Real-Time autonomous racing: the artificial race driver including initial contributions to shared control via the artificial race coach

Italian National Ph.D. Program in Autonomous Systems

ACADEMIC DISCIPLINE: SYSTEMS AND CONTROL ENGINEERING (IINF-04/A)

Final Dissertation

Real-Time Autonomous Racing: the Artificial Race Driver

Including Initial Contributions to Shared Control via the Artificial Race Coach

---

by

Sebastiano Taddei

Sebastiano Taddei

Administrative Headquarters:

Politecnico di Bari – Department of Electrical and Information Engineering

Hosting University:

University of Trento – Department of Industrial Engineering

Referees:

Prof. Johannes Betz

Prof. Basilio Lenzo

Supervisors:

Prof. Francesco Biral

Francesco Biral

Prof. Gastone Pietro Rosati Papini

Gastone Pietro Rosati Papini

Coordinator of Ph.D Program

Prof. Mariagrazia Dotoli

Course no. 38, 01/11/2022-31/10/2025

Politecnico di Bari

DAUSY

LIBERATORIA PER L'ARCHIVIAZIONE DELLA TESI DI DOTTORATO

Al Magnifico Rettore
del Politecnico di Bari

Il/la sottoscritto/a Sebastiano Taddei nato/a a Modena (MO) il 01/04/1998
residente a Borgo Valsugana (TN) in via Alessandro Spagolla 6 e-mail sebastianotaddei@gmail.com
iscritto al 3° anno di Corso di Dottorato di Ricerca in Autonomous Systems (DAUSY) ciclo 38
ed essendo stato ammesso a sostenere l'esame finale con la prevista discussione della tesi dal titolo:
Real-Time Autonomous Racing: the Artificial Race Driver, Including Initial Contributions to Shared Control via the Artificial Race Coach

DICHIARA

1) di essere consapevole che, ai sensi del D.P.R. n. 445 del 28.12.2000, le dichiarazioni mendaci, la falsità negli atti e l'uso di atti falsi sono puniti ai sensi del codice penale e delle Leggi speciali in materia, e che nel caso ricorressero dette ipotesi, decade fin dall'inizio e senza necessità di nessuna formalità dai benefici conseguenti al provvedimento emanato sulla base di tali dichiarazioni;
2) di essere iscritto al Corso di Dottorato di ricerca Autonomous Systems (DAUSY) ciclo 38, corso attivato ai sensi del "*Regolamento dei Corsi di Dottorato di ricerca del Politecnico di Bari*", emanato con D.R. n.286 del 01.07.2013;
3) di essere pienamente a conoscenza delle disposizioni contenute nel predetto Regolamento in merito alla procedura di deposito, pubblicazione e autoarchiviazione della tesi di dottorato nell'Archivio Istituzionale ad accesso aperto alla letteratura scientifica;
4) di essere consapevole che attraverso l'autoarchiviazione delle tesi nell'Archivio Istituzionale ad accesso aperto alla letteratura scientifica del Politecnico di Bari (IRIS-POLIBA), l'Ateneo archivierà e renderà consultabile in rete (nel rispetto della Policy di Ateneo di cui al D.R. 642 del 13.11.2015) il testo completo della tesi di dottorato, fatta salva la possibilità di sottoscrizione di apposite licenze per le relative condizioni di utilizzo (di cui al sito http://www.creativecommons.it/Licenze), e fatte salve, altresì, le eventuali esigenze di "embargo", legate a strette considerazioni sulla tutelabilità e sfruttamento industriale/commerciale dei contenuti della tesi, da rappresentarsi mediante compilazione e sottoscrizione del modulo in calce (Richiesta di embargo);
5) che la tesi da depositare in IRIS-POLIBA, in formato digitale (PDF/A) sarà del tutto identica a quelle **consegnate**/inviate/da inviarsi ai componenti della commissione per l'esame finale e a qualsiasi altra copia depositata presso gli Uffici del Politecnico di Bari in forma cartacea o digitale, ovvero a quella da discutere in sede di esame finale, a quella da depositare, a cura dell'Ateneo, presso le Biblioteche Nazionali Centrali di Roma e Firenze e presso tutti gli Uffici competenti per legge al momento del deposito stesso, e che di conseguenza va esclusa qualsiasi responsabilità del Politecnico di Bari per quanto riguarda eventuali errori, imprecisioni o omissioni nei contenuti della tesi;
6) che il contenuto e l'organizzazione della tesi è opera originale realizzata dal sottoscritto e non compromette in alcun modo i diritti di terzi, ivi compresi quelli relativi alla sicurezza dei dati personali; che pertanto il Politecnico di Bari ed i suoi funzionari sono in ogni caso esenti da responsabilità di qualsivoglia natura: civile, amministrativa e penale e saranno dal sottoscritto tenuti indenni da qualsiasi richiesta o rivendicazione da parte di terzi;
7) che il contenuto della tesi non infrange in alcun modo il diritto d'Autore né gli obblighi connessi alla salvaguardia di diritti morali od economici di altri autori o di altri aventi diritto, sia per testi, immagini, foto, tabelle, o altre parti di cui la tesi è composta.

Luogo e data Trento, 15/01/2025 Firma Sebastiano Taddei

Il/La sottoscritto, con l'autoarchiviazione della propria tesi di dottorato nell'Archivio Istituzionale ad accesso aperto del Politecnico di Bari (POLIBA-IRIS), pur mantenendo su di essa tutti i diritti d'autore, morali ed economici, ai sensi della normativa vigente (Legge 633/1941 e ss.mm.ii.),

CONCEDE

- al Politecnico di Bari il permesso di trasferire l'opera su qualsiasi supporto e di convertirla in qualsiasi formato al fine di una corretta conservazione nel tempo. Il Politecnico di Bari garantisce che non verrà effettuata alcuna modifica al contenuto e alla struttura dell'opera.
- al Politecnico di Bari la possibilità di riprodurre l'opera in più di una copia per fini di sicurezza, back-up e conservazione.

Luogo e data Trento, 15/01/2025 Firma Sebastiano Taddei

Sebastiano Taddei

Real-Time Autonomous Racing: the Artificial Race Driver

Including Initial Contributions to Shared Control via the Artificial Race Coach

Thesis submitted for the degree of Philosophiae Doctor

Italian National Ph.D. Program in Autonomous Systems
University of Trento and Politecnico di Bari

Tutors
Prof. *Francesco Biral*
Prof. *Gastone Pietro Rosati Papini*

2025

The doctoral scholarship was funded by the European Union - Next Generation EU, Mission 4 Component 1 CUP D93C22000500001.

Dissertation submitted for the degree of *Philosophiae Doctor*
Italian National Ph.D. Program in Autonomous Systems

Cycle:
38th

Administrative Headquarters:
Politecnico di Bari

Hosting University:
University of Trento

Title:
Real-Time Autonomous Racing: the Artificial Race Driver
Including Initial Contributions to Shared Control via the Artificial Race Coach

Ph.D Candidate:
Sebastiano Taddei, University of Trento (Trento, Italy), Politecnico di Bari (Bari, Italy)

Tutors:
Prof. Francesco Biral, University of Trento (Trento, Italy)
Prof. Gastone Pietro Rosati Papini, University of Trento (Trento, Italy)

Coordinator:
Prof. Engr. Mariagrazia Dotoli, Politecnico di Bari (Bari, Italy)

External Reviewers:
Prof. Dr.-Ing. Johannes Betz, Technical University of Munich (Munich, Germany)
Prof. Basilio Lenzo, University of Padua (Padua, Italy)

Last version:
January 11, 2026

Abstract

Autonomous racing is a highly challenging domain that pushes the limits of vehicle planning and control to achieve minimum lap times under extreme conditions. This thesis addresses these challenges by introducing the *Artificial Race Driver* (ARD), a hierarchical framework for time-optimal motion planning and control at the limits of handling. At the top level, ARD employs an *Economic Nonlinear Model Predictive Control* (E-NMPC) augmented with a physics-based lateral speed prediction model. At the low level, steering commands are executed by a physics-informed neural network controller, which embeds vehicle dynamics priors into a feedforward structure. Separate longitudinal and lateral feedback controllers are identified to correct the commanded longitudinal acceleration and steering angle. These modules are integrated with a detailed neural vehicle model acting as a digital twin, enabling closed-loop identification and testing under realistic dynamics.

Among the thesis's key contributions is a two-lap learning strategy. In this approach, ARD uses telemetry from one lap to perform the entire identification procedure. It then validates its identification on the telemetry of a second lap (preferably performed on a different track or track configuration). This enables rapid adaptation to new circuits and vehicles with minimal prior data. The work also explores shared-control concepts through an *Artificial Race Coach* (ARC) framework—an experimental parallel system that can interact with a human driver. Inspired by studies showing that autonomous coaching can improve driver performance, ARC is designed to blend the autonomous planner with human input, illustrating the potential of interactive human–machine collaboration in racing. (ARC remains exploratory, highlighting a novel avenue rather than a completed solution.)

ARD is evaluated through comprehensive simulation and real-world experiments. A custom high-fidelity driving simulator serves as a virtual testbed for closed-loop evaluation on multiple tracks and vehicles. In simulation, ARD's trajectories and lap times are benchmarked against offline optimal-control solutions to assess the validity of the proposed method. The framework is then deployed on a real autonomous vehicle to validate real-time performance on track. Results show that ARD can execute aggressive, time-optimal manoeuvres safely and reliably, achieving lap times close to the theoretical minimum. In summary, this thesis demonstrates that combining model-based planning, physics-informed learning, and a detailed neural vehicle model enables an artificial driver to drive at the limits of handling even with limited data.

Godo in sovrappiù a provarci che farcela
Leonardo da Vinci

Contents

Acronyms viii

Preface ix

List of Papers Written by the Author ix

Introduction

1 Introduction 2
- 1.1 Motivation and Contributions . . . 2
  - 1.1.1 Thesis Outline . . . 3
- 1.2 Background . . . 4
  - 1.2.1 Autonomous Driving . . . 4
  - 1.2.2 Autonomous Racing . . . 5
  - 1.2.3 Interpretable Machine Learning . . . 6
- 1.3 Autonomous Driving Literature Review . . . 7
  - 1.3.1 Offline Time-Optimal Vehicle Trajectory Planning . . . 7
  - 1.3.2 Online Time-Optimal Vehicle Trajectory Planning . . . 9
  - 1.3.3 Vehicle Trajectory Tracking . . . 11
  - 1.3.4 End-to-End Approaches for Time-Optimal Autonomous Driving . . . 13
  - 1.3.5 Interpretable Machine Learning for Autonomous Driving . 15
- 1.4 Shared Control Literature Review . . . 16
  - 1.4.1 Shared Control in the Context of Autonomous Driving and Racing . . . 16
  - 1.4.2 From Assistance to Coaching: Shifting the Focus to Driver Development . . . 17
  - 1.4.3 Approaches to shared control . . . 17
  - 1.4.4 Control and Interaction Design . . . 19
  - 1.4.5 Lessons from the Literature and Implications for ARC . . 21

Part I—The Artificial Race Driver Framework

2 System Overview and Design 24
- 2.1 Introduction . . . 24
- 2.2 System Architecture . . . 25
  - 2.2.1 High-Level Motion Planner . . . 26
  - 2.2.2 Low-Level Control Layer . . . 28
  - 2.2.3 Digital Twin and Real-Vehicle Interface . . . 28
  - 2.2.4 Integration with the Learning Framework . . . 28
  - 2.2.5 Communication Architecture and Runtime Infrastructure 29
  - 2.2.6 Simulation and Integration Testing Environment . . . 30
- 2.3 Engineering Decisions and Real-Time Trade-Offs . . . 30
  - 2.3.1 Modular vs. Monolithic Architecture . . . 30
  - 2.3.2 Broker-Centric Communication: Flexibility vs. Centralization 32
  - 2.3.3 Sync vs. Async Execution: Determinism vs. Responsiveness 33
  - 2.3.4 Choosing Custom Middleware over ROS . . . 34

3 Motion Planning 35
- 3.1 Introduction . . . 35

3.2 Problem Formulation . . . 36
3.2.1 Planning Objective and Cost Function . . . 37
3.2.2 Vehicle Modelling for Planning . . . 39
3.2.3 Constraints . . . 41
3.3 Implementation Details . . . 43
3.3.1 Solver Integration . . . 43
3.3.2 Warm-Starting and Horizon Management . . . 44
3.4 On the Formulation and Effects of g-g-v Constraints . . . 45
3.4.1 Motivation and Role in Planning . . . 45
3.4.2 Structure and Hybrid Formulation . . . 46

4 Vehicle Control 47
4.1 Introduction . . . 47
4.2 Longitudinal Control . . . 48
4.2.1 Motivation and Requirements . . . 48
4.2.2 Acceleration-Based Correction Controller . . . 48
4.2.3 Implementation and Real-World Usage . . . 48
4.3 Lateral Control . . . 49
4.3.1 Motivation and Requirements . . . 49
4.3.2 Low-Level Feedforward Steering Controller . . . 49
4.3.3 Low-Level Feedback Steering Controller . . . 55

5 Digital Twin Vehicle Model 57
5.1 Introduction . . . 57
5.2 High-Fidelity Double-Track Model . . . 58
5.2.1 Modelling Overview . . . 58
5.2.2 Role in Simulation and Benchmarking . . . 58
5.2.3 Validation on Real-World Data . . . 59
5.2.4 Challenges in Model Identification . . . 60
5.3 Neural Vehicle Model as a Lightweight Alternative . . . 62
5.3.1 Motivation and Scope . . . 62
5.3.2 Internal Structure and Modelling Assumptions . . . 62
5.3.3 Comparative Identification Results . . . 70

6 Learning to Race from Minimal Telemetry Data 73
6.1 Introduction . . . 73
6.2 Data Acquisition and Preprocessing . . . 74
6.2.1 Two-Lap Telemetry Collection . . . 76
6.2.2 Signal Preprocessing: Filtering and Corrections . . . 76
6.2.3 Computation of Derived Quantities . . . 77
6.3 Identification of the Kineto-Dynamical Model for E-NMPC . . . 77
6.3.1 Construction of the g-g-v Diagram . . . 78
6.3.2 Extraction of Lateral Velocity Characteristics . . . 80
6.3.3 Identification of First-Order Dynamics . . . 80
6.4 Low-Level Controller Identification . . . 81
6.4.1 Training the PhS-NN Lateral Controller . . . 81
6.4.2 Considerations for Longitudinal Control Identification . . . 82
6.5 Closed-Loop Identification . . . 83
6.5.1 Training the Neural Vehicle Model . . . 83
6.5.2 Closed-Loop Identification and Corrective Control . . . 85
6.6 On Error Detection and Interpretability of Physics-Informed Models 87

Part II—Simulation and Real-World Testing

7 Test Setup 91
7.1 Introduction . . . 91

7.2 Chosen Tracks . . . 92
7.2.1 UniBW Test Track . . . 92
7.2.2 Circuit de Barcelona—Catalunya . . . 92
7.2.3 UniBW Short Test Track . . . 92
7.3 Test Vehicles . . . 93
7.3.1 Volkswagen eCrafter . . . 94
7.3.2 Racecar . . . 94
7.4 Test Setup . . . 94

8 Evaluation in Simulation 96
8.1 Introduction . . . 96
8.2 MLT vs MPC . . . 97
8.2.1 Volkswagen eCrafter . . . 98
8.2.2 Racecar . . . 98
8.3 MPC vs CL-SYNC . . . 102
8.3.1 Volkswagen eCrafter . . . 103
8.3.2 Racecar . . . 103
8.4 CL-SYNC vs CL-ASYNC . . . 104
8.4.1 Volkswagen eCrafter . . . 104
8.4.2 Racecar . . . 104
8.5 Computation Times . . . 105
8.6 Evaluation under Non-Ideal Conditions . . . 114
8.7 Driving Style Biasing . . . 116

9 Real-World Deployment 120
9.1 Introduction . . . 120
9.1.1 Experimental Constraints and Limitations . . . 120
9.2 Conducted Experiments . . . 121
9.3 Discussion . . . 131

Part III—Early Work in Shared Control

10 The Artificial Race Coach 134
10.1 Introduction . . . 134
10.2 From Autonomous Driving to Shared Control . . . 135
10.2.1 Overview of Shared Control Paradigms . . . 135
10.2.2 Control Interface Assumptions . . . 135
10.3 ARC Architecture . . . 136
10.4 Control Allocation through Smoothstep-Based Guidance . . . 137
10.5 Preliminary Tests and Observations . . . 138
10.6 Discussion and Outlook . . . 139

Conclusions

11 Conclusions 142
11.1 Summary of Contributions . . . 142
11.2 Reflections on Autonomous Racing and Human-In-The-Loop Systems 143
11.3 Future Research Directions . . . 144

Appendices 146

A Driving Simulator 147
A.1 Introduction . . . 147
A.2 Hardware and Peripheral Setup . . . 148
A.3 Custom Software Stack . . . 149
A.3.1 Architecture Overview . . . 149
A.3.2 Core Services . . . 150

A.3.3 System Management and Hardware Integration 153
A.3.4 Simulation Backend 156
A.3.5 Development Utilities 158

Acronyms

ABS *Anti-lock Braking System.* 58, 94, 121

ARC *Artificial Race Coach.* ii, 2, 134, 143, 147

ARD *Artificial Race Driver.* ii, 2, 24, 35, 47, 57, 73, 91, 96, 120, 134, 142, 147

CoG *Centre of Gravity.* 62, 76

DoF *Degrees of Freedom.* 58, 148

E-NMPC *Economic Nonlinear Model Predictive Control.* ii, 2, 26, 35, 73, 105, 122, 142

ESP *Electronic Stability Programme.* 94, 121

HD *Handling Diagram.* 50

IL *Imitation Learning.* 13

LVD *Lateral Velocity Diagram.* 40, 80

MDK *Curvature-weighted Mean Deviation.* 118

MLP *Multi-Layer Perceptron.* 64

MLT *Minimum Lap Time.* 7, 59, 86, 96

MPC *Model Predictive Control.* 10

NMPC *Nonlinear Model Predictive Control.* 9

NN *Artificial Neural Network.* 12

NVM *Neural Vehicle Model.* 57, 73, 96, 123, 143

OCP *Optimal Control Problem.* 7, 37, 57, 77

PhS-NN *Physics-Informed Steering Neural Network.* 49, 73, 142

PID *Proportional-Integral-Derivative.* 3, 28, 47, 83

QSS *Quasi Steady-State.* 7

RL *Reinforcement Learning.* 6

RMSD *Root-Mean-Squared Deviation.* 118

ROS *Robotic Operating System.* 26

UniBW *Universität der Bundeswehr München.* 28, 45, 48, 63, 76, 91, 98, 124, 143

VMS *Vehicle Model used as a Simulator.* 59

Preface

This thesis is submitted in partial fulfilment of the requirements for the degree of *Philosophiae Doctor* in Autonomous Systems from the National Ph.D. program DAUSY.

The full list of papers written by the author is reported hereafter.

List of Papers Written by the Author

[1] **S. Taddei**, P. Gonnella, G. Fioraso, G. De Vecchi, M. Darin, F. Biral. "Local Misbehavior Detection for V2X-Enabled Autonomous Agents Leveraging Motion Primitives and Deep Learning on Real-World Data". *(to be submitted)*, 2025.

[2] **S. Taddei**, M. Piccinini, F. Biral. "Biasing the Driving Style of an Artificial Race Driver for Online Time-Optimal Maneuver Planning". *IEEE Intelligent Vehicles Symposium (IV)*, 2025. DOI: https://doi.org/10.1109/IV64158.2025.11097381

[3] M. Piccinini, **S. Taddei**, J. Betz, F. Biral. "Kineto-Dynamical Planning and Accurate Execution of Minimum-Time Maneuvers on Three-Dimensional Circuits". *IEEE International Conference on Robotics and Automation (ICRA)*, 2025. DOI: https://doi.org/10.1109/ICRA55743.2025.11127446

[4] M. Piazza, M. Piccinini, **S. Taddei**, F. Biral, E. Bertolazzi. "Real-Time Velocity Profile Optimization for Time-Optimal Maneuvering With Generic Acceleration Constraints". *IEEE Robotics and Automation Letters (RA-L)*, 2026. DOI: https://doi.org/10.1109/LRA.2025.3643297

[5] D. Fruet, **S. Taddei**, S. Cimignolo, F. Biral, G. Nollo. "Synchronized data acquisition system for behavioral analysis during simulated driving". *IX Congress of the National Group of Bioengineering (GNB 2025)*, 2025. Available: https://www.gnb2025.it/proceedings.html

[6] M. Piccinini, **S. Taddei**, E. Pagot, E. Bertolazzi, F. Biral. "How optimal is the minimum-time manoeuvre of an artificial race driver?". *Vehicle System Dynamics (VSD)*, 2024. DOI: https://doi.org/10.1080/00423114.2024.2407176

[7] M. Piccinini, **S. Taddei**, M. Piazza, F. Biral. "Impacts of g-g-v Constraints Formulations on Online Minimum-Time Vehicle Trajectory Planning". *IFAC Symposium on Control of Transportation Systems (CTS)*, 2024. DOI: https://doi.org/10.1016/j.ifacol.2024.07.323

[8] M. Piazza, M. Piccinini, **S. Taddei**, F. Biral. "MPTree: A Sampling-Based Vehicle Motion Planner for Real-Time Obstacle Avoidance". *IFAC Symposium on Control of Transportation Systems (CTS)*, 2024. DOI: https://doi.org/10.1016/j.ifacol.2024.07.332

[9] **S. Taddei**, F. Visintainer, F. Stoffella, F. Biral. "Multi-layered Local Dynamic Map for a Connected and Automated In-Vehicle System". *MDPI Sustainability*, 2024. DOI: https://doi.org/10.3390/su16031306

[10] M. Piccinini, **S. Taddei**, M. Larcher, M. Piazza, F. Biral. "A Physics-Driven Artificial Agent for Online Time-Optimal Vehicle Motion Planning and Control". *IEEE Access*, 2023. DOI: https://doi.org/10.1109/ACCESS.2023.3274836

Acknowledgements

I am deeply grateful to my supervisor, Francesco Biral, for his guidance and support throughout my PhD, and to my co-supervisor, Gastone Pietro Rosati Papini, for his invaluable advice and encouragement. I would also like to thank Enrico Bertolazzi for his inspiring teaching and mentorship, which have greatly influenced my approach to research. My sincere thanks go to my colleagues, in particular Mattia Piccinini, my former mentor; Matteo Larcher, one of the most capable engineers I have ever met; Mattia Piazza, for sharing the challenges of our PhD journey from the very beginning; and Davide Stocco for his collaboration and support. Finally, I am thankful to my mum for her constant support, to my girlfriend for her patience and encouragement, and to my friends and family who have always been there for me during this journey.

Disclaimer: AI tools were used to refine grammar and improve readability.

Introduction

Chapter 1

Introduction

Autonomous driving represents one of the hardest challenges in modern control engineering and artificial intelligence. While significant progress has been made in ensuring safety, comfort, and robustness for urban autonomous vehicles, achieving high-performance driving near the handling limits of a vehicle remains an open and complex problem. Autonomous racing, where a vehicle must travel around the track in the minimum possible time, offers a compelling research environment for developing and testing algorithms that operate under extreme conditions. Beyond its competitive nature, this domain also provides valuable insights that can translate into safer and more reliable control strategies for everyday autonomous driving. It brings together the demands of online trajectory planning, real-time control, system identification, and interaction with uncertain environments—all at the edge of feasibility.

This thesis focuses on the development of a complete system for autonomous racing, built around a framework named the *Artificial Race Driver* (ARD). ARD is designed to compute and execute time-optimal manoeuvres using a combination of *Economic Nonlinear Model Predictive Control* (E-NMPC) and learning-based control. By operating within this performance-critical domain, the work presented in this thesis not only advances the capabilities of autonomous vehicles in racing, but might also offer insights and methods applicable to broader contexts such as emergency manoeuvres.

In addition to developing ARD, this thesis also addresses the challenge of rapid adaptation to new circuits and vehicles through a minimal-data learning strategy, and explores early efforts in shared control between human drivers and autonomous systems via the *Artificial Race Coach* (ARC).

Through a combination of theoretical modelling, extensive simulation, and real-world deployment, the work aims to contribute a comprehensive, experimentally validated approach for autonomous high-performance driving.

1.1 Motivation and Contributions

The ability to drive at the handling limits of a vehicle is traditionally reserved for expert human drivers, particularly in the context of motorsports. Translating this capability to autonomous systems presents a unique and demanding challenge. It requires the integration of high-fidelity vehicle modelling, real-time optimal control, and robust decision-making under uncertainty. While the primary testbed for such systems is often the racetrack, the motivation for this research extends beyond competitive driving alone.

High-performance autonomous driving has the potential to significantly advance the safety and responsiveness of autonomous vehicles in general settings. Emergency manoeuvres in urban or highway environments—such as obstacle avoidance, evasive braking, or rapid lane changes—require precise execution under time and space constraints. These scenarios demand the same control precision and situational awareness that autonomous racing systems are designed to master. By pushing the boundaries of vehicle performance in a structured environment, autonomous racing serves as a fertile ground for developing algorithms and architectures that may ultimately improve the safety of everyday autonomous transportation.

This thesis addresses the challenge by proposing ARD: a hierarchical framework that combines time-optimal planning, physics-informed control, and realistic simulation into a cohesive system for autonomous racing. The design prioritizes adaptability, modularity, and data efficiency, with the goal of building a generalizable artificial driver capable of performing with minimal prior knowledge of the track, environment, and vehicle.

Hereafter a comprehensive overview of the main contributions of this thesis.

Design and Implementation of ARD: A modular, real-time framework is developed for autonomous racing, consisting of:

- A high-level motion planner based on E-NMPC, incorporating a physics-based lateral speed prediction model.
- A low-level lateral controller using a physics-informed neural network that embeds vehicle dynamics priors into a feedforward structure.
- Two low-level learned *Proportional-Integral-Derivative* (PID) corrective controllers for lateral and longitudinal dynamics.
- A neural vehicle model that serves as a digital twin for high-fidelity simulation and closed-loop identification.

Development of a Two-Lap Learning Strategy: A novel learning approach enables ARD to infer all driving-relevant properties of a vehicle from a single lap of telemetry, then validate the identification on a second lap—preferably performed on a different track or track configuration. This strategy allows for rapid adaptation to new circuits and vehicles with minimal data, supporting broader deployment with reduced calibration requirements.

Comprehensive Simulation and Real-World Validation: The system is rigorously evaluated in simulation, using a custom high-fidelity driving simulator, and in real-world experiments on an autonomous vehicle. The simulation analysis extends beyond ideal conditions, introducing packet loss and measurement noise to assess robustness under realistic operating scenarios. This investigation provides new insight into the system's behaviour in the presence of communication delays and sensing imperfections. Results demonstrate the system's ability to execute time-optimal trajectories safely and with performance close to theoretical limits.

Exploratory Work on Shared Control: The thesis also introduces ARC, a framework designed to explore shared control and human-in-the-loop strategies in racing. ARC provides an early demonstration of how autonomous planning can blend with human driving styles, laying the foundation for future work in cooperative and assistive driving.

Together, these contributions advance the state of the art in autonomous racing and highlight its relevance to both high-performance applications and future urban driving systems.

1.1.1 Thesis Outline

This thesis is structured into three main parts, each progressively building upon the development, validation, and extension of ARD.

Part I: the Artificial Race Driver Framework

- Chapter 2 gives an overview of ARD's system design. It introduces the architecture and modular composition of the framework, highlighting the principles that guide its real-time performance and its suitability for aggressive, time-optimal driving.
- Chapter 3 presents ARD's high-level motion planning module, centred on E-NMPC. It details the time-optimal trajectory planning process and the role of dynamic constraints, including the incorporation of 3D road geometry.
- Chapter 4 describes ARD's low-level vehicle controllers. This component relies on a physics-informed neural network to translate planned trajectories into steering actions, as well as learned PID corrective controllers.
- Chapter 5 covers the digital twin neural vehicle model that supports ARD's planning and control stack. This simulated model is used to train control components, test planning strategies, and validate system-level performance.

- Chapter 6 introduces a two-lap learning strategy that enables ARD to rapidly adapt to unknown circuits and vehicles using minimal telemetry data. This data-efficient approach significantly reduces the identification and setup burden for deploying ARD on new tracks and vehicles.

Part II: Simulation and Real-World Testing

- Chapter 7 details both the simulation and real-world test setups.
- Chapter 8 evaluates ARD in a high-fidelity simulation environment, benchmarking it against theoretical baselines to assess its optimality.
- Chapter 9 presents the real-world deployment of ARD. It outlines the integration steps, safety mechanisms, and results from on-track experiments, demonstrating ARD's capacity to execute aggressive driving manoeuvres reliably and safely outside of simulation.

Part III: Early Work in Shared Control

- Chapter 10 introduces ARC: an experimental companion framework designed to explore shared control between an autonomous system and a human driver. While still in early stages, ARC demonstrates the potential for exploiting ARD's planning capabilities to improve the human driver performance.

Finally, Chapter 11 concludes the thesis by summarizing the contributions, reflecting on ARD's development and deployment, and outlining promising directions for future research, both in autonomous racing and its application to broader driving contexts. An appendix is also included, detailing the driving simulator infrastructure that supports much of the simulation-based work.

1.2 Background

1.2.1 Autonomous Driving

Over the past decade, autonomous driving has emerged as a major focus of research and development in both academia and the automotive industry [1], [2], [3], [4]. The motivation behind this technological shift is multi-faceted: reducing traffic accidents caused by human error, improving traffic efficiency, and enabling more productive or comfortable use of travel time for vehicle occupants.

Human error is responsible for approximately 90% of all traffic accidents [5], with common causes including misjudgments in planning, misinterpretation of the environment, and driver distraction. In response to this public safety concern, autonomous vehicles are expected to play a significant role in reducing both the frequency and severity of road accidents. As part of its "Vision Zero" initiative [6], the European Union has committed to halving road fatalities by 2030 and eliminating them entirely by 2050. These ambitious goals further underscore the societal importance of developing reliable and safe autonomous driving systems.

