning from positive to negative values, generating a smooth and continuous braking phase. The speed decreases without oscillations, stabilizing around 4 m/s between 28 and 30 m — a value that respects the lateral acceleration limit during the high-curvature section. In this region, longitudinal acceleration remains close to zero, indicating a quasi-steady-state dynamic balance. Once the most demanding curve segment is completed (around 48 m), the preview window progressively “releases” the high-curvature section, and the controller resumes throttle action: acceleration becomes positive again, producing a clean and gradual reacceleration up to the 9 m/s target.

Both deceleration and acceleration ramps are smooth and nearly symmetric, as the longitudinal action is rate- and jerk-limited, avoiding spikes or discontinuities. Overall, the combined analysis of speed and acceleration confirms a clearly predictive and dynamically coherent behavior: the controller does not react a posteriori to curvature, but rather anticipates it, coordinating braking, cornering, and reacceleration harmoniously to maximize stability and comfort.

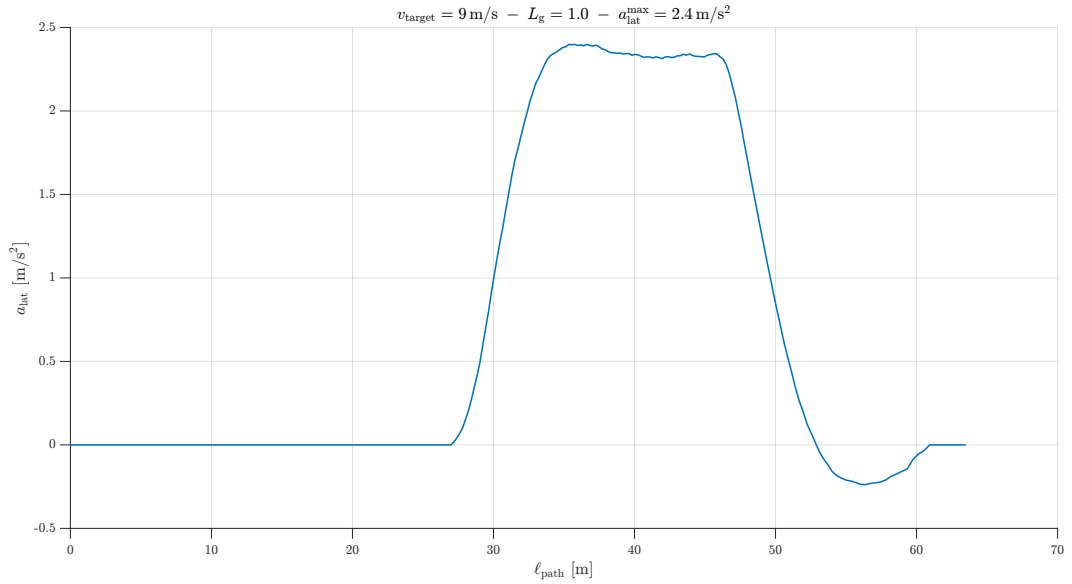


Figure 6.8: Pure Pursuit – U-shaped path - Lateral acceleration

Lateral acceleration The lateral acceleration profile follows the dynamics of the U-curve consistently. In the initial straight section, lateral acceleration remains close to zero; it then increases progressively as the controller initiates steering input, reaching the limiting value of approximately 2.4 m/s^2 , corresponding to the

predefined comfort and stability threshold. This peak is maintained throughout the central portion of the curve, indicating that the system effectively exploits available tire adhesion without exceeding dynamic constraints. In the exit phase, the transition is symmetric and equally smooth: lateral acceleration gradually decreases toward zero as the steering angle is released and the vehicle realigns with the straight section. The absence of discontinuities or residual peaks confirms that steering modulation is perfectly coordinated with velocity control, producing a continuous and predictable vehicle response. In summary, the lateral acceleration profile demonstrates effective management of lateral dynamics: the vehicle enters, negotiates, and exits the curve while consistently operating at the imposed limit without exceeding it.

S-shaped path

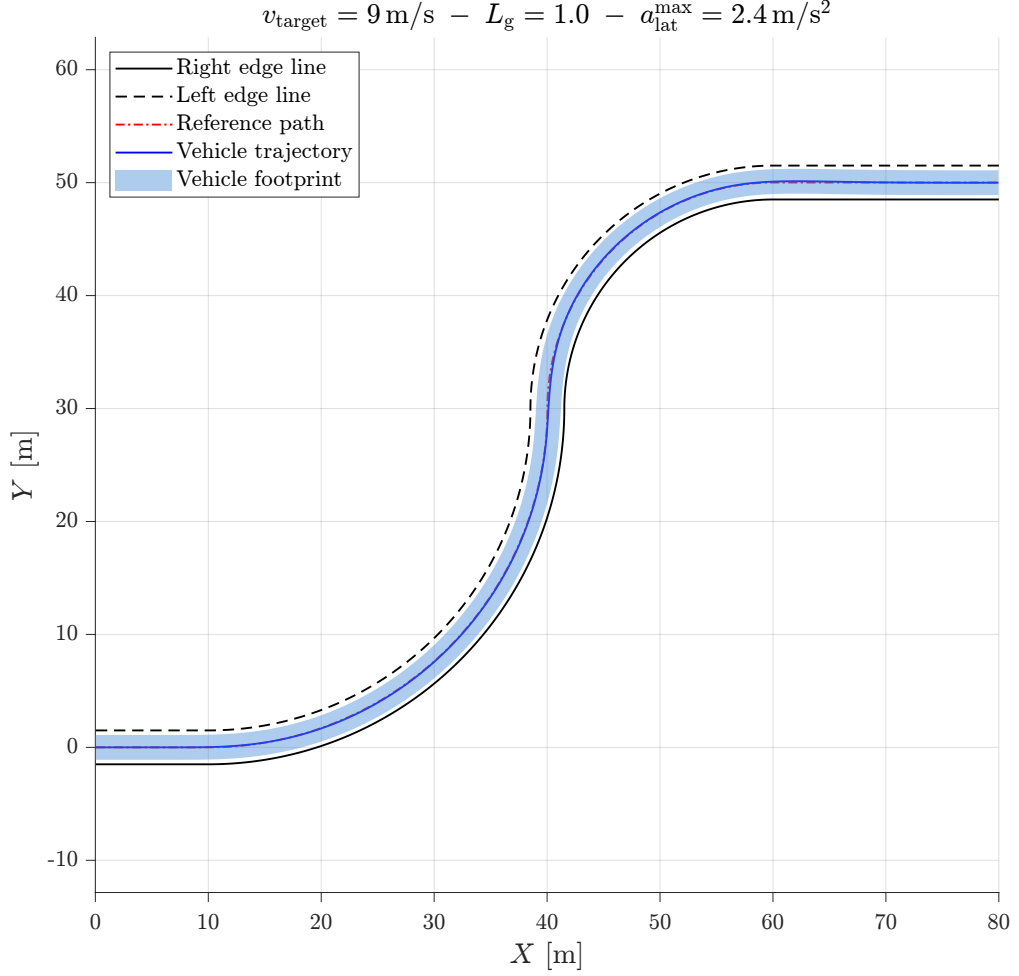


Figure 6.9: Pure Pursuit – S-shaped path - Vehicle trajectory and footprint

Trajectory The vehicle trajectory adheres very well to the reference path along the entire S-curve, maintaining the footprint entirely within the lane boundaries. During the traversal of each curve, the dynamics remain smooth and stable: the vehicle follows the intended curvature precisely, without oscillations or cutting the path. The most interesting part is observed in the central region, where the curvature changes sign. In this transition, the vehicle cannot instantly reproduce the new direction of the path, exhibiting a very slight alignment delay. This is due to the steering rate and jerk limits, which prevent overly rapid command variations and slow down the transition between opposite steering angles. The result is a very soft and controlled curvature inversion, where the trajectory shows

no discontinuities but only a small natural phase shift — a direct expression of the system’s dynamic constraints and the priority given to motion continuity over perfect geometric adherence.

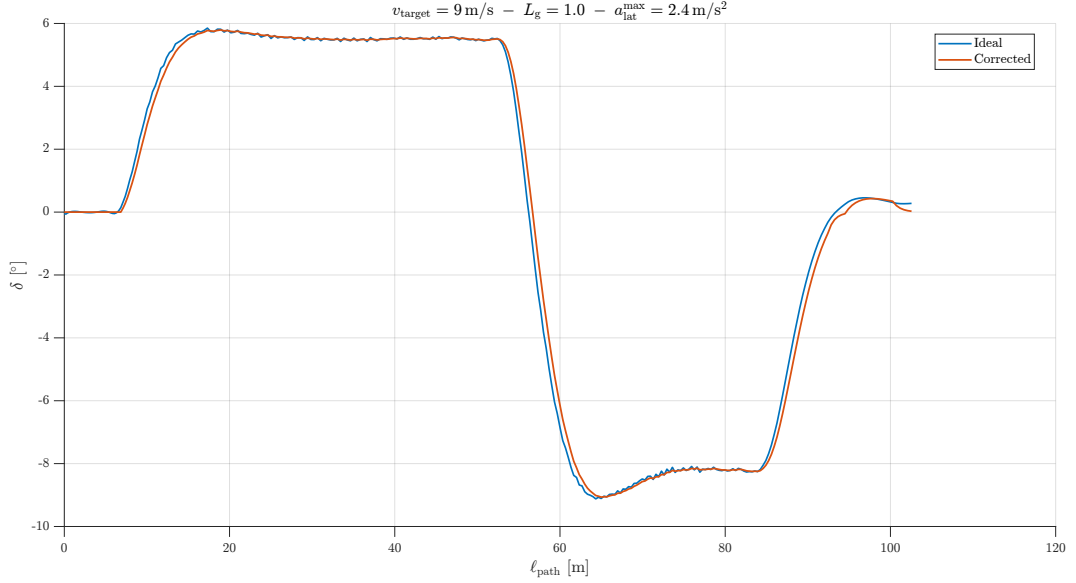


Figure 6.10: Pure Pursuit – S-shaped path - Steering angle

Steering angle The steering angle plot clearly highlights the two sign inversions corresponding to the opposite curves of the S-shape. During the first curve, the steering increases steadily until it reaches the value required to generate the prescribed curvature, then progressively reverses direction to tackle the second curve. The corrected steering command appears smoother than the ideal one: the slopes are gentler and the peaks less pronounced, indicating that the steering rate and jerk limits reduce the rate of variation, improving stability and comfort but introducing a slight delay in the transient phases.

During the curvature inversion, the transition from a positive to a negative steering angle does not occur instantaneously but through a softer intermediate phase, in which the steering smoothly crosses zero. This behavior explains the small phase shift observed in the trajectory: the vehicle does not “cut through” the S but follows a more gradual path. At the exit of the second curve, the steering returns toward zero in a controlled manner, without overshoot or residual oscillations, confirming that the output filtering works correctly in ensuring a smooth release of the command.

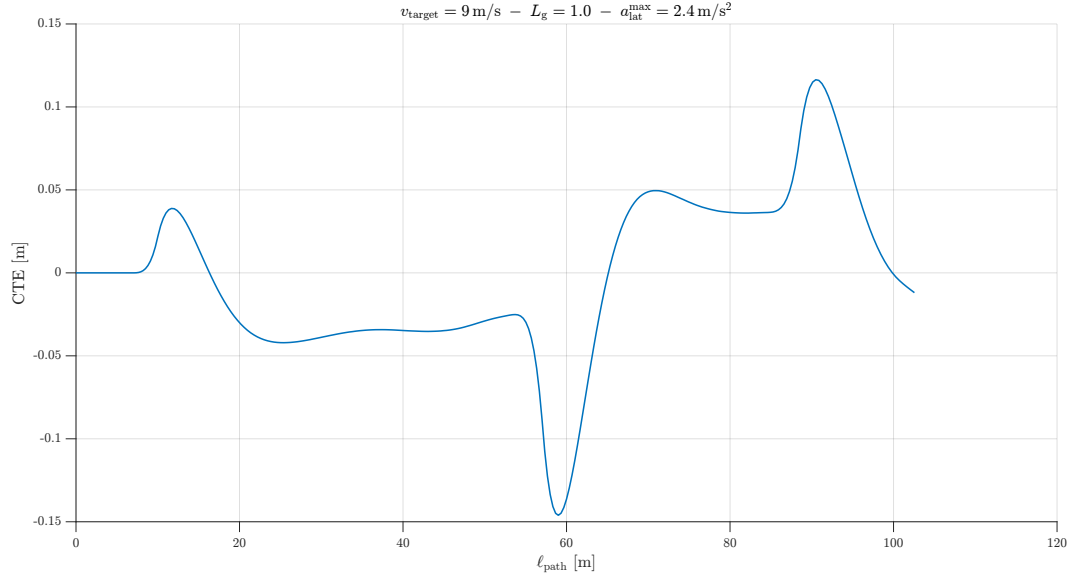


Figure 6.11: Pure Pursuit – S-shaped path - Cross-Track Error

Cross-Track Error The Cross-Track Error remains small along the entire path, with oscillations well within 15 cm. During each curve, the error remains low and stable, confirming accurate lateral alignment. The most evident peaks appear at the entry and exit of the S and, in particular, at the curvature inversion point, where the vehicle must simultaneously reverse both the steering direction and the lateral acceleration component. In these regions, the CTE reaches its maximum values due to the limited steering rate and jerk, which delay the generation of the new curvature. After the transition, the error quickly returns to below 5 cm, maintaining a smooth and oscillation-free trend. The symmetry of the two deviations and their rapid attenuation confirm that the controller preserves a stable and balanced behavior even during fast geometric transitions of the path.