Despite rapid progress in sensing, planning, and control technologies, several challenges still limit the widespread deployment of autonomous vehicles. These challenges span regulatory, ethical, and societal dimensions, but from a technical standpoint, the most critical issues involve the design of robust and interpretable decision-making systems capable of operating safely in complex, uncertain, and dynamic environments.

As a result, autonomous driving continues to be one of the most active and fast-evolving research areas in robotics and control. The rapid growth in scientific publications over the past decade reflects this momentum—particularly after 2016, when the number of papers indexed under "autonomous driving" increased nearly fourteenfold by 2024. This upward trend has only accelerated in recent years, with publication counts rising by approximately 1000 papers per year since 2022. Figure 1.1 illustrates recent trends in the Scopus database for key topics including "autonomous driving", "autonomous racing",

"interpretable machine learning", "explainable machine learning", and "explainable neural networks".

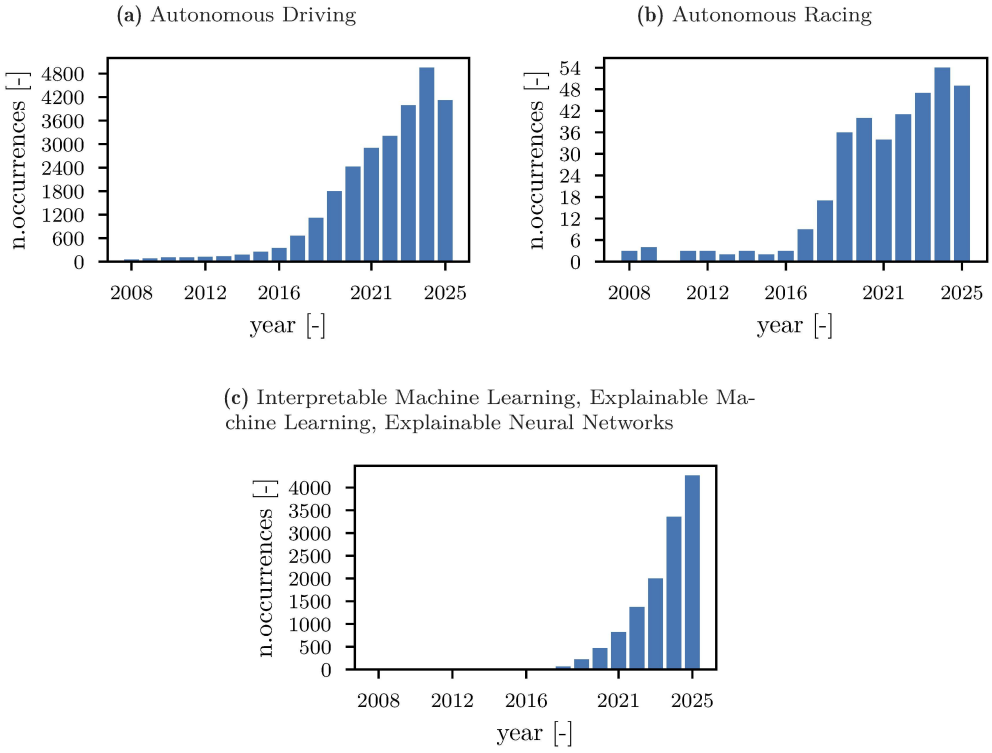


Figure 1.1: Recent trends in the Scopus database for key topics including "autonomous driving", "autonomous racing", "interpretable machine learning", "explainable machine learning", and "explainable neural networks".

1.2.2 Autonomous Racing

In much the same way that Formula 1 has historically served as a testbed for innovations later adopted in consumer vehicles, autonomous racing [7] has emerged as a valuable experimental platform for advancing high-performance driving algorithms. Conducted within the safety of closed circuits, autonomous racing enables the development and testing of decision-making, planning, and control strategies under extreme conditions that would be unsafe or infeasible to reproduce on public roads.

Autonomous racing introduces a unique set of challenges to the field of autonomous driving. Vehicles operate at high speeds and in proximity to their physical handling limits, where non-linearities in vehicle dynamics become pronounced. Under these conditions, traditional conservative planning and control strategies fall short. Instead, systems must rapidly generate and adapt time-optimal trajectories, manage tight actuation constraints, and remain robust in the face of uncertainty or disturbances. These requirements demand sophisticated, tightly coupled planning and control pipelines capable of operating in real-time.

Compared to urban or highway autonomous driving, autonomous racing calls for more aggressive and anticipatory behaviour. It relies on accurate dynamic models, fast solvers, and predictive techniques capable of handling complex and adversarial interactions. While algorithms designed for autonomous racing can often be generalized to more conventional driving contexts, the reverse is not always true—many standard approaches lack the responsiveness and precision needed in performance-critical environments. As such, autonomous racing has become a proving ground for advanced methods that may eventually benefit safety-critical applications such as emergency manoeuvring.

Interest in autonomous racing has grown rapidly over the past years. Despite being a relatively niche research area compared to mainstream autonomous driving, its momentum is visible through a surge of competitions and benchmark platforms. Events such as Formula Student Driverless [8], F1TENTH [9], Roborace [10], the Indy Autonomous Challenge [11], and the AbuDhabi Autonomous Racing League [12] have encouraged the development of standardized racing scenarios. These competitions have recently begun incorporating more complex, adversarial formats, where multiple autonomous agents must compete under real-time constraints, further pushing the boundaries of control and strategy.

By embracing the demanding nature of the racing domain, researchers aim not only to push the state of the art in autonomy but also to extract methods that are transferable to broader driving contexts. In particular, those involving evasive or emergency manoeuvres near the limits of handling.

1.2.3 Interpretable Machine Learning

In recent years, deep learning and deep *Reinforcement Learning* (RL) have gained significant traction in the field of autonomous driving. These methods have been applied to a wide range of tasks [13], including perception [14], motion planning [15], prediction [16], control [17], and even end-to-end autonomous driving [18]. Leveraging large datasets and high-capacity models, deep neural networks have demonstrated impressive capabilities in approximating complex functions and learning behaviour from data. Within the context of autonomous racing, learning-based approaches have also been explored for tasks such as dynamics modelling [19], [20], [21], [22], [23], control [17], [24], [25], and high-speed policy learning [26], [27].

Despite these advancements, the widespread adoption of deep learning in safety-critical domains like autonomous driving remains limited by several challenges. Chief among these is the issue of interpretability. Deep neural networks are often regarded as black-box models, whose internal representations are difficult to understand, analyse, or validate. This lack of transparency poses a significant obstacle when such models are used in control loops where trust, accountability, and robustness are paramount. Moreover, general-purpose architectures typically require large and diverse datasets to generalize well to novel scenarios [28]—an impractical requirement in domains where data is expensive, safety-critical, or scenario-specific.

To address these issues, a growing body of research has focused on interpretable machine learning [29]—the design of models and learning procedures that are more transparent, structured, or physically grounded. In contrast to post-hoc explainability methods, which aim to interpret decisions made by opaque models, interpretable learning seeks to embed structure and prior knowledge into the model itself. This includes techniques such as modular architectures, physics-informed neural networks, and hybrid model-based/data-driven approaches that encode known system dynamics into the learning process.

Part of this thesis follows a research direction that our group began exploring in 2019, ahead of what later became a broader trend in the field: the use of physics-informed neural networks for control and vehicle dynamic modelling. These architectures incorporate domain-specific priors and constraints into the design of the network, enabling improved data efficiency, better generalization, and enhanced interpretability. By embedding physical knowledge directly into the network's structure, the resulting models remain consistent with known vehicle behaviour and produce physically grounded predictions even outside the distribution of the training data, while being easier to analyse and integrate within a real-time autonomous driving stack.

The growing importance of interpretable machine learning is reflected in the recent increase in related publications, see Figure 1.1c. As the demand for reliable, transparent, and certifiable AI systems continues to rise—particularly in regulated fields such as automotive safety—it is expected that interpretable learning will become a central component of next-generation autonomous systems.

1.3 Autonomous Driving Literature Review

1.3.1 Offline Time-Optimal Vehicle Trajectory Planning

Minimum Lap Time (MLT) simulations have long been used to evaluate the performance limits of vehicles [30], [31]. These simulations serve as a powerful benchmarking tool across various applications, such as race car setup optimization [32], [33], energy management in motorsports [34], optimal torque allocation for electric powertrains [35], vehicle model validation [36], and parametric sensitivity analysis [37]. More recently, MLT simulation techniques have also been applied to the generation of time-optimal reference trajectories for autonomous vehicles [7], [24], [38], [39], [40].

A substantial body of literature highlights the use of *Optimal Control Problem* (OCP) methods as particularly effective for addressing MLT problems. Surveys in the field [7], [30], [31], [41], [42], [43] emphasize OCP's flexibility in handling nonlinear dynamics, non-convex cost functions, and complex constraint sets, including path boundaries, control limits, and physical vehicle limitations. This capability has made OCP one of the most widely adopted approaches for offline trajectory optimization in racing and performance-oriented driving.

The following sections present an overview of the main formulations and methods used for solving MLT problems in an offline setting. Specifically:

- Section 1.3.1.1 covers *Quasi Steady-State* (QSS) optimization techniques, which simplify the vehicle dynamics for computational efficiency;
- Section 1.3.1.2 focuses on optimal control formulations, following established taxonomies;
- Section 1.3.1.3 discusses forward-backward integration methods and other hybrid or alternative strategies used in offline MLT simulations.

1.3.1.1 Quasi Steady-State Methods

Some of the earliest approaches to MLT vehicle simulations employed QSS techniques [44], [45], [46]. Foundational works in this area focused on optimizing a vehicle's speed profile along a predefined or previously computed path, while neglecting transient dynamic effects. These early methods simplified the problem considerably, making them suitable for rapid analysis, albeit with limited physical fidelity.

At the core of QSS techniques is the use of the g-g diagram [47], a representation of the maximum longitudinal and lateral accelerations a vehicle can achieve on flat terrain. This diagram can be extended to improve accuracy under varying operating conditions. For instance:

- The g-g-v envelope [24], [25], [40], [48], [49] introduces velocity dependence into the acceleration limits, accounting for aerodynamic and tire effects;
- The more recent g-g-g-v [50], [51], [52], [53], [54] representation models the effects three-dimensional track topologies.

These acceleration envelopes can be derived in several ways:

- Through offline optimization procedures [44], [52], [55], [56] or optimal control problems [57], which solve for the vehicle's maximum achievable accelerations under physical and environmental constraints;
- Or by fitting the envelopes to empirical data or simulation outputs [24], [25], [34], [40], [49], using experimental measurements or synthetic data from high-fidelity models.

While QSS-based methods offer computational simplicity and are still widely used in performance studies and design optimization, their primary limitation lies in their neglect of transient vehicle dynamics. This simplification can result in solutions that are

physically infeasible or overly optimistic when evaluated against full dynamic models. To address this, recent efforts [48], [50], [58] have proposed free-trajectory QSS methods, where the race line itself is optimized in addition to the speed profile. These approaches typically use simplified kinematic models to explore trajectory flexibility while retaining some computational advantages of QSS.

Although inherently approximate, QSS techniques remain relevant for exploratory studies, concept validation, and initial condition generation for more complex optimization pipelines.

1.3.1.2 Optimal Control-based Methods

A more detailed and physically accurate approach to MLT simulations involves the formulation and offline solution of OCP problems using full-vehicle transient dynamic models. In contrast to quasi steady-state methods, these formulations capture the complete time-dependent behaviour of the vehicle, including suspension dynamics, load transfer, and transient responses to steering, throttle, and braking inputs. Notable examples in the literature [32], [33], [36], [59], [60], [61], [62], [63] have successfully applied this methodology to MLT problems using complex vehicle models, including multibody representations and high-order nonlinear systems.

A comprehensive overview of free-trajectory MLT-OCP studies can be found in [31], which catalogue a wide range of formulations, vehicle models, and optimization techniques.

Despite their accuracy and modelling richness, these high-fidelity OCPs present several challenges that limit their practical application:

- **Computational Cost:** These problems are inherently large-scale and computationally intensive. As such, they can only be solved offline, making them unsuitable for real-time applications.
- **Numerical Convergence:** Achieving convergence often requires careful problem-specific tuning of solver parameters. Performance is sensitive to the initial guess, problem scaling, and the choice of discretization methods.
- **Practical Feasibility:** The resulting time-optimal trajectories may be difficult to execute, either by a human driver or an autonomous system. Small deviations from the planned trajectory can lead to unstable or infeasible behaviour, as optimality typically hinges on exploiting narrow performance margins.

These limitations, particularly regarding real-time feasibility and execution robustness, underscore the gap between theoretical optimality and deployable solutions. While OCP-based methods remain essential tools for generating performance benchmarks and understanding vehicle limits, their direct use in onboard planning and control systems is constrained by their computational demands.

1.3.1.3 Approximate Methods to Speed Up the Solution of MLT Problems

As discussed in the previous sections, both QSS and OCP methods are widely used for offline MLT trajectory optimization. However, when high-fidelity vehicle models are involved, these methods can become computationally demanding, making them unsuitable for fast or repeated execution—particularly in real-time or iterative design workflows.

To overcome these limitations, the past decade has seen the development of approximate methods aimed at reducing the computational burden of solving MLT problems. These methods typically focus on two common simplifications:

- Fixing the path and optimizing the speed profile along it;
- Sequentially optimizing the path geometry and speed profile in a decoupled and iterative manner.

A popular class of these techniques relies on forward-backward integration of the vehicle dynamics, where the optimal speed profile is computed by propagating the vehicle state forward and backward in time while enforcing acceleration limits, aerodynamic drag, and powertrain constraints.

Several notable works illustrate this approach:

- One method [64] proposes a two-step iterative scheme that alternates between updating a minimum-curvature path via convex optimization and computing a time-optimal speed profile through forward-backward integration. This approach enables efficient refinement of both path and velocity in sequence.
- In another formulation [39], a quadratic optimization problem (QP) method is used to compute the optimal path curvature, followed by speed profile optimization through forward-backward integration techniques.
- A different strategy [65] employs a mono-dimensional QSS formulation, where the speed profile is iteratively refined to account for combined acceleration envelopes, aerodynamic effects, and powertrain limitations. This method retains the simplicity of QSS models while incorporating key physical constraints.

In parallel, researchers have also explored semi-analytical solutions for simplified MLT problems. For instance, one approach [66] derives closed-form expressions for the minimum-time speed profile of a vehicle constrained to follow a clothoid segment, under acceleration bounds. This work was later extended [67] to generic path geometries, resulting in a highly efficient forward-backward solver suitable for time-optimal speed planning under physical constraints. Finally, a recent work extended previous approaches to generic acceleration constraints [68].

Due to their computational efficiency, these approximate methods are often employed in applications where fast trajectory generation is essential. While they may sacrifice some optimality or generality, they offer valuable trade-offs in speed and simplicity—especially when used as initializations for more accurate optimization routines, or in the context of online planning with dynamic obstacle avoidance [69].

1.3.2 Online Time-Optimal Vehicle Trajectory Planning

The MLT methods discussed in the previous section are primarily designed for offline trajectory optimization. Due to their high computational complexity—especially when using high-fidelity vehicle models—these approaches are generally unsuitable for real-time deployment in autonomous driving applications.

This section shifts focus to the online setting, analysing the literature on real-time, time-optimal trajectory planning for autonomous vehicles. The goal is to generate executable, dynamically feasible trajectories at runtime, while operating under stringent computational and physical constraints.

Within the broader spectrum of motion planning techniques surveyed in the literature [7], [42], [70], [71], OCP-based approaches have emerged as particularly well-suited for solving minimum-time trajectory planning problems. These methods directly incorporate vehicle dynamics, constraints, and performance objectives, making them inherently applicable to high-performance and racing scenarios.

A widely adopted implementation of OCP in the online setting is *Nonlinear Model Predictive Control* (NMPC). NMPC [72], [73] operates in a receding-horizon fashion, repeatedly solving a finite-horizon optimization problem at each control step to generate updated control commands. This framework allows for the integration of dynamic models, time-varying constraints, and objective functions that reflect both safety and performance goals. As a result, NMPC has become a central tool for online planning and control in autonomous vehicles, particularly when time-optimality or near-limit handling is required.

1.3.2.1 Time-Optimal Trajectory Planning vs Trajectory Tracking with Online NMPC

Within the framework of NMPC, two primary paradigms can be distinguished: tracking NMPC and E-NMPC [74].

- Tracking NMPC is designed to steer the system—often referred to as the plant in control theory—toward a predefined reference trajectory or setpoint.
- In contrast, E-NMPC seeks to optimize a performance-related cost function directly, without requiring a reference trajectory. In the context of autonomous racing and high-performance driving, this cost function is typically chosen to minimize travel time.

Most research efforts in online trajectory generation for autonomous vehicles have employed tracking NMPC [75], [76], [77], [78], [79], [80], [81], where the controller follows a race line computed offline. This strategy has been widely adopted due to its relative simplicity and lower computational demands, as the optimization typically minimizes deviations from a fixed path. Notable examples include applications of tracking NMPC for real-time race line following, using models of varying complexity.

In contrast, a smaller but growing number of studies have explored E-NMPC for online time-optimal planning [25], [40], [52], [82], [83], in which the system continually recomputes the trajectory that minimizes lap time from its current state. This formulation offers significantly more flexibility, allowing the vehicle to dynamically adjust its path and speed profile in response to new conditions, model discrepancies, or execution errors.

While tracking NMPC generally leads to more tractable optimization problems due to its quadratic cost structure, it lacks the replanning capabilities characteristic of expert human drivers. Studies [25], [78], [84], [85] have highlighted that the ability to continuously adapt the planned trajectory—rather than rigidly follow a predefined one—is crucial for achieving true time-optimal performance. In scenarios involving dynamic disturbances or minor deviations from the nominal path, a tracking controller may attempt to return to the original race line, even when doing so is suboptimal or physically infeasible.

This thesis demonstrates that E-NMPC, when combined with a sufficiently long prediction horizon, enables online generation of time-optimal manoeuvres that adapt to the current vehicle state in real-time. This self-adaptive planning behaviour better mimics the strategies employed by professional drivers and is essential for autonomous systems operating near the limits of handling.

1.3.2.2 Vehicle Dynamics Models for Online Time-Optimal Trajectory Planning

Implementing online minimum-time motion planning—whether using E-NMPC or tracking NMPC—requires careful consideration of the trade-offs between model fidelity, computational efficiency, and problem tractability. The choice of vehicle dynamics model directly influences the non-linearity and dimensionality of the optimization problem, as well as the feasibility and robustness of the resulting trajectories.

To reduce the computational burden, many authors have adopted simplified point-mass models. For instance in [76], point-mass dynamics have been used to compute terminal velocity constraints to then track a precomputed minimum-curvature path. In several studies [48], [52], [78], [80], [82], [86], [87], these models are paired with g-g-v envelopes, which encode the velocity-dependent acceleration limits of the vehicle. While these approaches offer low computational complexity, they often neglect yaw dynamics, or rely on oversimplified kinematic representations. As a result, the planned trajectories may violate physical feasibility when executed on real vehicles. Tube-based *Model Predictive Control* (MPC) formulations were used in [80], [88] in combination with simple linear point-mass models. Such techniques have been applied in real-world competitions such as the Indy Autonomous Challenge [11].

Beyond point-mass formulations, other authors have explored dynamic or kineto-dynamical models for improved accuracy. For example, in [78] the authors embedded

the longitudinal load transfer effects in a simplified vehicle dynamic model and into the g-g diagram. However, to meet real-time constraints for trajectory tracking with static obstacle avoidance, they had to linearize the model. Notably, in [79] they used a single track model, but again they had to perform trajectory tracking rather than planning, where the time-optimal trajectory computed offline and then tracked online.

Another emerging direction involves the integration of machine learning models into NMPC frameworks [89]. Learning-based approaches have been employed to augment nominal vehicle models online, using techniques such as Gaussian processes [90], [91], neural networks [21], [77], [92], and iterative learning MPC [93], [94], [95]. These methods aim to improve prediction accuracy or adapt cost functions and constraints based on observed data. While many of these approaches are still in experimental phases, they offer promising tools for compensating model uncertainty and improving adaptability near the limits of handling.

While most of the aforementioned works rely on direct OCP methods, our recent contributions [24], [25], [40], [49], [54], [96], [97] adopt an indirect OCP approach to online time-optimal E-NMPC.

Finally, several works have begun to explore robust control formulations [80], [88], such as tube MPC and uncertainty-aware planning, to account explicitly for model mismatch and disturbances. The impact of model fidelity on time-optimal trajectory generation—especially when operating near physical limits—has also been rigorously studied [98], highlighting the importance of choosing a model that balances accuracy and computational efficiency.

1.3.3 Vehicle Trajectory Tracking

In modular autonomous driving architectures, trajectory tracking plays a central role in ensuring that a vehicle follows the path generated by a higher-level motion planner. Whether the trajectory is computed offline as a minimum-time reference or planned online in response to dynamic changes, a reliable tracking controller is essential for accurate and safe execution.

A wide range of trajectory tracking strategies have been explored in the literature, each offering different trade-offs in terms of complexity, interpretability, robustness, and performance. Comprehensive surveys [7], [99], [100], [101] are available that categorize and compare these approaches across various application domains.

This section focuses specifically on trajectory tracking in high-performance and autonomous racing scenarios. The following subsections review tracking techniques based on NMPC, neural networks, and alternative control methods, with particular attention to how each performs under the demands of aggressive, near-limit driving.

1.3.3.1 Trajectory Tracking with NMPC

NMPC has been widely adopted as a low-level trajectory tracking method in modular autonomous driving architectures, particularly in the context of autonomous racing. In these frameworks, NMPC is typically responsible for following reference trajectories generated by a separate high-level planning module. This separation allows the controller to focus on enforcing dynamic feasibility and constraint satisfaction in real-time.

Several works [76], [79], [102] have successfully applied NMPC to high-performance driving scenarios, demonstrating its effectiveness in controlling vehicles along aggressive trajectories. These applications rely on NMPC's ability to handle nonlinear vehicle dynamics and to anticipate future behaviour over a finite prediction horizon, making it a natural choice for tracking fast, time-optimal trajectories.

To enhance the accuracy of NMPC tracking, some researchers have integrated machine learning techniques into the control pipeline. For example, Gaussian process regression has been used—in [90], [91]—to improve nominal vehicle models online, allowing for data-driven correction of model errors as new observations become available. Other approaches [21], [77] have employed neural networks to model vehicle lateral dynamics

and augment the prediction models used within NMPC. These hybrid techniques aim to compensate for modelling inaccuracies while preserving the structure and interpretability of model-based control.

Despite these advances, NMPC faces several practical limitations in real-time applications. A key challenge is the trade-off between model fidelity and computational complexity. High-fidelity vehicle models increase prediction accuracy but result in large-scale optimization problems that are computationally demanding and sensitive to solver settings. In time-critical environments such as autonomous racing, this can lead to convergence issues, particularly when the vehicle operates near its dynamic limits or under rapidly changing conditions.

As a result, many NMPC-based tracking controllers adopt simplified vehicle models to strike a balance between accuracy and real-time feasibility. The performance of such controllers depends heavily on how well these simplified models approximate real vehicle behaviour under high-stress scenarios. Recent studies have shown that robust and stochastic formulations of NMPC can mitigate these limitations by explicitly accounting for model and parameter uncertainties [103], [104]. These approaches enable simplified models to achieve reliable performance even in the presence of disturbances and modelling errors.

1.3.3.2 Trajectory Tracking with Neural Networks

In addition to traditional control techniques, various researchers have explored the use of *Artificial Neural Network*s (NNs) as low-level controllers for tracking vehicle trajectories [7], [17], [25]. These networks are typically designed to control either the longitudinal or lateral dynamics—or both—by directly mapping system observations to control actions such as steering angles or torque commands.

A number of studies have investigated the effectiveness of different neural architectures in this context. For instance, comparisons have been made [105] between simple feedforward networks and more complex convolutional architectures for computing control inputs during path tracking at constant speed. In more advanced applications [85], stochastic neural policies have been trained to imitate expert human driving behaviour near the vehicle's handling limits.

Other works [24] have employed recurrent neural networks to model temporal dependencies in vehicle behaviour, allowing the controller to take into account the system's dynamic history. In certain cases [22], neural network controllers have been deployed in tandem with classical feedback loops, where the NN handles nominal behaviour and the feedback controller compensates for disturbances or modelling errors.

More recently, attention has shifted toward improving the interpretability and data efficiency of NN-based controllers. One promising direction involves designing physics-informed neural networks, which embed prior knowledge of the vehicle's dynamics directly into the network architecture [25], [106], [107]. These structured models constrain the learning process to obey known physical relationships, such as the equations governing longitudinal or lateral motion. As a result, they often outperform general-purpose NNs in terms of both generalization and transparency, while requiring fewer trainable parameters.

This thesis adopts a similar philosophy, leveraging physics-informed neural networks to ensure that learned control policies remain consistent with the underlying vehicle dynamics. These models offer a compelling compromise between the flexibility of learning-based methods and the structure and safety guarantees of model-based control.

1.3.3.3 Trajectory Tracking with Other Methods

Beyond NMPC and NN-based approaches, a wide variety of control strategies for vehicle trajectory tracking have been proposed in the literature [7], [99], [100], [101]. Several of these methods, while perhaps simpler in structure, have proven effective in specific contexts, particularly within autonomous racing applications where robustness and computational efficiency are critical.

Experimental comparisons have been conducted [108] to evaluate classical control strategies for lateral vehicle dynamics, such as pure pursuit [109] and clothoid-based tracking algorithms. These methods offer intuitive geometric formulations and are often favoured in resource-constrained embedded systems. In other works [110], PID controllers have been used successfully for trajectory tracking in real-vehicle prototypes. Although relatively simple, PID controllers can be finely tuned for acceptable performance under moderate conditions.

More advanced approaches combine model-based reasoning with feedback adaptation. One notable example is the use of feedforward-feedback control structures for high-performance path tracking [111]. In that framework, the feedforward component was derived from a vehicle's handling diagram, and they integrated the desired side-slip angle behaviour in the control loop.

Recent years have also seen increased interest in RL for trajectory tracking. RL-based controllers are capable of learning from interaction and adapting to complex, nonlinear dynamics without requiring explicit models. Techniques such as soft actor-critic have been applied to miniature racing platforms [112], demonstrating promising generalization across tracks and speeds. Other contributions [113] leverage iterative learning control to refine performance over repeated laps, using data from previous trials to improve both lateral and longitudinal tracking accuracy.

Hybrid strategies have also emerged, such as model-based policy learning, where RL techniques are used to tune or augment conventional controllers. In these approaches [114], a nominal feedforward-feedback controller serves as a baseline, and policy gradients—estimated using simplified vehicle models—are applied to enhance performance in real-world trials.

While these methods vary in complexity and design philosophy, they all contribute to the broader objective of achieving precise, stable, and adaptable control for autonomous vehicles operating near the handling limits. Their diversity reflects the multi-faceted nature of trajectory tracking and underscores the need to balance performance, interpretability, and real-time feasibility in high-speed applications.

1.3.4 End-to-End Approaches for Time-Optimal Autonomous Driving

The majority of works discussed in previous sections (1.3.2, 1.3.3) follow a modular architecture for autonomous driving and racing, typically composed of distinct perception, planning, and control modules. This structure offers clarity, interpretability, and the ability to tune each component independently. However, it also relies on well-calibrated interfaces and strong prior knowledge of system dynamics, often requiring careful design and integration effort.

In contrast, recent research has explored end-to-end learning-based approaches, which aim to bypass intermediate representations by directly mapping sensory inputs to control actions. These methods typically leverage *Imitation Learning* (IL) or deep RL to train driving policies that replicate expert behaviour or optimize performance directly.

In the context of time-optimal autonomous driving, such end-to-end strategies have been applied to learn the driving style of professional race drivers, typically using data collected from gaming platforms or simulation environments. These methods offer the potential for faster development cycles and improved adaptability, but they also raise challenges related to generalization, interpretability, and safety—especially when deployed outside the conditions observed during training.

1.3.4.1 Imitation Learning Approaches

As an alternative to hierarchical motion planning and control frameworks, some researchers have begun exploring IL as an end-to-end strategy for time-optimal autonomous driving. Rather than decomposing the driving task into perception, planning, and tracking, IL methods aim to directly learn a control policy from expert demonstrations, mapping sensory observations to actions in a single model.

Recent studies [115] have benchmarked the performance of IL in the context of aggressive or high-performance driving. In one approach [85], stochastic policies were trained to imitate the behaviour of professional race drivers, with the goal of replicating human-like manoeuvres near the handling limits. While these models demonstrated promising performance in familiar environments, they often struggled to generalize to circuits not encountered during training—highlighting a common limitation of IL methods.

Other works [116] have evaluated various combinations of IL and RL techniques on scaled-down autonomous racing platforms, such as F1Tenth vehicles. Although these studies provide valuable insights into algorithmic trade-offs, the experimental setups are typically simplified and do not reflect the full complexity of time-optimal racing. As a result, the resulting trajectories are typically far from optimal when compared to solutions derived from model-based planning.

A fundamental limitation of IL is its dependence on expert demonstrations. The quality, diversity, and representativeness of the training data strongly influence the performance of the learned policy. Moreover, IL methods tend to replicate the behaviour seen during training without a mechanism for active improvement or adaptation in unseen scenarios, which can be problematic in dynamic or safety-critical applications. Recent developments aim to mitigate these shortcomings by incorporating physical or safety constraints directly into the learning process. For instance, [117] proposed a constraint-aware IL framework for autonomous racing that enforces safety into the IL objective.

Despite these limitations, IL offers a compelling paradigm for learning complex behaviours in a data-driven way, and continues to be an active area of research in autonomous racing.

1.3.4.2 Reinforcement Learning Approaches

RL has recently gained traction as an end-to-end strategy for time-optimal autonomous driving, particularly within simulation environments and gaming platforms. RL methods aim to learn control policies through interaction, without relying on explicit supervision or predefined trajectories. However, their effectiveness in high-performance scenarios remains limited by several practical and methodological challenges.

Early applications of RL to autonomous racing have used popular actor-critic algorithms to control vehicles in commercial racing games or simulators. In some cases [118], asynchronous advantage actor-critic methods were used to train agents in environments like World Rally Championship 6, while soft actor-critic algorithms were applied to control vehicles in simulators such as TORCS [119]. However, these studies typically involved oversimplified vehicle dynamics, and no performance comparison was made against expert human drivers or optimization-based reference trajectories—making it difficult to assess the time-optimality of the learned behaviours.

More recent works have attempted to bridge this gap by incorporating offline-computed minimum-time trajectories into the training process. In one study [120], an RL agent was trained to minimize deviations from these precomputed trajectories across various circuits. While this approach improved structure in the learning process, the resulting performance still fell significantly short of the optimal reference, indicating the difficulty of learning aggressive driving policies solely through interaction.

To improve learning efficiency and generalization, some researchers have combined IL with RL. For example in [121], hybrid IL+RL methods have been used for vision-based autonomous control of scaled-down vehicles, achieving better performance than either approach in isolation.

High-profile demonstrations of RL in racing environments have also been presented using commercial video games. Notably in [26], [27], soft-actor-critic agents trained in Gran Turismo Sport were able to defeat professional human drivers—but only on the circuits used during training. These successes came at the cost of extremely high data requirements, with training spanning several days and relying on multiple parallel gaming consoles. Moreover, these agents failed to generalize to unseen circuits, exposing a key limitation of current RL methods in terms of adaptability.

A more recent effort [122] introduced contextual RL to model a distribution over driving styles based on professional demonstration data. While the trained agents were able to outperform human drivers in simulation, they still relied heavily on expert demonstrations and, crucially, remained unable to generalize to new circuits outside the training set.

Taken together, these studies show that while RL can produce high-performance behaviour in tightly controlled environments, its application to real-world or general-purpose autonomous racing is still limited by sample inefficiency, poor generalization, and high computational cost.

1.3.5 Interpretable Machine Learning for Autonomous Driving

As introduced in Section 1.2.3, interpretable machine learning has recently emerged as a promising avenue to enhance the safety, transparency, and societal acceptance of autonomous driving systems. While deep learning techniques have shown great potential in control and perception, their widespread deployment in safety-critical applications is often hindered by their opaque nature and limited robustness. Interpretable machine learning seeks to bridge this gap by embedding structure, prior knowledge, or transparency into the learning process.

This evolving field encompasses several research directions. The following categories are particularly relevant in the context of autonomous driving.

- **Explainable AI**: This area focuses on post-hoc methods that aim to make the decisions of black-box AI systems more understandable to humans. Explainable AI is especially important in safety-critical applications like autonomous driving, where clarity and accountability are essential for certification and public trust. A recent survey in this category can be found in [29], [123].
- **Physics-Informed Neural Networks (PINNs):** Introduced in the context of solving partial differential equations (PDEs) [124], PINNs incorporate physical knowledge directly into the training process by augmenting the loss function with the residuals of governing equations. While PINNs have been applied in various physics domains [125], including state estimation, their internal structure typically remains generic. The key innovation lies in the training process, which enables data-efficient learning of physically consistent models.
- **Hybrid Neural-Analytical Models:** This approach combines the expressive power of neural networks with the interpretability and structure of classical models. For example, a neural network may be used to approximate unknown dynamics—such as tire forces—while the overall system behaviour remains governed by an analytical vehicle model [126]. This hybridization provides a clear interface between learned and known components, improving interpretability and trustworthiness.
- **Symbolic Distillation:** In this line of work, interpretable symbolic rules are extracted from complex neural models [127]. These distilled models can reduce inference time and increase transparency, offering a pathway to deploy simpler yet human-understandable control policies derived from richer learned representations.
- **Neural Networks with Domain-Specific Internal Architectures:** Rather than relying on generic architectures, this approach designs neural networks whose internal structure reflects domain knowledge—such as the laws of vehicle dynamics and kinematics. By embedding inductive biases into the architecture itself, these networks improve generalization, interpretability, and sample efficiency. This technique—described in [128] as active interpretability intervention—has shown promising results in autonomous driving and related domains [25], [106], [107], [129], [130], [131].

This thesis focuses primarily on this final category. Specifically, domain-informed neural networks are developed and deployed both as low-level controllers and as neural

vehicle models that act as digital twins within the time-optimal autonomous racing framework. These models capture the underlying vehicle dynamics while remaining interpretable and physically grounded, enabling their use for both control and simulation purposes. Designed to operate near the vehicle's handling limits, their architecture is explicitly shaped by physical insight into vehicle behaviour.

By embedding physical knowledge into the learning architecture, this work aims to combine the flexibility of data-driven methods with the transparency and reliability of model-based design, required in high-performance autonomous driving.

Lessons from the Literature and Implications for ARD

Across the reviewed literature, several limitations remain evident in the domains of planning, control, and vehicle modelling. Offline methods achieve high accuracy but lack real-time feasibility, while online approaches often rely on simplified dynamics that sacrifice physical realism for computational speed. Similarly, learning-based controllers offer adaptability but typically lack interpretability and physical consistency, limiting their trustworthiness in safety-critical applications.

These gaps motivate the development of ARD, which combines model-based optimization with physics-informed learning to achieve both real-time feasibility and physical fidelity. By integrating interpretable neural vehicle models, an E-NMPC planner, and physics-informed control, this work bridges the divide between theoretical time-optimality and deployable real-world performance. The resulting architecture demonstrates that interpretable and data-efficient autonomous driving can achieve high performance with minimal prior knowledge, while maintaining adaptability and consistency near the vehicle's handling limits.

1.4 Shared Control Literature Review

1.4.1 Shared Control in the Context of Autonomous Driving and Racing

As autonomous driving technology continues to mature, the role of the human driver is being redefined. While many systems aim to replace human input entirely, an alternative and increasingly relevant approach is shared control, where control authority is dynamically distributed between human and machine. In this paradigm, the goal is not only to assist the driver in routine tasks but to actively enhance their performance, safety, and situational awareness.

In autonomous driving, shared control is often framed as a way to reduce accidents caused by human error, to support drivers in fatigue or distraction, or to manage transitions between manual and automated modes. Most applications focus on maintaining lane discipline, regulating speed, or intervening in critical scenarios to avoid hazards. These systems aim to balance safety and usability, sometimes taking full control and other times subtly correcting or guiding the driver.

In the domain of autonomous racing, however, the motivations for shared control expand beyond safety. Racing requires driving at the limits of handling, where optimal trajectories, timing, and control precision matter immensely. In such conditions, even a skilled human driver may struggle to match the consistency and speed of an optimal autonomous agent. Yet, replacing the human entirely overlooks the opportunity to enhance both the driver and the automated system capabilities through collaboration. Here, shared control can become a tool for real-time coaching, not just intervention. It enables the machine to assist, guide, and nudge the driver toward optimal behaviour, while preserving the driver's engagement and learning experience.

This concept underlies the motivation for ARC. Rather than taking over, ARC aims to share control in a way that teaches the driver how to drive better. By integrating shared control techniques with an understanding of optimal vehicle behaviour—such as that provided by ARD—ARC seeks to blend machine intelligence with human intent, creating

a collaborative dynamic that is especially valuable in high-performance or emergency scenarios.

1.4.2 From Assistance to Coaching: Shifting the Focus to Driver Development

Historically, shared control systems have been developed with the primary goal of assisting the driver—reducing workload, mitigating error, and improving safety in challenging or hazardous scenarios. A wide variety of approaches have emerged from this safety-oriented mindset, such as lane-keeping assistance [132], road departure avoidance systems [133], and hazard-avoidance interventions based on predictive or constraint-based planning [134], [135].

These systems often rely on continuous monitoring of the driver's actions and environment to allocate control authority dynamically. For instance, authority can be adjusted based on threat levels [135], distraction states [136], or predicted collision risks [137], [138]. Most operate under metaphors such as "guardian angel" [139], in which the automation subtly or forcefully corrects the driver's actions to ensure safety. Feedback is typically provided through haptic cues, particularly on the steering wheel [137], [140], or on pedals for longitudinal control [141].

However, these frameworks are generally not designed to teach the driver how to improve. They intervene or correct rather than instruct or explain. Exceptions exist—for example, the use of AI-based coaching to provide explanatory feedback has been explored [142], but these systems usually operate in observation or passive modes and do not implement shared control directly. In that context, the role of shared control as a coaching mechanism remains underexplored.

Coaching reframes the objective: the system no longer intervenes only to avoid danger but actively helps the driver learn and internalize better driving strategies. This shift in purpose affects how control is allocated, how feedback is presented, and how human performance is evaluated. The instructor-student metaphor highlighted in the review by [143] offers a compelling conceptual foundation for this kind of relationship. Here, the automation may apply guidance or resist deviations—not to override the driver, but to communicate best practices and accelerate skill acquisition.

This reframing is particularly important in high-performance driving contexts such as racing, where the objective is not simply to avoid failure, but to optimize performance. In such settings, a coaching-oriented shared control system must balance real-time performance with the opportunity for the driver to explore, adapt, and learn. It must offer transparency, consistency, and a level of authority that supports—rather than dominates—the driver. The ARC framework proposed in this thesis is designed around these principles, aiming to enable a novel form of real-time machine-assisted driver development.

1.4.3 Approaches to shared control

1.4.3.1 Coupled and Decoupled Control Systems

One of the fundamental distinctions in shared control design lies in how the driver and automation interact with the vehicle's actuators—often referred to as coupled versus decoupled control architectures. In coupled systems, both the driver and the automation simultaneously apply control inputs to the same physical interface, such as a steering wheel or pedal. This creates the possibility for rich, bidirectional interaction through haptic feedback. In contrast, decoupled systems process driver and machine commands independently and fuse them algorithmically, typically assuming a steer-by-wire or drive-by-wire setup. This typically means that the driver feels only the guiding feedback and not the one of the vehicle.

Several works explored the benefits and challenges of coupled systems, especially in lateral control scenarios. For example, in [136], both the driver and the automation apply torque to the steering wheel, with a fuzzy controller computing the assistive torque in

real-time based on a trajectory planning module and inferred driver intent. Similarly, [140] examined different levels of haptic authority in shared steering, concluding that lower authority can result in better cooperation and driver comfort—though higher authority may be preferred in challenging conditions. Force feedback on the steering wheel was also used in [137], [144], enabling smoother transitions during handover between manual and automated modes.

Coupled architectures also exist in longitudinal control, albeit less frequently. In [141], shared control is applied to the accelerator pedal using counteracting forces to reduce acceleration in response to the environment. While this setup only permits the system to reduce—but not increase—acceleration, it demonstrates the feasibility of haptic shared control in the longitudinal domain. The Toyota Guardian concept, although less technical in available publications [139], also embodies this philosophy of amplifying human intent through physical interfaces.

Conversely, decoupled systems typically rely on control blending formulas of the form

$$u = \alpha\, u_{\text{driver}} + (1 - \alpha)\, u_{\text{machine}}, \tag{1.1}$$

where the allocation coefficient α is computed dynamically or held constant. This formulation assumes the control actions can be safely combined, which is generally feasible in steer-by-wire contexts. For instance, [138] used fuzzy logic to determine α in a high-speed obstacle avoidance scenario, allowing flexible authority allocation between a simulated driver and machine. Similarly, [145] used a fuzzy control allocation based on estimated trust, and [146] trained a RL agent to compute the allocation coefficient adaptively.

While decoupled systems are easier to implement in simulation and offer greater architectural modularity, they often fail to provide meaningful real-time feedback to the driver. This can lead to misunderstandings or loss of situational awareness, especially in edge-case scenarios. Coupled systems, by contrast, can encode feedback in the resistance or feel of the interface, but they require more complex design and hardware assumptions. Interestingly, [134] used a coupled-like interface but with unilateral authority: the driver could feel the system's action through the steering wheel, but the automation would take over control if needed, introducing challenges in transparency and driver acceptance.

Choosing between these paradigms ultimately depends on the application. For cooperative or assistive systems, both approaches are viable. However, for coaching-focused systems—as proposed in ARC—a coupled interface is likely more appropriate, as it allows the automation to communicate guidance through feel, rather than override.

1.4.3.2 Dynamic Authority Allocation Strategies

A central challenge in shared control design is determining how control authority is allocated between the human driver and the automated system. Fixed authority systems—where either the driver or automation has a preset dominance—often fail to adapt to the dynamic nature of real-world driving. Hence, dynamic authority allocation strategies are emerging as a key enabler for flexible and adaptive cooperation.

Several papers explored mechanisms for computing the allocation coefficient α dynamically, either through rule-based, fuzzy logic, or learning-based methods. In [138], a fuzzy logic controller modulates the authority allocation based on the vehicle's lateral error, allowing the automation to intervene more strongly as deviation from the desired path increases. Similarly, [145] proposed a fuzzy logic controller that adapts authority in response to estimated driver trust. This approach models the trust-performance relationship to mitigate human–machine conflict, although their system was evaluated only with a simulated driver model.

In [146], RL was used to dynamically allocate control authority by optimizing a coordination strategy between two RL agents—one representing the driver, the other the automation. The learned policy determines the authority weighting based on driving context. A related idea was implemented in [147], where RL is guided by inferred driver intention. While promising, these approaches introduce complexity and require

substantial data for training, and often lack interpretability or transparency—crucial factors in trust-building with human users.

MPC has also been used as a means to derive control allocation strategies. In [135], the system computes a "threat index" based on predicted vehicle states (*e.g.,* front-wheel slip) over the MPC horizon. This threat level is then used to scale control authority dynamically, letting the automation intervene more as the predicted risk increases. A similar scheme is presented in [134], where the system allocates between 0–100% authority based on real-time safety constraints while leaving manoeuvring freedom within safe homotopies.

Some strategies consider context-aware cues like time-to-collision to compute allocation. In [137], a Nash equilibrium formulation is used to derive cooperative actions between the human and automation, with control allocation modulated by time-to-collision. This makes the system responsive to risk without resorting to full override.

Interestingly, few of these systems incorporate learning from individual driver behaviour to personalize authority allocation over time. An exception is [141], where they adapt the model of each driver's control behaviour, allowing the shared longitudinal controller to adjust its strategy per user.

In sum, dynamic control allocation is widely recognized as critical for effective shared control, especially in systems where the driver must remain engaged. However, most reviewed approaches either operate with limited real-time feedback, rely on simulated drivers, or overlook driver-centric factors like trust, expertise, and intent.

1.4.4 Control and Interaction Design

1.4.4.1 Feedback Modalities: Haptic, Visual, and Beyond

A well-designed feedback system is essential in shared control to ensure transparency, maintain driver engagement, and foster trust. While many shared control frameworks focus heavily on the control algorithm, several of the reviewed works highlight the importance of how the system communicates its actions and intentions to the human driver—whether through haptic cues, visual indicators, or less conventional modalities.

Haptic feedback is the most commonly explored modality. Various systems applied assistive torques directly to the steering wheel, enabling the driver to perceive the automated system's intentions without requiring visual attention. For example, [136] implemented a fuzzy steering controller that applies assistive torque based on predicted driver intention, offering continuous guidance during highway driving. In [137], torque feedback was used to support coordination derived from a non-cooperative Nash game formulation. Notably, [148] emphasized using haptic cues to inform the driver of the system's intended path, rather than override or correct their input—a subtle but important distinction that promotes cooperation over conflict.

Others, such as [133], applied haptic feedback as a corrective cue in a road departure avoidance system. While effective in improving safety, the system risked confusing users without prior training—highlighting that too much control or unclear feedback can lead to reduced performance or even mistrust. This aligns with findings from [140], where the best driver–machine cooperation occurred with low-to-moderate haptic authority; higher levels degraded subjective experience and had diminishing returns in terms of control accuracy.

Visual feedback was less commonly used but showed potential for conveying richer information. In [134], a superimposed safe corridor was projected in the driver's field of view, visually indicating the permissible area for manoeuvring. This helped the driver remain within safe bounds while maintaining a sense of agency. Likewise, [142] showed that multimodal explanations, combining visual and auditory cues, led to improved driver performance and trust in AI coaching scenarios, even outside a control-sharing context.

However, many reviewed systems did not provide any form of real-time feedback. This includes learning-based approaches like [147], where control allocation is dynamically computed but not actively communicated to the driver. This lack of transparency can

limit the interpretability and acceptance of such systems—particularly problematic in high-speed or high-risk contexts like racing.

Finally, the idea of "snap" feedback, as discussed conceptually in [138], opens up a novel direction for signalling threshold breaches in control allocation. In this design, exceeding a predefined virtual boundary would result in a sharp feedback cue (*e.g.,* a jolt on the steering wheel), clearly informing the driver that they have exited the system's support envelope and must fully take over.

In summary, effective feedback is as crucial as control itself in shared systems. Across the reviewed literature, haptic cues remain the most prevalent due to their immediacy and non-intrusiveness, but visual and multimodal feedback—particularly when linked to interpretable internal models—may hold the key to scalable and intuitive human–machine cooperation. However, it is worth noting that most existing studies have been conducted under nominal driving conditions, and the suitability of these feedback modalities near the vehicle's handling limits remains largely unexplored. Understanding how feedback should behave in such extreme regimes thus remains an open challenge for high-performance driving applications.

1.4.4.2 Designing Interactions for Safety, Trust, and Performance

A shared control system is only as effective as the quality of interaction it enables between human and machine. Designing this interaction involves more than tuning controller gains or implementing clever authority allocation strategies—it requires a deep understanding of how the driver perceives the system's behaviour, builds trust, and maintains situational awareness. Safety, trust, and performance are tightly coupled, and trade-offs between them are shaped by the interaction design itself.

A central challenge noted in several works is how to avoid human–machine conflict, especially when the system intervenes unexpectedly or too aggressively. For instance, [140] found that excessive haptic authority could reduce driver acceptance, even if it improved objective vehicle control. This supports the principle that drivers should feel in control, even when they're being assisted—an insight that is particularly relevant to high-performance settings like racing, where overbearing intervention may degrade rather than enhance performance.

Conversely, many studies found that some level of automation can enhance safety, especially in difficult scenarios. For example, [133] demonstrated that timely system intervention could reduce the risk of road departure—but only if the driver had prior experience with the system. Without this familiarity, performance actually worsened. Similarly, [145] proposed adjusting authority based on inferred driver trust, suggesting that adaptivity to user confidence and experience is critical to maintaining safety without undermining trust.

While most of the reviewed literature aimed to assist the driver in staying safe, very few systems were designed to teach the driver to perform better. One notable exception is [142], where an AI coach provided explanations that significantly improved driver performance and confidence. Although their work did not involve real-time shared control, it points toward the importance of transparent and supportive feedback, especially when the goal is long-term skill development.

Interaction metaphors also play a subtle but important role in shaping expectations. [143] discussed several metaphors for shared control, with the instructor–student metaphor being particularly aligned with ARC's vision. In this paradigm, the system does not "save" the driver but instead guides them, allowing for active learning through experience. This approach is distinct from more paternalistic metaphors (like guardian or autopilot), which assume the driver is a fallback or passive participant.

In some cases, interaction strategies were used to deliberately modulate driver engagement. For example, [149] implemented a shared control policy where control was actively taken from the driver to force attention, then gradually handed back. This was intended to re-engage distracted drivers—an approach that may not apply directly

to racing, but still raises valuable questions about how control transitions can be used to influence human behaviour.

Lastly, the lack of discussion on driver expertise, cognitive load, and emotional state in many studies—including those using advanced techniques like RL [146], [147]—highlights a persistent gap. Few systems explicitly modelled these human factors, even though they significantly affect how the driver interprets and responds to system behaviour. This suggests a need for greater emphasis on human-centred evaluation, especially when moving from assistance to skill transfer.

In summary, designing effective human–machine interactions in shared control systems requires more than optimal control theory. It calls for thoughtful integration of feedback, authority sharing, transparency, and user modelling—all aimed at ensuring that drivers remain engaged, confident, and ultimately safer and more skilled behind the wheel.

1.4.5 Lessons from the Literature and Implications for ARC

1.4.5.1 Evaluation Practices and Observed Limitations

Across the reviewed body of literature, evaluation practices for shared control systems vary widely—from driver-in-the-loop simulator tests to real-world experiments and pure simulation. However, common limitations persist, particularly in how human–machine interaction is assessed and how generalizable the findings are.

A majority of the experimental work was carried out in driving simulators, which offer controlled environments for evaluating system behaviour without physical risk. Notable examples include [150], [136], and [141], all of which demonstrated the benefits of their shared control systems in tasks like lane keeping or longitudinal following. Simulators also enabled rapid prototyping and safe iteration, which was essential in early-stage research.

However, real-world validation remains sparse. While a few works did go beyond simulation–*e.g.,* [135], [137], [138]–most of these tests involved limited scenarios or constrained control setups (*e.g.,* fixed speed, pre-defined manoeuvres). This restricts the applicability of findings to more dynamic or aggressive contexts like racing, where variability in both the vehicle and driver is higher.

The choice of evaluation metrics was similarly narrow in many cases. Some studies focused on physical quantities like trajectory deviation, lateral error, or steering smoothness [132], [149], while others looked at system-level metrics like task completion time or collision rate [148], [151]. However, subjective measures such as trust, cognitive load, perceived cooperation, and user acceptance were often missing or underdeveloped. Exceptions include [140] and [142], which attempted to measure these human factors—though even there, quantifying these aspects proved difficult and context-dependent.

Another critical gap in many studies is the absence of diverse or adaptive evaluation scenarios. Tests were typically limited to well-defined, low-variability driving tasks. For instance, in [136], [141], the driver was restricted to either the steering wheel or accelerator pedal, with fixed speeds and no unexpected disturbances. This raises concerns about how these systems would behave in more realistic or competitive situations where the driver's strategy, intent, or performance evolves over time.

Few systems have been tested for their ability to support driver learning—an aspect central to ARC's mission. While [142] showed the potential of AI coaching in improving driver skill, no study systematically assessed whether shared control can enhance human learning over time. Most works evaluated short-term performance improvements, neglecting longitudinal effects, which are key to coaching-oriented applications.

Finally, user variability—in terms of driving style, skill, and adaptability—is seldom addressed rigorously. Several papers, like [145], [146], employed driver models or simulations of multiple users but did not include real-time adaptation to individual users. Even when systems allowed for personalization (*e.g.,* tuning parameters [135]), these features were not tested with a broad range of drivers or linked to performance trends.

In conclusion, while current evaluation practices provide a foundation for assessing shared control systems, they often fall short in capturing the complexity of human–machine collaboration, especially in dynamic or high-performance contexts. In particular, the case of shared support near the vehicle's handling limits remains virtually unexplored, leaving open important questions on how assistance should be designed and evaluated in such extreme conditions.

1.4.5.2 Opportunities for Real-Time Coaching and Adaptive Guidance

While much of the literature on shared control emphasizes safety and stability, a compelling opportunity lies in repurposing these frameworks toward real-time coaching and adaptive driver guidance—an area that remains largely unexplored. Unlike conventional shared control systems that aim to correct or override driver behaviour [133], [145], ARC aspires to teach, offering drivers feedback that helps improve their skills over time while preserving their agency.

A few precedents in this direction exist. Notably, [142] demonstrated that AI-based coaching, even in a passive observation setting, can positively impact human driving performance. Their study showed that multimodal feedback (visual and auditory) improved driving metrics and user confidence, reinforcing the idea that machine support need not be intrusive to be effective. However, the coaching in that case was limited to an explanatory role, with no shared control or dynamic interaction.

On the control side, some systems hinted at adaptive guidance by adjusting authority based on risk, trust, or predicted conflict [134], [137], [138]. For example, [134] proposed a threat-based authority allocation scheme where the system intervened more strongly as risk increased, while [140] suggested that driver preferences shift under difficulty, favouring more assertive support during challenging scenarios. These ideas support the notion of a continuous support, adaptable not just to road conditions but also to driver skill level and intent.

Interestingly, a few authors proposed control metaphors that align more naturally with coaching. The "instructor–student" metaphor discussed in [143] describes a system where the machine acts as a more experienced pilot—offering guidance either actively or passively. This framework resonates with ARC's ambition: to be a partner in the loop, not a fallback mechanism. However, most systems implementing such metaphors still lacked a concrete mechanism for delivering nuanced feedback beyond control actions.

What's largely missing—and what ARC seeks to explore—is a layer of interaction designed around pedagogy: feedback that evolves based on user progression, personalized interventions informed by real-time performance, and mechanisms for phasing out support as competence increases. Current literature offers a few ingredients for such a system—*e.g.,* learning of driver parameters [141], adaptive authority scaling [146], and haptic signalling of intent [136]. However, these elements have yet to be integrated into a unified, driver-centric coaching framework.

There is also untapped potential in leveraging multi-modal feedback. While haptic feedback on the steering wheel or pedals is commonly used [136], [140], few systems combine it meaningfully with visual cues, such as overlays or AR-based guidance [134]. For a coaching application, communicative clarity is paramount: the driver must understand not only what the system is doing, but why and how to improve.

In summary, the reviewed literature highlights promising building blocks but falls short of realizing real-time coaching systems. ARC's design can draw from these lessons to position itself not solely as a safety-enhancing intervention but as a teaching partner, capable of nudging, guiding, and ultimately stepping back as the human driver matures. This opens new research avenues at the intersection of control, human–machine interaction, and behavioural adaptation.

Part I—The Artificial Race Driver Framework

Chapter 2

System Overview and Design

Abstract

This chapter introduces the *Artificial Race Driver* (ARD), a modular framework for autonomous racing that combines predictive planning, learning-based control, and validated vehicle models. ARD is organized hierarchically, with a high-level motion planner generating minimum-time trajectories based on a lightweight kineto-dynamical model of the vehicle. These trajectories are then executed by low-level controllers, which compute steering and acceleration actions in real-time through physics-informed learning-based modules. To support development and evaluation, ARD includes a high-fidelity digital twin—a validated vehicle dynamics model used in simulation—which can easily be replaced by a real vehicle when the system is deployed on actual hardware. This chapter provides an architectural overview of ARD, detailing the interactions among its core components and the design principles that enable its real-time performance. Emphasis is placed on modularity, interpretability, and the balance between computational efficiency and physical accuracy, setting the foundation for the following chapters on planning, control, and modelling.

Contents

2.1 Introduction 24
2.2 System Architecture 25
2.2.1 High-Level Motion Planner 26
2.2.2 Low-Level Control Layer 28
2.2.3 Digital Twin and Real-Vehicle Interface 28
2.2.4 Integration with the Learning Framework 28
2.2.5 Communication Architecture and Runtime Infrastructure 29
2.2.6 Simulation and Integration Testing Environment 30
2.3 Engineering Decisions and Real-Time Trade-Offs 30
2.3.1 Modular vs. Monolithic Architecture 30
2.3.2 Broker-Centric Communication: Flexibility vs. Centralization 32
2.3.3 Sync vs. Async Execution: Determinism vs. Responsiveness 33
2.3.4 Choosing Custom Middleware over ROS 34

2.1 Introduction

Autonomous racing poses a unique set of challenges that distinguish it from traditional autonomous driving. The goal is not only to navigate safely, but to extract the highest possible performance from the vehicle, operating near the limits of handling. These requirements demand a system capable of fast and reliable decision-making, tight integration with vehicle dynamics, and real-time replanning capabilities—all within a physically accurate and computationally efficient framework.

ARD is a hierarchical architecture developed to address these challenges. It consists of three main components:

- a high-level motion planner that generates minimum-time trajectories using a kineto-dynamical vehicle model;
- a set of low-level controllers that translate these trajectories into real-time steering and acceleration commands, leveraging physics-informed neural networks;

- and a validated digital twin that models the full vehicle dynamics for identification and simulation purposes.

When deployed on real hardware, the digital twin is replaced by the actual vehicle system, enabling seamless transition from simulation to real-world experiments.

The work presented in this thesis builds on and significantly extends the original ARD framework developed by Piccinini during his doctoral research [152]. While we collaborated during the initial phases of the project—particularly on time-optimal planning and neural network-based control—this version of ARD reflects a complete redesign. The architecture was revised to improve modularity, support high-frequency real-time control, and enable easier integration with both simulation tools and real vehicles. Every component was rewritten from scratch to better support this vision, including middleware interfaces, identification pipelines, and simulation tools. Additionally, new concepts were introduced at both the modelling and control levels, leading to more flexible learning-based controllers and vehicle models. As a result, the framework now constitutes a unified pipeline that can operate seamlessly from simulation to on-vehicle testing.

This chapter provides a system-level overview of ARD. Section 2.2 presents the architecture in detail, describing the roles and interactions of each module, while Section 2.3 outlines the key design principles, with particular attention to real-time performance, modularity, and the integration of model-based and learning-based components. The following chapters will then examine the three core aspects of ARD—planning, control, and modelling—in greater depth.

2.2 System Architecture

ARD, illustrated in Figure 2.1, is a modular and hierarchical framework designed for autonomous racing. Its architecture is tailored to the unique demands of high-performance driving: rapid and repeatable decision-making, and close integration with physically accurate vehicle behaviour.

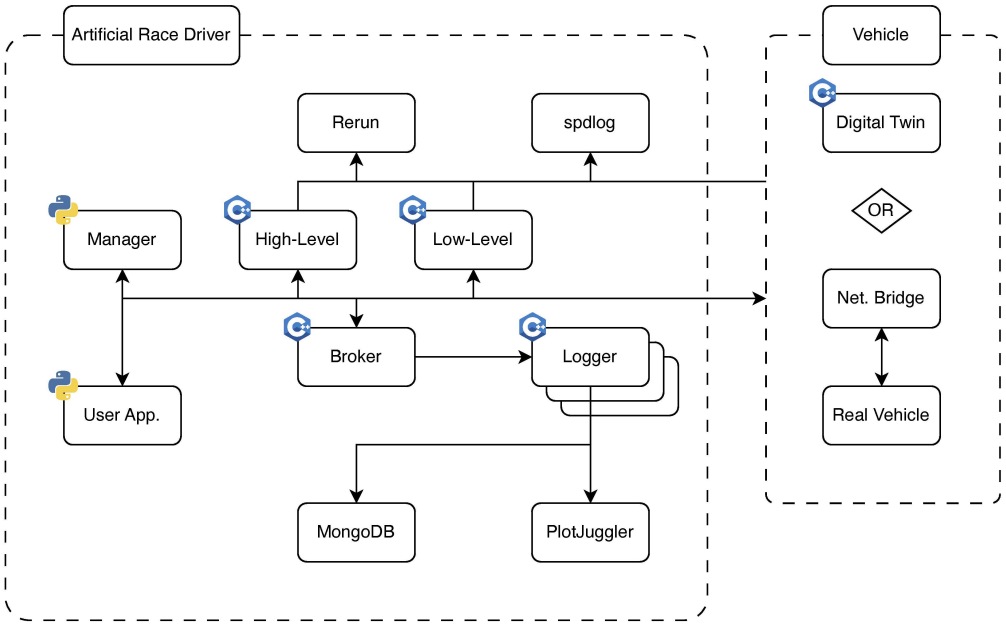


Figure 2.1: Overall ARD system diagram.