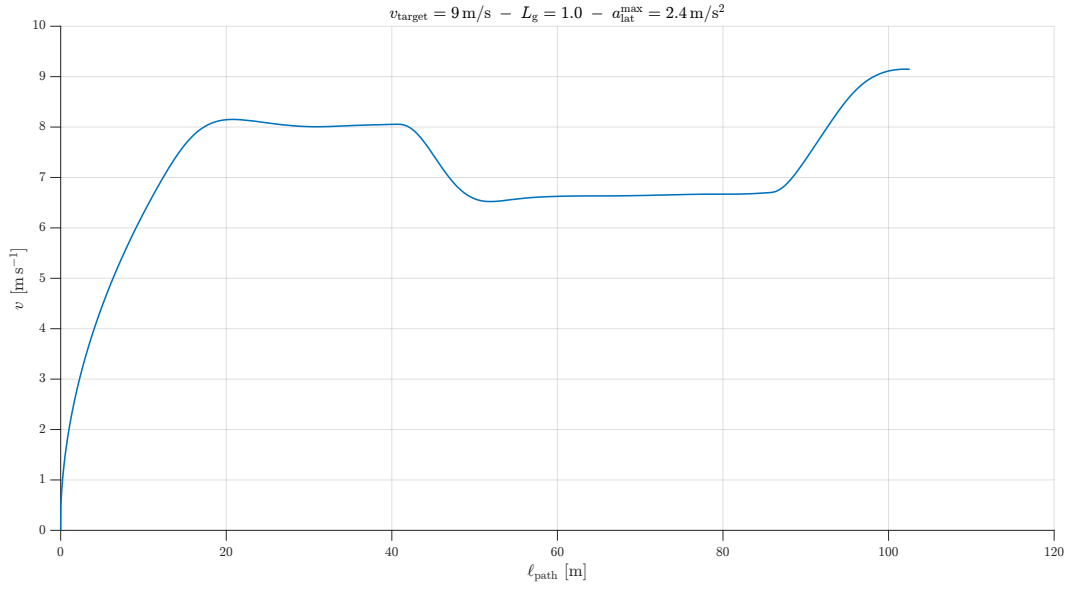


Figure 6.12: Pure Pursuit – S-shaped path - Longitudinal speed

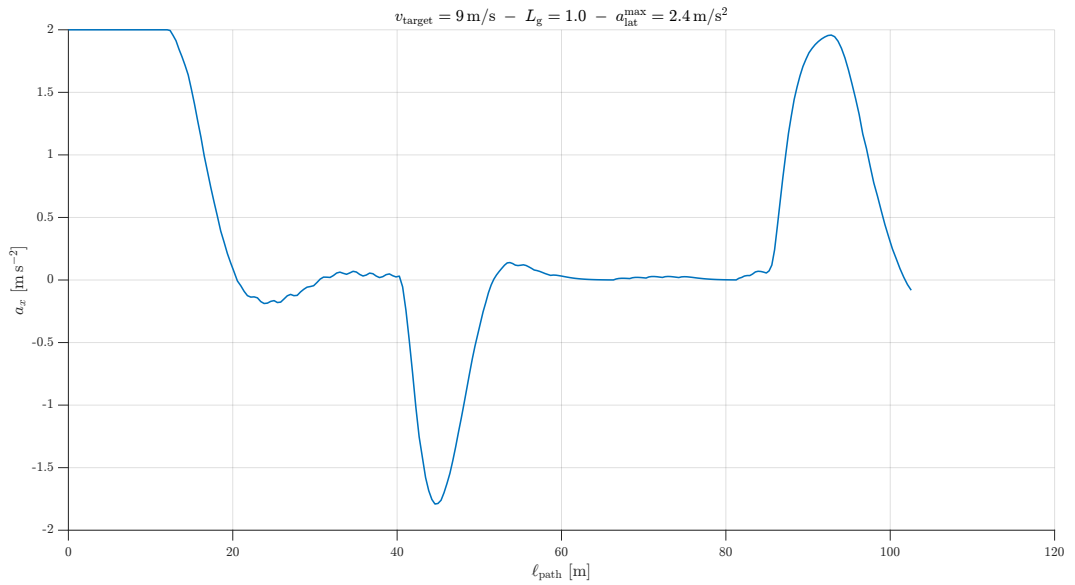


Figure 6.13: Pure Pursuit – S-shaped path - Longitudinal acceleration

Longitudinal speed and acceleration The longitudinal speed profile appears extremely smooth and well-modulated, with two main constant-speed segments corresponding to the two curves of the S. Unlike the U-turn scenario, in this case the first curve already lies within the 20 m longitudinal preview window from the start

of the simulation. For this reason, the controller does not accelerate up to the 9 m/s target speed from the beginning, but instead stabilizes around 8 m/s — already adequate for the upcoming curvature. The velocity remains stable throughout the first curve, confirming that curvature and corresponding speed estimation are correctly computed. Subsequently, a more pronounced speed reduction is observed as the preview window intercepts the second curve, characterized by a smaller radius. In this phase, speed stabilizes at approximately 7 m/s until the end of the second curve, before gradually increasing again to reach the 9 m/s target at the end of the path.

The longitudinal acceleration profile mirrors this behavior, alternating brief phases of anticipatory deceleration and progressive reacceleration, perfectly synchronized with the sequence of curves. Values remain moderate — generally within 2 m/s^2 — indicating smooth and well-filtered control. The transition between braking and acceleration occurs symmetrically and without discontinuities, thanks to the rate and jerk limits imposed on acceleration, which prevent abrupt variations. Overall, the combined analysis of speed and acceleration shows a clearly predictive and dynamically coherent behavior: the system sets the appropriate velocity for the first curve from the beginning, maintains stability during both curvature inversions, and gradually restores the target speed only once the vehicle returns to the straight segment.

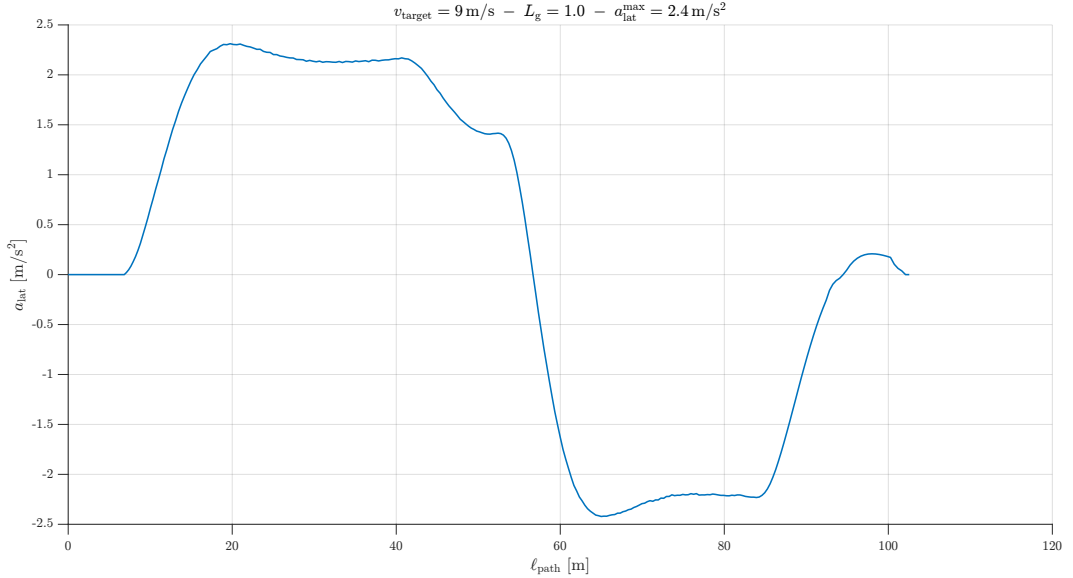


Figure 6.14: Pure Pursuit – S-shaped path - Lateral acceleration

Lateral acceleration The lateral acceleration plot shows two opposite lobes corresponding to the two curves of the S. In both cases, the value stabilizes around

2.4 m/s², the maximum limit imposed by the controller, indicating that the system fully exploits available tire grip while maintaining vehicle stability. Although the two curves have different radii, the lateral acceleration reaches similar magnitudes due to automatic compensation between curvature and velocity: in the first, wider curve, the steering angle is smaller while the velocity is slightly higher; in the second, tighter curve, the steering increases while the velocity decreases, keeping the resulting centrifugal force approximately constant. The transition between the two lobes is smooth and continuous: lateral acceleration crosses zero gently, without peaks or discontinuities, reflecting the perfect coordination between steering variation and velocity modulation. The entire profile appears symmetric and well balanced, showing that the vehicle remains consistently at the prescribed lateral limit without exceeding it or introducing instability.

Overall, the *Pure Pursuit* controller exhibits an extremely smooth and harmonious behavior, characterized by continuous trajectories and the complete absence of irregularities. Steering actions are always gradual and progressive, with soft transitions both at curve entry and exit, while longitudinal modulation ensures a balanced management of speed. The controller’s design clearly prioritizes stability and motion continuity, favoring a comfortable and well-damped behavior even under varying curvature conditions. This approach entails a slight loss of geometric precision, visible as small phase shifts during transitions, but enables a consistent, predictable, and physically realistic driving behavior. The *Pure Pursuit* therefore stands out for its smoothness and stability, qualities that make it particularly suitable for scenarios where control fluidity is prioritized over absolute lateral precision.

6.2.2 Stanley Controller

The *Stanley* controller represents a geometric approach distinct from the *Pure Pursuit*, based on the simultaneous correction of both the lateral and heading errors between the vehicle and the reference trajectory. Its control law generates a steering command proportional to the sum of two main contributions: a heading alignment term, which minimizes the orientation difference between the vehicle’s longitudinal axis and the path tangent, and a lateral correction term, which tends to reduce the Cross-Track Error.

The desired steering angle is computed according to the following relationship:

$$\delta = \theta_e + \tan^{-1} \left(\frac{k \cdot e_y}{v + \epsilon} \right)$$

where ϵ is a small regularization term to prevent singularities at low speed.

In the code implementation, the lateral error e_y is defined with respect to the front axle, as in the classical *Stanley* formulation. However, for plotting purposes

only, the CTE shown in the simulation results was recomputed with respect to the geometric center of the vehicle, so as to represent a physically meaningful lateral deviation. This adjustment does not affect the controller's operation but allows for a more intuitive and consistent interpretation of the results, since the plotted CTE directly corresponds to the lateral offset of the vehicle body relative to the reference trajectory.

Similarly to the *Pure Pursuit* controller, the *Stanley* implementation employs a dynamic *lookahead* parameter defined as:

$$L_d = \max(L_{\min}, k_L v^\gamma)$$

where $L_{\min}$ and k_L represent respectively the minimum lookahead distance and the corresponding gain. This parameter influences the trajectory sampling window and therefore the controller's responsiveness: larger values enhance stability but reduce agility, while smaller values increase reactivity at the expense of potential sensitivity to noise or abrupt curvature changes.

The parameters adopted for the simulations are summarized in Table 6.3. They represent a balanced compromise between stability, accuracy, and smoothness, ensuring consistent tracking performance under varying dynamic conditions.

Table 6.3: Stanley controller parameters.

Parameter	Symbol	Value	Unit
Proportional gain on CTE	k	2.5	–
Minimum lookahead distance	$L_{\min}$	0.5	m
Lookahead gain	k_L	0.5	–
Velocity exponent	γ	0.83	–

U-shaped path

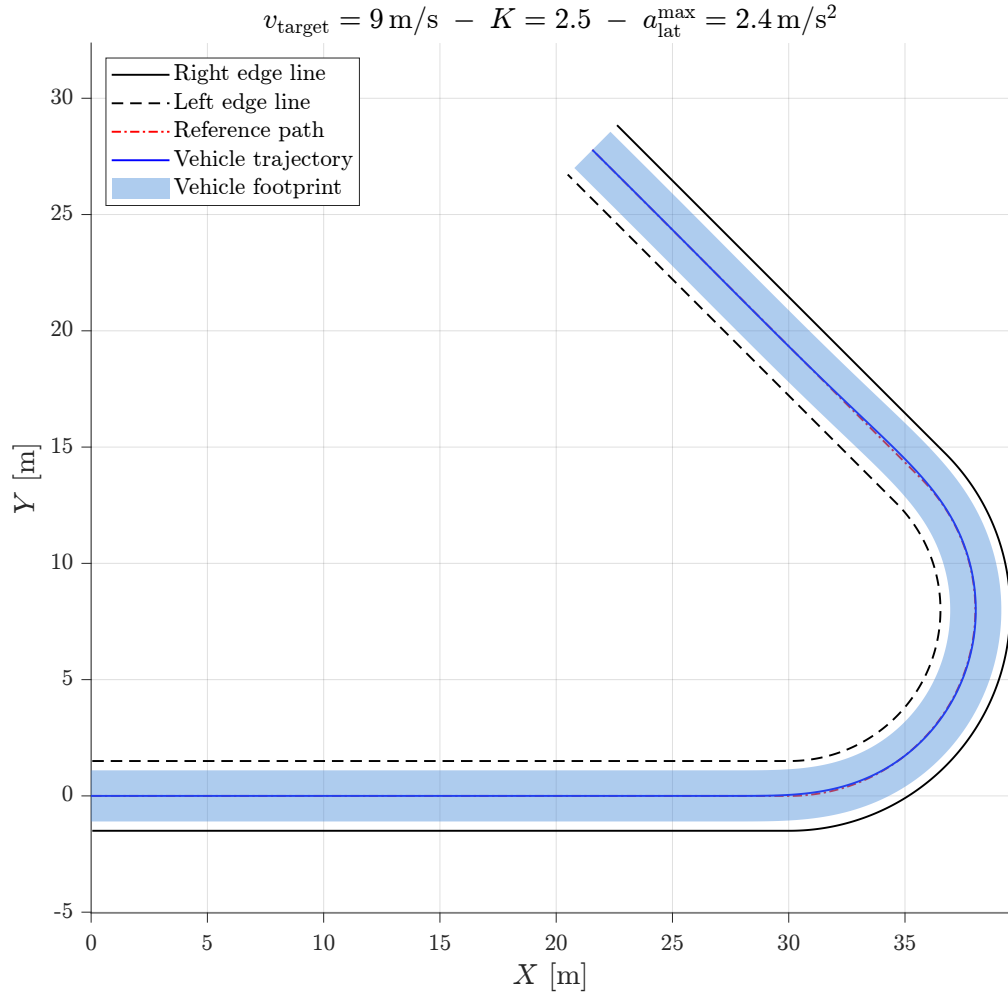


Figure 6.15: Stanley Controller – U-shaped path - Vehicle trajectory and footprint

Trajectory The vehicle trajectory follows the reference path with good accuracy throughout the entire U-turn, keeping the footprint fully within the lane boundaries. At the curve entry, the response is prompt: the vehicle initiates steering decisively, reducing alignment delay and minimizing initial deviations. During cornering, the trajectory remains stable and regular, indicating effective curvature control. At the exit, the return toward the straight section is fast yet well-modulated, with no overshoot or path cutting. The slight lateral deviations observed at the curve edges result from the combined effect of the anticipatory lookahead and the rate and jerk limits imposed on the steering, which slightly delay realignment.

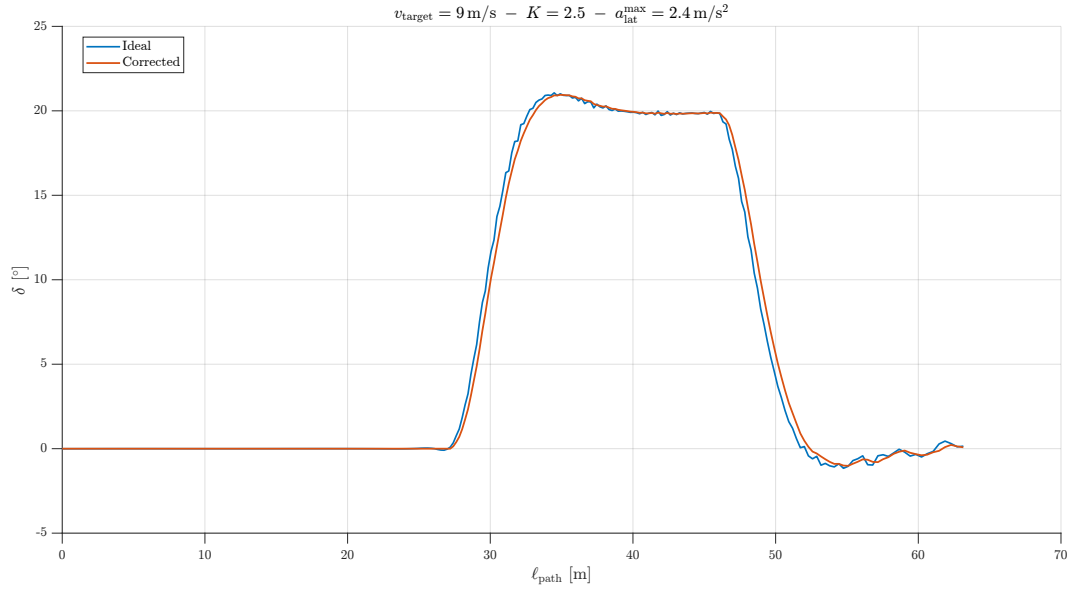


Figure 6.16: Stanley Controller – U-shaped path - Steering angle

Steering angle The steering angle profile exhibits a direct and continuous response: the command increases decisively at the curve entry, reaches the value required for the prescribed curvature, and then returns quickly and smoothly at the exit. The corrected steering trace closely follows the ideal one, with only minor differences due to the rate and jerk limits, which smooth out steep slopes without introducing overshoot. The overall trend appears linear and oscillation-free, though slightly more “tense” than the *Pure Pursuit* during transients — the command favors prompt alignment with the curve while maintaining a clean and consistent steering profile up to full realignment.