ARD is composed of three primary modules: a high-level planner, a low-level control stack, and a vehicle simulator. These components communicate through a central broker, enabling strict decoupling and real-time data exchange. Around this core, a custom-built

infrastructure provides logging, visualization, and system management. All data passing through the broker is captured by a distributed logging system, which records messages to a MongoDB [153] backend and optionally streams them to PlotJuggler [154] for real-time analysis. Each module also maintains local logs via spdlog [155] and can stream structured internal state data to Rerun [156] for storage and live introspection. An example of such a visualization is given in Figure 2.2.

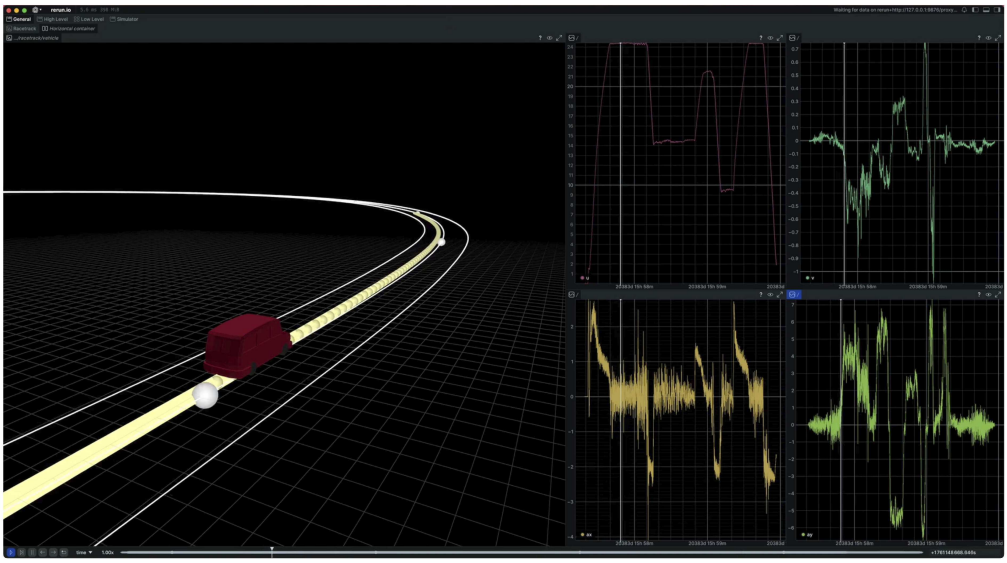


Figure 2.2: Example visualization of ARD via Rerun

The entire system is designed to be remotely controlled. A lightweight manager interface—currently implemented in Python for flexibility and rapid prototyping—can be used to run and configure experiments. This interface is message-driven and fully decoupled from the rest of the stack, allowing alternative implementations (*e.g.,* in C++) when runtime performance of the manager is critical.

Although the digital twin serves as the primary testbed during development, it can be seamlessly replaced by a real vehicle in real-world deployments. Communication with the physical platform can occur either through ARD's native messaging or via a dedicated *Robotic Operating System* (ROS) [157] bridge. Extensions to other communication protocols can of course be easily implemented.

The following sections present a closer look at each of ARD's main components, their interactions, and the rationale behind their design.

2.2.1 High-Level Motion Planner

At the top of the ARD hierarchy lies the high-level motion planner (Figure 2.3a), responsible for computing minimum-time trajectories that push the vehicle to its physical limits while ensuring feasibility and safety. This module is built around an *Economic Nonlinear Model Predictive Control* (E-NMPC) formulation, tailored for real-time operation in racing scenarios.

Following a receding-horizon paradigm, the planner solves a constrained optimization problem over a finite spatial window at each cycle. It continuously replans based on the most recent state estimates, adapting to control deviations and changes in the environment. The result is a full time-parameterized trajectory—including future poses, velocities, and curvature profiles—which is broadcast through the broker without regard for downstream consumer needs. This decoupled design ensures modularity: the planner focuses solely on generating optimal motion, while the low-level controller extracts and interprets the information it requires.

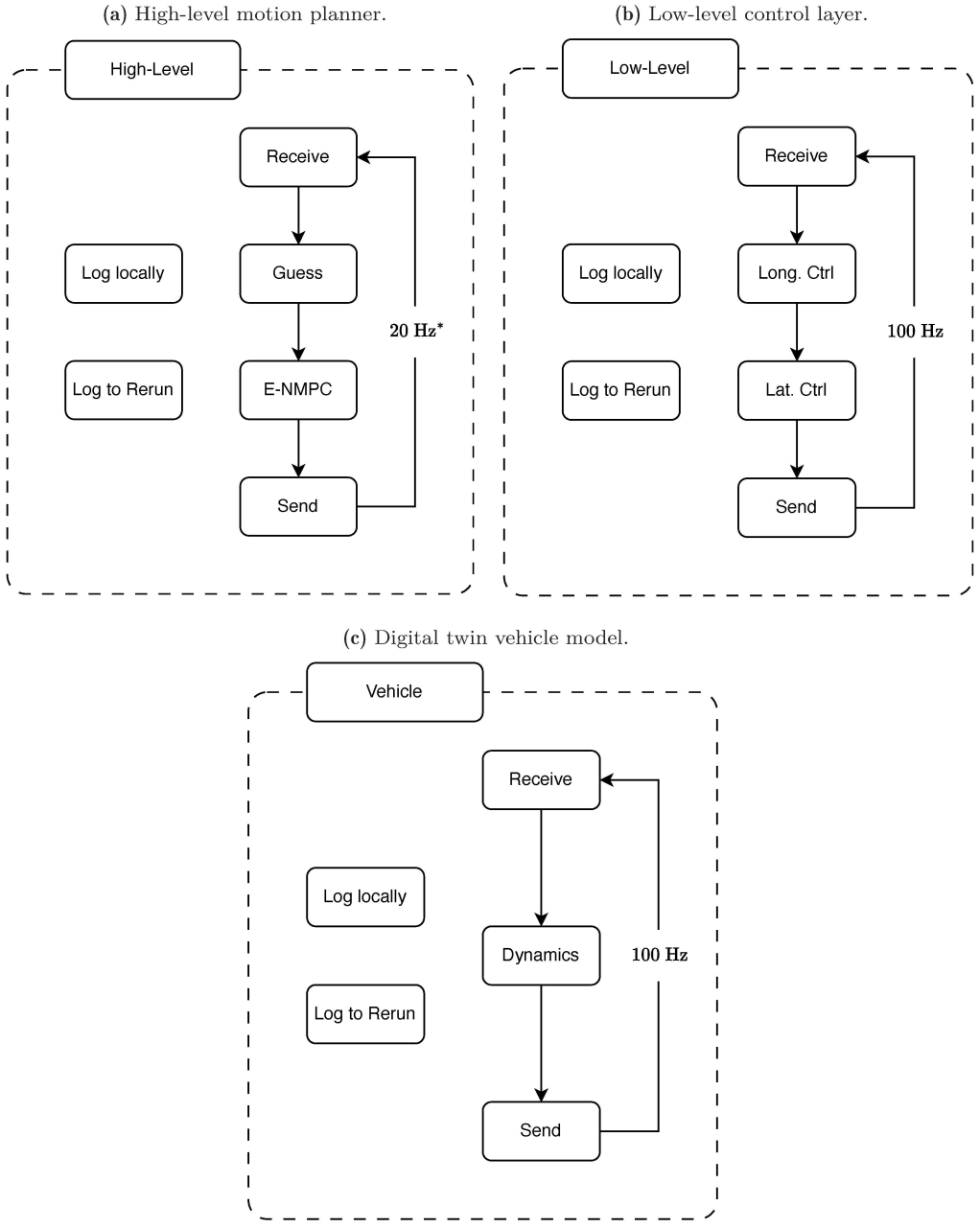


Figure 2.3: Core modules of the ARD system.

In typical use the planner runs synchronously at a fixed rate of 20 Hz, which simplifies analysis and ensures deterministic behaviour. In this mode the solver is allowed to take as long as needed to converge, since all other modules wait for its output, and the cycle time is assumed at 20 Hz regardless of the actual computation time. In asynchronous operation—whether in simulation or on a real vehicle—the planner is executed at a configurable maximum frequency (typically 20 Hz). The solver may exceed the nominal cycle time, in which case the effective update rate decreases. The maximum frequency can also be removed entirely, allowing the solver to run as fast as possible; this can be advantageous in some cases, as consecutive solutions are closer together and warm-starting from the previous trajectory becomes more effective. Whether a minimum cycle time is enforced therefore depends on the application.

2.2.2 Low-Level Control Layer

The low-level control layer (Figure 2.3b) is responsible for translating the planned minimum-time trajectory into real-time steering and acceleration commands. Designed for high responsiveness, this module can run up to 1 kHz, but is usually capped at the highest achievable frequency on physical platforms (*e.g.,* 100 Hz on the *Universität der Bundeswehr München* (UniBW) vehicles), ensuring fast feedback and stability even in high-performance driving.

The control stack is divided into two main subsystems: a lateral controller based on a physics-informed neural network, and a longitudinal controller implemented as a learnable *Proportional-Integral-Derivative* (PID).

The lateral controller predicts the steering input required to follow the reference trajectory. It uses a learned model that maps trajectory features to either wheel or steering wheel angle, depending on the configuration. This feedforward prediction is paired with a learned PID controller that applies local feedback corrections based on tracking error. The curvature reference for feedback can reflect either the steady-state or transient dynamics of the vehicle, depending on available telemetry.

The longitudinal controller governs longitudinal acceleration commands. It consists of a learnable PID which tracks the desired longitudinal speed.

As with the rest of ARD, this module adheres to a publish-subscribe model and operates independently of the planner's update rate. It can be run either synchronously—waiting for required data to ensure deterministic behaviour—or asynchronously, reacting to new data as it becomes available. This flexibility supports both reproducible debugging and real-world deployment.

2.2.3 Digital Twin and Real-Vehicle Interface

The digital twin in ARD (Figure 2.3c) acts as the reference environment for training, validation, and evaluation. It simulates full-vehicle behaviour using a high-fidelity neural vehicle model that has been validated against real-world telemetry data.

Within the system, the digital twin is treated as a generic black-box vehicle model: it receives control commands—steering and longitudinal acceleration—and outputs vehicle telemetry such as pose, velocity, and acceleration. Its integration interface is deliberately minimal, enabling the model to be swapped without disrupting the rest of the architecture. This design supports seamless transitions between simulation and real-world deployment.

The digital twin module is implemented to run up to 1 kHz, matching the frequency of the low-level control stack and ensuring realistic closed-loop interactions. However, this module is also usually capped at the highest achievable frequency on physical platforms (*e.g.,* 100 Hz on the UniBW vehicles).

To deploy ARD on a physical vehicle, the digital twin is simply replaced with an interface module that transmits control commands to the real system and receives telemetry in return. If the vehicle supports ARD's native communication stack, no adaptation is needed—as is the case for our custom driving simulator, described in Appendix A. If the vehicle relies on a different middleware, a lightweight bridge module suffices. For example, to integrate with UniBW's test vehicles, which use ROS 1 (Noetic) [157], ARD includes a dedicated ROS 1 bridge. This makes it possible to interface with any ROS-based robotic platform with minimal integration effort.

2.2.4 Integration with the Learning Framework

Although ARD operates independently of the learning process, its modular design enables the seamless integration of learned components identified from minimal human-driven data. In this workflow, the learning pipeline functions as an upstream stage that produces trained models subsequently deployed within the real-time system.

As illustrated in Figure 2.4, human-driven sessions provide the raw telemetry used to identify and train individual modules, such as vehicle dynamics models and low-level

controllers. Once learned, these models are exported in a compact runtime format (*e.g.,* ONNX) and loaded directly by their corresponding modules in ARD. The framework automatically detects and employs the latest available models without modification to the surrounding architecture.

This separation ensures that the learning process never interferes with real-time operation while enabling continuous refinement of the underlying components. Each subsystem—planner, controllers, or simulator—can therefore be fine-tuned or replaced independently.

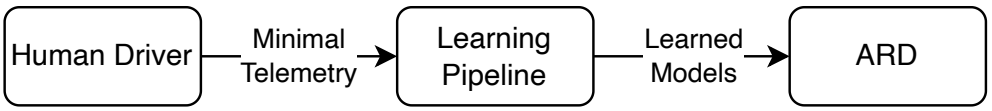


Figure 2.4: High-level interaction between the learning pipeline and the ARD framework. Data collected from human-driven runs are used offline to train and identify models, which are then deployed within the real-time system.

2.2.5 Communication Architecture and Runtime Infrastructure

The modular components of ARD communicate through a custom infrastructure based on ZeroMQ [158] and FlatBuffers [159], as illustrated in Figure 2.1. At its core lies a broker-centric PUB/SUB architecture: each module publishes its outputs and subscribes to the data it requires, without directly issuing commands or making requests. This enforces strict decoupling between modules, which in turn promotes modularity, reusability, and flexibility.

A central principle of this design is that no module controls another. Instead, every component broadcasts its state or outputs to the broker, while other modules subscribe only to the streams relevant to their operation. This ensures that no producer depends on a particular consumer and vice versa, allowing modules to be swapped or extended independently. For example, the high-level planner publishes full time-parameterised trajectories without knowledge of how they will be used; the low-level controller subscribes and extracts only the variables it requires (*e.g.,* curvature, velocity). Likewise, the low-level controller outputs generic actuation signals (*e.g.,* steering and desired acceleration), and the vehicle interface interprets them according to the target platform. This pattern is applied consistently in both simulation and real-vehicle deployments.

Messages are encoded with FlatBuffers for zero-copy binary transmission, enabling fast serialization with minimal overhead. Each message carries a timestamp, either provided by the sender or injected by the logger. The broker does not alter or inspect the data, acting purely as a relay.

Observability is supported through a distributed logging infrastructure. Dedicated logger modules can subscribe to the broker, convert passing messages to JSON [160], and store them in a MongoDB instance. These logs are used for post-processing, evaluation, or real-time visualization with tools such as PlotJuggler [154]. Because loggers are external, they can be started, stopped, or deployed across machines without affecting the core system. In addition, each module can stream internal states to Rerun or record local diagnostics with spdlog. This combination of external message-level introspection and internal module-level diagnostics provides comprehensive insight during development, while allowing unnecessary logging to be disabled in real deployments to meet real-time requirements.

Although all modules run asynchronously during deployment, synchronous operation is supported for development and debugging. In this mode, modules wait for incoming data before proceeding, which simplifies integration tests and ensures reproducibility. This flexibility allows ARD to span different use cases—from controlled debugging sessions to high-speed simulation and real-time vehicle control. To ensure safety, the low-level controller includes an emergency mode that activates when no new planned trajectory is

received and the current one has been fully executed. In this condition, the controller commands full braking and centres the steering, bringing the vehicle to a safe stop.

2.2.6 Simulation and Integration Testing Environment

Simulation is a cornerstone of the ARD framework, supporting continuous development, validation, and debugging before deployment on real vehicles.

The stack supports multiple levels of fidelity, depending on the development stage. At the base is the digital twin vehicle model, which serves as the core closed-loop simulation layer. This lightweight configuration enables rapid iteration and can run faster than real-time, making it well suited for testing new features and control strategies efficiently.

The next level adds real-time 3D visualization via Rerun. Beyond plotting internal module states (*e.g.,* steering commands, vehicle pose), Rerun provides spatial visualization of the vehicle's motion, the planned trajectory, and the executed path. This is valuable for diagnosing discrepancies not evident from logs alone. Despite the visual overhead, this configuration can still operate faster than real-time. Figure 2.2 shows an example of such a visualization.

For high-fidelity evaluation, ARD is integrated with a customized version of the Carla simulator [161]. Carla is modified to support external vehicle dynamics within the Unreal Engine environment [162], reducing latency while retaining visual realism. Custom racetracks, such as a digital reconstruction of the Mugello circuit, were developed to enable repeatable experiments under realistic conditions.

All simulation modes support synchronous execution, which is useful for debugging and deterministic behaviour. However, asynchronous operation is essential for realistic evaluation. Running ARD on a laptop and Carla on a separate machine emulates the distributed nature of real automotive systems, exposing the framework to message delays and network latency. This topology mirrors the configuration used in our custom driving simulator, shown in Appendix A, where visualization and control are also distributed across distinct platforms.

This progressive simulation pipeline—from lightweight closed-loop tests to full-system distributed evaluation—allows ARD to transition smoothly into real-world experiments. The framework is thus validated not only under ideal conditions but also in latency-rich environments, strengthening its readiness for deployment.

2.3 Engineering Decisions and Real-Time Trade-Offs

The previous sections described the structure and components of ARD in functional terms. This section shifts the focus to the key architectural and engineering decisions that shaped its design. Rather than relying on off-the-shelf frameworks, ARD was built from the ground up to meet the specific demands of autonomous racing: modularity, real-time responsiveness, ease of debugging, and seamless simulation-to-reality deployment. Each of these goals required deliberate trade-offs. The following subsections reflect on those trade-offs, providing rationale for the communication model, execution strategies, middleware choices, and testing workflow adopted throughout the project.

2.3.1 Modular vs. Monolithic Architecture

The decision to adopt a modular architecture in ARD was primarily driven by the need for flexibility in developing and integrating multiple subsystems (*e.g.,* alternative low-level controllers), and by the practical advantages of multiprocessing over multithreading. While a monolithic design—here intended as a single-process implementation where all functionalities are compiled into one executable—has appealing strengths, it ultimately presents limitations that conflict with the goals of real-time autonomous racing.

Advantages of a monolithic architecture:

- **Performance:** Communication between components incurs virtually zero latency and compiler optimizations can be applied globally.
- **Rapid prototyping:** In early development stages, integrating subsystems is straightforward and does not require designing explicit communication protocols or interfaces.

Drawbacks of a monolithic architecture:

- **Scalability limitations:** Even small modifications in a single subsystem require recompiling and restarting the entire application. As the system grows, this tight coupling makes design decisions increasingly complex and difficult to manage.
- **Complex multithreading:** Real-time performance often requires multithreading, which introduces concurrency issues that are difficult to test, debug, and maintain.
- **Language constraints:** All code must be written in the same language[1], limiting the ability to prototype or optimize individual modules using domain-appropriate tools.
- **Deployment inflexibility:** Subsystems cannot be distributed across multiple machines, making it difficult to balance computational load or mirror real-world deployment conditions.

Given these trade-offs, ARD was designed around a modular, node-based architecture in which each subsystem is a fully independent application communicating over the network. This design comes with its own challenges.

Drawbacks of a modular architecture:

- **Steeper initial development:** Building a modular system requires familiarity with networking, data serialization, and inter-process communication.
- **Higher overhead:** Each node must handle its own lifecycle, messaging, and state management, resulting in additional code and maintenance effort.
- **Lower theoretical performance:** Inter-process messaging can introduce latency, memory copies, and scheduling unpredictability, particularly when spread across machines.

Benefits of a modular architecture:

- **Parallel development and testing:** Nodes can be developed, launched, and even hot-swapped independently.
- **Better system-level load balancing:** The operating system can manage CPU affinity and resource scheduling automatically.
- **Distributed operation:** Nodes can run on different machines without modification.
- **Multi-language support:** Modules can be implemented in any language, enabling rapid prototyping and fine-tuned optimization.
- **Minimal multithreading:** Each node typically requires only lightweight concurrency (*e.g.,* for background I/O or logging), simplifying logic and reducing risk.
- **Scalable latency control:** While network-based communication may introduce latency, this can be minimized using inter-process transport methods like IPC or shared memory.

[1]This is partially true, as different languages can be compiled and linked together, but that is out of the scope of the current discussion.

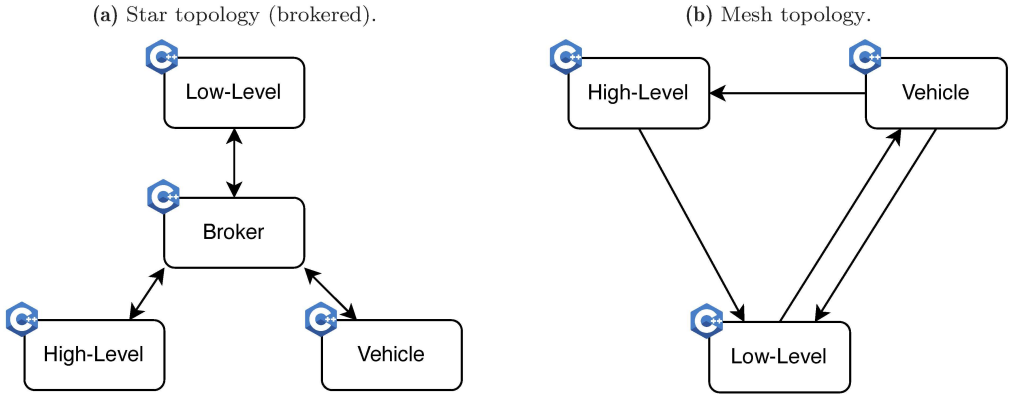


Figure 2.5: Illustrative comparison between star and mesh topologies applied to ARD's core modules.

Given these considerations—and the observation that even monolithic applications must eventually interface with external systems over a network during real-world deployment—we opted for a modular design. This choice has proven essential for enabling rapid experimentation, robust debugging, and deployment flexibility across both simulation and physical platforms.

2.3.2 Broker-Centric Communication: Flexibility vs. Centralization

A key design decision in ARD was the choice of communication topology. In general, two broad categories exist: centralized topologies (*e.g.,* star) and decentralized ones (*e.g.,* mesh) [163]. In the context of research systems, decentralized mesh-like structures are commonly adopted. This is often not because of their long-term advantages, but rather due to their simplicity during initial development.

To illustrate this, consider the three core modules of ARD: the high-level planner, the low-level controller, and the digital twin (Figure 2.5). Each of these needs to communicate with the others: the planner requires state feedback from the vehicle, the controller needs both plans and telemetry, and the simulator needs actuation commands. The intuitive solution is to connect these nodes directly, forming a small mesh (Figure 2.5b).

A mesh topology comes with some appealing advantages:

- **Minimal latency:** Messages travel directly from sender to receiver in a single hop.
- **Failure isolation:** If a node crashes, it does not bring down the rest of the network.

However, these advantages come at significant costs:

- **Tight coupling:** Each node must be manually configured with the addresses of the nodes it depends on.
- **Poor scalability:** Changing or replacing a node often requires modifying others, particularly across machines.
- **Difficult peer discovery:** Dynamic environments or runtime node discovery are hard to support.
- **Limited flexibility:** Hot-plugging components or duplicating nodes for debugging is non-trivial.
- **Fragile in practice:** Although a node crash does not affect the network, it typically halts the system since the missing data is still critical.

To overcome these limitations, ARD adopts a centralized star topology (Figure 2.5a), placing a broker at the centre of the communication graph. This architecture comes with its own trade-offs:

- **Single point of failure:** If the broker crashes, the network collapses.
- **Higher baseline latency:** Each message incurs two hops: publisher to broker, broker to subscriber. However, when all modules run on the same machine, the resulting communication delay is negligible given the high throughput of local message passing.
- **Broker bottleneck:** The broker must handle the aggregate bandwidth of all messages.

However, these are outweighed by substantial benefits:

- **Loose coupling:** Nodes do not need to know about each other—only the broker—greatly simplifying configuration and deployment.
- **Automatic peer discovery:** Nodes can join and leave the system at runtime without reconfiguration.
- **Hot-plugging and redundancy:** Tools like loggers and visualizers can be launched dynamically and connected without affecting core execution.
- **Debugging and observability:** Centralized traffic enables powerful monitoring and logging strategies with minimal additional infrastructure.

This architecture reflects a trade-off between absolute minimal latency and system-level modularity, flexibility, and introspectability. While such flexibility can, in principle, introduce communication delays that may impact closed-loop stability, this aspect was explicitly investigated in simulation by introducing message dropouts and noise to assess the system's robustness under adverse conditions (see Section 8.6).

In real-time autonomous systems, these properties are not secondary: they are essential for safe iteration, reliable debugging, and robust deployment across diverse hardware setups. In practice, the added latency of a broker is negligible for ARD's update rates and well worth the architectural clarity it enables.

2.3.3 Sync vs. Async Execution: Determinism vs. Responsiveness

Another key architectural choice in designing a modular real-time system like ARD is how to coordinate the execution of its subsystems. In particular, a system must decide whether to operate synchronously—where each module runs at a fixed, coordinated rate (Figure 2.6a)—or asynchronously—where each module runs independently, at its own pace (Figure 2.6b). Both modes have their strengths and weaknesses, and ARD is explicitly designed to support both.

Synchronous execution is invaluable during development. It ensures deterministic timing: all modules receive fresh data at known intervals, and the entire system can be debugged with clear causality and reproducibility. Errors are easier to isolate, and behaviour across runs is consistent, which simplifies testing and validation.

However, the real world is not synchronous. Sensors, actuators, and compute units all operate on their own clocks, often at varying frequencies and with non-negligible delays. Once deployed on real hardware, systems must cope with message jitter and delays. For example, a control module cannot pause to wait for a new sensor reading—it must act based on the best information currently available. As such, ARD's modules are also designed to function asynchronously, consuming data when it is available and falling back to the last known values when it is not.

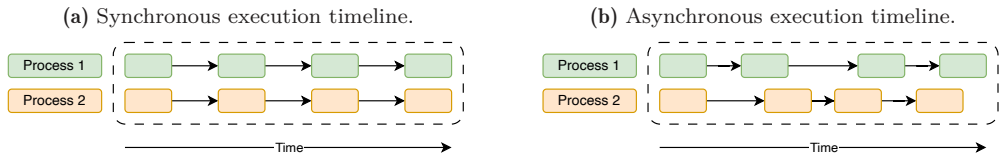


Figure 2.6: Illustration of synchronous (a) vs. asynchronous (b) execution timelines.

This dual-mode execution strategy plays a critical role in ARD's development and deployment cycle. During early-stage testing and simulation, synchronous execution is preferred to isolate issues and ensure consistent behaviour across iterations. Once confidence is gained, the same system is tested asynchronously to verify robustness to real-world conditions: timing noise, communication jitter, delayed messages.

In asynchronous mode, modules are designed to make decisions based on the most recent data available, rather than assuming fresh inputs will always arrive. In practice, this means using the last received message and updating it opportunistically.

By supporting both modes, ARD offers the best of both worlds: deterministic development and flexible, resilient deployment. This also eases the transition from simulation to real-world testing, reducing the need for architectural rewrites or unsafe assumptions about timing guarantees.

2.3.4 Choosing Custom Middleware over ROS

ROS is a cornerstone of modern robotics research. It offers an extensive feature set, broad community support, and an impressive collection of tools and libraries. In particular, ROS excels when all team members, robots, and infrastructure use the same Linux distribution and version, allowing for smooth integration and leveraging of its ecosystem.

However, for projects requiring cross-platform compatibility, real-time performance, or lightweight deployments, ROS can quickly become a limiting factor. Although ROS 2 addresses some of these concerns with improved middleware abstraction and modularity, platform support remains uneven. At the time of writing, the official ROS 2 Kilted documentation [164] (the current version of ROS 2) still does not provide prebuilt binaries for macOS—a significant barrier given that most of our development is conducted on macOS machines. Our group also frequently runs simulations on Windows and different Linux distributions, and occasionally targets embedded systems such as the Raspberry Pi. These heterogeneous constraints demanded a more portable and lightweight middleware solution.

For this reason, we opted to develop a fully custom middleware tailored to the needs of ARD. A detailed description of it is provided in Appendix A. At its core, the system is built on two robust open-source libraries: ZeroMQ for messaging and FlatBuffers for serialization. ZeroMQ provides a fast and flexible communication layer, eliminating many of the complexities typically associated with networking (*e.g.,* handshaking and reconnect logic). FlatBuffers enables zero-copy, schema-based serialization with minimal overhead, allowing real-time transmission of structured data without incurring the performance penalties of more well-known formats like Protocol Buffers [165] or JSON.

Everything else—including the broker, the logger, and the automatic code generation infrastructure—was developed in-house. This has given us full control over the performance characteristics, deployment model, and cross-platform support. As a result, the middleware can be compiled and executed on all major platforms, including macOS, Linux, Windows, and embedded systems like the Raspberry Pi. It also supports all major programming languages supported by ZeroMQ and FlatBuffers, allowing developers to for example prototype in Python or Go and later port performance-critical modules to C++ without changing interfaces or message definitions.

While developing custom infrastructure comes with an upfront cost, this approach has allowed us to sidestep the limitations of ROS while preserving modularity, real-time performance, and maintainability. In doing so, we have created a middleware that is purpose-built for autonomous racing, simulation, and experimentation in heterogeneous environments.

Chapter 3

Motion Planning

Abstract

This chapter describes the motion planning subsystem of the *Artificial Race Driver* (ARD), which computes minimum-time trajectories while respecting both physical limitations and environmental constraints. The planner is based on an *Economic Nonlinear Model Predictive Control* (E-NMPC) framework, using a kineto-dynamical point-mass vehicle model combined with learned g-g-v performance constraints. The chapter also briefly outlines our extension of the framework to three-dimensional road geometries.

Contents

3.1 Introduction 35
3.2 Problem Formulation 36
3.2.1 Planning Objective and Cost Function 37
3.2.2 Vehicle Modelling for Planning 39
3.2.3 Constraints 41
3.3 Implementation Details 43
3.3.1 Solver Integration 43
3.3.2 Warm-Starting and Horizon Management 44
3.4 On the Formulation and Effects of g-g-v Constraints 45
3.4.1 Motivation and Role in Planning 45
3.4.2 Structure and Hybrid Formulation 46

3.1 Introduction

Motion planning in autonomous racing differs fundamentally from that in conventional autonomous driving. Instead of prioritizing safety margins or passenger comfort, the objective is to complete a lap in the shortest possible time while operating the vehicle near its physical limits. This demands trajectories that are dynamically feasible and computationally tractable.

In the ARD framework, motion planning is handled by a high-level module that solves an E-NMPC problem in real-time. The formulation is based on a kineto-dynamical point-mass vehicle model, with feasibility enforced via a combination of physical constraints and a data-driven g-g-v envelope. This envelope encodes the maximum achievable longitudinal and lateral accelerations as a function of speed, capturing the nonlinear performance limits of the vehicle in a compact and efficient manner.

This chapter introduces the motion planning formulation adopted in ARD, with an emphasis on modelling assumptions, constraint structures, and real-time implementation choices. Particular attention is devoted to the formulation and role of the g-g-v constraints. While the main focus remains on flat circuits, we also recall the extension of the planner to three-dimensional roads, as introduced in our previous work [54].

On Planning over Three-Dimensional Roads

Real-world circuits often include non-negligible three-dimensional features such as slopes and banked corners, which can significantly influence vehicle dynamics and limit performance. These effects become especially relevant when operating near the physical boundaries of the vehicle, as is common in racing scenarios. Notable examples include the

Las Vegas Motor Speedway, with banking angles of up to 20°, Zandvoort (18° banking), and Eau Rouge at Spa (9° slope). In such cases, planar approximations fail to capture the true behaviour of the system.

Following prior work in the optimal control literature [166], [167], [168], we adopt a ribbon-based model of the road in $\mathbb{R}^3$ (Figure 3.1), in which the track is described as a sequence of locally planar sections aligned with a moving Frenet-Serret frame $\{\vec{\boldsymbol{\zeta}}, \vec{\boldsymbol{n}}, \vec{\boldsymbol{z}}\}$. The orientation of this frame along the curvilinear abscissa ζ is specified using yaw, pitch, and roll angles $\{\theta(\zeta), \theta_{\text{sl}}(\zeta), \theta_{\text{bk}}(\zeta)\}$, corresponding to heading, slope, and banking. Under a small-angle approximation—which holds for most public roads and circuits like Mugello, where angles remain below 5°—the resulting rotation matrix $\boldsymbol{R}_c(\zeta)$ simplifies to a linearized form. The rate of change of road orientation is then described by a skew-symmetric matrix $\boldsymbol{W}_c(\zeta) = \frac{d\boldsymbol{R}_c}{d\zeta}\boldsymbol{R}_c^\top$, which leads to the curvature relations:

$$\begin{aligned} \frac{d\theta}{d\zeta} &= \kappa + \theta_{\text{bk}}\, \upsilon, \\ \frac{d\theta_{\text{sl}}}{d\zeta} &= \upsilon - \kappa\, \theta_{\text{bk}}, \\ \frac{d\theta_{\text{bk}}}{d\zeta} &= \tau + \kappa\, \theta_{\text{sl}}, \end{aligned} \tag{3.1}$$

where κ, υ, and τ represent the geodesic, sagittal, and frontal curvatures of the road. Together with the initial conditions and width profiles $M_L(\zeta)$, $M_R(\zeta)$, these functions fully define the 3D road geometry.

This 3D modelling approach was implemented and validated as part of our previous work in [54]. Although ARD is fully capable of planning over such terrains in real-time, the focus of this thesis remains on flat circuits. The 3D extension will therefore be discussed only briefly after each relevant section (*e.g.*, the g-g-v extension to 3D roads).

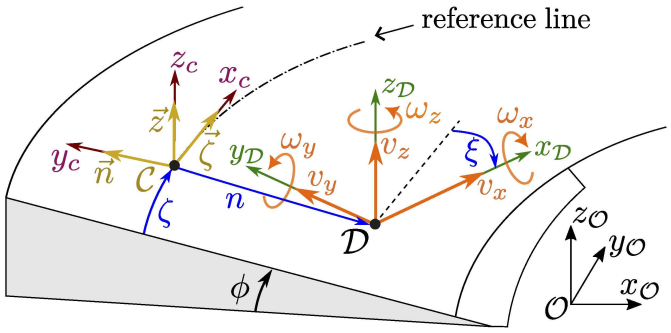


Figure 3.1: Ribbon-based 3D road model: the road centreline is framed by local orientation vectors $\vec{\boldsymbol{\zeta}}, \vec{\boldsymbol{n}}, \vec{\boldsymbol{z}}$, with slope θ_{sl} and banking θ_{bk}. Note that, for clarity, the signs of n and ξ have been inverted in this illustration; this is not the case in practice.

3.2 Problem Formulation

The motion planning task addressed in this chapter is formulated as a constrained optimal control problem, solved online within an E-NMPC framework. The objective is to compute minimum-time trajectories that are dynamically feasible, respect the road geometry, and operate the vehicle near its performance limits.

This section introduces the overall structure of the formulation used in ARD. It begins by defining the optimal control problem and its key components, including the cost function, vehicle model, and constraint set. We then present the kineto-dynamical vehicle model in detail, which describes the system evolution in curvilinear coordinates and provides a balance between physical expressiveness and computational tractability. The formulation of the cost function is discussed in detail, including the treatment of initial

and terminal conditions, as well as integral terms. Finally, we describe the constraint set, which encodes track boundaries, actuation limits, and dynamic feasibility via a learned g-g-v envelope.

3.2.1 Planning Objective and Cost Function

The objective of ARD's motion planner is to compute dynamically feasible, minimum-time trajectories within a receding-horizon framework. To this end, the problem is posed as an E-NMPC problem, where the planner directly minimizes travel time rather than tracking a predefined reference.

At each planning step, the system solves an *Optimal Control Problem* (OCP) over a finite spatial horizon. The optimization minimizes a cost function composed of a Mayer term and a Lagrange term, while satisfying dynamic constraints, boundary conditions, track limits, and a learned g-g-v envelope representing the vehicle's performance capabilities. The underlying dynamics are governed by a kineto-dynamical point-mass model (see Section 3.2.2), and all quantities are expressed in curvilinear coordinates (see Paragraph 3.2.1).

The OCP takes the form:

$$\min_{\boldsymbol{u}(\zeta)} \quad \Phi(\zeta_f) + \Phi(\zeta_i) + \int_{\zeta_i}^{\zeta_f} \mathcal{L}(\boldsymbol{x}(\zeta), \boldsymbol{u}(\zeta))\, d\zeta \tag{3.2a}$$

$$\text{s.t.} \quad \frac{d\boldsymbol{x}(\zeta)}{d\zeta} = \boldsymbol{f}(\boldsymbol{x}(\zeta), \boldsymbol{u}(\zeta)), \quad \text{kineto-dynamical model,} \tag{3.2b}$$

$$\boldsymbol{b}(\boldsymbol{x}(\zeta_i)) = 0, \quad \text{initial boundary conditions,} \tag{3.2c}$$

$$n(\zeta) \in \mathcal{R}(\zeta), \quad \text{track boundaries,} \tag{3.2d}$$

$$(\boldsymbol{x}(\zeta), \boldsymbol{u}(\zeta)) \in \mathcal{G}(\cdot), \quad \text{g-g-v envelope} \tag{3.2e}$$

The state and control vectors are:

$$\boldsymbol{x}(\zeta) = \begin{bmatrix} n(\zeta) & \xi(\zeta) & v_x(\zeta) & v_y(\zeta) & a_x(\zeta) & \Omega(\zeta) \end{bmatrix}^\top$$
$$\boldsymbol{u}(\zeta) = \begin{bmatrix} a_{x_{\text{ctrl}}}(\zeta) & \Omega_{\text{ctrl}}(\zeta) \end{bmatrix}^\top.$$

Here, n is the lateral deviation from the reference path, ξ the relative yaw angle, v_x and v_y the longitudinal and lateral velocities, a_x the longitudinal acceleration, and Ω the yaw rate. The controls $a_{x_{\text{ctrl}}}$ and Ω_{ctrl} are the commanded longitudinal acceleration and yaw rate, respectively. All variables are defined in the vehicle's body-fixed frame, consistent with ISO 8855 [169] (with x pointing forward, y to the left, and z upward, forming a right-handed coordinate system).

Curvilinear Coordinates. To remove the need to explicitly optimize over the final time, the planning problem is expressed in curvilinear coordinates. This reformulation replaces time as the independent variable with the curvilinear abscissa ζ, which denotes the arc length along the track centreline. The additional states introduced by this formulation are:

$$\begin{aligned} \dot{\zeta}(t) &= \frac{v_x(t)\cos(\xi(t)) - v_y(t)\sin(\xi(t))}{1 - n(t)\kappa(\zeta(t))}, \\ \dot{n}(t) &= v_x(t)\sin(\xi(t)) + v_y(t)\cos(\xi(t)), \\ \dot{\xi}(t) &= \Omega(t) - \kappa(\zeta(t)) \cdot \dot{\zeta}(t), \end{aligned} \tag{3.3}$$

where n is the lateral deviation from the track centreline, ξ is the relative yaw angle between the vehicle heading and the local orientation of the track, and κ is the curvature of the centreline.

These kinematic equations are well established in the racing literature and are particularly effective for enforcing track constraints and constructing cost functions

in a spatially invariant form. They also simplify horizon definition and allow the optimal control problem to be solved over a fixed spatial window.

The full state vector in time-parametrised form becomes:

$$\boldsymbol{x}(t) = \begin{bmatrix} \zeta(t) & n(t) & \xi(t) & v_x(t) & v_y(t) & a_x(t) & \Omega(t) \end{bmatrix}^\top$$

which reduces to:

$$\boldsymbol{x}(\zeta) = \begin{bmatrix} n(\zeta) & \xi(\zeta) & v_x(\zeta) & v_y(\zeta) & a_x(\zeta) & \Omega(\zeta) \end{bmatrix}^\top$$

once the system is reformulated with respect to the arc length ζ.

Mayer Term: Soft Boundary Conditions

The Mayer term softly enforces desired initial and final conditions:

$$\sum_{j=1}^{6} \mathcal{W}_{f_j} \left(\frac{\boldsymbol{x}_j(\zeta_f) - \boldsymbol{x}_{j_f}}{\boldsymbol{x}_{j_{\max}}} \right)^2 + \sum_{j=1}^{6} \mathcal{W}_{i_j} \left(\frac{\boldsymbol{x}_j(\zeta_i) - \boldsymbol{x}_{j_i}}{\boldsymbol{x}_{j_{\max}}} \right)^2, \tag{3.4}$$

where $\boldsymbol{x}_j$ is the j-th component of the state vector, ζ_i and ζ_f denote the initial and final abscissae, $\boldsymbol{x}_{j_i}$ and $\boldsymbol{x}_{j_f}$ are the corresponding desired values, and $\boldsymbol{x}_{j_{\max}}$ are normalizing constants used to balance the influence of each state. The weights $\mathcal{W}_{i_j}$ and $\mathcal{W}_{f_j}$ determine the relative importance of enforcing the initial and final conditions for each state component.

These soft boundary conditions provide resilience against noise and modelling errors, particularly in real-world settings where state measurements—especially accelerations—may be unreliable. Enforcing exact values in such conditions often leads to instability or infeasibility. The soft formulation allows deviations when beneficial, while discouraging unphysical behaviour. For instance, without appropriate weights, the optimizer might exploit the initial conditions to begin the horizon with unrealistic speeds or accelerations. Tuning these weights is therefore essential for balancing robustness and realism.

Typically, the terminal conditions are imposed only on a subset of the state, such as n and ξ, to loosely guide the final vehicle orientation. These can be kept equal to their last value at the end of the horizon, fixed to the centreline values, estimated from a clothoid-based guess or a short forward integration of the dynamics, following the approach in [97]. While not strictly necessary, these conditions have been found to improve convergence in practice.

Lagrange Term: Time, Continuity, and Smoothness

The Lagrange term accumulates cost terms over the planning horizon:

$$\int_{\zeta_i}^{\zeta_f} w_T + \mathcal{P}_\mathcal{C}(\boldsymbol{x}(\zeta)) + \mathcal{W}_{da_x} \left(\frac{da_x}{d\zeta} \right)^2 + \mathcal{W}_{d\Omega} \left(\frac{d\Omega}{d\zeta} \right)^2 \, \mathrm{d}\zeta. \tag{3.5}$$

The first term, w_T, is a constant that drives the minimum-time behaviour, which forms the core of the E-NMPC approach. The second term enforces spatial consistency by penalizing deviation from part of the previous solution:

$$\mathcal{P}_\mathcal{C}(\boldsymbol{x}_j(\zeta)) = \sum_{j=1}^{6} \left(\frac{\boldsymbol{x}_j(\zeta) - \boldsymbol{x}_j^{\text{prev}}(\zeta)}{\boldsymbol{x}_{j_{\max}}} \right)^2 e^{-\frac{(\zeta - \zeta_0)}{\alpha}\lambda}. \tag{3.6}$$

This term improves warm-starting and reduces solver oscillations across iterations. The decay parameters λ and α limit the influence of this term further down the horizon, allowing the solution to evolve freely where needed. A visualization that shows the effects of $\mathcal{P}_\mathcal{C}$ is shown in Figure 3.2.

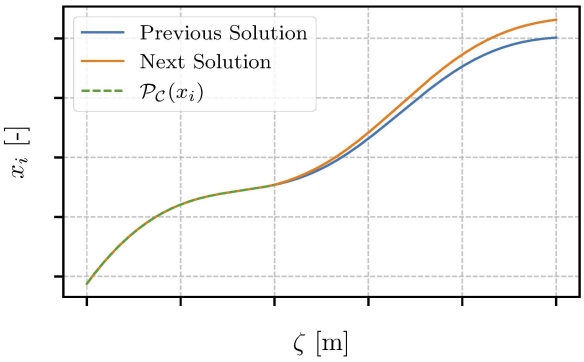


Figure 3.2: Effects of the continuity penalty $\mathcal{P}_\mathcal{C}$ on state x_i from the previous solution to the next one.

Finally, the squared derivatives of a_x and Ω reduce control chatter by limiting sharp variations in the commanded acceleration and yaw rate. These terms are weighted by $\mathcal{W}_{da_x}$ and $\mathcal{W}_{d\Omega}$, respectively. Since both a_x and Ω are governed by first-order dynamics (as we will see in Section 3.2.2), enforcing their smoothness indirectly promotes smooth control behaviour without explicitly penalizing the controls.

3.2.2 Vehicle Modelling for Planning

This section presents the kineto-dynamical vehicle model used within ARD's high-level planner. The formulation is based on our prior work [25] but has been adapted to prioritize real-time performance and ease of identification directly from telemetry data. While it abstracts away most of the classical vehicle dynamics, the model retains sufficient expressiveness to support time-optimal planning near the physical limits, without relying on detailed tyre models or chassis parameters.

The system is described by four first-order differential equations:

$$\dot{v}_x(t) = a_x(t), \tag{3.7a}$$
$$\tau_{v_y}\dot{v}_y(t) + v_y(t) = \mathcal{F}_{v_y}(a_y(t), v_x(t), a_x(t)), \tag{3.7b}$$
$$\tau_\Omega\dot{\Omega}(t) + \Omega(t) = \Omega_{\text{ctrl}}(t), \tag{3.7c}$$
$$\tau_{a_x}\dot{a}_x(t) + a_x(t) = a_{x_{\text{ctrl}}}(t), \tag{3.7d}$$

where $a_y(t) = v_x(t)\Omega(t)$ is the approximated lateral acceleration in the vehicle's body frame. The state and control vectors are:

$$\boldsymbol{x}(t) = \begin{bmatrix} v_x(t) & v_y(t) & \Omega(t) & a_x(t) \end{bmatrix}^\top,$$
$$\boldsymbol{u}(t) = \begin{bmatrix} a_{x_{\text{ctrl}}}(t) & \Omega_{\text{ctrl}}(t) \end{bmatrix}^\top.$$

All signals are expressed in the vehicle's body-fixed reference frame, as defined by the ISO 8855 [169]. The planner augments this model with curvilinear coordinates $\{\zeta, n, \xi\}$ and with a reformulation in the curvilinear abscissa ζ, which was previously described in Section 3.2.1.

Model Overview. Each state evolves under a first-order relation with its respective input or steady-state target. This structure allows compact implementation and predictable transient responses. The time constants τ_{v_y}, τ_Ω, and τ_{a_x} are identified directly from telemetry data and capture the system's effective response delays. In contrast to physics-based models that rely on suspension and tyre parameters, this formulation learns effective input-output mappings, which are easier to adapt across different vehicles and racetracks.

Longitudinal Dynamics. The longitudinal velocity v_x is governed by a simple kinematic relation with the current acceleration a_x (3.7a), which itself follows a first-order lag driven by the commanded acceleration $a_{x_{\text{ctrl}}}$ (3.7d). This cascade structure introduces a delay consistent with observed actuator dynamics.

Effects such as aerodynamic drag, tyre saturation, and drivetrain limits are not explicitly modelled here. Instead, they are captured indirectly via the learned g-g-v constraint (see Section 3.4), which bound the feasible combinations of longitudinal and lateral accelerations.

Yaw Rate Dynamics. The yaw rate Ω is driven by a first-order response to the commanded yaw rate Ω_{ctrl} (3.7c). This avoids the complexity and parameter sensitivity of moment-based rotational dynamics, while still allowing the planner to exploit yaw transients during aggressive manoeuvres. Just like the longitudinal acceleration, the nonlinear effects are modelled and imposed on the yaw rate via the learned g-g-v constraint.

Lateral Velocity Dynamics and the Lateral Velocity Diagram. The lateral velocity v_y follows a first-order dynamic, where the steady-state value is provided by a nonlinear map $\mathcal{F}_{v_y}(a_y, v_x, a_x)$ (3.7b). This map, termed the *Lateral Velocity Diagram* (LVD), captures how v_y varies as a function of lateral acceleration a_y, longitudinal velocity v_x, and longitudinal acceleration a_x. Its structure is:

$$\begin{aligned}\mathcal{F}_{v_y}(a_y, v_x, a_x) = \quad & f_1(v_x)(1 + b_1 a_x + b_2 a_x^2) \cdot a_y \\ & + f_3(v_x)(1 + b_3 a_x + b_4 a_x^2) \cdot a_y^3 \\ & + f_5(v_x)(1 + b_5 a_x + b_6 a_x^2) \cdot a_y^5,\end{aligned} \tag{3.8}$$

where each function $f_i(v_x)$ is a cubic spline defined over the vehicle's operating speed range:

$$f_i(v_x) = \text{cubic spline}_i, \quad i \in \{1, 3, 5\}.$$

The coefficients b_k modulate the influence of longitudinal acceleration, allowing the lateral response to adapt under combined manoeuvres. Only odd powers of a_y are used to preserve symmetry across left and right turns. This structure enables smooth, efficient evaluation and ensures generalization across the entire dynamic envelope.

The diagram $\mathcal{F}_{v_y}$ is learned offline directly from telemetry data and does not require any dedicated test to be performed. Chapter 6 describes the learning procedure in detail. Figure 3.3 show an example representation of this diagram.

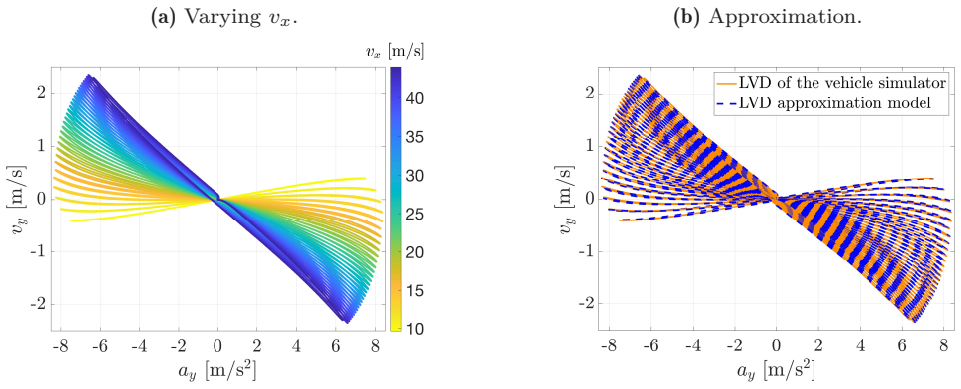


Figure 3.3: Lateral velocity diagram as a function of longitudinal speed: (a) varying v_x, (b) approximation. Figures taken from our work [25] © 2025 IEEE.

Remarks. The model (3.7) is referred to as *kineto-dynamical* because it merges kinematic and dynamic elements. Kinematic components define the fundamental geometry of motion, while the dynamic components are captured through first-order dynamics and steady-state maps derived from real data. This hybrid approach enables low computational cost and high representational power—both essential for fast E-NMPC-based planning.

Extension to 3D Roads

A key strength of the kineto-dynamical model adopted in ARD is that its formulation remains nearly invariant under 3D road geometry. The vehicle dynamics, expressed in curvilinear coordinates, do not require modification when transitioning from planar to spatially complex tracks. This is possible because the effects of slope and banking are fully captured at the constraint level, specifically through acceleration limits, rather than requiring changes to the system dynamics.

In particular, the vehicle is still modelled as a point-mass system evolving under curvilinear kinematics:

$$\begin{aligned} \dot{\zeta} &= \frac{v_x \cos(\xi) - v_y \sin(\xi)}{1 - n\kappa(\zeta)}, \\ \dot{n} &= v_x \sin(\xi) + v_y \cos(\xi), \\ \dot{\xi} &= \Omega - \kappa(\zeta)\dot{\zeta}, \end{aligned} \tag{3.9}$$

with no explicit dependence on slope θ_{sl} or banking θ_{bk}. These geometric effects instead modulate the net accelerations experienced by the vehicle due to gravity and local road curvature, and are incorporated directly into the feasible acceleration regions defined by the g-g-v constraints.

The only structural change in the formulation arises in the lateral velocity constraint, which depends on the lateral acceleration a_y, vehicle longitudinal speed v_x, longitudinal acceleration a_x, and—critically in the 3D case—vertical acceleration a_z. The extended constraint function becomes:

$$\begin{aligned} \tilde{F}_{v_y}(a_y, v_x, a_x, a_z) = \quad & f_1(v_x) \cdot (1 + b_1 a_x + b_2 a_x^2) \cdot (1 + c_1 a_z + c_2 a_z^2) \cdot a_y \\ & + f_3(v_x) \cdot (1 + b_3 a_x + b_4 a_x^2) \cdot (1 + c_3 a_z + c_4 a_z^2) \cdot a_y^3 \\ & + f_5(v_x) \cdot (1 + b_5 a_x + b_6 a_x^2) \cdot (1 + c_5 a_z + c_6 a_z^2) \cdot a_y^5, \end{aligned} \tag{3.10}$$

where the newly introduced b_{z_i} coefficients model the influence of vertical dynamics due to road slope and banking.

This design preserves the computational structure and solver efficiency of the original planner, while enabling ARD to plan on 3D tracks. Crucially, if the vertical acceleration is assumed zero, the formulation falls back to the 2D case. Therefore, it remains perfectly compatible with both 2D and 3D scenarios.

3.2.3 Constraints

3.2.3.1 Track Boundary Constraints

To ensure the vehicle remains within the drivable area, we impose hard constraints on the lateral deviation n from the track centreline, expressed in curvilinear coordinates as:

$$n_{\min}(\zeta) \leq n(\zeta) \leq n_{\max}(\zeta), \tag{3.11}$$

where $n_{\min}$ and $n_{\max}$ denote the left and right track boundaries, respectively. These are encoded as quintic splines fitted to recorded track margins, yielding smooth and differentiable constraints along ζ.

Constraint Relaxation Near the Horizon Start. In practice, the track constraints are slightly relaxed at the beginning of the planning horizon. This accounts for situations where the vehicle starts close to or marginally outside the feasible region, due to

noise, discretization, or recovery from off-track scenarios. Without this relaxation, the optimization problem may become infeasible, despite the situation being physically manageable.

The relaxed margins are defined via an exponentially decaying function:

$$\text{Right margin:} \quad n(\zeta) \geq w_R(\zeta) + \delta_R(\zeta) - 0.5 w_{\text{veh. width}}, \tag{3.12}$$

$$\text{Left margin:} \quad n(\zeta) \leq w_L(\zeta) - \delta_L(\zeta) + 0.5 w_{\text{veh. width}}, \tag{3.13}$$

where:

- $w_R(\zeta)$, $w_L(\zeta)$ are the lateral distances from the centreline to the right and left boundaries;
- the relaxation terms are given by:

$$\delta_{R,L}(\zeta) = \alpha w_{R,L}(\zeta_0) \exp\left(-\frac{1}{\lambda}(\zeta - \zeta_0)\right),$$

  with ζ_0 the start of the horizon, and $[\alpha, \lambda]$ the amplitude and decay parameters;
- a fixed margin of $0.5 w_{\text{veh. width}}$ is subtracted from both bounds to account for the vehicle footprint.

This relaxation rapidly vanishes after a few metres, ensuring that mid- and long-horizon planning is unaffected.

3.2.3.2 Dynamic Feasibility via g-g-v Constraint

The g-g-v constraint encodes the feasible region of longitudinal acceleration a_x, lateral acceleration $a_y = v_x \Omega$, and longitudinal velocity v_x. It serves to enforce dynamic feasibility without requiring explicit modelling of tyres, drivetrains, or other components. This yields a three-dimensional envelope—the g-g-v diagram—that bounds feasible combinations of (a_x, a_y) at each v_x. The constraint implicitly captures nonlinear effects such as aerodynamic drag and powertrain saturation.

This constraint is modelled as a convex polytope imposed on the planned states:

$$\boldsymbol{A}_{\text{ggv}} \begin{bmatrix} a_y \\ a_x \\ v_x \end{bmatrix} + \boldsymbol{b}_{\text{ggv}} \leq 0, \tag{3.14}$$

which yields a numerically efficient approximation of the feasible envelope.

While effective in the coupled regime, convex polytopes cannot capture sharp non-convex nonlinearities at the pure acceleration limits, caused for example by engine saturation. Such limitations will be addressed with actuation limits in Paragraph 3.2.3.3.

3D Extension of the g-g-v Constraint. The g-g-v constraint can be extended to account for 3D road geometry, as introduced in [54]. Road slope and banking angles produce additive contributions to a_x and a_y, while the vertical acceleration a_z scales the achievable lateral force. The result is a translated and scaled version of the original constraint:

$$\boldsymbol{A}_{\text{ggv}} \begin{bmatrix} \dfrac{a_y - g(\cos(\xi)\theta_{\text{bk}} + \sin(\xi)\theta_{\text{sl}})}{S_y(a_z)} \\ a_x - g(\sin(\xi)\theta_{\text{bk}} - \cos(\xi)\theta_{\text{sl}}) \\ v_x \end{bmatrix} + \boldsymbol{b}_{\text{ggv}} \leq 0. \tag{3.15}$$

This formulation preserves the convex structure while embedding key terrain effects. The scaling function $S_y(a_z)$ is a cubic spline embedding the effects of a_z on a_y which is iteratively learned when driving along the track.

3.2.3.3 Actuation Limits

To correct for the limitations of the convex polytope at the pure acceleration limits, we introduce bounds on the control inputs. These constraints model the nonlinear and asymmetric actuation limits, and are derived from data once again.

Longitudinal Acceleration Limits. The longitudinal control input $a_{x_{\text{ctrl}}}(t)$ is bounded by velocity-dependent splines:

$$a_{x_{\min}}(v_x(t)) \leq a_{x_{\text{ctrl}}}(t) \leq a_{x_{\max}}(v_x(t)), \tag{3.16}$$

where $a_{x_{\max}}(v_x)$ and $a_{x_{\min}}(v_x)$ model maximum traction and braking respectively. Both bounds are modelled as cubic splines for smoothness.

Lateral (Yaw Rate) Limits. The lateral control input $\Omega_{\text{ctrl}}(t)$ is similarly bounded:

$$-\Omega_{\max}(v_x(t)) \leq \Omega_{\text{ctrl}}(t) \leq \Omega_{\max}(v_x(t)). \tag{3.17}$$

Here, $\Omega_{\max}(v_x)$ is derived from the lateral acceleration envelope via the simplified relation $a_y = v_x\Omega$. The constraint is enforced symmetrically, as experimental data generally shows minimal asymmetry between left and right turning capabilities.

3D Extension of the Actuation Limits. The actuation limits can be consistently extended to account for road slope and banking effects using the same additive terms introduced in the 3D g-g-v formulation (3.15). For the longitudinal input bounds $a_{x_{\min}}(v_x)$ and $a_{x_{\max}}(v_x)$, the gravitational components along the road tangent are added to the data-driven splines, as shown in (3.16). This results in a rigid shift of the acceleration and braking limits based on the local road geometry. The lateral control input $\Omega_{\text{ctrl}}(t)$ is affected similarly. Through the approximation $a_y = v_x\Omega$, the same gravitational and scaling terms for the a_y are applied to (3.17). This unified treatment preserves the structure of the original formulation while improving fidelity on non-planar roads.

3.3 Implementation Details

3.3.1 Solver Integration

The E-NMPC problem described in this chapter is implemented using Pins [170], the in-house optimal control solver developed by our group. Pins is based on Pontryagin's Maximum Principle and adopts an indirect approach to optimal control. Originally developed for high-fidelity offline OCP simulations, it has been extended to support real-time receding-horizon E-NMPC, as used in ARD.

The setup and integration process follows these steps:

1. **Symbolic Problem Description.** The OCP is first defined in Maple [171], specifying the problem formulation from the system dynamics to the boundary conditions.
2. **C++ Code Generation.** Pins generates problem-specific high-performance C++ code, which can be compiled into shared libraries.
3. **Minimum-Lap-Time Benchmark.** A full-lap minimum-time trajectory is computed offline. This solution provides a performance baseline and aids in tuning weights and constraints for the online E-NMPC formulation.
4. **E-NMPC Interface Wrapping.** A code generation tool developed for this thesis wraps the low-level solver API into a standard E-NMPC interface, facilitating integration and reusability.
5. **Integration into ARD.** The generated E-NMPC module is embedded into ARD's planning stack. Multiple problem variants (*e.g.,* different weights, constraint sets, or cost terms) are supported simultaneously, provided the model structure remains mostly unchanged. The planner is selected at runtime through a configuration file.