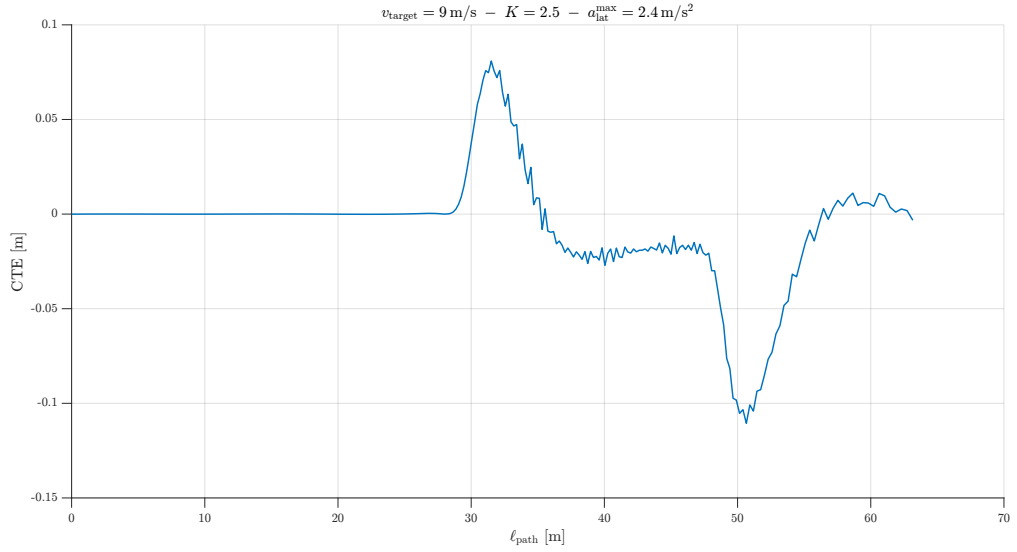


Figure 6.17: Stanley Controller – U-shaped path - Cross-Track Error

Cross-Track Error The Cross-Track Error remains small throughout the maneuver, with values within 10 cm during the curve entry and exit phases. During steady-state cornering, the error stays nearly constant at around 2 cm, with no oscillations or drift. The controller’s ability to correct deviations is evident: corrections are fast and precise yet well-damped, avoiding abrupt reactions or instability. Overall, lateral tracking appears firm and responsive, with behavior that prioritizes geometric precision while preserving dynamic stability.

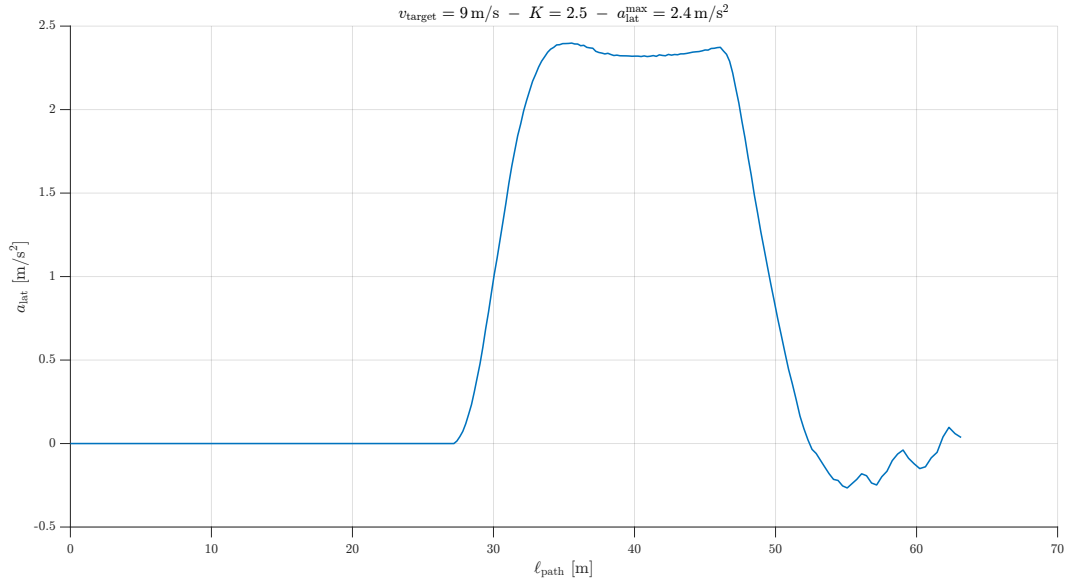


Figure 6.18: Stanley Controller – U-shaped path - Lateral acceleration

Lateral acceleration The lateral acceleration profile clearly reflects the dynamic nature of the maneuver: it rises progressively to values close to the maximum limit of 2.4 m/s^2 , remains steady over most of the curve, and then decreases symmetrically at the exit. The rise toward the peak is faster than in the *Pure Pursuit* case, yet still smooth thanks to the rate and jerk constraints that prevent abrupt steering variations. The absence of oscillations or overshoot confirms proper coordination between steering and velocity, ensuring that the vehicle maintains a consistent lateral dynamic behavior — operating right at the grip limit without exceeding it.

S-shaped path

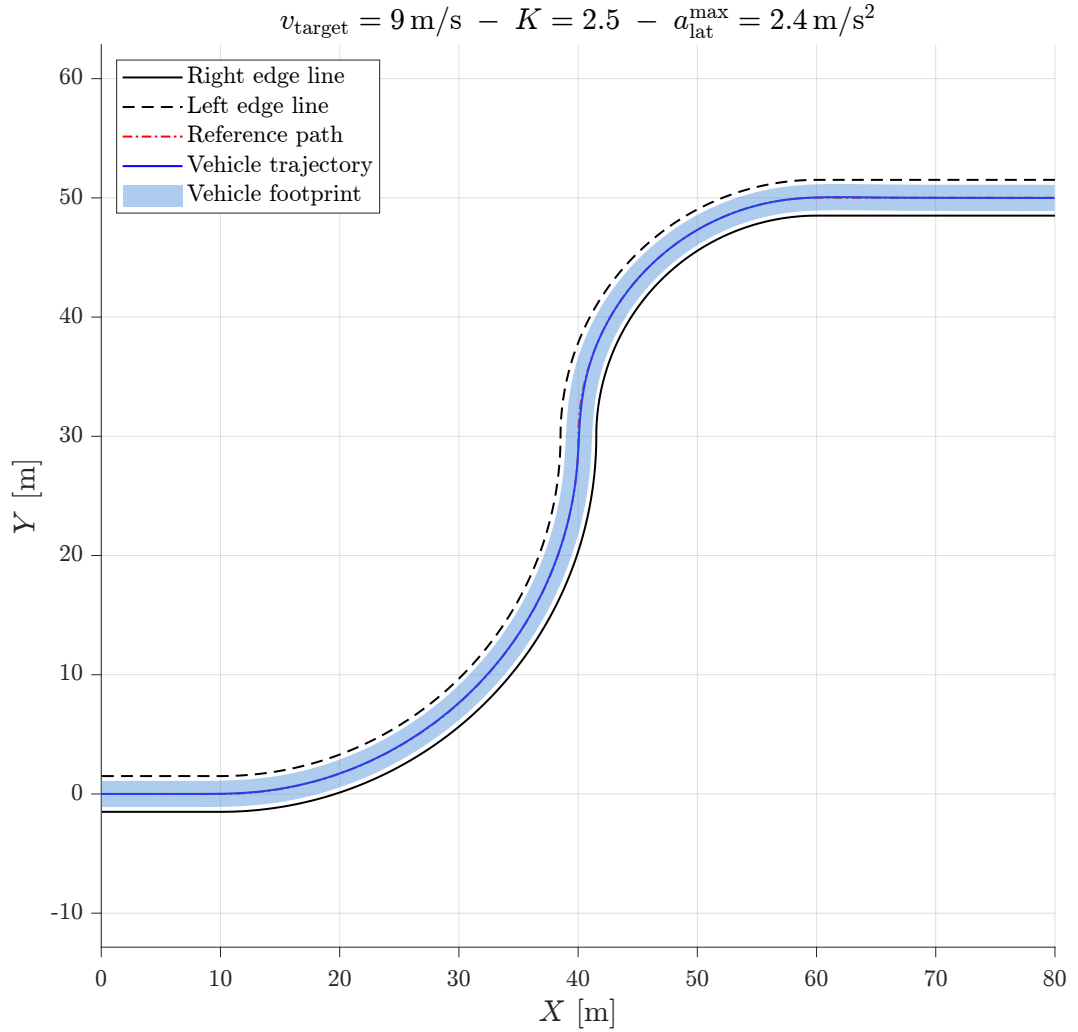


Figure 6.19: Stanley Controller – S-shaped path - Vehicle trajectory and footprint

Trajectory The vehicle trajectory follows the reference path with high precision, keeping the footprint perfectly within the lane boundaries. The path appears smooth and continuous, a sign of effective curvature control, though with a more prompt and decisive response compared to other controllers. In the central section, where the curvature changes sign, the vehicle quickly realigns to the opposite direction, reducing the lateral phase shift to just a few centimeters. This behavior confirms the high reactivity of the *Stanley* controller, which corrects deviations promptly but may produce a slightly tenser dynamic during transients.

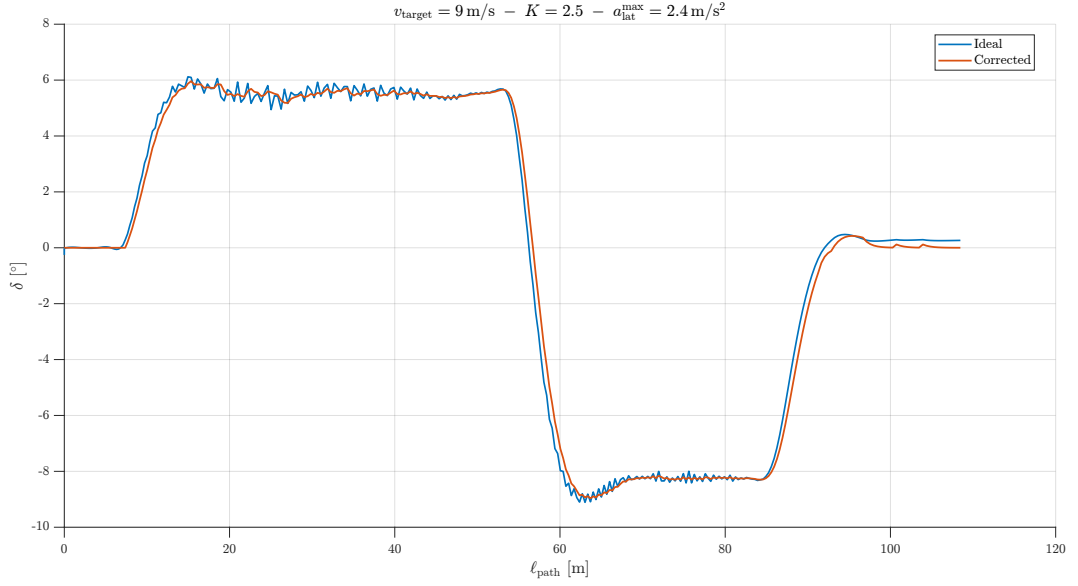


Figure 6.20: Stanley Controller – S-shaped path - Steering angle

Steering angle The steering angle profile shows a direct and well-proportioned response to the path curvature. The corrected command closely reproduces the ideal profile, though with small “micro-segmentation” visible mainly during cornering and at the curvature inversion point. These small irregularities stem from the controller’s high sensitivity to variations in both lateral and heading error: the corrective action is extremely fast and tends to react even to minimal deviations. The result is a precise steering command consistent with the required dynamics, but with a slightly more “nervous” response due to the combination of the high proportional gain and the steering rate and jerk limits that discretize the output signal.

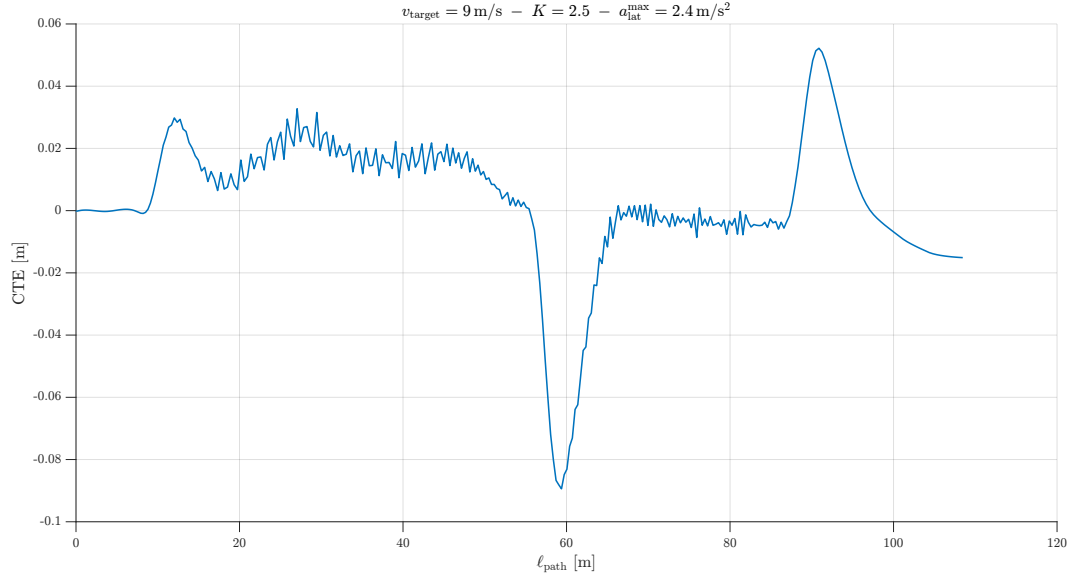


Figure 6.21: Stanley Controller – S-shaped path - Cross-Track Error

Cross-Track Error The CTE plot confirms the controller’s tracking precision: the lateral error remains very small, around 2 cm during the curved sections, with a narrow and short peak at the transition between the two curves. These minimal deviations are the direct consequence of the controller’s high responsiveness, which tends to correct even the smallest departures through rapid interventions. The error profile is clean and regular, showing no drift over time, indicating that the vehicle remains consistently centered on the reference path. Overall, the CTE trend highlights extremely accurate tracking, achieved at the cost of a slightly more energetic and less filtered control action during transients.

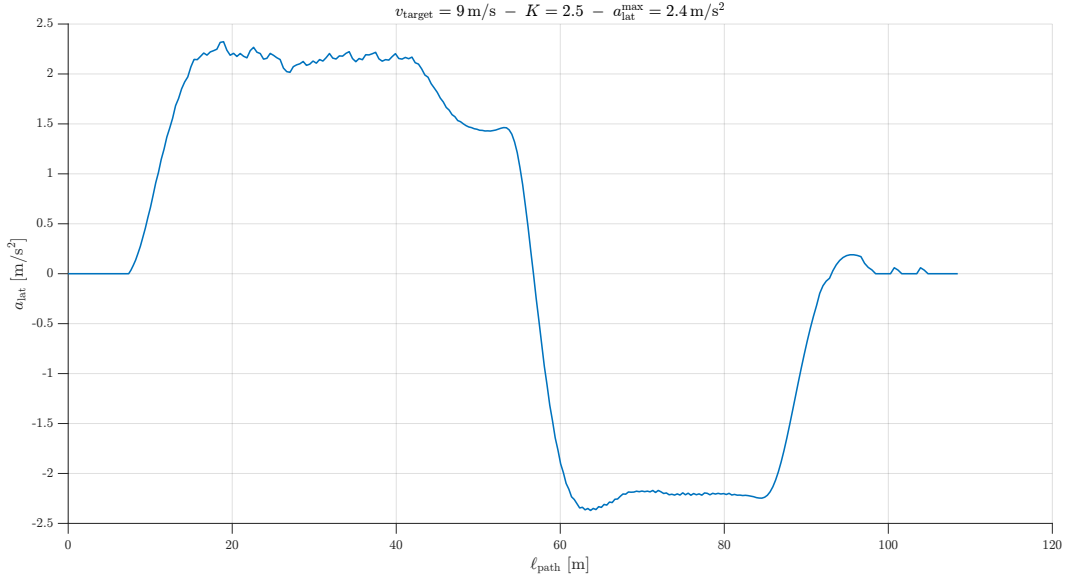


Figure 6.22: Stanley Controller – S-shaped path - Lateral acceleration

Lateral acceleration The lateral acceleration plot shows two opposite lobes corresponding to the two curves of the S, with values close to the imposed limit of 2.4 m/s^2 . The overall trend is correct and consistent with the maneuver, though small oscillations appear along the sides of the lobes, caused by the same reactivity observed in the steering command. Each micro-correction of the steering angle is directly reflected in the lateral acceleration, producing minor local oscillations. Nevertheless, the overall shape remains regular and symmetric: no abnormal peaks or discontinuities are present, and the vehicle consistently maintains the limit value without exceeding it. This behavior shows that the controller operates at maximum responsiveness, maintaining strong grip and stability but with a slightly stiffer and more sensitive dynamic profile.

In summary, the *Stanley* controller demonstrates solid and consistent behavior, capable of following the reference path with very high precision in both the U and S curves. The lateral error remains consistently low, and path adherence is excellent, confirming precise and stable tracking. The system's response is prompt and decisive: the steering reacts quickly to curvature variations, keeping the vehicle well centered, though this high reactivity introduces small oscillations in the steering and lateral acceleration plots due to the controller's sensitivity and the applied rate and jerk limits. The longitudinal control is not shown here, as it is identical to that used for the *Pure Pursuit* controller; therefore, speed and acceleration are managed using the same predictive logic and dynamic constraints. Overall, the *Stanley* controller ensures accurate and reactive path tracking, with a tighter yet stable dynamic behavior, fully consistent with the vehicle's physical limitations.

6.2.3 Model Predictive Control (MPC)

For the simulations of the *Model Predictive Control* (MPC) algorithm, the same standard configuration adopted in the previous tests was used. The controller operates based on the kinematic bicycle model and follows a receding-horizon formulation, computing at each control step the optimal sequence of steering and acceleration commands while accounting for the vehicle's kinematic and dynamic constraints.

In the setup used for the simulations, the prediction horizon was set to $N = 10$ steps, with a sampling interval of $\Delta t = 0.1$ s. This corresponds to a total prediction window of one second, providing a balanced compromise between predictive capability and computational efficiency. The optimization problem is solved at every control cycle, updating the solution in real time based on the current vehicle state.

The main parameters used for the MPC configuration are summarized in Table 6.4.

Table 6.4: Main MPC controller parameters.

Parameter	Symbol	Value	Unit
Prediction horizon	N	10	–
Sampling time	Δt	0.1	s

U shaped path

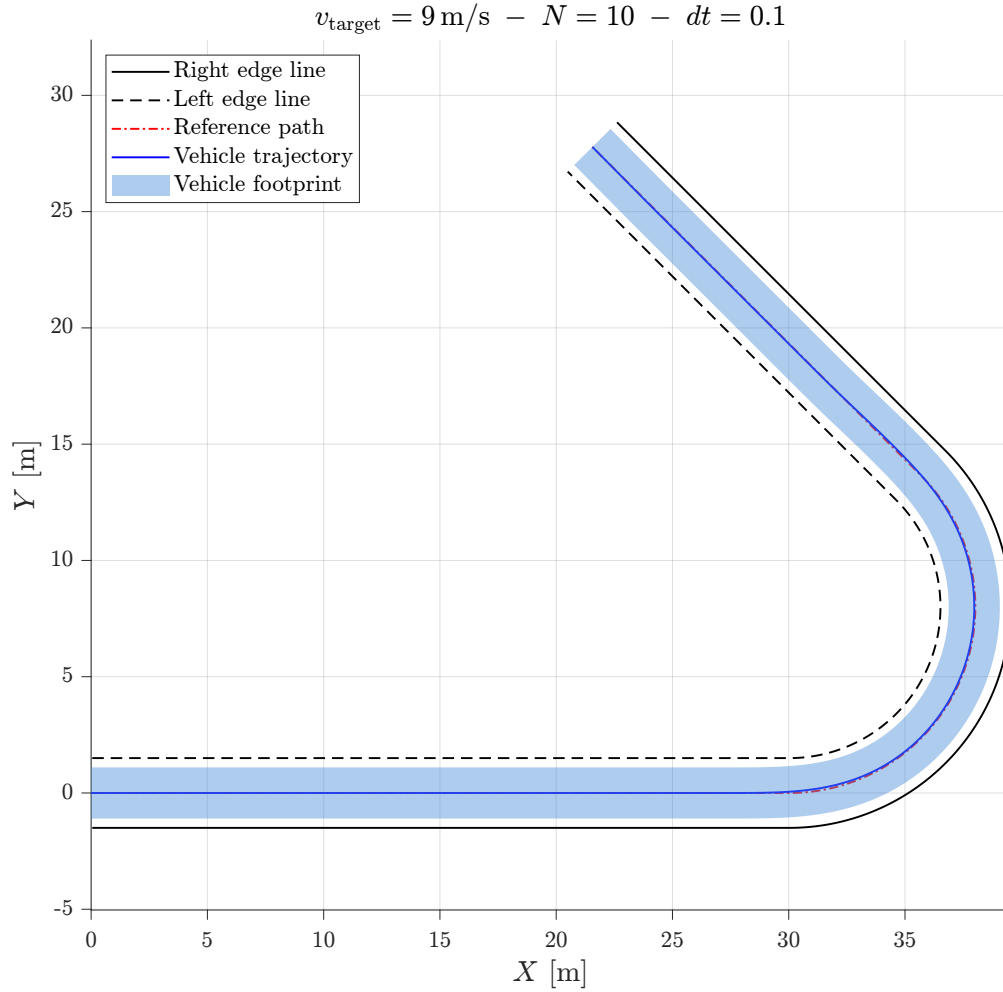


Figure 6.23: MPC – U-shaped path - Vehicle trajectory and footprint

Trajectory The vehicle trajectory adheres extremely well to the reference path, remaining perfectly centered within the lane boundaries without noticeable deviations in either the entry or exit phase. The overall curvature is smooth and continuous, indicating a geometrically consistent and dynamically stable motion throughout the entire path. The vehicle negotiates the tight turn (radius 8 m) with high accuracy, confirming that the predictive planner effectively coordinates steering and velocity. The absence of overshoot or understeer demonstrates that the MPC anticipates curvature variations and adjusts its trajectory in advance while fully respecting comfort and stability limits.

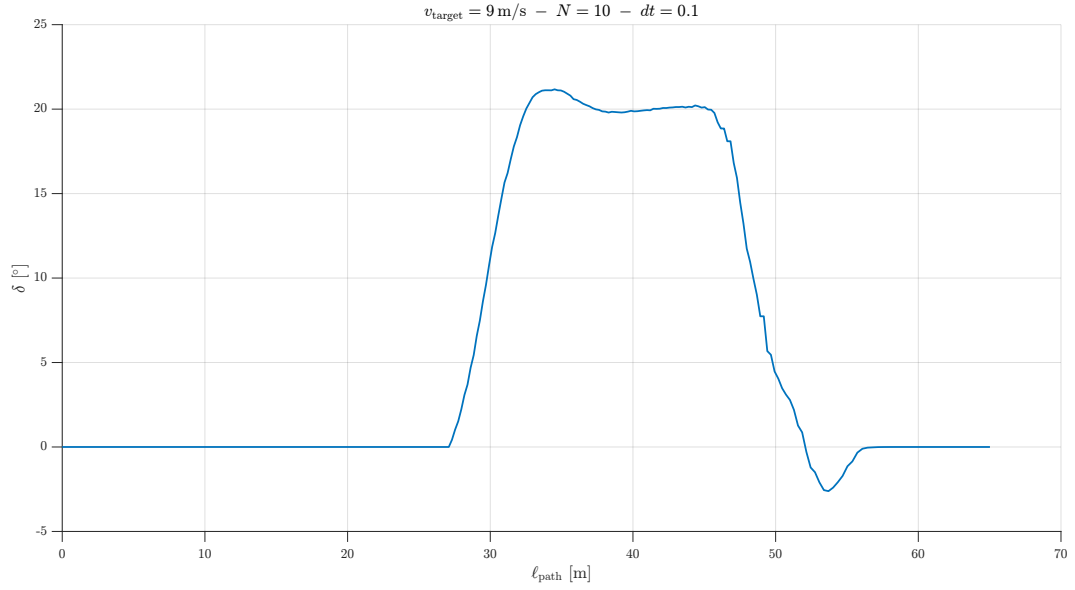


Figure 6.24: MPC – U-shaped path - Steering Angle

Steering angle The steering angle profile is regular and well-proportioned to the path curvature. The command increases progressively up to a peak of approximately 20° , which is consistent with the geometric requirements of the curve. During cornering, the steering remains nearly constant around $19\text{--}20^\circ$, then gradually decreases toward zero in a symmetric manner. The absence of oscillations or discontinuities confirms that the rate- and jerk-penalization terms are effective in ensuring a smooth yet precise response. Overall, the steering behavior appears kinematically consistent and dynamically stable, as expected from an optimized predictive controller.

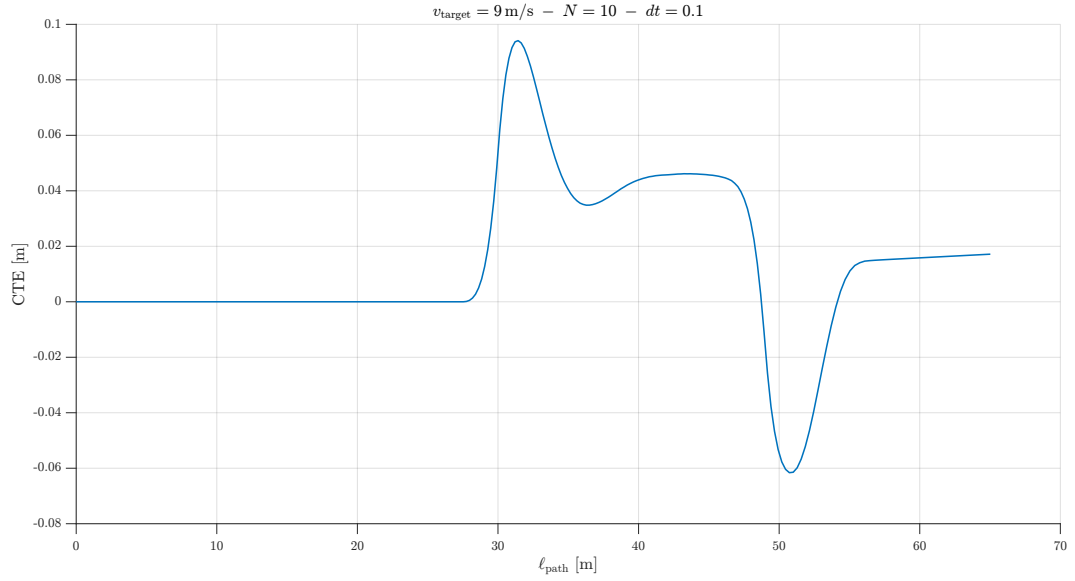


Figure 6.25: MPC – U-shaped path - Cross-Track Error

Cross-Track Error The cross-track error remains extremely low throughout the maneuver, with oscillations limited to about 7–8 cm. The error increases slightly at curve entry, reaching a maximum of approximately +0.09 m, then reverses sign during cornering, settling around -0.07 m before converging back to zero in the final section. This symmetric and well-damped trend indicates a proper balance between the controller’s anticipatory and corrective actions. The vehicle maintains a highly stable and accurate lateral tracking performance, without long-term drift or overshoot.