This approach combines the flexibility of symbolic modelling and offline benchmarking with the efficiency of compiled code and modular solver integration. It enables real-time execution, fast iteration, and comparative evaluation across different formulations.

3.3.2 Warm-Starting and Horizon Management

Robust warm-starting and effective horizon management are critical for achieving real-time performance and stability in E-NMPC, especially in the presence of nonlinear dynamics, non-convex constraints, and road geometry variations.

Warm-Starting Strategy. Each E-NMPC problem is initialized by reusing the previous solution as a guess. However, the receding-horizon nature of the problem introduces three main challenges:

- **Mesh Misalignment.** The prediction horizon is re-meshed at each time step. The previous solution must be interpolated to match the new mesh.
- **Partial Horizon Coverage.** The prior solution covers only a portion of the new horizon. The remainder must be extrapolated.
- **Feasibility Risk.** Interpolation and extrapolation may lead to guesses that violate constraints, particularly the g-g-v constraint in 3D terrain, where the admissible region varies along the road.

The following techniques are used to mitigate these issues:

- **Linear Interpolation.** The overlapping portion of the previous solution is linearly interpolated to the current mesh. This preserves feasibility for convex constraints, but not for non-convex ones.
- **Tail Extrapolation.** For the unreached section of the horizon, several methods are available:
  1. Constant-value extrapolation (default),
  2. Clothoid-based extrapolation,
  3. Forward integration of the curvilinear dynamics.
- **Boundary Clamping.** The lateral deviation n is clamped (with padding) within track limits to avoid off-track initializations.
- **State Scaling for 3D Roads.** On non-planar tracks, accelerations $[a_x, a_y]$ and Ω are scaled when needed to reduce the likelihood of violating the g-g-v envelope. Although projection onto the constraint set would be more principled, it is computationally expensive. Empirical evidence shows that scaling is a sufficiently effective and simple alternative. On 2D tracks, such scaling is not necessary: the state polytope is convex, and the nonlinear actuation limits apply to controls that are analytically computed using the kineto-dynamical model, making them less susceptible to guess violations.

Despite these measures, warm-starting remains sensitive to sudden geometry changes. A typical failure case is a sharp corner following a long straight, where the past trajectory no longer aligns with the feasible set. These situations are the subject of ongoing investigation.

To partially mitigate this issue, we introduced an additional integral cost term in the Lagrangian that penalizes excessive longitudinal velocity. The term activates whenever v_x exceeds the maximum speed along the centreline, here approximated as

$$v_{x,\max} = \sqrt{\frac{a_{y_{max}}}{|\kappa|}}, \tag{3.18}$$

where $a_{y,\max}$ is the maximum admissible lateral acceleration and κ is the track curvature. The corresponding cost contribution is

$$\mathcal{W}\,\mathrm{pospart}(v_x - v_{x,\max})\,, \tag{3.19}$$

with $\mathcal{W}$ a tuning weight and $\mathrm{pospart}(\cdot)$ the positive-part operator. Although this approximation leads to slightly slower lap times, it provides the solver with an implicit preview of how the track is evolving and reduces the risk of divergence in sharp-corner scenarios. This strategy thus acts as a practical safeguard in edge cases, improving convergence at the cost of some optimality. Given this trade-off, we use this term on a per-case basis.

Horizon Mesh Construction. The layout and resolution of the prediction horizon are defined by the chosen mesh. Several strategies were explored:

- **Hybrid Mesh (Legacy).** This mesh matched the low-level controller structure:
  - 30 points uniformly spaced in time for lateral control resolution;
  - 10 points spaced at 1 m intervals to capture horizon-end effects;
  - The remaining points are spaced uniformly in ζ until the desired horizon length is reached.

  This approach performed well on wide circuits like Catalunya and Mugello.
- **Uniform Spatial Mesh (Current).** On more narrow circuits (*e.g., Universität der Bundeswehr München* (UniBW)), the hybrid mesh was less stable due to corners with sudden high curvatures. A uniform spatial mesh of evenly spaced points along ζ is now used, offering improved performance likely due to the solution aligning more closely to the guess.
- **Alternative Meshes (Discarded).** Adaptive spacing based on curvature and variable-length meshes were tested but discarded. These strategies increased warm-start mismatch and reduced solver convergence.

3.4 On the Formulation and Effects of g-g-v Constraints

The g-g-v constraint encodes the maximum achievable accelerations of the vehicle as a function of longitudinal speed. It defines a three-dimensional envelope in the space of lateral acceleration a_y, longitudinal acceleration a_x, and longitudinal velocity v_x, effectively capturing the interaction between tyres, powertrain, and aerodynamics, without requiring explicit modelling of these components. By constraining the planner within this envelope, ARD ensures physical feasibility while maintaining a formulation that is compact and suitable for real-time optimization.

3.4.1 Motivation and Role in Planning

The role of the g-g-v constraint is to bound the trajectory planning problem within the dynamic capabilities of the vehicle. This is particularly relevant near the handling limits, where the feasible acceleration region is highly nonlinear and speed-dependent. A formulation that neglects these limitations either risks infeasibility (if overestimated) or yields overly conservative trajectories (if underestimated).

In ARD, the g-g-v constraint enables the planner to reason about trade-offs between longitudinal and lateral acceleration, especially in aggressive manoeuvres such as braking into corners or accelerating out of them. Compared to explicit force-based models, the g-g-v formulation offers a lower computational burden and more straightforward identification from data. However, it does not describe how the vehicle dynamics evolve within the feasible envelope itself. This aspect is left to the equations of motion, and if a simplified point-mass model is employed, specific characteristics such as understeer or oversteer behaviour cannot be captured.

3.4.2 Structure and Hybrid Formulation

To balance computational efficiency and model fidelity, we adopt a hybrid constraint formulation composed of:

- A convex polytope acting on the planned states $[a_y, a_x, v_x]$, capturing the dominant coupled constraints;
- Nonlinear bounds on the control inputs, correcting the envelope near the pure acceleration limits where the polytope tends to overestimate feasibility.

Convex Polytope Representation. The convex component is a set of linear inequalities in the space $[a_y, a_x, v_x]$, defined as:

$$\boldsymbol{A}_{\text{ggv}} \begin{bmatrix} a_y \\ a_x \\ v_x \end{bmatrix} \leq \boldsymbol{b}_{\text{ggv}}. \tag{3.20}$$

This representation is derived as the convex hull of the identified g-g-v points, which may be obtained via simulation or data-driven identification. It captures the feasible acceleration set over a wide operating range, including coupled regimes.

However, due to the convex nature of this formulation, the polytope generally overestimates accelerations near the pure longitudinal and lateral axes. For example, engine torque saturation and aerodynamic drag limit acceleration at high speeds. These nonlinear effects are not well represented by a convex envelope.

Nonlinear Pure Acceleration Bounds. To correct for the limitations of the polytope near the axes, we impose additional nonlinear bounds on the control inputs:

$$a_{x_{\min}}(v_x) \leq a_{x_{\text{ctrl}}}(t) \leq a_{x_{\max}}(v_x), \tag{3.21}$$

$$-\Omega_{\max}(v_x) \leq \Omega_{\text{ctrl}}(t) \leq \Omega_{\max}(v_x). \tag{3.22}$$

The longitudinal limits $a_{x_{\min}}$, $a_{x_{\max}}$ are identified from either physics-based simulation or data, and fitted using cubic splines to retain smoothness. The lateral bounds are derived from the lateral acceleration envelope via the relation $a_y = v_x\Omega$, and likewise represented as speed-dependent splines. This correction ensures realistic saturation behaviour in the control space without complicating the structure of the state constraints.

Impact on Optimization and Feasibility. The hybrid formulation balances feasibility and numerical tractability. The convex polytope constrains the state evolution, preserving a structure that is friendly to the E-NMPC solver, whilst the nonlinear control bounds correct local inaccuracies. This modularity also allows the bounds to be updated independently when the vehicle, surface, or operating conditions change. We showed the impacts of different g-g-v formulations in [49]. We compared our polytopic formulation against: our previous super-ellipse formulation [25], Veneri and Massaro's polar spline representation [48], and Rowold *et al.* diamond shapes [52]. Ultimately establishing that our formulation outperformed the others in the resulting lap time whilst remaining computationally feasible.

Chapter 4

Vehicle Control

Abstract

This chapter presents the low-level control architecture that enables the *Artificial Race Driver* (ARD) to accurately execute the time-optimal manoeuvres planned by the high-level layer. The controller stack is modular and tailored to the different nature of longitudinal and lateral vehicle dynamics. Longitudinal control is handled via a learned *Proportional-Integral-Derivative* (PID) controller. For lateral dynamics, we deploy a physics-informed neural network trained to compute steering actions, augmented by a PID feedback loop for local correction. The chapter details the design rationale, structure, and implementation of each component, highlighting how prior knowledge and learning-based models are effectively combined to achieve accurate, real-time vehicle control in both simulation and real-world conditions.

Contents

4.1 Introduction 47
4.2 Longitudinal Control 48
4.2.1 Motivation and Requirements 48
4.2.2 Acceleration-Based Correction Controller 48
4.2.3 Implementation and Real-World Usage 48
4.3 Lateral Control 49
4.3.1 Motivation and Requirements 49
4.3.2 Low-Level Feedforward Steering Controller 49
4.3.3 Low-Level Feedback Steering Controller 55

4.1 Introduction

The low-level control layer is responsible for executing the time-optimal trajectories generated by the high-level planner while respecting the physical limits of the vehicle. Within the ARD framework, this layer plays a critical role in bridging the gap between the model predictive planner and the actual vehicle control, both in simulation and in the real world. It must deliver fast and reliable actuation in highly dynamic scenarios, where the vehicle operates close to its handling limits.

Longitudinal control is realized through a learned PID controller. This choice represents a practical compromise between simplicity, interpretability, and performance. The controller is automatically tuned through closed-loop learning, with an emphasis on achieving responsive yet stable behaviour across the entire speed range. Although classical in structure, this component has proven sufficient for the scope of time-optimal acceleration and braking within ARD, and its modular formulation facilitates future extensions toward more advanced schemes.

Lateral control, by contrast, is handled through a physics-informed neural network that maps the planned vehicle state to an appropriate steering command. The learning process is guided entirely by minimal telemetry data, and the resulting network operates in a pure feedforward manner. To ensure proper closed-loop performance, a PID controller is integrated in parallel as a correction loop. This hybrid design allows the neural controller to leverage the modelling capacity of learning-based techniques while relying on the PID to handle unseen conditions and discrepancies. This approach is especially relevant in real-world deployments, where the exact system dynamics will not be fully known and environment uncertainties cannot be ignored.

4.2 Longitudinal Control

4.2.1 Motivation and Requirements

The longitudinal controller is tasked with tracking the time-optimal speed profiles produced by the high-level planner. Depending on the specific platform, the control output may be either a normalized pedal command or a direct acceleration request. In the current ARD setup, the control signal is a desired acceleration command, with positive values corresponding to throttle and negative values to braking. This design choice was driven by the *Universität der Bundeswehr München* (UniBW) test vehicles which already feature an internal controller capable of accepting desired longitudinal acceleration or velocity. In the case of pedal control, we typically use a gain-scheduling version of this PID controller, which converts the desired acceleration signal into a pedal position.

Given ARD's modular structure, the longitudinal control block is designed to be lightweight, interpretable, and portable across different platforms. While longitudinal and lateral dynamics are inherently coupled, the longitudinal controller is implemented independently to simplify deployment and tuning. More tightly integrated approaches may be explored in future developments, as discussed in subsequent chapters.

4.2.2 Acceleration-Based Correction Controller

The longitudinal controller applies a corrective action to the desired acceleration computed by the high-level planner to ensure that the vehicle follows the target speed profile. Its structure mirrors that of the lateral correction controller described in Section 4.3.3, operating in parallel to the planner and injecting an additive correction term when velocity deviations are detected.

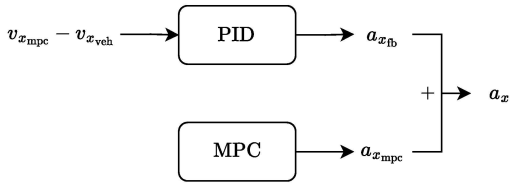


Figure 4.1: Schematic of the longitudinal control loop. The controller applies a corrective term to the acceleration planned by the high-level to reduce deviations from the target speed profile. In the case of pedal based control, we typically use a gain-scheduling version of this PID which converts the desired acceleration signal into a pedal position.

As shown in Figure 4.1, the feedback correction $a_{x_{\text{fb}}}$ is added to the planned acceleration to obtain the final command:

$$a_x = a_{x_{\text{mpc}}} + a_{x_{\text{fb}}}.$$

The error is driven by the difference between the planned and actual vehicle velocity, $v_{x_{\text{mpc}}}$ and $v_{x_{\text{veh}}}$ respectively.

The tuning of the PID controller is described in Chapter 6.

4.2.3 Implementation and Real-World Usage

In simulation, the longitudinal correction controller ensures tracking of the planned speed profile and compensates for minor modelling inaccuracies. The description of the learning procedure can be found in Chapter 6. In real-world experiments, the longitudinal control is delegated to the vehicle's built-in controller, which directly interprets the desired acceleration or velocity requests.

4.3 Lateral Control

4.3.1 Motivation and Requirements

The lateral control module is responsible for generating steering commands that execute the high-level planner's intended trajectory as accurately as possible, especially in aggressive manoeuvres near the vehicle's physical limits. A purely model-based approach is difficult to maintain across vehicle platforms and operating conditions without extensive identification. Conversely, purely learning-based methods often suffer from poor generalization or lack of structure. ARD addresses these issues through a hybrid strategy that combines a physics-informed feedforward neural controller with a classical feedback loop for error correction. The goal is to leverage learning where it is most beneficial—capturing complex lateral interactions—while preserving the interpretability of classical modelling and control techniques.

4.3.2 Low-Level Feedforward Steering Controller

The network, referred to as *Physics-Informed Steering Neural Network* (PhS-NN), is structured around known laws of vehicle dynamics, resulting in a compact, interpretable, and generalizable inverse model of the lateral dynamics.

PhS-NN is trained via supervised learning using synthetic or real telemetry data. Its architecture reflects both steady-state and transient lateral behaviours, integrating vehicle-specific knowledge directly into the network topology. The resulting controller outputs a feedforward steering command δ_{ff_k} at each discrete time step k, based on future planned motion.

PhS-NN operates as a discrete-time system with sampling time T_{sNN} and computes:

$$\delta_{\mathrm{ff}_k} = \mathcal{F}_N\big(\boldsymbol{\rho}_k, \boldsymbol{v_x}_k, \boldsymbol{a_y}_k, \boldsymbol{a_x}_k, \boldsymbol{a_z}_k\big) \tag{4.1}$$

where is $\mathcal{F}_N(\cdot)$ a physics-informed neural functional trained to model the inverse lateral dynamics. The input signals to the model, consist of windows of predicted quantities:

$$\begin{aligned}
\boldsymbol{\rho}_k &= \big[\rho_k, \ldots, \rho_{k+q}\big]\,, \\
\boldsymbol{v_x}_k &= \big[v_{x_k}, \ldots, v_{x_{k+q}}\big]\,, \\
\boldsymbol{a_y}_k &= \big[a_{y_k}, \ldots, a_{y_{k+q}}\big]\,, \\
\boldsymbol{a_x}_k &= \big[a_{x_k}, \ldots, a_{x_{k+q}}\big]\,, \\
\boldsymbol{a_z}_k &= \big[a_{z_k}, \ldots, a_{z_{k+q}}\big]\,.
\end{aligned} \tag{4.2}$$

These vectors contain the planned trajectories of transient curvature ρ, accelerations $[a_x, a_y, a_z]$, and longitudinal velocity v_x. The vertical acceleration a_z accounts for three-dimensional road effects and is zero on flat tracks.

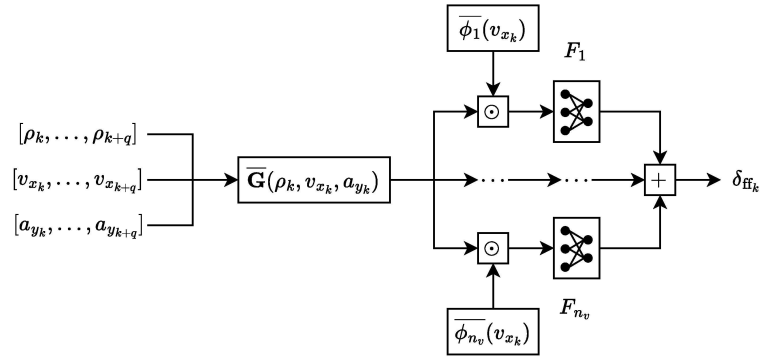
The design of $\mathcal{F}_N(\cdot)$ follows a structured local modelling approach inspired by neuro-fuzzy systems [172], [173], but adapted to vehicle dynamics. Local models are derived from simplified nonlinear vehicle equations and combined via smooth partition-of-unity activations. This ensures that the learned model maintains physically meaningful behaviours across the operating domain.

We now describe the design of PhS-NN in detail, starting from its core formulation and progressively extending it to incorporate additional dynamical effects.

4.3.2.1 PhS-NN: Modelling the Pure Lateral Vehicle Dynamics

We begin with the foundational form of PhS-NN, which models the inverse lateral dynamics of the vehicle on a flat, two-dimensional road under the assumption of negligible longitudinal and vertical dynamic effects. In this initial configuration, the network receives only $[\rho, v_x, a_y]$ as inputs, omitting a_x and a_z.

(a) Base structure.



(b) Full structure.

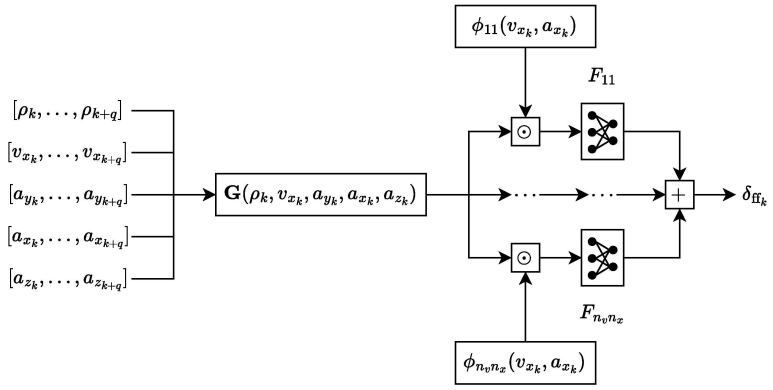


Figure 4.2: Schematics of the PhS-NN architectures: (a) base structure and (b) full structure.

The structure, depicted in Figure 4.2a, is formally described by:

$$\delta_{\mathrm{ff}_k} = \sum_{j=1}^{n_v} \left(\bar{\boldsymbol{G}}(\boldsymbol{\rho}_k, \boldsymbol{v}_{\boldsymbol{x}k}, \boldsymbol{a}_{\boldsymbol{y}k}) \odot \bar{\boldsymbol{\phi}}_j(\boldsymbol{v}_{\boldsymbol{x}k}) \right) \begin{bmatrix} \mathrm{F}_{j_1} \\ \vdots \\ \mathrm{F}_{j_{q+1}} \end{bmatrix} \tag{4.3}$$

Here, $\bar{\boldsymbol{G}}(\boldsymbol{\rho}_k, \boldsymbol{v}_{\boldsymbol{x}k}, \boldsymbol{a}_{\boldsymbol{y}k})$ is a vector-valued function that models the steady-state steering behaviour over a time horizon of $q+1$ steps. The vector functions $\bar{\boldsymbol{\phi}}_j(\boldsymbol{v}_{\boldsymbol{x}k})$ partition the velocity domain into n_v overlapping regions, each activating a fully connected layer F_j to capture the transient response characteristics within that range. Figure 4.3a shows the structure of $\bar{\boldsymbol{G}}(\cdot)$.

The terms in Eq. (4.3) serve distinct roles:

1. $\bar{\boldsymbol{G}}(\boldsymbol{\rho}_k, \boldsymbol{v}_{\boldsymbol{x}k}, \boldsymbol{a}_{\boldsymbol{y}k})$ encodes steady-state lateral dynamics through a locally weighted combination of n_y models representing the nonlinear *Handling Diagram* (HD).
2. Each layer F_j learns to combine the future predicted values from $\bar{\boldsymbol{G}}$, capturing transient effects and delays in steering response.
3. The activation functions $\bar{\boldsymbol{\phi}}_j$ define a partition of unity across the v_x domain, ensuring smooth transitions between the different F_j layers.

This formulation constitutes the foundational block of the PhS-NN controller. It enables a compact and physically consistent approximation of pure lateral dynamics by combining local models of the vehicle's steady-state behaviour with structured transient corrections. In the following subsections, we first describe how the HD is encoded and approximated through these local models. The extension to account for longitudinal and vertical effects is addressed subsequently.

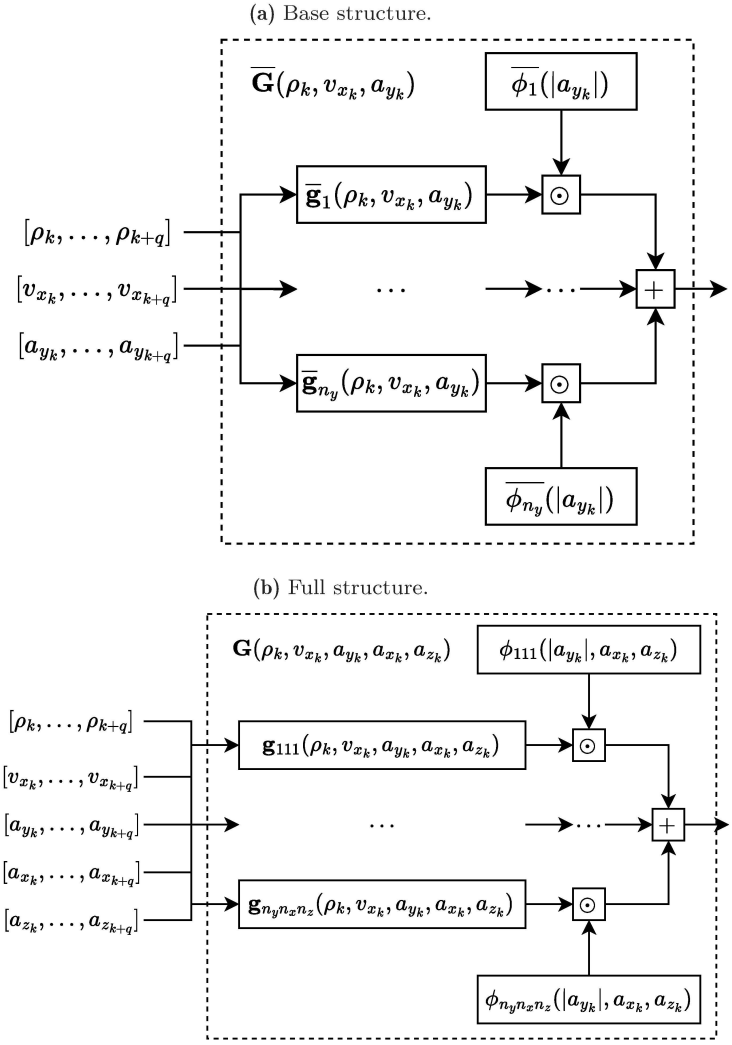


Figure 4.3: Schematics of the $\boldsymbol{G}(\cdot)$ network architectures: (a) base structure and (b) full structure.

Steady-State Lateral Dynamics Modelling

PhS-NN learns the steady-state lateral behaviour of the vehicle via the function $\bar{\boldsymbol{G}}(\boldsymbol{\rho}_k, \boldsymbol{v}_{\boldsymbol{x}k}, \boldsymbol{a}_{\boldsymbol{y}_k})$ introduced in Eq. (4.3). This formulation is inspired by the nonlinear HD [24], [174], which characterizes the vehicle's under- or oversteering response as a function of speed and lateral acceleration. The HD is typically expressed as:

$$\delta - \rho L = K_{\text{us}}(a_y, v_x) \tag{4.4}$$

where δ is the front axle steering angle, ρ is the trajectory curvature, L is the vehicle's wheelbase, and K_{us} denotes the understeering gradient. Positive values of K_{us} indicate understeering behaviour, while negative values correspond to oversteering. Figure 4.4a shows how the HD varies with vehicle speed v_x.

Locally Weighted Models of the Handling Diagram To approximate $K_{\text{us}}(a_y, v_x)$, the network includes n_y local linear models. Each model is centred at a nominal operating point $|a_y| = a_{y_{0_i}}$ and has the form:

$$\delta = \rho L + \mathrm{k}_{1_i} \operatorname{sign}(a_y) + \mathrm{k}_{2_i}\left(a_y - a_{y_{0_i}} \operatorname{sign}(a_y)\right) \tag{4.5}$$

where k_{1_i} and k_{2_i} are learnable parameters. Symmetry in the response for left- and right-hand turns is enforced using $\operatorname{sign}(a_y)$.

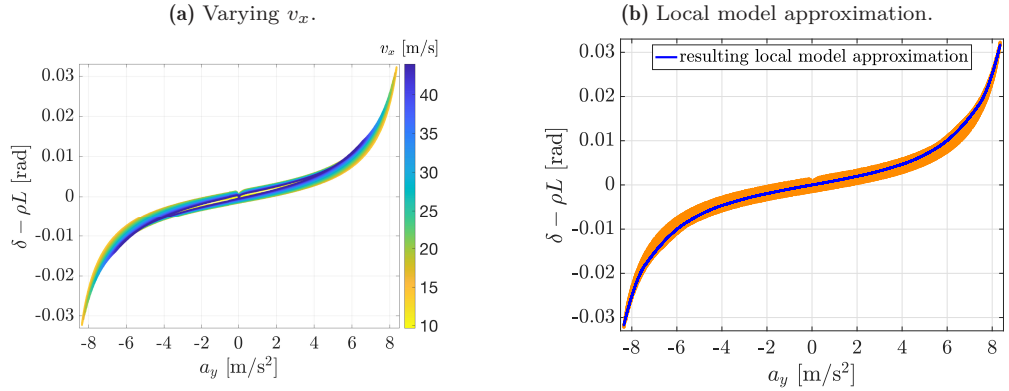


Figure 4.4: Handling diagram as a function of longitudinal speed: (a) varying v_x, (b) local model approximation. Figures taken from our work [25] © 2025 IEEE.

To account for the variation of the HD with speed v_x, the local models are extended as:

$$\delta = \bar{g}_i(\rho, v_x, a_y) = \rho L + \mathrm{k}_{1_i}(v_x)\,\mathrm{sign}(a_y) + \mathrm{k}_{2_i}(v_x)\big(a_y - a_{y_{0_i}}\mathrm{sign}(a_y)\big) \tag{4.6}$$

with speed-dependent gains parameterized by polynomials:

$$\begin{cases} \mathrm{k}_{1_i}(v_x) = \sum_{j=0}^{n_{p1}} \mathrm{c}_{1_{ij}} v_x^j & \text{(4.7a)} \\ \mathrm{k}_{2_i}(v_x) = \sum_{q=0}^{n_{p2}} \mathrm{c}_{2_{iq}} v_x^q & \text{(4.7b)} \end{cases}$$

where $\{\mathrm{c}_{1_{ij}}, \mathrm{c}_{2_{iq}}\}$ are learnable coefficients. The polynomial orders n_{p_1} and n_{p_2} are selected empirically.

Each local model $\bar{g}_i(\rho, v_x, a_y)$ is activated by a piecewise linear function $\bar{\phi}_i(|a_y|)$, which ensures a smooth transition between models and satisfies the partition-of-unity property:

$$\sum_{i=1}^{n_y} \bar{\phi}_i(|a_y|) = 1 \tag{4.8}$$

The resulting approximation of the nonlinear HD is defined as:

$$\delta = \bar{G}(\rho, v_x, a_y) = \sum_{i=1}^{n_y} \bar{g}_i(\rho, v_x, a_y)\,\bar{\phi}_i(|a_y|) \tag{4.9}$$

Figure 4.4b shows an example of this local approximation.

Vector Formulation for Transient Dynamics PhS-NN operates on planned trajectories of future accelerations, provided by the high-level planner. Hence, the previously scalar formulation of the HD approximation is generalized to vectors:

$$\begin{cases} \bar{\boldsymbol{g}}_i(\boldsymbol{\rho}_k, \boldsymbol{v_x}_k, \boldsymbol{a_y}_k) = \boldsymbol{\rho}_k L + \mathbf{k}_{1_i}(\boldsymbol{v_x}_k) \odot \mathrm{sign}(\boldsymbol{a_y}_k) \\ \qquad\qquad + \mathbf{k}_{2_i}(\boldsymbol{v_x}_k) \odot \big(\boldsymbol{a_y}_k - a_{y_{0_i}}\,\mathrm{sign}(\boldsymbol{a_y}_k)\big) & \text{(4.10a)} \\ \bar{\boldsymbol{G}}(\boldsymbol{\rho}_k, \boldsymbol{v_x}_k, \boldsymbol{a_y}_k) = \sum_{i=1}^{n_y} \Big(\bar{\boldsymbol{g}}_i(\boldsymbol{\rho}_k, \boldsymbol{v_x}_k, \boldsymbol{a_y}_k) \odot \bar{\boldsymbol{\phi}}_i(|\boldsymbol{a_y}_k|)\Big) & \text{(4.10b)} \end{cases}$$

The functions $\bar{\boldsymbol{\phi}}_i(|\boldsymbol{a_y}_k|)$ are defined element-wise across the prediction horizon:

$$\bar{\boldsymbol{\phi}}_i(|\boldsymbol{a_y}_k|) = \big[\bar{\phi}_i(|a_{y_k}|), \ldots, \bar{\phi}_i(|a_{y_{k+q}}|)\big] \tag{4.11}$$

This yields a vector of predicted front axle angles, spanning from k to $k+q$, which is processed by a set of fully connected layers. Each layer F_j computes a linear combination of the predicted values, effectively capturing the transient dynamics and the evolution of the response over the prediction horizon.

Speed-Dependent Transient Response To account for variation in vehicle response at different speeds, the network includes n_v fully connected layers $\{\mathrm{F}_1, \ldots, \mathrm{F}_{n_v}\}$. Each is activated by a speed-dependent vector function:

$$\bar{\boldsymbol{\phi}}_j(\boldsymbol{v}_{\boldsymbol{x}k}) = \left[\bar{\phi}_j(v_{x_k}), \ldots, \bar{\phi}_j(v_{x_{k+q}})\right] \tag{4.12}$$

The scalar functions $\bar{\phi}_j(\cdot)$ partition the velocity space and ensure that each layer is active only within a local region. This modular design enables PhS-NN to learn transient behaviour in a speed-dependent manner.

4.3.2.2 PhS-NN: Modelling the Combined Lateral Dynamics

The previous section introduced the pure lateral dynamics captured by PhS-NN under the assumption of flat terrain and negligible coupling. This section completes the formulation by modelling both the combined effects of longitudinal and vertical accelerations on lateral vehicle dynamics.

Coupled Effects of Longitudinal and Vertical Accelerations To handle combined lateral dynamics, PhS-NN extends its model to incorporate the influence of longitudinal a_x and vertical a_z accelerations. The extended formulation is:

$$\delta_{\mathrm{ff}_k} = \sum_{j=1}^{n_v}\sum_{l=1}^{n_x}\left(\boldsymbol{G}(\boldsymbol{\rho}_k, \boldsymbol{v}_{\boldsymbol{x}k}, \boldsymbol{a}_{\boldsymbol{y}k}, \boldsymbol{a}_{\boldsymbol{x}k}, \boldsymbol{a}_{\boldsymbol{z}k}) \odot \boldsymbol{\phi}_{jl}(\boldsymbol{v}_{\boldsymbol{x}k}, \boldsymbol{a}_{\boldsymbol{x}k})\right) \begin{bmatrix} \mathrm{F}_{jl_1} \\ \vdots \\ \mathrm{F}_{jl_{q+1}} \end{bmatrix} \tag{4.13}$$

This formulation introduces:

- Additional input sequences $\boldsymbol{a}_{\boldsymbol{x}k}$ and $\boldsymbol{a}_{\boldsymbol{z}k}$, planned by the high-level.
- A generalized vector function $\boldsymbol{G}(\cdot)$, which replaces $\bar{\boldsymbol{G}}(\cdot)$ and includes local models of the lateral response under varying a_x and a_z.
- A set of n_x regions in a_x and n_z in a_z, each with their corresponding activation functions and model parameters.

This combined modelling approach equips PhS-NN with the capability to handle highly nonlinear, coupled conditions as found in aggressive manoeuvres and over non-flat surfaces.

Quasi Steady-State Combined Lateral Nonlinear Dynamics

We now describe the vector function $\boldsymbol{G}(\cdot)$, depicted in Figure 4.3b and introduced in Eq. (4.13), which generalizes the HD formulation to model the quasi steady-state combined lateral dynamics under the influence of longitudinal and vertical accelerations.

$$\begin{aligned}\boldsymbol{G}(\boldsymbol{\rho}_k, \boldsymbol{v}_{\boldsymbol{x}k}, \boldsymbol{a}_{\boldsymbol{y}k}, \boldsymbol{a}_{\boldsymbol{x}k}, \boldsymbol{a}_{\boldsymbol{z}k}) = \sum_{i=1}^{n_y}\sum_{l=1}^{n_x}\sum_{m=1}^{n_z} \boldsymbol{g}_{ilm}(\boldsymbol{\rho}_k, \boldsymbol{v}_{\boldsymbol{x}k}, \boldsymbol{a}_{\boldsymbol{y}k}, \boldsymbol{a}_{\boldsymbol{x}k}, \boldsymbol{a}_{\boldsymbol{z}k}) \odot \\ \odot\, \boldsymbol{\phi}_{ilm}\big(|\boldsymbol{a}_{\boldsymbol{y}k}|, \boldsymbol{a}_{\boldsymbol{x}k}, \boldsymbol{a}_{\boldsymbol{z}k}\big)\end{aligned} \tag{4.14}$$

Each local model $\boldsymbol{g}_{ilm}(\cdot)$ is activated around a nominal condition $\{|a_y|, a_x, a_z\} = \{a_{y_{0_i}}, a_{x_{0_l}}, a_{z_{0_m}}\}$, with activation functions defined by:

$$\boldsymbol{\phi}_{ilm}(|\boldsymbol{a}_{\boldsymbol{y}k}|, \boldsymbol{a}_{\boldsymbol{x}k}, \boldsymbol{a}_{\boldsymbol{z}k}) = \bar{\boldsymbol{\phi}}_i(|\boldsymbol{a}_{\boldsymbol{y}k}|) \odot \bar{\boldsymbol{\phi}}_l(\boldsymbol{a}_{\boldsymbol{x}k}) \odot \bar{\boldsymbol{\phi}}_m(\boldsymbol{a}_{\boldsymbol{z}k}) \tag{4.15}$$

The total number of local models is $n_y \cdot n_x \cdot n_z$, with n_y capturing lateral acceleration effects, n_x accounting for longitudinal influence, and n_z representing road-induced vertical load transfer. The activation functions follow the same partition-of-unity principle discussed earlier, ensuring smooth blending between local models.

Each local function $\boldsymbol{g}_{ilm}(\cdot)$ extends the base handling diagram model by introducing additional terms for a_x and a_z, derived from a local approximation of the nonlinear double-track vehicle model near quasi steady-state. The scalar version reads:

$$
\begin{aligned}
& g_{ilm}(\rho, v_x, a_y, a_x, a_z) = \\
& \rho L + \mathrm{k}_{1_i}(v_x)\mathrm{sign}(a_y) + \mathrm{k}_{2_i}(v_x)\left(a_y - a_{y_{0_i}}\mathrm{sign}(a_y)\right) \\
& + \mathrm{k}_{z_{1_m}}\left(a_y - a_{y_{0_i}}\mathrm{sign}(a_y)\right)(a_z - a_{z_{0_m}}) + \mathrm{k}_{z_{2_m}}\left(a_y - a_{y_{0_i}}\mathrm{sign}(a_y)\right)(a_z - a_{z_{0_m}})^2 \\
& + \mathrm{k}_{y_{1_i}}\mathrm{k}_{x_{1_l}}(1 + \mathrm{k}_{z_{3_m}})\Big(a_y - (a_{y_{0_i}} + \mathrm{k}_{y_{2_i}})\mathrm{sign}(a_y)\Big) \\
& \quad \times \Big(\mathrm{k}_{x_{2_l}} + a_x - a_{x_{0_l}}\Big)\big[1 + \mathrm{k}_{y_{3_i}}\left(a_y - a_{y_{0_i}}\mathrm{sign}(a_y)\right) \\
& \quad + \mathrm{k}_{x_{3_l}}(a_x - a_{x_{0_l}}) + \mathrm{k}_{x_{4_l}}(a_x - a_{x_{0_l}})^2 \\
& \quad + \mathrm{k}_{z_{4_m}}(a_z - a_{z_{0_m}}) + \mathrm{k}_{z_{5_m}}(a_z - a_{z_{0_m}})^2 \\
& \quad + \mathrm{k}_{z_{6_m}}(a_z - a_{z_{0_m}})(a_x - a_{x_{0_l}}) \\
& \quad + \mathrm{k}_{y_{4_i}}\mathrm{k}_{x_{5_l}}\left(a_y - a_{y_{0_i}}\mathrm{sign}(a_y)\right)(a_x - a_{x_{0_l}})\big]
\end{aligned}
\tag{4.16}
$$

The full vector formulation follows the same logic and is omitted here for brevity. The sign functions ensure symmetry with respect to the direction of a_y.

Parameter Count and Reduction Strategy The number of parameters per local model is:

$$
N_{\mathrm{pars}_g} = (n_{p_1} + 1) + (n_{p_2} + 1) + 4 + 5 + 6 = n_{p_1} + n_{p_2} + 17 \tag{4.17}
$$

To avoid parameter redundancy, the model adopts a structured decomposition in which terms related to lateral, longitudinal, and vertical dynamics are shared across combinations. The resulting total parameter count becomes:

$$
N_{\mathrm{pars}_{\boldsymbol{G}}} = (n_{p_1} + 1 + n_{p_2} + 1)n_y + 4n_y + 5n_x + 6n_z \tag{4.18}
$$

instead of $N_{\mathrm{pars}_g} \cdot n_y n_x n_z$, with no measurable loss in approximation quality. This decomposition supports efficient learning while preserving expressiveness in high-dimensional input spaces.

Transient Combined Lateral Nonlinear Dynamics

By combining the local models $\boldsymbol{g}_{ilm}(\cdot)$, the function $\boldsymbol{G}(\cdot)$ in Eq. (4.14) returns a vector of present and future steering angle predictions. As in the initial formulation of PhS-NN, a bank of fully connected layers is used to model the transient lateral dynamics and refine the underlying quasi steady-state estimates.

Specifically, PhS-NN employs $n_v \cdot n_x$ fully connected layers $\{\mathrm{F}_{11}, \ldots, \mathrm{F}_{n_v n_x}\}$, illustrated in Figure 4.2b, each of which computes a linear combination of the entries of the vector $\boldsymbol{G}(\boldsymbol{\rho}_k, \boldsymbol{v}_{\boldsymbol{x}k}, \boldsymbol{a}_{\boldsymbol{y}k}, \boldsymbol{a}_{\boldsymbol{x}k}, \boldsymbol{a}_{\boldsymbol{z}k})$. This architecture allows the network to learn speed- and acceleration-dependent corrections that account for dynamic effects such as actuation delays and transient tyre responses.

Each layer F_{jl}, with $j \in \{1, \ldots, n_v\}$ and $l \in \{1, \ldots, n_x\}$, has a single neuron with $q+1$ learnable weights $\{\mathrm{F}_{jl_1}, \ldots, \mathrm{F}_{jl_{q+1}}\}$, operating on the corresponding $q+1$-dimensional slice of $\boldsymbol{G}$. Their inputs are selectively activated via vector-valued functions:

$$
\boldsymbol{\phi}_{jl}(\boldsymbol{v}_{\boldsymbol{x}k}, \boldsymbol{a}_{\boldsymbol{x}k}) = \bar{\boldsymbol{\phi}}_j(\boldsymbol{v}_{\boldsymbol{x}k}) \odot \bar{\boldsymbol{\phi}}_l(\boldsymbol{a}_{\boldsymbol{x}k}) \tag{4.19}
$$

where $\bar{\boldsymbol{\phi}}_j$ and $\bar{\boldsymbol{\phi}}_l$ are vector-valued activations based on the scalar partition-of-unity functions $\bar{\phi}_j(v_x)$ and $\bar{\phi}_l(a_x)$.

The scalar outputs of the activated layers are summed to produce a single steering angle estimate.

4.3.2.3 Remarks on the Physics-Driven Internal Structure of PhS-NN

The previous subsections described the architecture of the proposed neural steering controller, PhS-NN, which is designed according to physics-driven principles. These principles leverage prior knowledge of vehicle dynamics to guide the network's structure, ensuring consistency with known behaviour while retaining the flexibility of a data-driven model. This embedding of domain knowledge forms a central contribution of our approach.

We now summarize the key vehicle dynamics insights that informed the design of PhS-NN.

1. **Steady-state pure lateral dynamics**: PhS-NN learns the vehicle's HD, a canonical representation of the understeer/oversteer relationship [175]. Classical vehicle dynamics indicate that the nonlinearity of the HD is primarily governed by lateral acceleration a_y. PhS-NN exploits this by employing multiple locally activated models defined over overlapping intervals of a_y, while also accounting for speed dependency through v_x.
2. **Combined lateral dynamics on 3D roads**: When longitudinal acceleration a_x and vertical acceleration a_z are non-zero, PhS-NN extends the local models with physics-inspired terms derived from a local approximation of a nonlinear double-track vehicle model near quasi steady-state conditions. This enables the controller to capture the effects of load transfers and road-induced vertical dynamics.
3. **Transient lateral dynamics**: The lateral dynamic response varies with both vehicle speed v_x and longitudinal acceleration a_x, due to physical phenomena such as tyre relaxation behaviour [175], [176]. To reflect this, PhS-NN includes locally activated fully connected layers that model transient effects in different regions of the v_x–a_x space.

This structured design enables PhS-NN to learn both linear and nonlinear lateral dynamics, and to generalize effectively even in data-scarce regimes.

4.3.3 Low-Level Feedback Steering Controller

The PhS-NN steering controller described in Section 4.3.2 operates in a feedforward fashion, computing the control signal δ_{ff_k} based on the high-level planned trajectories. However, model mismatches and external disturbances (*e.g.,* local variations in friction) can still introduce trajectory tracking errors.

To mitigate these effects, a PID feedback controller is used to reduce the trajectory curvature tracking error, defined as $\rho_{s_{\mathrm{err},k}} = \rho_{s_k} - \hat{\rho}_{s_k}$, where ρ_{s_k} and $\hat{\rho}_{s_k}$ denote the planned and measured trajectory curvatures at time step k, respectively. The curvature ρ_s is computed as the steady-state approximation Ω/V, with Ω the yaw rate and $V = \sqrt{v_x^2 + v_y^2}$ the total velocity.

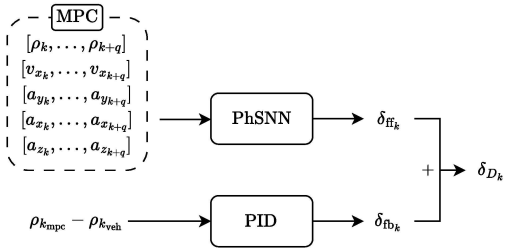


Figure 4.5: Schematic of the lateral controller. Given the planned trajectory from the high-level planner, it first computes the feedforward steering angle δ_{ff_k}. Then it computes a correcting feedback δ_{fb_k}, using a PID with anti-windup and filtered derivatives based on the steady-state curvature error between the planned and current trajectory curvatures. The final steering angle δ_{D_k} is obtained as the sum of the two contributions.

As illustrated in Figure 4.5, the feedback correction δ_{fb_k} is added to the feedforward term δ_{ff_k} to yield the final steering command:

$$\delta_{D_k} = \delta_{\mathrm{ff}_k} + \delta_{\mathrm{fb}_k}.$$

The tuning of the PID controller is described in Chapter 6.

Chapter 5

Digital Twin Vehicle Model

Abstract

This chapter presents the vehicle model underpinning the simulation component of the thesis. A high-fidelity double-track model of a rear-wheel-drive race car is first introduced. It features realistic powertrain, suspension, and tyre dynamics, supports three-dimensional circuit topologies, and has been validated extensively against telemetry data. While this model has served as a reliable digital twin for closed-loop simulation in the *Artificial Race Driver* (ARD), offline *Optimal Control Problem* (OCP), and performance benchmarking in the past, it also presents practical drawbacks: it is vehicle-specific, requires expert calibration, and is not easily differentiable, limiting its use in gradient-based optimization or rapid adaptation to new platforms. To address these limitations, a *Neural Vehicle Model* (NVM) is introduced as a compact and fully differentiable approximation. Its design balances physical structure with learnable components, enabling efficient training from telemetry data. Comparative results show that it reproduces the predictive accuracy of the high-fidelity model while offering greater flexibility and scalability.

Contents

5.1 Introduction 57
5.2 High-Fidelity Double-Track Model 58
5.2.1 Modelling Overview 58
5.2.2 Role in Simulation and Benchmarking 58
5.2.3 Validation on Real-World Data 59
5.2.4 Challenges in Model Identification 60
5.3 Neural Vehicle Model as a Lightweight Alternative 62
5.3.1 Motivation and Scope 62
5.3.2 Internal Structure and Modelling Assumptions 62
5.3.3 Comparative Identification Results 70

5.1 Introduction

Accurate vehicle models are essential for the development and evaluation of autonomous driving systems. In earlier work, a high-fidelity physics-based model of a racecar was employed as a digital twin to support simulation and offline optimization. The model enabled closed-loop evaluation in a realistic simulation environment and was also used to generate minimum-lap-time benchmark solutions via offline OCP.

The former digital twin was a validated double-track representation of a rear-wheel-drive racecar equipped with a combustion engine. It captures key non-linearities and couplings across suspension, tyre dynamics, powertrain, and load transfer, and supports arbitrary three-dimensional track geometries. Although its development is not a direct contribution of this thesis, it has been employed extensively in past studies to analyse vehicle dynamics and benchmark planning strategies.

This approach, however, presents several limitations. The model is vehicle-specific, requires expert calibration and high-quality instrumentation, and is not easily differentiable. These drawbacks restrict its applicability in contexts where rapid adaptation across different vehicles or gradient-based optimization is needed. In real-world scenarios, where flexibility and generalizability are crucial, such constraints limit its practical usefulness.

To address these challenges, this chapter introduces a NVM as a compact and fully differentiable alternative. Its design combines physical structure with learnable components, enabling efficient training from telemetry data while preserving interpretability. The neural model offers a scalable way to approximate vehicle behaviour across platforms, with reduced identification effort and predictive accuracy comparable to the double-track model.

The remainder of the chapter presents both models, their validation results, and discusses the trade-offs between fidelity and practicality, motivating the adoption of the NVM in the rest of the thesis.

5.2 High-Fidelity Double-Track Model

5.2.1 Modelling Overview

The vehicle model is a high-fidelity, physics-based model of a rear-wheel-drive racecar. It is formulated as a 7 *Degrees of Freedom* (DoF) double-track model with roll dynamics and supports simulation over three-dimensional circuits. The model captures the main nonlinear interactions governing vehicle behaviour at the limit, including tyre force generation and load transfers.

The model includes the following key subsystems:

- *Tyre model:* Each wheel is modelled using the Pacejka Magic Formula 5.2, implemented with the PAC2002 parametrization [176]. This formulation captures combined slip, camber effects, and load sensitivity, enabling accurate simulation of grip saturation.
- *Powertrain and differential:* The engine model reproduces the measured torque curve of the real vehicle and includes a limited-slip differential with a nonlinear saturation characteristic.
- *Braking system:* A simplified *Anti-lock Braking System* (ABS) logic is implemented. It reduces the brake torque individually at each wheel based on slip thresholds to prevent wheel lock-up.
- *Traction control:* A basic traction control logic reduces engine torque when excessive longitudinal slip is detected at the driven (rear) wheels.

The model has 18 dynamic states and 2 control inputs, grouped in the vectors $\boldsymbol{x}$ and $\boldsymbol{u}$, respectively:

$$\begin{aligned} \boldsymbol{x} &= \left[n, \xi, v_x, v_y, \omega_x, \omega_y, \Omega, a_{x_e}, F_{z_{fl}}, F_{z_{fr}}, F_{z_{rl}}, F_{z_{rr}}, \omega_{fl}, \omega_{fr}, \omega_{rl}, \omega_{rr}, \delta, p\right]^\top \\ \boldsymbol{u} &= \left[\delta_{\text{dot}}, p_{\text{dot}}\right]^\top \end{aligned} \tag{5.1}$$

Here, n and ξ are the lateral displacement and relative yaw angle in curvilinear coordinates, v_x and v_y are the longitudinal and lateral velocities, and $(\omega_x, \omega_y, \Omega)$ denote the roll, pitch, and yaw rates. The states $F_{z_{ij}}$ represent the vertical loads at each wheel, and ω_{ij} are the corresponding wheel angular speeds. The control inputs are the steering wheel angle rate δ_{dot} and the pedal rate p_{dot}, where $p \in [-1, 1]$ denotes the throttle/brake command, and δ the steering angle at the front wheels.

The model captures the full-body and wheel dynamics of a high-performance vehicle operating at the limits of handling. It supports speeds exceeding 300 km/h and accelerations beyond $1g$. This level of fidelity makes it suitable for both closed-loop simulation of autonomous agents and for the formulation of benchmark trajectories via OCP.

5.2.2 Role in Simulation and Benchmarking

The high-fidelity double-track model described above has played a central role in the development of our research on autonomous racing. In earlier work, it served as the

simulated environment in which ARD was evaluated, enabling a systematic study of closed-loop performance across different circuits and driving conditions. This role as a *Vehicle Model used as a Simulator* (VMS) ensured that planning and control strategies were tested against realistic nonlinear and coupled vehicle dynamics, representative of behaviour at the limit.

Beyond its function as a simulator, the VMS has also been used in the past to define and solve a *Minimum Lap Time* (MLT) OCP [40]. By solving this problem offline with full knowledge of the circuit and system dynamics, time-optimal benchmark trajectories were obtained that served as performance upper bounds. These results, computed with the PINS software developed by our group, provided valuable context for evaluating the lap times achieved by ARD.

In the scope of this thesis, the VMS is primarily introduced as a reference model for validation and comparison. While its fidelity is unmatched, its vehicle-specific nature, identification requirements, and hard differentiability limit its suitability for the broader objectives of this work. This motivates the introduction of a NVM, which retains predictive accuracy while being more adaptable and easier to integrate into optimisation-based methods.

5.2.3 Validation on Real-World Data

Although the development of the VMS lies outside the scope of this thesis, its accuracy has been extensively validated using experimental data collected from a high-performance vehicle driven on track by a professional driver. The validation aims to ensure that the model faithfully reproduces the dynamics of the real system, thereby supporting its role as a credible digital twin for both simulation and offline benchmarking.

Due to confidentiality agreements, specific details about the vehicle and test environment cannot be disclosed. Nonetheless, the validation procedure is described here for completeness. Telemetry was recorded using a high-precision measurement setup, including RTK-GPS, inertial sensors, wheel encoders, and measurements of driver inputs such as steering and pedal commands.

The validation is formulated as an OCP in the spatial domain, where the model's control inputs are optimized to track the experimental telemetry data. The objective function penalizes deviations from key signals: vehicle speed, lateral displacement, and yaw rate. An additional filtering term is included to regularize the noisy yaw rate measurements.

The OCP is defined as:

$$\underset{\mathbf{u} \in \mathcal{U}}{\text{minimize}} \quad \int_0^{L_{\text{track}}} \frac{1}{v_\zeta(\zeta)} \Big(J_{v_G} + J_n + J_\Omega + J_{\text{filt}} \Big) \mathrm{d}\zeta \tag{5.2a}$$

$$\text{subject to} \quad \begin{cases} \dfrac{d\mathbf{x}(\zeta)}{d\zeta} = \dfrac{\boldsymbol{f}(\mathbf{x}(\zeta), \mathbf{u}(\zeta))}{v_\zeta(\zeta)}, & \text{(5.2b)} \\ \boldsymbol{b}(\mathbf{x}(0)) = 0, & \text{(5.2c)} \\ \boldsymbol{c}(\mathbf{u}(\zeta)) \leq 0 & \text{(5.2d)} \end{cases}$$

The cost terms in the objective function are defined as:

$$\begin{aligned} J_{v_G} &= \left(\frac{v_G - \hat{v}_G}{W_v} \right)^2, \\ J_n &= \left(\frac{n - \hat{n}}{W_n} \right)^2, \\ J_\Omega &= \left(\frac{\Omega - \hat{\Omega}_{\text{filt}}}{W_\Omega} \right)^2, \\ J_{\text{filt}} &= \left(\frac{\Delta\Omega}{W_{\text{filt}}} \right)^2 \end{aligned} \tag{5.3}$$

Here, v_G is the vehicle's absolute velocity, n is its lateral displacement, and Ω is its yaw rate. The symbols $\hat{v}_G$, $\hat{n}$, and $\hat{\Omega}$ denote the corresponding telemetry measurements, with $\hat{\Omega}_{\text{filt}}$ representing a filtered version of the yaw rate to attenuate noise and driver-induced transients. The filtering is handled via an auxiliary control variable $\Delta\Omega$, included in the cost with a weight W_{filt}.

Validation results are shown in Figures (5.1, 5.2, 5.3), where all signals have been normalized to preserve confidentiality. The model reproduces the measured dynamics with high fidelity across key states. In particular, the longitudinal speed v_x, yaw rate Ω, and both longitudinal and lateral accelerations show excellent agreement with telemetry. Vertical wheel loads and wheel angular speeds are also accurately captured.

These results confirm that the VMS is a suitable digital twin for simulation-based evaluation and offline benchmark generation. Its predictive accuracy supports its use as a trusted environment for testing autonomous planning algorithms and for solving minimum-lap-time OCPs.

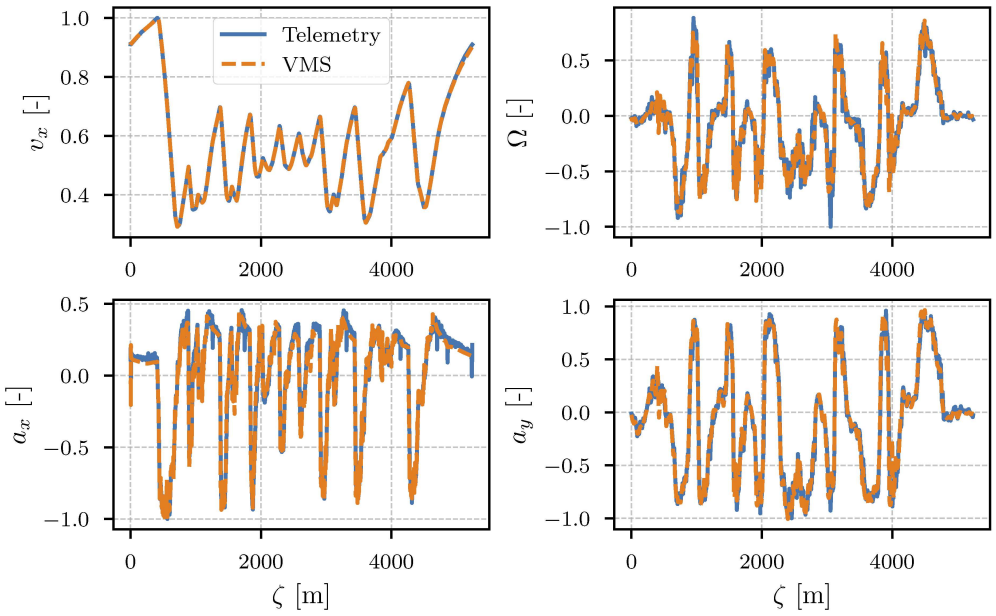


Figure 5.1: Normalized profiles of vehicle longitudinal speed v_x, yaw rate Ω, longitudinal acceleration a_x and lateral acceleration a_y, comparing the VMS with the real telemetry data.

5.2.4 Challenges in Model Identification

While the validated double-track model provides a highly accurate representation of the vehicle dynamics, its complexity poses significant challenges for identification and deployment on new platforms. The model includes numerous parameters related to suspension geometry, mass distribution, tyre behaviour, and powertrain characteristics. Many of these parameters are difficult to measure directly and require specialized equipment, controlled experiments, and expert knowledge to estimate accurately.

The identification process is further complicated by the strong coupling between parameters. Small errors in quantities such as tyre stiffness, aerodynamic coefficients, or suspension kinematics can lead to substantial deviations in the predicted dynamics. Achieving high-fidelity performance typically requires iterative calibration using extensive telemetry data collected under a wide range of manoeuvres. Such procedures are time-consuming and rely on domain expertise that may not always be available.

Another practical limitation is that the identified model is specific to a particular vehicle configuration and operating environment. If the vehicle is modified, or when transferring to a different platform, the identification process must be repeated (at least

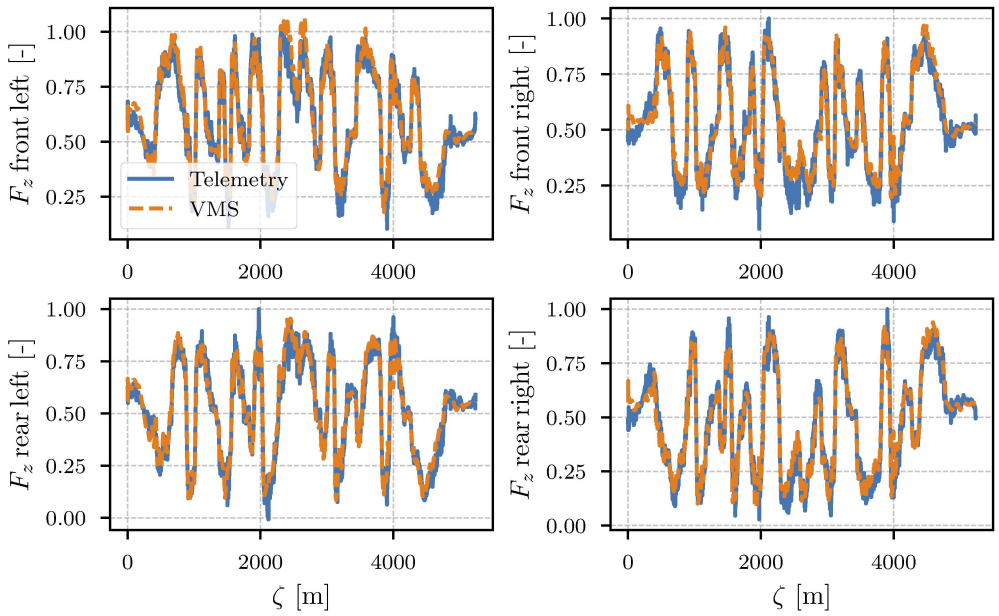


Figure 5.2: Normalized vertical wheel loads F_z on each wheel, comparing the VMS with the real telemetry data.