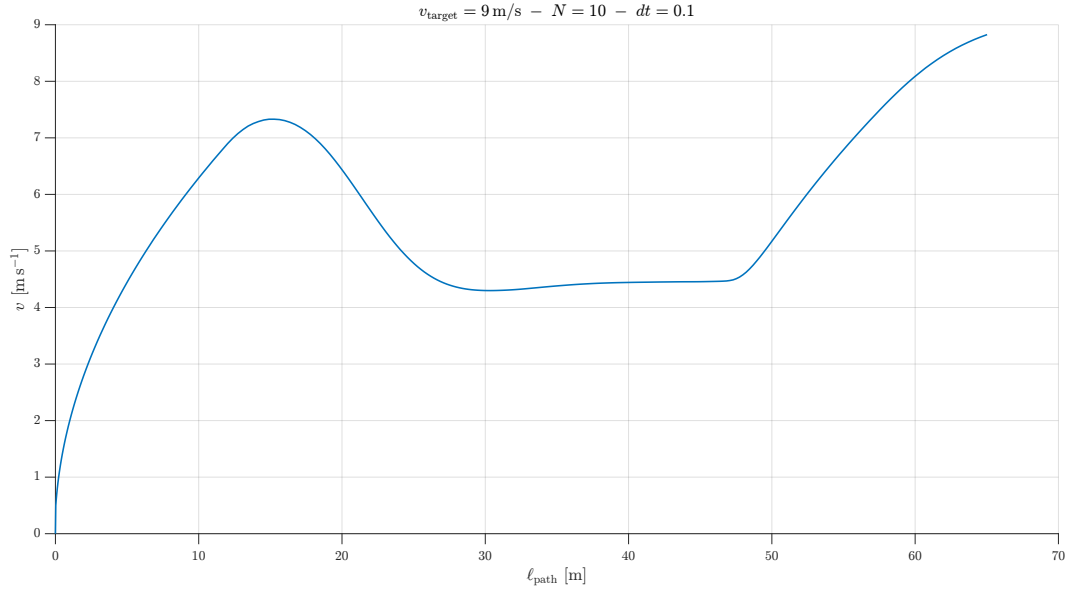


Figure 6.26: MPC – U-shaped path - Longitudinal speed

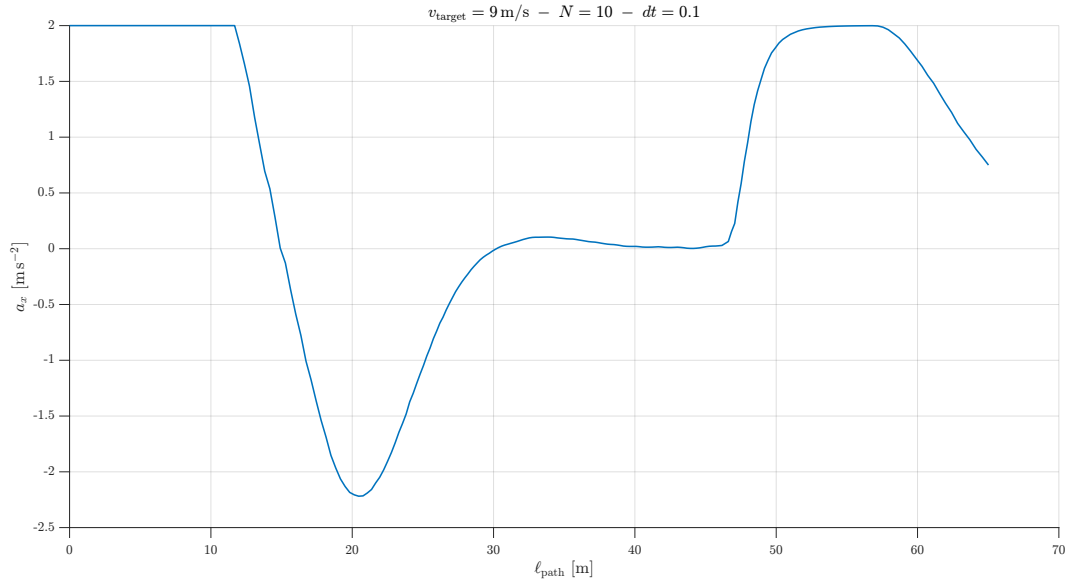


Figure 6.27: MPC – U-shaped path - Longitudinal acceleration

Longitudinal speed and acceleration The longitudinal velocity profile clearly highlights the predictive nature of the MPC. After an initial acceleration phase, the speed rises to about 7.3 m/s within the first 15–20 m of travel, then progressively decreases to around 4 m/s while approaching the sharp bend. This anticipatory

deceleration demonstrates that the controller reacts ahead of time to the curvature variations within the prediction horizon, reducing velocity to maintain constant lateral acceleration. After the curve, the vehicle accelerates smoothly back to the target speed of 9 m/s, with symmetric and continuous transitions free of abrupt changes. The longitudinal acceleration behavior is regular and fully consistent with the velocity trend. It starts from the upper limit of $+2 \text{ m/s}^2$, then decreases steadily to a minimum of about -2.3 m/s^2 during the braking phase preceding the curve. Throughout the central portion of the turn, the acceleration remains close to zero, indicating a dynamically balanced state. In the exit phase, it gradually returns to positive values up to $+2 \text{ m/s}^2$, confirming a controlled and progressive re-acceleration. The longitudinal action is therefore perfectly synchronized with the lateral dynamics, highlighting the MPC's ability to jointly optimize comfort and stability.

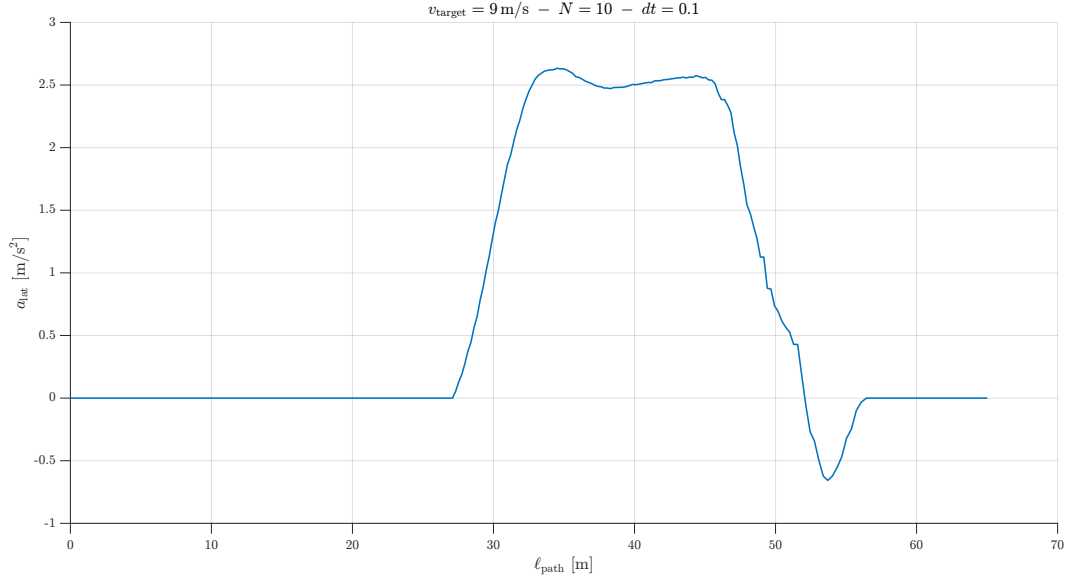


Figure 6.28: MPC – U-shaped path - Lateral acceleration

Lateral Acceleration The lateral acceleration profile closely mirrors the curve dynamics: it increases progressively up to approximately 2.6 m/s^2 , remaining stable throughout the cornering phase before decreasing symmetrically back to zero. The absence of discontinuities or local peaks indicates that the variation of steering is well-coordinated with speed modulation. The fact that the maximum value slightly exceeds the comfort threshold (2.4 m/s^2) shows that the controller exploits the available dynamic margin efficiently, without violating the physical limits of the vehicle.

In summary, the MPC exhibits a highly smooth, stable, and predictive behavior.

The trajectory is continuous and precise, the lateral error minimal, and the coordinated modulation of steering and velocity ensures a realistic and dynamically coherent response. Compared with the other controllers, the MPC provides the best trade-off between geometric accuracy and dynamic comfort, thanks to its joint optimization of steering and acceleration over the prediction horizon.

S-shaped path

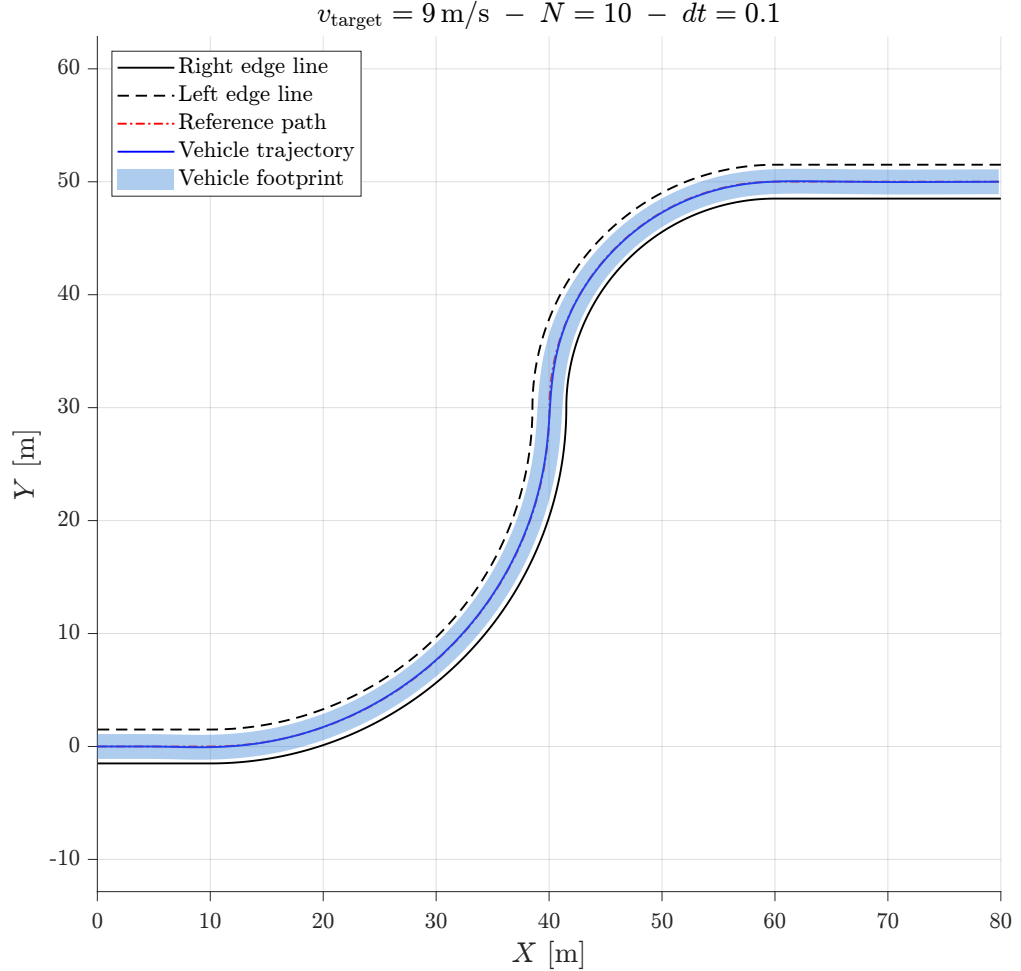


Figure 6.29: MPC – S-shaped path - Trajectory

Trajectory The vehicle trajectory adheres excellently to the reference path along the entire S-shaped curve, maintaining the footprint perfectly within the lane boundaries. The motion appears smooth and continuous, demonstrating the controller’s strong ability to adapt to alternating curvature changes. During the

curvature inversion, around 55–60 m, the vehicle maintains a seamless transition without discontinuities, indicating that the predictive optimization effectively coordinates steering and velocity to ensure kinematic continuity. The absence of overshoot or visible lateral deviations confirms the stability of the planning process and the MPC’s capacity to manage both control domains simultaneously.

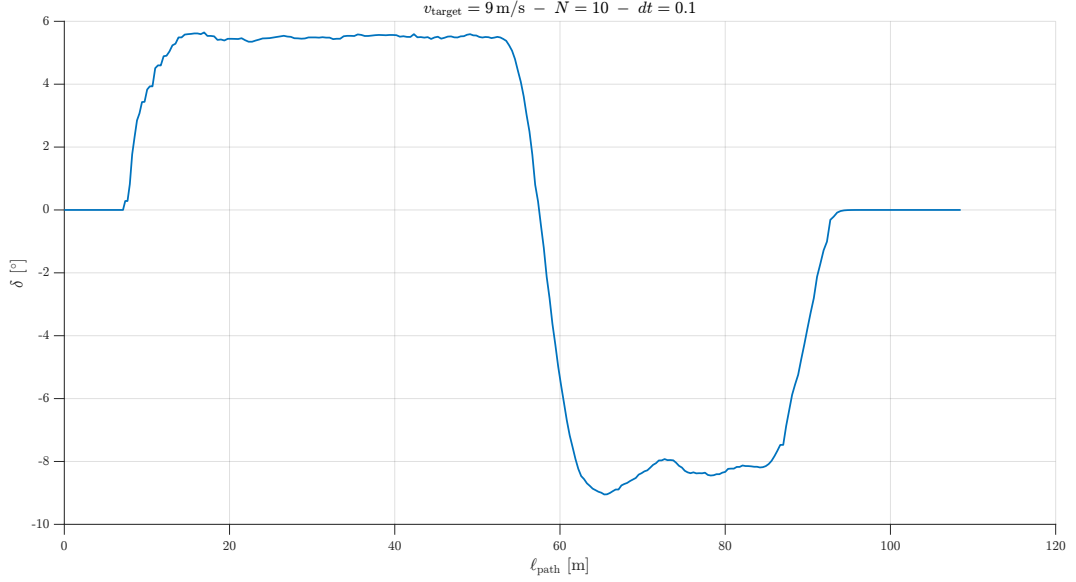


Figure 6.30: MPC – S-shaped path - Steering angle

Steering angle The steering angle profile shows two main variations of opposite sign, corresponding to the two bends of the S-curve. The command increases progressively up to approximately $+5.8^\circ$ during the first left-hand curve, maintaining stability for several meters before reversing direction smoothly and reaching about -8° in the second right-hand curve. The transitions are smooth and symmetric, with the steering crossing zero gradually at the curvature inversion point. The absence of peaks or oscillations confirms the effectiveness of the rate and jerk constraints in smoothing the steering command, while the prompt response highlights the predictive accuracy of the vehicle motion estimation within the horizon.

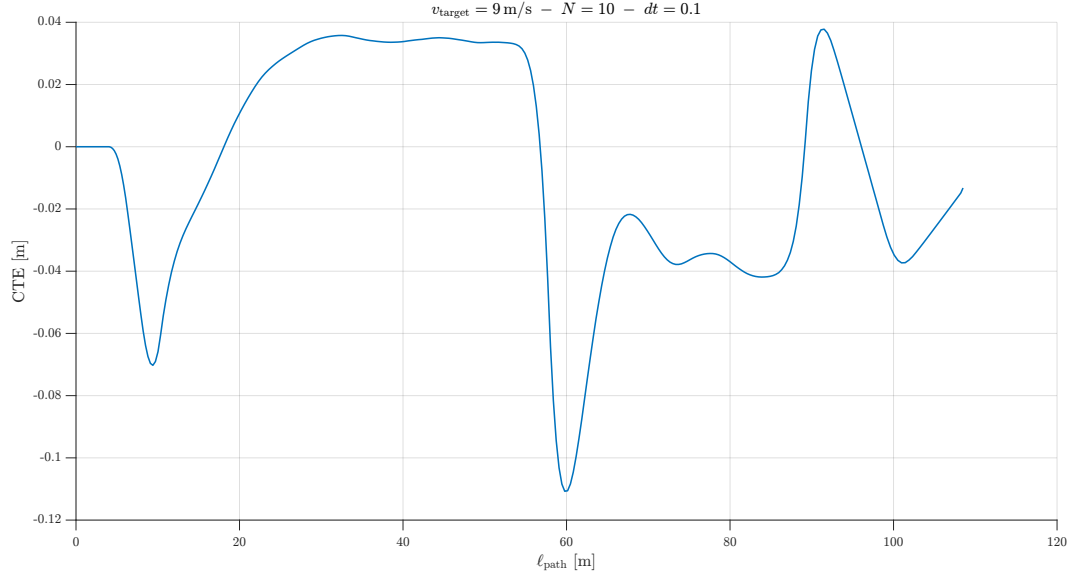


Figure 6.31: MPC – S-shaped path - Cross-Track Error

Cross-Track Error The lateral error remains extremely low throughout the maneuver, with oscillations below ± 0.04 m during cornering and localized peaks not exceeding 0.1 m at curvature transitions. A first minimum of about -0.07 m is observed at the entry of the left-hand curve, followed by a small positive deviation of approximately $+0.03$ m and a second negative peak (around -0.1 m) during the right-hand curve. This symmetric and well-damped behavior highlights the proper balance between the anticipatory and corrective actions of the controller. The vehicle remains centered on the reference path at all times, with no drift or residual offset at the end of the maneuver.

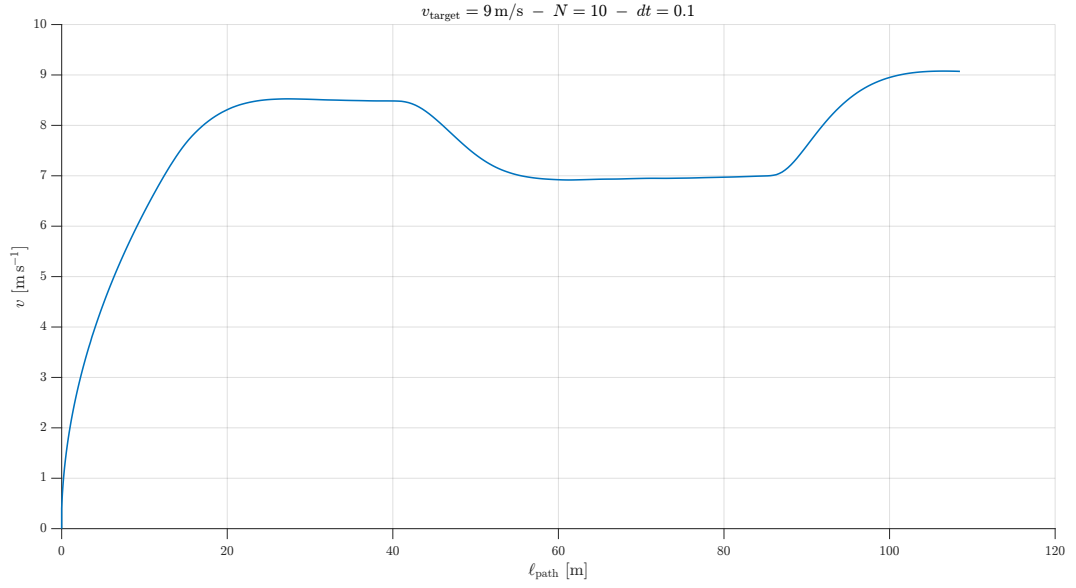


Figure 6.32: MPC – S-shaped path - Longitudinal speed

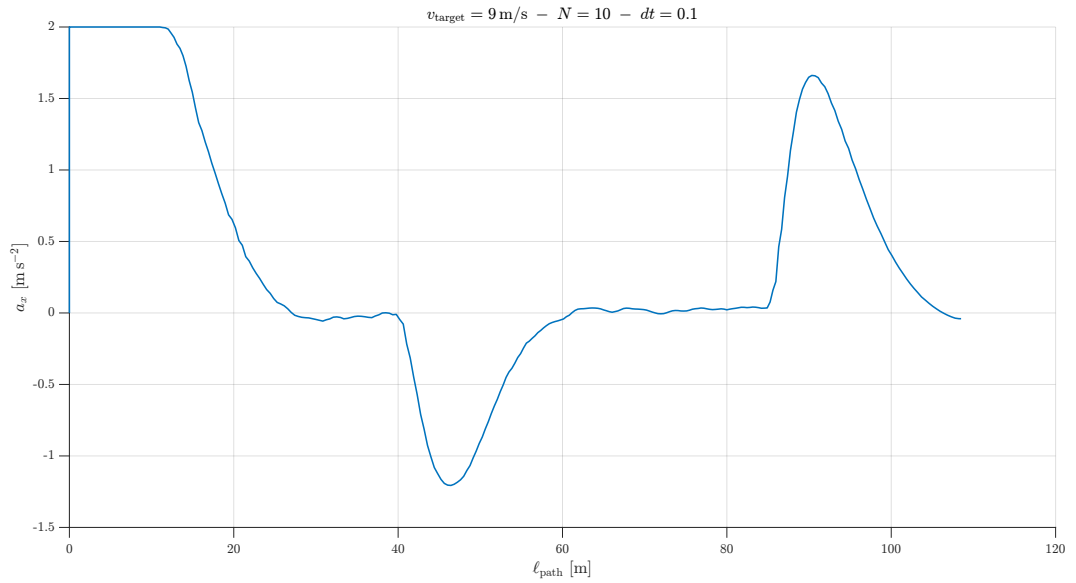


Figure 6.33: MPC – S-shaped path - Longitudinal acceleration

Longitudinal speed and acceleration The longitudinal velocity profile further confirms the predictive behavior of the MPC. After an initial acceleration phase up to approximately 8.5 m/s, the controller progressively reduces speed during the first curve, stabilizing it around 7 m/s. In the central straight segment, the velocity

remains nearly constant, then decreases further to approximately 6.5 m/s during the second curve, which has a smaller radius. After completing the S-shaped section, the vehicle accelerates again smoothly to reach the target speed of 9 m/s. The transitions are perfectly symmetric and continuous, confirming a fluid management of longitudinal dynamics.

The longitudinal acceleration reflects the same dynamic pattern as the velocity profile. Starting at approximately $+2 \text{ m/s}^2$, it gradually decreases to near zero during the first curve and reaches a minimum of around -1.3 m/s^2 in the braking phase. After a brief steady-state region ($a_x \approx 0$), the acceleration becomes positive again in the exit phase, peaking at about $+1.8 \text{ m/s}^2$ as the vehicle regains speed. The signal remains continuous and well-filtered, with no discontinuities, confirming an excellent coordination between longitudinal and lateral control domains. The MPC modulates speed optimally to respect lateral acceleration limits and ensure passenger comfort.

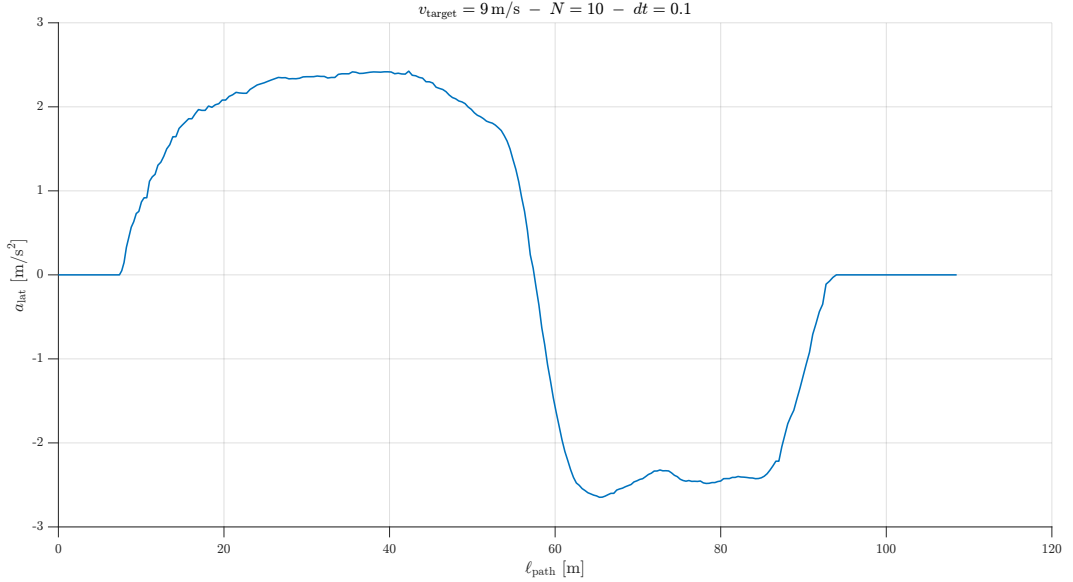


Figure 6.34: MPC – S-shaped path - Lateral Acceleration

Lateral Acceleration The lateral acceleration profile presents two main lobes of opposite sign, corresponding to the two bends of the S-shaped path. In the first section, acceleration increases progressively up to about $+2.4 \text{ m/s}^2$, remaining nearly constant throughout the first curve, then crosses zero smoothly and reaches around -2.5 m/s^2 in the second curve. The slopes are regular and free from oscillations, indicating that steering and velocity are perfectly synchronized. The symmetry of the two lobes and the absence of overshoot confirm an effective control of lateral dynamics: the vehicle consistently operates at the desired limit without

exceeding it. Overall, the lateral dynamics are fully coherent with the kinematic model predictions and the comfort constraints defined in the system.

In summary, the MPC demonstrates highly predictive and harmonious behavior on the S-curve, characterized by precise trajectory tracking, coordinated control action, and continuous response. Its ability to anticipate and handle the effects of opposite curvature variations ensures dynamic stability even during direction reversals, providing smooth, consistent, and well-balanced motion that optimally combines geometric accuracy and dynamic comfort.

6.3 Controllers Comparison

This section presents a comparative analysis of the implemented controllers, evaluating their performance in both lateral and longitudinal domains. The discussion focuses on trajectory tracking accuracy, control smoothness, and dynamic response under different path geometries, in order to highlight the specific strengths and limitations of each approach.

6.3.1 Lateral behaviour

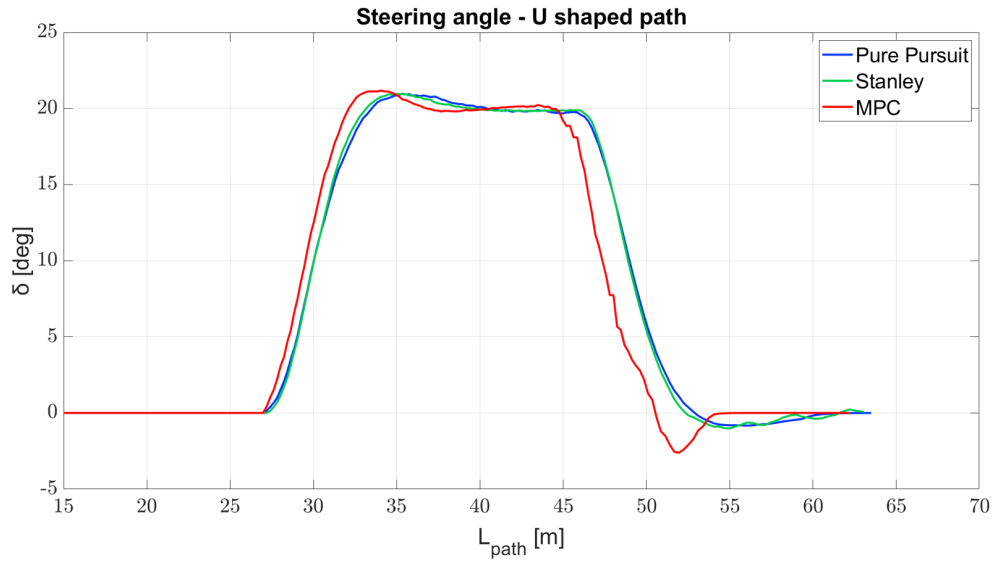


Figure 6.35: Steering angle comparison - U shaped path

In the steering angle plot, all controllers exhibit a response consistent with the geometry of the trajectory, characterized by two sharp transitions corresponding to the low-radius turns. The *Pure Pursuit* and *Stanley* controllers produce smoother

and more symmetric steering profiles, while the *MPC* shows a more anticipatory behavior: the steering angle increases earlier at the curve entry and remains higher throughout the central phase. This behavior originates from the predictive nature of the MPC, which evaluates the future evolution of the vehicle state and anticipates trajectory changes to minimize the global error across the prediction horizon. In the post-turn section, the MPC profile exhibits a rapid inversion of the steering command followed by small compensatory oscillations. This phenomenon is typical of predictive controllers with limited horizons, where the locally optimal solution may appear slightly aggressive during the recovery phase, as the algorithm prioritizes the minimization of the global cost over the smoothness of the actuation. Conversely, the *Pure Pursuit*, based purely on geometric reasoning, performs a more gradual yet delayed correction, whereas the *Stanley* controller, thanks to its proportional term on the lateral error, achieves a balanced response, effectively positioned between the two extremes.

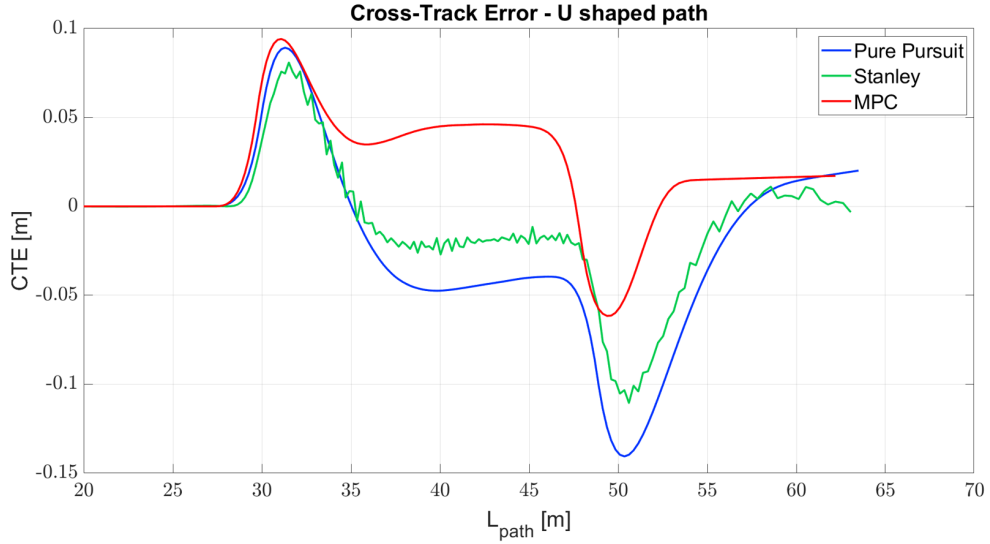


Figure 6.36: Cross-track error comparison - U shaped path

The comparison of the Cross-Track Error (CTE) confirms these observations. The *MPC* maintains the smallest error in the initial and central parts of the curve but exhibits slight instability in the exit phase, where prediction accuracy is affected by residual error accumulation and dynamic constraints. The *Stanley* controller achieves the best overall performance, maintaining a low and stable lateral error along the entire path with a smooth, well-damped trend. In contrast, the *Pure Pursuit* shows larger deviations in tighter turns, indicating a delay in convergence to the desired path, which is typical of purely kinematic formulations.

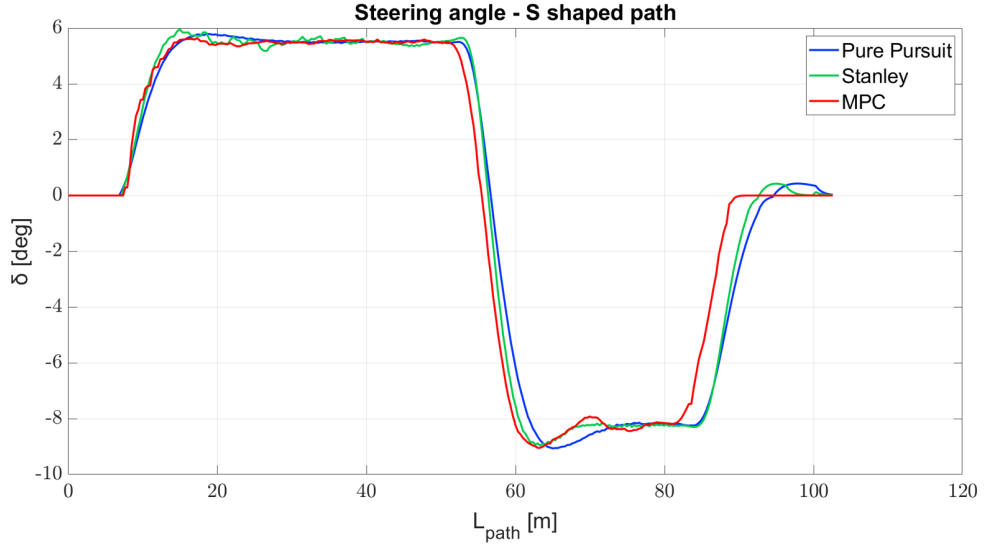


Figure 6.37: Steering angle comparison - U shaped path

For the S-shaped trajectory, all controllers correctly reproduce the alternating curvature of the path, showing two consecutive sign inversions corresponding to the left–right maneuver. The *Pure Pursuit* again exhibits a delayed reaction in both transitions, with smoother but temporally shifted steering profiles. This behavior reflects its purely geometric formulation, which reacts to the current lookahead point rather than predicting upcoming curvature changes. The *Stanley* controller demonstrates an improved dynamic response: it adjusts the steering angle earlier in each transition and maintains a closer alignment between the commanded and desired curvature. Its proportional correction on the cross-track error allows for faster convergence without overshoot, resulting in smooth yet responsive trajectory tracking. The *MPC*, on the other hand, anticipates both directional changes by predicting the vehicle’s future evolution across the horizon. Consequently, the steering command exhibits faster and more symmetric transitions with slightly higher peak amplitudes. This anticipatory nature reduces the delay observed in the other two controllers, though it may introduce small oscillations at curvature inflection points due to the finite prediction horizon and actuation limits.

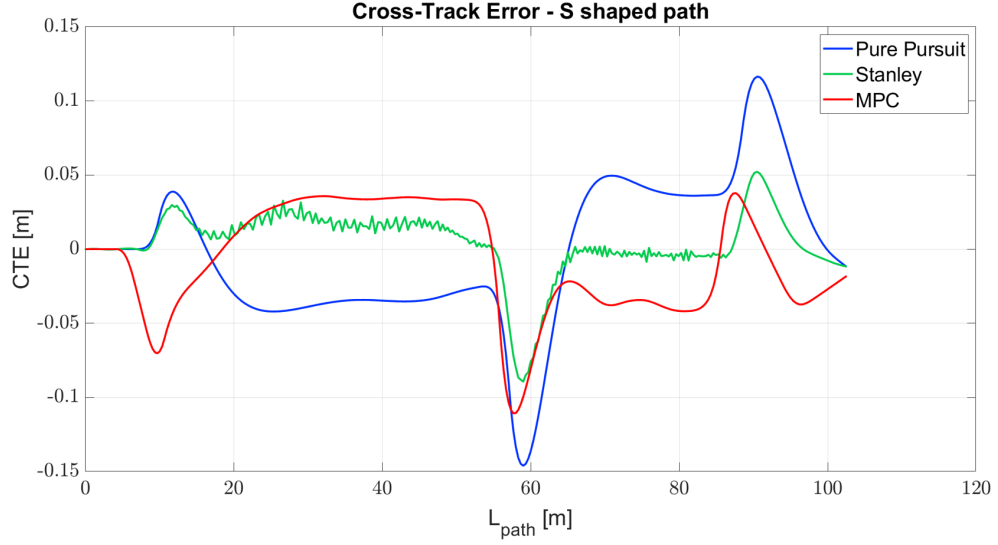


Figure 6.38: Cross-track error comparison - S shaped path

Regarding the Cross-Track Error, the *MPC* achieves the lowest mean lateral error throughout the maneuver, confirming its ability to anticipate and adapt to curvature reversals. However, small oscillations appear during direction changes, revealing the sensitivity of the predictive model to high-frequency dynamics. The *Stanley* controller again provides a stable and well-damped response, maintaining a low error amplitude and avoiding overshoot, while the *Pure Pursuit* accumulates larger deviations at the beginning of each turn, with slower convergence back to the reference centerline.

Overall, the S-shaped trajectory confirms and reinforces the trends already observed in the U-shaped scenario. The *MPC* provides the most accurate and anticipatory trajectory tracking, effectively minimizing the average lateral error by predicting future vehicle states along the horizon. However, this advantage comes at the cost of higher computational complexity and a slightly more aggressive steering response, which can introduce minor oscillations during curvature reversals. The *Stanley* controller achieves the best compromise between accuracy and stability, offering fast convergence and smooth steering corrections without overshoot. Finally, the *Pure Pursuit*, although simple and robust, remains the least precise in highly dynamic or high-curvature segments, where its geometric formulation limits its ability to anticipate rapid directional changes.

6.3.2 Longitudinal behaviour

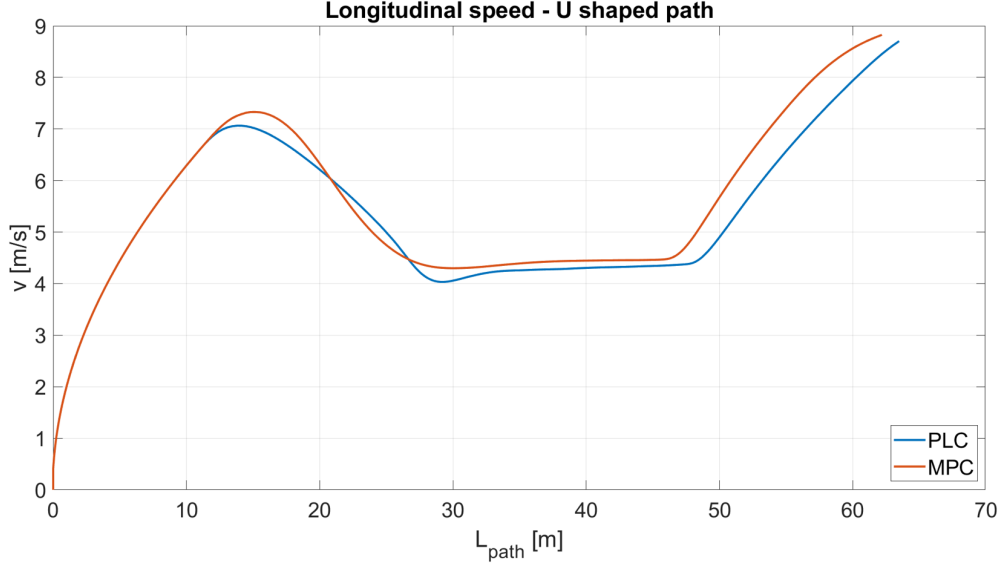


Figure 6.39: Longitudinal speed comparison - U shaped path

In the U-shaped path scenario, both longitudinal controllers allow the vehicle to reach and stabilize around the desired speed profile when approaching the curve. However, some clear differences can be observed in their dynamic response.

The Proportional controller exhibits a more conservative behaviour, characterized by a sharper and slightly delayed deceleration phase. The vehicle maintains a lower speed within the curve, strictly respecting the lateral acceleration limit and ensuring stable but less smooth dynamics. Conversely, the MPC controller shows a smoother and more anticipatory response: the deceleration starts as soon as the curve enters the prediction window of approximately 20 m ahead of the vehicle. This anticipatory behaviour results from the model's predictive capability. The steady-state speed in the curved section is slightly higher, since the MPC handles lateral acceleration as a soft constraint, which can be temporarily exceeded when the optimizer determines that doing so minimizes the overall cost. This flexibility leads to a faster yet stable curve negotiation, maintaining overall comfort and safety.

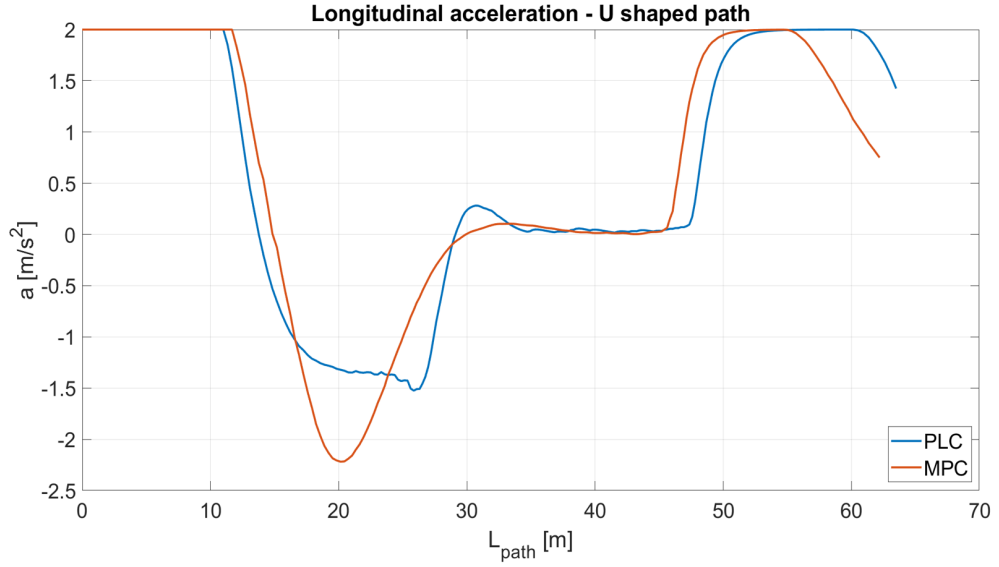


Figure 6.40: Longitudinal acceleration comparison - U shaped path

The longitudinal acceleration profile highlights the different ways in which the two controllers manage transient dynamics.

The PLC exhibits sharper variations: the deceleration phase at the curve entry is more intense and characterized by higher-frequency oscillations, indicating a more reactive and less optimized response in terms of comfort. This behaviour arises because the proportional controller reacts directly to the instantaneous velocity error without anticipating future curvature changes.

In contrast, the MPC distributes the deceleration over a longer horizon, producing a smoother profile with no abrupt peaks. The softer shape of the acceleration curve reflects the influence of the cost function weights, which penalize large variations of control effort and jerk. As a result, the vehicle slows down and accelerates more progressively, improving both comfort and longitudinal stability.

The difference in amplitude between the two curves mirrors their control philosophy: the MPC prefers minor temporary violations of dynamic limits rather than aggressive corrective actions, whereas the PLC enforces stricter but less predictive adjustments.

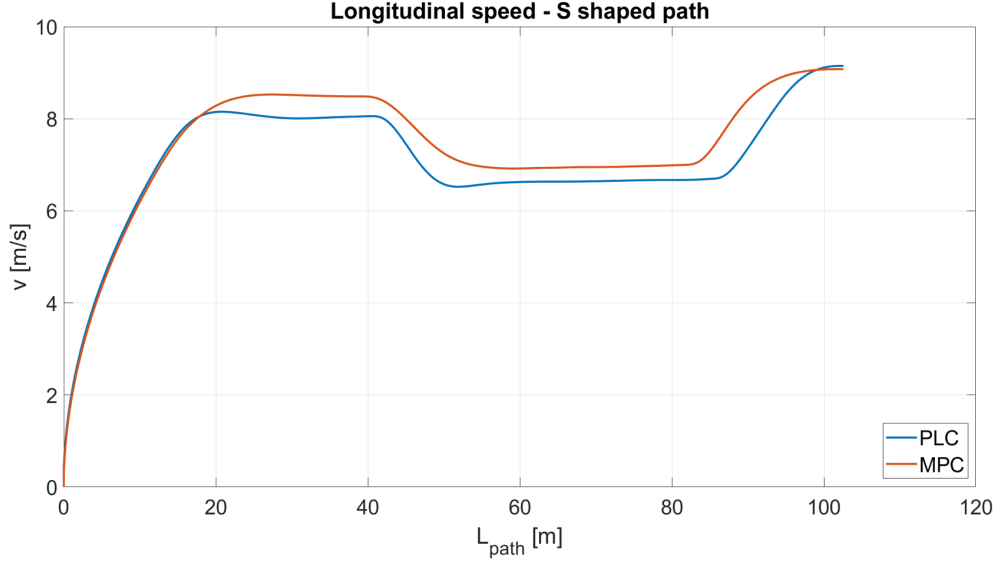


Figure 6.41: Longitudinal speed comparison - S shaped path

For the S-shaped path, which involves two consecutive curvature inversions, the longitudinal dynamics become even more representative of the controllers' differences.

The PLC modulates speed in a more discontinuous way: after the initial deceleration, the vehicle keeps an almost constant velocity but exhibits visible fluctuations around the curvature transition, followed by a relatively abrupt acceleration at the exit. This behaviour reveals the absence of prediction, as the controller reacts locally to curvature variations.

The MPC, instead, anticipates both curvature changes. The deceleration phase is smoother, and the mean speed remains about **0.3–0.5 m/s higher** than that of the PLC. Again, the presence of a soft constraint on lateral acceleration allows the vehicle to maintain slightly higher speeds without compromising stability, dynamically choosing the optimal trade-off between travel time and comfort. The resulting longitudinal profile is smoother, with reduced velocity oscillations between the two curves and a more continuous transition phase.

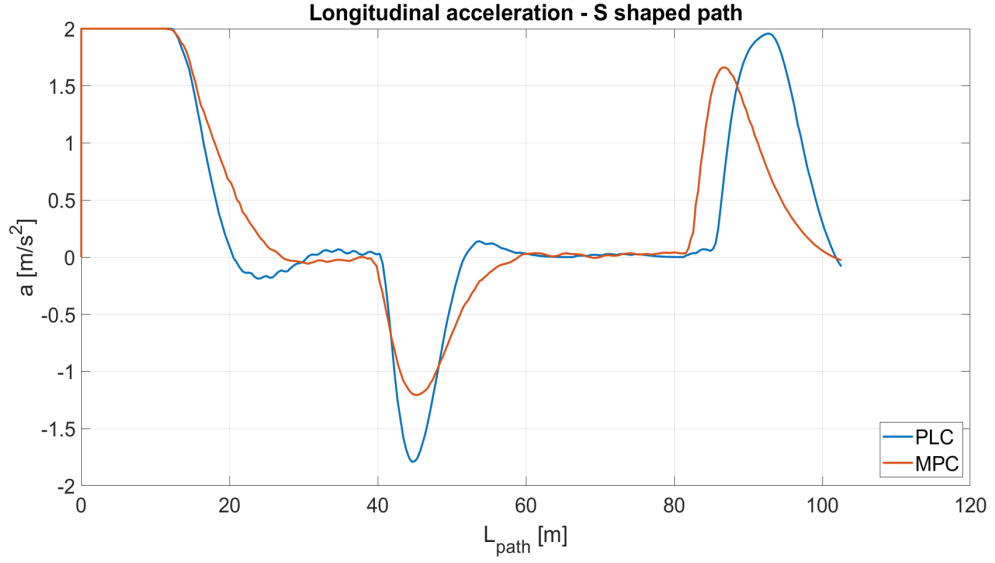


Figure 6.42: Longitudinal acceleration comparison - S shaped path

The MPC, on the other hand, produces a more harmonic profile with smoother transitions and smaller magnitude of acceleration. The acceleration–deceleration phases are symmetric and limited, as the optimization penalizes sudden changes over the prediction horizon. This predictive mechanism leads to a longitudinal response that is globally more balanced and comfortable.

Overall, the MPC provides a smoother and more anticipative behaviour, while the PLC remains stiffer and more conservative. The MPC’s predictive strategy results in milder accelerations, slightly higher cornering speeds, and improved longitudinal comfort.

Chapter 7

Conclusions and Future Developments

Throughout this work, a complete control module for autonomous vehicles has been designed, implemented, and validated, capable of managing both Path Following and Low-Level Control in an integrated and coherent manner. The goal was to create a modular, physically realistic, and reproducible framework in which the main control strategies could be implemented, compared, and evaluated within a ROS 2 environment entirely designed to reproduce real vehicle dynamics. The result is a system where control theory meets engineering practice, enabling an in-depth analysis of how different algorithms affect the vehicle’s stability, smoothness, and comfort.

The comparison between the implemented controllers — Pure Pursuit, Stanley, and Model Predictive Control — revealed substantial differences in both philosophy and performance. The Pure Pursuit algorithm stood out for its simplicity and regularity: it produced smooth, progressive trajectories with a natural and harmonious motion that prioritizes comfort over absolute precision. The Stanley controller, by contrast, exhibited a sharper reactivity and higher adherence to the reference path, especially in tight or high-curvature turns. However, this precision comes with a slightly more nervous behavior on straight segments, as the controller aggressively corrects even minimal lateral deviations. The Model Predictive Control (MPC) approach proved to be the most advanced synthesis: thanks to its predictive and constrained formulation, it anticipates the evolution of the trajectory, optimizes both lateral and longitudinal dynamics in a coordinated way, and ensures continuous compliance with the vehicle’s physical limits. The resulting behavior is stable, fluid, and remarkably human-like, striking a refined balance between accuracy, comfort, and stability.

Quantitative analyses confirmed these tendencies. Pure Pursuit achieved the

lowest jerk values and the smoothest transitions between maneuvers, albeit at the cost of slightly higher lateral error in rapid curvature changes. Stanley minimized cross-track error effectively, maintaining precise alignment with the reference path, though with more noticeable steering oscillations. MPC delivered the best overall performance, achieving the optimal compromise between tracking accuracy, stability, and smooth control effort. Its predictive nature allows the vehicle to anticipate curvature variations, reducing sudden corrections and maintaining a more continuous, predictable response.

From an architectural perspective, the developed framework demonstrated the effectiveness of the proposed modular paradigm. Each ROS 2 node — from trajectory generation to kinematic simulation, from the controller to the data analysis module — retains a clear and independent role, allowing each component to be analyzed and validated in isolation. This organization ensures transparency, scalability, and scientific rigor, making the entire system ready for integration with the physical *eWolf* platform. The framework thus bridges theoretical modeling and practical validation, offering a flexible experimental base for both research and industrial development.

The contribution of this thesis extends beyond the comparative analysis of control strategies: it provides a complete experimental environment that can be expanded and enriched in several directions. Future developments will focus on integrating the control module with perception and planning layers, thus completing the full decision-making chain of the autonomous driving architecture. In the context of the *eWolf* platform, this will translate into the implementation of an obstacle and pedestrian detection system capable of real-time environment recognition and adaptive trajectory correction. By incorporating LiDAR, radar, and camera-based perception, the current path-following framework can evolve into an adaptive control system, dynamically adjusting its reference trajectory in response to environmental conditions.

In this perspective, the control architecture could evolve toward an integrated predictive system in which trajectories are not merely followed but continuously recalculated based on environmental interactions and surrounding agents. Additionally, the adoption of *Machine Learning* and *Reinforcement Learning* techniques could enable automatic tuning of control parameters, allowing the system to learn from driving data and progressively improve its performance and adaptability. On the low-level side, future extensions may include more detailed actuator models and integrated energy management, enabling the controller to account in real time for the electrical and thermal state of the vehicle components. The integration with the Speedgoat hardware will mark the transition from simulation to real-world testing, maintaining the same ROS 2-based communication and control structure developed in this thesis.

In a broader perspective, the proposed platform could become the foundation for

an advanced predictive control framework in which the vehicle continuously plans and adapts its trajectory, anticipating obstacles, negotiating dynamic environments, and collaborating with other road users. This evolution will allow *eWolf* to serve as a real-world testbed for high-level autonomous driving algorithms, contributing to the development of safer, more efficient, and more sustainable mobility solutions.

Ultimately, this thesis demonstrates that the success of autonomous driving does not depend on a single algorithm, but on the synergy between intelligent, physically consistent, and well-integrated modules capable of communicating through a scalable architecture. The work bridges the gap between control theory and experimental validation, offering a concrete contribution to the realization of autonomous vehicles that are not only able to move on their own, but to do so in a way that is smooth, predictable, and deeply human.

Bibliography

- [1] World Health Organization. *Global Status Report on Road Safety 2023*. World Health Organization, 2023. URL: <https://www.who.int/publications/i/item/9789240068643> (cit. on p. 2).
- [2] National Highway Traffic Safety Administration. *Critical Reasons for Crashes Investigated in the National Motor Vehicle Crash Causation Survey*. U.S. Department of Transportation, 2015. URL: <https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812115> (cit. on p. 2).
- [3] European Commission. *EU Road Safety Policy Framework 2021–2030: Next Steps Towards Vision Zero*. European Commission, Directorate-General for Mobility and Transport, 2021. URL: https://transport.ec.europa.eu/transport-themes/road-safety/road-safety-policy-framework-2021-2030_en (cit. on p. 2).
- [4] International Energy Agency. *Global EV Outlook 2023: Catching Up with Climate Ambitions*. International Energy Agency, 2023. URL: <https://www.iea.org/reports/global-ev-outlook-2023> (cit. on p. 2).
- [5] R. Zhang, E. Yao, and J. Hu. «Eco-Driving Control of Connected and Automated Vehicles: Methods, Challenges, and Opportunities». In: *Transportation Research Part C: Emerging Technologies* 114 (2020), pp. 794–816. DOI: 10.1016/j.trc.2020.03.002 (cit. on p. 2).
- [6] McKinsey and Company. *Autonomous Driving Industry Outlook 2023: The Road to 2035*. McKinsey and Company, 2023. URL: <https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/autonomous-driving-road-to-2035> (cit. on p. 2).
- [7] International Road Transport Union. *European Road Transport Report 2023: Professional Driver Shortage*. IRU, 2023. URL: <https://www.iru.org/resources/newsroom/driver-shortages-europe-2023> (cit. on p. 3).

- [8] World Economic Forum. *The Inclusive Mobility Initiative: Making Transportation Accessible for All*. World Economic Forum, 2021. URL: <https://www.weforum.org/projects/the-inclusive-mobility-initiative> (cit. on p. 3).
- [9] M. Elbanhawi and M. Simic. «Advanced Inertial Measurement and Sensor Fusion for Automated Vehicles: A Review». In: *Sensors* 23.12 (2023), p. 5563. DOI: 10.3390/s23125563 (cit. on p. 3).
- [10] NVIDIA Corporation. *NVIDIA DRIVE Platform: AI Computing for Autonomous Vehicles*. 2023. URL: <https://www.nvidia.com/en-us/self-driving-cars/drive-platform/> (cit. on p. 3).
- [11] Open Robotics. *ROS 2 Documentation: Humble and Rolling Distributions*. 2024. URL: <https://docs.ros.org/en/> (cit. on p. 3).
- [12] Forschungsinstitut für Kraftfahrwesen und Fahrzeugmotoren Stuttgart (FKFS). *eWolf Autonomous Driving Research Platform Overview*. FKFS, University of Stuttgart, 2024. URL: <https://www.fkfs.de/en/research/ewolf/> (cit. on pp. 4, 10).
- [13] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli. «A Survey of Motion Planning and Control Techniques for Self-Driving Urban Vehicles». In: *IEEE Transactions on Intelligent Vehicles* 1.1 (2016), pp. 33–55. DOI: 10.1109/TIV.2016.2578706 (cit. on pp. 4, 7, 10).
- [14] General Motors Corporation. *Highways and Horizons (Futurama Exhibit at the 1939 New York World's Fair)*. 1939. URL: https://digital.hagley.org/Futurama_1939 (cit. on p. 5).
- [15] Radio Corporation of America. «Electronic Highway Control of Automobiles». In: *RCA Review* 14.1 (1953), pp. 33–40 (cit. on p. 5).
- [16] State of Nebraska Department of Roads. *Automated Highway Test Segment Using Buried Wire Guidance*. Experimental project report, Nebraska Department of Roads. 1958 (cit. on p. 5).
- [17] E. D. Dickmanns, B. Mysliwetz, and T. Christians. «An Integrated Spatio-Temporal Approach to Automatic Visual Guidance of Autonomous Vehicles». In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 1988, pp. 538–539. DOI: 10.1109/CVPR.1988.196293 (cit. on p. 5).
- [18] A. Broggi, M. Bertozzi, and A. Fascioli. «The ARGO Autonomous Vehicle's Vision and Control Systems». In: *IEEE Transactions on Intelligent Transportation Systems* 3.4 (2002), pp. 401–410. DOI: 10.1109/TITS.2002.806840 (cit. on p. 5).

- [19] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, et al. «Stanley: The Robot That Won the DARPA Grand Challenge». In: *Journal of Field Robotics*. Vol. 23. 9. 2006, pp. 661–692. DOI: 10.1002/rob.20147 (cit. on p. 5).
- [20] C. Urmson et al. «Autonomous Driving in Urban Environments: Boss and the DARPA Urban Challenge». In: *Journal of Field Robotics* 25.8 (2008), pp. 425–466. DOI: 10.1002/rob.20255 (cit. on pp. 5, 7).
- [21] SAE International. *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles, SAE Standard J3016*. 2021. URL: https://www.sae.org/standards/content/j3016_202104/ (cit. on p. 6).
- [22] Tesla Inc. *Occupancy Network and End-to-End Perception for Full Self-Driving*. 2023. URL: <https://www.tesla.com/AI> (cit. on p. 8).
- [23] Waymo LLC. *Waymo Safety Report: The Trusted Driver*. Waymo LLC, 2021. URL: <https://waymo.com/safety/> (cit. on p. 9).
- [24] International Organization for Standardization. *ISO/PAS 21448: Road Vehicles — Safety of the Intended Functionality (SOTIF)*. 2022 (cit. on p. 9).
- [25] Steven E Shladover. «Connected and automated vehicle systems: Introduction and overview». In: *Journal of Intelligent Transportation Systems* 22.3 (2018), pp. 190–200 (cit. on p. 12).
- [26] Rajesh Rajamani. *Vehicle Dynamics and Control*. 3rd. New York: Springer, 2020. ISBN: 978-1-0716-0689-9. DOI: 10.1007/978-1-0716-0689-9 (cit. on pp. 25, 31, 37).
- [27] Jarrod M. Snider. *Automatic Steering Methods for Autonomous Automobile Path Tracking*. Tech. rep. CMU-RI-TR-09-08. Pittsburgh, PA: Carnegie Mellon University, Robotics Institute, 2009. URL: <https://repository.cmu.edu/robotics/26> (cit. on pp. 29, 30, 33, 37).
- [28] R. Craig Coulter. *Implementation of the Pure Pursuit Path Tracking Algorithm*. Tech. rep. CMU-RI-TR-92-01. Pittsburgh, PA: Carnegie Mellon University, Robotics Institute, 1992. URL: <https://repository.cmu.edu/robotics/549> (cit. on pp. 29, 33).
- [29] Sebastian Thrun et al. «Stanley: The Robot That Won the DARPA Grand Challenge». In: *Journal of Field Robotics* 23.9 (2006), pp. 661–692. DOI: 10.1002/rob.20147 (cit. on pp. 29, 37).
- [30] Paolo Falcone, Francesco Borrelli, Jamshid Asgari, H. Eric Tseng, and Davor Hrovat. «Predictive Active Steering Control for Autonomous Vehicles». In: *IEEE Transactions on Control Systems Technology* 15.3 (2007), pp. 566–580. DOI: 10.1109/TCST.2007.894653 (cit. on pp. 43, 46, 47).

- [31] James B. Rawlings and David Q. Mayne. *Model Predictive Control: Theory and Design*. Madison, WI: Nob Hill Publishing, 2009. ISBN: 978-0-9759377-0-9 (cit. on pp. 43, 44).
- [32] David Q. Mayne. «Model predictive control: Recent developments and future promise». In: *Automatica* 50.12 (2014), pp. 2967–2986. DOI: 10.1016/j.automatica.2014.10.128 (cit. on pp. 44, 47).
- [33] A. Marouf, J.C. Fauroux, F. Plestan, and A. Glumineau. «Control of an Electric Power Assisted Steering system using reference model». In: *Mechatronics* 21.1 (2011), pp. 239–250. DOI: 10.1016/j.mechatronics.2010.09.003 (cit. on p. 53).
- [34] Pavan B. Kandula. «Dynamics and Control of an Electric Power Assist Steering System». MA thesis. Cleveland State University, 2010. URL: <https://engagedscholarship.csuohio.edu/etdarchive/412/> (cit. on p. 53).
- [35] D. Lee. «A Novel Electric Power Steering (EPS) Control Algorithm Development and Reference-Steering-Feel Tracking». In: *SAE Technical Paper 2016-01-1546*. 2016. DOI: 10.4271/2016-01-1546 (cit. on p. 53).
- [36] Y. Jang, T. Kim, and J. Lee. «User Evaluation of Comfortable Deceleration Profiles for Highly Automated Vehicles». In: *Transportation Research Part C* 162 (2024), p. 104194. DOI: 10.1016/j.trc.2024.104194 (cit. on p. 54).
- [37] H. B. Pacejka. *Tire and Vehicle Dynamics*. 3rd. Elsevier, 2012 (cit. on p. 54).
- [38] *ISO 26262:2018 – Road vehicles – Functional safety*. International Organization for Standardization, 2018 (cit. on p. 55).
- [39] *ISO 15622:2019 – Intelligent transport systems – Adaptive Cruise Control systems – Performance requirements and test procedures*. International Organization for Standardization, 2019 (cit. on p. 55).
- [40] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. «ROS: an open-source Robot Operating System». In: *ICRA Workshop on Open Source Software*. 2009, pp. 1–6 (cit. on p. 58).
- [41] Yuya Maruyama, Shinpei Kato, and Takuya Azumi. «Exploring the performance of ROS2». In: *Proceedings of the 13th International Conference on Embedded Software (EMSOFT)*. 2016, pp. 1–10. DOI: 10.1145/2968478.2968502 (cit. on p. 59).
- [42] Steven Macenski, Tully Foote, Brian Gerkey, Christophe Lalancette, and William Woodall. «Robot Operating System 2: Design, architecture, and uses in the wild». In: *Science Robotics* 7.66 (2022), eabm6074. DOI: 10.1126/scirobotics.abm6074 (cit. on pp. 59, 62).

- [43] Gerardo Pardo-Castellote, William Warren, and Joe Marrero. «An Introduction to DDS and the Data-Centric Paradigm». In: *Proceedings of the Real-Time Innovations Conference on Distributed Systems*. 2021 (cit. on pp. 59, 62).
- [44] Tully Foote. «tf: The transform library». In: *IEEE International Conference on Technologies for Practical Robot Applications (TePRA)*. 2013, pp. 1–6. DOI: 10.1109/TePRA.2013.6556373 (cit. on p. 62).