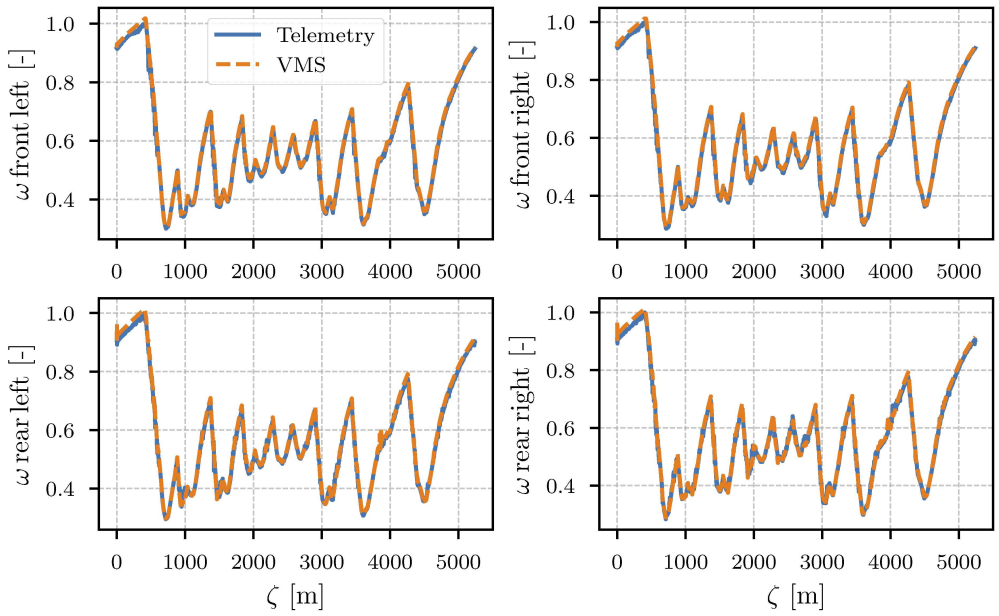


Figure 5.3: Normalized wheel rates for each wheel, comparing the VMS with the real telemetry data.

partially). This hampers transferability, especially in applications where rapid deployment is desirable.

These challenges motivate the exploration of alternative approaches to vehicle modelling that reduce the need for manual identification while retaining sufficient accuracy for simulation and control design. In particular, models that can be partially or fully learned from data offer the potential for semi-automatic identification, making them more practical for scenarios where extensive calibration efforts are not feasible.

5.3 Neural Vehicle Model as a Lightweight Alternative

5.3.1 Motivation and Scope

To address the double-track model limitations, we developed a NVM that combines a simplified vehicle structure with data-driven parameter identification. By retaining the physics of a single-track formulation with load transfer and tyre force generation, the model remains interpretable while requiring far fewer explicit parameters. It is implemented entirely in PyTorch [177] as a differentiable module, enabling gradient-based identification directly from telemetry data.

Only a small subset of quantities, such as vehicle mass or basic dimensions, must be specified manually, while all other parameters are trainable within physically plausible bounds. This structure not only simplifies the identification process but also enables the model to be fine-tuned as new data become available, either offline or during operation. As a result, the NVM can follow the gradual evolution of the vehicle's behaviour over time—for instance, due to tyre wear, thermal effects, or varying surface conditions—opening new possibilities for adaptive control within the ARD framework.

This greatly reduces the effort required to obtain an accurate model for a given vehicle. In the remainder of this section, we describe the structure of the model in detail and demonstrate its predictive capabilities by comparing it with a highly accurate simulation dataset.

Training and Validation Dataset

The dataset used for validating the NVM was collected in a high-fidelity driving simulator. Professional sim-racing drivers operated a detailed multi-body model of the same vehicle employed in the real-world dataset used for the high-fidelity double-track model. The use of simulated telemetry was motivated by the availability of a richer set of signals, which were required to validate the model thoroughly. Although only a subset of those signals is shown here for confidentiality reasons, we used telemetry such as lateral and longitudinal forces to ensure our model was learning the correct dynamics. By contrast, the signals needed for training form only a small subset that can be easily acquired in practice. The simulator and its underlying multi-body model are of sufficiently high accuracy that they are routinely employed in vehicle development prior to physical prototyping, lending strong credibility to the data adopted in this work.

5.3.2 Internal Structure and Modelling Assumptions

The model retains the same physical equations of a quasi-steady state double-track formulation, but departs from classical approaches in the derivation of their components. The derivation explicitly accounts for longitudinal and lateral load transfers, ABS and traction control mechanisms, as well as differential effects. This is achieved by considering the contributions of all four wheels throughout most of the derivation and then condensing them into a highly nonlinear single-track-like representation. The resulting formulation captures several key effects of high-performance vehicles with greater fidelity, while preserving the relative simplicity of single-track equations structure. Such a balance makes the model suitable for future use in OCP formulations. The entire implementation is developed in PyTorch, allowing efficient gradient computation and direct identification of unknown parameters from telemetry data.

5.3.2.1 State, Inputs, and Dynamics

The NVM models the following seven states:

$$\boldsymbol{x} = \begin{bmatrix} x & y & \psi & v_x & v_y & \Omega & \delta_f \end{bmatrix}^\top,$$

where (x, y) denote the position of the vehicle's *Centre of Gravity* (CoG) in the global frame, ψ is the yaw angle, (v_x, v_y) are the longitudinal and lateral components of the

velocity expressed in the vehicle body frame, Ω is the yaw rate, and δ_f is the front steering angle. The body frame definition follows the ISO 8855 [169] convention for ground-vehicles.

The control inputs are

$$\boldsymbol{u} = \begin{bmatrix} a_x & \delta_{f_{\text{ctrl}}} \end{bmatrix}^\top,$$

where a_x denotes the commanded longitudinal acceleration and $\delta_{f_{\text{ctrl}}}$ the commanded steering angle at the front wheels. Positive a_x corresponds to throttle, while negative values correspond to braking. This choice is consistent with the actuation interface of the experimental vehicles at the *Universität der Bundeswehr München* (UniBW), which accept longitudinal acceleration and front wheel angle as inputs. Using a_x directly avoids modelling intermediate powertrain and braking dynamics, which are not relevant for the present scope of closed-loop planning simulations. If required for other applications, the formulation could be extended by introducing a drivetrain model to handle separate throttle and brake commands.

The dynamics are derived from classical formulations [174], [178] as follows:

$$\begin{aligned}
\dot{x} &= v_x \cos\psi - v_y \sin\psi \\
\dot{y} &= v_x \sin\psi + v_y \cos\psi \\
\dot{\psi} &= \Omega \\
m(\dot{v}_x - v_y\Omega) &= F_{x_r} - F_{x_{\text{aero}}} - F_{\text{roll}} - F_{y_f}\sin\delta_f + F_{x_f}\cos\delta_f \\
m(\dot{v}_y + v_x\Omega) &= F_{y_r} + F_{y_f}\cos\delta_f + F_{x_f}\sin\delta_f \\
I_z\dot{\Omega} &= \ell_f(F_{y_f}\cos\delta_f + F_{x_f}\sin\delta_f) - \ell_r F_{y_r} + \Delta F_{x_r} b_r + \Delta F_{x_f} b_f + M_z \\
\dot{\delta}_f &= \text{clamp}\left(\frac{\delta_{f_{\text{ctrl}}} - \delta_f}{\tau_\delta}, -\dot{\delta}_{f_{\max}}, \dot{\delta}_{f_{\max}}\right)
\end{aligned} \tag{5.4}$$

Where F_{x_f} and F_{y_f} denote the longitudinal and lateral tyre forces at the front axle, while F_{x_r} and F_{y_r} are the corresponding forces at the rear axle, obtained as the sum of the individual tyre contributions (modelled or estimated) as described later. $F_{x_{\text{aero}}}$ represents the longitudinal aerodynamic drag, and F_{roll} the rolling resistance force. ℓ_f and ℓ_r are the distances from the CoG to the front and rear axles, b_f and b_r the front and rear track widths, m is the vehicle mass, and I_z the yaw inertia. τ_δ models the actuator time constant, and $\dot{\delta}_{f_{\max}}$ the maximum steering rate. The additional terms $\Delta F_{x_r} b_r$ and $\Delta F_{x_f} b_f$ account for the yaw moments induced by longitudinal force differences at the rear and front axles, respectively, while M_z collects the self-aligning moments contributions at the wheels. The modelling of each component is detailed in the following subsections.

5.3.2.2 Aerodynamic and Rolling Forces

The longitudinal aerodynamic drag [178] is computed as

$$F_{x,\text{aero}} = \tfrac{1}{2}\rho C_d v_x^2,$$

where ρ is the air density, C_d is a trainable drag coefficient (which already includes the surface area), and v_x is the longitudinal speed in the body frame.

The rolling resistance force [178] is modelled as

$$F_{\text{roll}} = c_0 mg,$$

where c_0 is a trainable coefficient, m is the vehicle mass, and g is the gravitational acceleration.

5.3.2.3 Longitudinal Forces

The commanded longitudinal force is first computed by combining the effect of the desired acceleration input a_x with the resistive forces and a steering correction term:

$$F_{\text{ctrl}} = ma_x + F_{x,\text{aero}} + F_{\text{roll}} + \frac{\ell_r}{\ell} ma_y \sin\delta_f,$$

where $a_y = v_x\Omega$ is the approximated lateral acceleration at the CoG, and δ_f is the front steering angle. The term $\frac{\ell_r}{\ell} m a_y \sin\delta_f$ approximates the longitudinal component of the front lateral force generated when the vehicle is cornering with a non-zero steering angle. We remark that F_{ctrl} is expressed in the vehicle reference frame, and thus needs to be projected in the tire reference frames before continuing (F^T_{ctrl}). This is done by accounting for the steering angle at the front wheels and the driving configuration as follows:

$$\begin{cases} F^T_{\text{ctrl}} = \frac{F_{\text{ctrl}}\cos\delta + \frac{\ell_r}{\ell} m a_y \sin\delta_f}{(1-\alpha)\cos\delta + \alpha}, & F_{\text{ctrl}} \geq 0 \quad \text{(throttle)}, \\ F^T_{\text{ctrl}} = \frac{F_{\text{ctrl}}\cos\delta + \frac{\ell_r}{\ell} m a_y \sin\delta_f}{(1-\beta)\cos\delta + \beta}, & F_{\text{ctrl}} < 0 \quad \text{(braking)}, \end{cases}$$

where $\alpha \in [0,1]$ is the driving distribution factor, fixed to reflect whether the vehicle is front- or rear-wheel driven, and $\beta \in [0,1]$ is a trainable braking distribution factor. Thus, positive (throttle) commands are applied entirely at the driven axle (rear in this case), while braking commands are split between the two axles.

The commanded longitudinal force F^T_{ctrl}, projected in the tire reference frames, is distributed between the front and rear axles depending on its sign, reflecting the vehicle's drive configuration:

$$F_{x_f} = \begin{cases} \alpha\,|F^T_{\text{ctrl}}|, & F_{\text{ctrl}} \geq 0 \quad \text{(throttle)}, \\ \beta\,|F^T_{\text{ctrl}}|, & F_{\text{ctrl}} < 0 \quad \text{(braking)}, \end{cases} \qquad F_{x_r} = \begin{cases} (1-\alpha)\,|F^T_{\text{ctrl}}|, & F_{\text{ctrl}} \geq 0, \\ (1-\beta)\,|F^T_{\text{ctrl}}|, & F_{\text{ctrl}} < 0, \end{cases}$$

Once the longitudinal force split between front and rear axles is defined, it is further distributed to the individual wheels. Since the total F_x is computed directly from a_x, the global effects of ABS, traction control, and differential mechanisms are already embedded. However, the resulting lateral distribution of F_x between the left and right wheels of each axle follows a highly nonlinear relationship that is difficult to express analytically. To capture this dependency explicitly, we employ two independent *Multi-Layer Perceptrons* (MLPs), one for the front and one for the rear axle. Each network has the common structure

$$\Delta F_x = \text{MLP}(a_x, a_y), \tag{5.5}$$

where the inputs are the commanded longitudinal acceleration a_x and the approximated lateral acceleration a_y. In practice, the yaw rate Ω should also be included among the inputs as it has a noticeable effect at low speeds, but it was not considered in the present implementation. Each MLP consists of a fully connected layer with 32 hidden units, followed by a tanh activation, and a final fully connected layer mapping to a single scalar output. This compact architecture is sufficient to capture the non-linearities governing the distribution.

The longitudinal force at each wheel is then computed as:

$$F_{x_{\text{fl}}} = \tfrac{1}{2}F_{x_f} - \Delta F_{x_f}, \qquad F_{x_{\text{rl}}} = \tfrac{1}{2}F_{x_r} - \Delta F_{x_r}, \tag{5.6}$$

$$F_{x_{\text{fr}}} = \tfrac{1}{2}F_{x_f} + \Delta F_{x_f}, \qquad F_{x_{\text{rr}}} = \tfrac{1}{2}F_{x_r} + \Delta F_{x_r}, \tag{5.7}$$

with appropriate clamping to preserve the total force per axle. This formulation allows the model to approximate the effective longitudinal forces generated at each wheel while keeping the overall representation lightweight. Figure 5.4 shows a visual representation of the longitudinal forces' computation.

5.3.2.4 Vertical Forces

The vertical load on each wheel is obtained by combining the static axle loads with the effects of aerodynamic downforce, and load transfers [176]. The static front and rear axle loads are given by

$$F^{\text{stat}}_{z_{\text{f}}} = mg\frac{\ell_r}{\ell_f + \ell_r}, \quad F^{\text{stat}}_{z_{\text{r}}} = mg\frac{\ell_f}{\ell_f + \ell_r},$$

where ℓ_f and ℓ_r are the distances from the CoG to the front and rear axles, respectively.

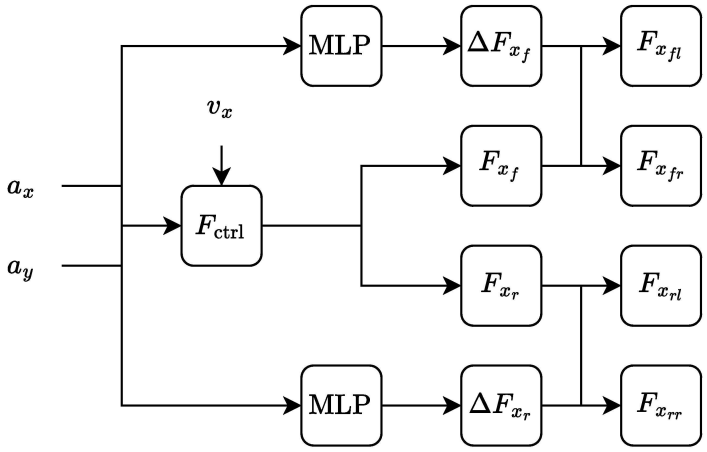


Figure 5.4: Schematic for the computation of the longitudinal forces.

The longitudinal load transfer is computed as

$$F_z^{\text{long}} = m a_x \frac{h_{\text{cg}}}{\ell_f + \ell_r},$$

where h_{cg} is the CoG height and a_x is the commanded longitudinal acceleration. Positive a_x causes a rearward load transfer, while negative a_x transfers load to the front axle.

The aerodynamic downforce is modelled as

$$F_z^{\text{aero}} = \tfrac{1}{2}\,\rho\, C_{\text{down}}\, v_x^2,$$

where ρ is the air density, C_{down} is a trainable coefficient that already includes the vehicle's effective reference area, and v_x is the longitudinal velocity. The distribution of the aerodynamic load between front and rear axles is defined by a second trainable parameter, $C_{\text{down\_split}} \in [0, 1]$, such that

$$F_{z_{\text{f}}}^{\text{aero}} = C_{\text{down\_split}}\, F_z^{\text{aero}}, \quad F_{z_{\text{r}}}^{\text{aero}} = (1 - C_{\text{down\_split}})\, F_z^{\text{aero}}.$$

The lateral load transfer at the front and rear axles is computed as:

$$F_z^{\text{lat}_f} = \frac{m a_y}{b_{\text{f}}} \left(\frac{\ell_r}{\ell_f + \ell_r} h_{\text{rc}} + \epsilon_\phi h_s \right) - \frac{h_{\text{rc}}}{b_f \ell} (\Delta F_{x_r} b_r + \Delta F_{x_f} b_f + M_z),$$

$$F_z^{\text{lat}_r} = \frac{m a_y}{b_{\text{r}}} \left(\frac{\ell_f}{\ell_f + \ell_r} h_{\text{rc}} + (1 - \epsilon_\phi) h_s \right) + \frac{h_{\text{rc}}}{b_r \ell} (\Delta F_{x_r} b_r + \Delta F_{x_f} b_f + M_z),$$

where $a_y = v_x \Omega$ is the approximated lateral acceleration, h_{rc} is the roll-centre height, and h_s is the vertical distance between the CoG and the roll axis. The parameter $\epsilon_\phi \in [0, 1]$ defines the roll stiffness ratio. The quantities b_{f} and b_{r} are the front and rear track widths, while ℓ_f and ℓ_r denote the distances from the CoG to the front and rear axles, respectively. The last term in each equation accounts for yaw-moment-induced load transfer: ΔF_{x_r} and ΔF_{x_f} are the longitudinal force asymmetries between left and right wheels on the rear and front axles, respectively, which, together with the self-aligning moment M_z, generate additional lateral load redistribution. We remark that computing ΔF_{x_f} from the front longitudinal tire forces is an approximation. One should first project them back to the vehicle reference frame, but this is not possible given we do not know the left/right front lateral tire forces at this stage.

Combining these contributions, the vertical loads at each wheel are

$$F_{z_{\text{fl}}} = \tfrac{1}{2} F_{z_{\text{f}}}^{\text{stat}} - \tfrac{1}{2} F_z^{\text{long}} - F_z^{\text{lat}_f} + F_{z_{\text{f}}}^{\text{aero}}, \tag{5.8}$$

$$F_{z_{\text{fr}}} = \tfrac{1}{2} F_{z_{\text{f}}}^{\text{stat}} - \tfrac{1}{2} F_z^{\text{long}} + F_z^{\text{lat}_f} + F_{z_{\text{f}}}^{\text{aero}}, \tag{5.9}$$

$$F_{z_{\text{rl}}} = \tfrac{1}{2} F_{z_{\text{r}}}^{\text{stat}} + \tfrac{1}{2} F_z^{\text{long}} - F_z^{\text{lat}_r} + F_{z_{\text{r}}}^{\text{aero}}, \tag{5.10}$$

$$F_{z_{\text{rr}}} = \tfrac{1}{2} F_{z_{\text{r}}}^{\text{stat}} + \tfrac{1}{2} F_z^{\text{long}} + F_z^{\text{lat}_r} + F_{z_{\text{r}}}^{\text{aero}}. \tag{5.11}$$

The use of both longitudinal and lateral load transfer represents a significant extension compared to traditional single-track models, which typically only account for longitudinal transfer. Each wheel load is lower-bounded by a minimum value to avoid lift-off.

5.3.2.5 Steering Dynamics

The steering actuator dynamics are represented by a first-order system with a rate limit, describing the evolution of the front-wheel steering angle δ_f in response to the commanded steering input $\delta_{f_{\text{ctrl}}}$:

$$\dot{\delta}_f = \text{clamp}\left(\frac{\delta_{f_{\text{ctrl}}} - \delta_f}{\tau_\delta}, -\dot{\delta}_{f_{\max}}, \dot{\delta}_{f_{\max}}\right),$$

where τ_δ is the actuator time constant and $\dot{\delta}_{f_{\max}}$ is the maximum steering rate. Both quantities can be treated as trainable parameters, but were not in this thesis (see Paragraph 5.3.2.11). This formulation captures the essential dynamics of real steering actuators, including finite response speed and rate saturation.

Once the mean front steering angle δ_f is obtained, the individual left and right wheel angles are computed to reflect asymmetries. These are defined as

$$\begin{aligned}
\delta_{\text{s}} &= \begin{cases} 1, & \delta_f \geq 0, \\ 0, & \delta_f < 0, \end{cases} \\
\delta_{\text{l}} &= -a_0 + \delta_{\text{s}}\, a_1\, \delta_f + (1 - \delta_{\text{s}})(2 - a_1)\, \delta_f + a_2\, \delta_f^2, \\
\delta_{\text{r}} &= 2\, \delta_f - \delta_{\text{l}},
\end{aligned} \tag{5.12}$$

where a_0, a_1, and a_2 are trainable coefficients governing the lateral distribution of steering between the left and right wheels, due to steering kinematics and suspension effect such as toe angles and roll steer.

This simple yet flexible parametrization allows the model to reproduce realistic steering asymmetries. Clamping is applied internally to preserve physical consistency between both wheel angles.

5.3.2.6 Lateral Slip Angles

The slip angle α_i [174], [176], [178] of each wheel i is computed as a function of the longitudinal and lateral vehicle velocities, yaw rate, and steering angle. The formulation accounts for the longitudinal position of the axles and the half track-widths, allowing the computation of individual slip angles for each wheel:

$$\alpha_{\text{fl}} = -\arctan\left(\frac{v_y + \ell_f \Omega - \delta_{\text{fl}}\big(v_x - \Omega b_{\text{f}}/2\big)}{v_x - \Omega b_{\text{f}}/2 + \delta_{\text{fl}} \ell_f \Omega}\right), \tag{5.13}$$

$$\alpha_{\text{fr}} = -\arctan\left(\frac{v_y + \ell_f \Omega - \delta_{\text{fr}}\big(v_x + \Omega b_{\text{f}}/2\big)}{v_x + \Omega b_{\text{f}}/2 + \delta_{\text{fl}} \ell_f \Omega}\right), \tag{5.14}$$

$$\alpha_{\text{rl}} = -\arctan\left(\frac{v_y - \ell_r \Omega}{v_x - \Omega b_{\text{r}}/2}\right), \tag{5.15}$$

$$\alpha_{\text{rr}} = -\arctan\left(\frac{v_y - \ell_r \Omega}{v_x + \Omega b_{\text{r}}/2}\right), \tag{5.16}$$

where b_{f} and b_{r} are the front and rear track widths, respectively.

This computation differs from the simplifications often employed in basic single-track models, where a single slip angle is associated with each axle. Here, the individual wheel slip angles allow for a more accurate evaluation of the combined slip effects when computing the lateral tyre forces.

5.3.2.7 Estimation of Longitudinal Slips

The estimation of longitudinal slips is based on an approximate inversion of the MF96 tyre model [176]. Instead of computing slip ratios from wheel kinematics, the slip is inferred directly from the relationship between longitudinal force and slip, thereby ensuring consistency with the tyre model used in the vehicle dynamics.

The pure longitudinal force in the MF96 formulation is expressed as

$$F_x(\kappa) = d_x \, \sin\big(c_x \, \arctan\big(b_x\kappa - e_x(b_x\kappa - \arctan(b_x\kappa))\big)\big)\,, \tag{5.17}$$

where κ is the longitudinal slip, b_x is the stiffness factor, c_x the shape factor, d_x the peak factor, and e_x the curvature factor.

For inversion, we assume $e_x = 0$, which simplifies Eq. (5.17) to

$$F_x(\kappa) \approx d_x \, \sin(c_x \, \arctan(b_x\kappa))\,. \tag{5.18}$$

Rearranging, the slip can be estimated as

$$\kappa \approx \frac{1}{b_x} \, \tan\left(\frac{1}{c_x} \arcsin\left(\frac{F_x}{d_x}\right)\right). \tag{5.19}$$

The approximation captures the increasing nonlinear stiffness of the tyre up to the peak force, but does not model the characteristic force reduction and subsequent plateau occurring at higher slips. For the purposes of parameter identification and closed-loop simulations, this level of approximation has proven sufficient. Importantly, only a reduced subset of MF96 parameters is required:

$$\{p_{cx1},\ p_{dx1},\ p_{dx2},\ p_{kx1},\ p_{kx3}\},$$

which define the effective stiffness, shape, and peak factors.

5.3.2.8 Lateral Forces

Once the vertical loads and slip angles are known, the lateral forces at each wheel are computed using the MF96 tyre formulation with combined-slip behaviour [176]. The pure lateral force is first evaluated as

$$F_{y0}(\alpha, F_z) = d_y \, \sin\Big(c_y \, \arctan\big(b_y\alpha - e_y(b_y\alpha - \arctan(b_y\alpha))\big)\Big), \tag{5.20}$$

where α is the side slip angle, and b_y, c_y, d_y, e_y are the stiffness, shape, peak, and curvature factors, respectively.

Combined-slip effects are introduced by scaling F_{y0} with the weighting function $G_{y\kappa}$:

$$G_{y\kappa}(\kappa, \alpha, F_z) = \frac{\cos\big(c_{yk} \, \arctan(b_{yk}\,\kappa)\big)}{\cos\big(c_{yk} \, \arctan(b_{yk}\, s_{hy\kappa})\big)}, \tag{5.21}$$

where κ is the longitudinal slip, F_z is the vertical load, b_{yk}, c_{yk} are shape and stiffness-like coefficients, and $s_{hy\kappa}$ is a small offset term. The resulting combined-slip lateral force is

$$F_y = G_{y\kappa}(\kappa, \alpha, F_z)\, F_{y0}(\alpha, F_z). \tag{5.22}$$

In this formulation, only a reduced subset of $G_{y\kappa}$ parameters is required for training:

$$\{rcy1,\ rby1,\ rby2\},$$

while other terms (*i.e.,* $rhy1$, $rhy2$, $rby3$) are set to zero. This keeps the parameter set compact and identifiable while retaining the ability to capture the key nonlinear effects of combined slip.

The lateral force at each wheel i is then evaluated as

$$F_{y_i} = G_{y\kappa_i}(\kappa_i, \alpha_i, F_{z_i})\, F_{y0_i}(\alpha_i, F_{z_i}), \tag{5.23}$$

where F_{z_i} is the vertical load on the wheel, κ_i is the estimated longitudinal slip, and α_i is the slip angle. Finally, the total lateral forces at the front and rear axles are obtained by summing the contributions of the left and right wheels:

$$F_{y_f} = F_{y_{\mathrm{fl}}} + F_{y_{\mathrm{fr}}}, \qquad F_{y_r} = F_{y_{\mathrm{rl}}} + F_{y_{\mathrm{rr}}}. \tag{5.24}$$

This approach captures load-dependent and combined-slip effects consistently with the MF96 formulation, while keeping the number of trainable parameters limited for efficient identification. Figure 5.5 visually shows the computation of the lateral forces.

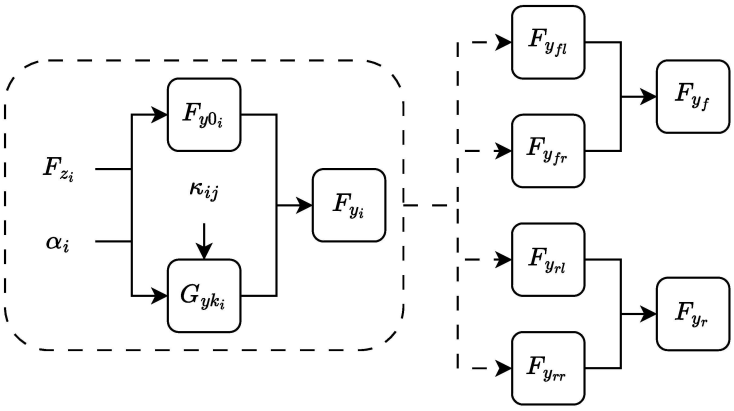


Figure 5.5: Schematic for the computation of the lateral forces.

5.3.2.9 Implementation in PyTorch

The NVM is implemented in Python using the PyTorch framework, which provides a convenient balance between modelling flexibility and computational efficiency. The implementation is modularly organized, where each main component is encapsulated within a dedicated `nn.Module`. In particular, the tyre model and the vehicle model are implemented as separate modules, allowing them to be trained, tested, and reused independently.

The use of PyTorch enables automatic differentiation of all model quantities, allowing gradient-based identification of physical parameters directly from telemetry data. At the same time, the modular design makes it straightforward to swap or refine individual components without altering the overall model interface. While the current implementation runs entirely in Python, its design is compatible with a future migration of critical modules to C++ for improved learning performance.

5.3.2.10 Parameter Constraints

All trainable parameters of the model are implemented as `torch.nn.Parameter` objects and constrained using PyTorch's `torch.nn.utils.parametrize` functionality. Each parameter is associated with lower and upper bounds through the utility function `bound_parameter`, which registers a custom parametrization module. The raw parameter is optimized in an unconstrained space, while the parametrization maps it to the valid interval using a scaled sigmoid function. This ensures that parameters remain within their bounds throughout training while preserving smooth gradients, even when approaching the limits.

The bounds for the tyre parameters are derived from the reference values reported in Pacejka's book [176, Annex 3]. Bounds for vehicle-level parameters such as roll-centre and CoG heights are chosen based on typical values for passenger or high-performance vehicles. Wherever possible, nominal values are obtained directly from available technical information or estimated using simple relations. For instance, the mass and geometric

dimensions can usually be measured or found in technical specifications, while the yaw inertia can be approximated as

$$I_z = m\left(\frac{\ell_f}{\ell}\ell_r^2 + \frac{\ell_r}{\ell}\ell_f^2\right),$$

where $\ell = \ell_f + \ell_r$ is the wheelbase.

This approach provides a clean and efficient way to enforce constraints directly within PyTorch's optimization pipeline. Because the parametrization is part of the computation graph, gradients flow seamlessly through the constrained parameters, enabling fully differentiable identification without manual projection or clamping. This is particularly beneficial for tyre parameters, where unconstrained optimization could otherwise lead to unrealistic values and poor generalization outside the training dataset.

5.3.2.11 Model Parameters

Table 5.1 lists all parameters of the NVM, together with their meaning, units, and whether they were treated as trainable in this thesis. Most parameters were optimized directly from telemetry data, with only a few kept fixed. Tyre parameters follow the MF96 formulation and are initialized within bounds derived from Pacejka's book [176, Annex 3].

Table 5.1: Parameters of the NVM. The last column indicates which parameters were treated as trainable in this thesis. Tyre parameters are provided separately for front and rear axles.

Parameter	Meaning	Units	Trainable
m	Vehicle mass	kg	No
I_z	Yaw inertia	$\mathrm{kg\,m^2}$	No
ℓ_f, ℓ_r	Distances from CG to front/rear axles	m	No
b_f, b_r	Front/rear track widths	m	No
r_f, r_r	Front/rear rolling radii	m	No
v_max	Maximum allowed longitudinal speed	m/s	No
δ_min, δ_max	Steering angle limits	rad	No
$\dot{\delta}_\mathrm{max}$	Max steering rate	rad/s	No
τ_δ	Steering time constant	s	No
h_cg	CoG height	m	Yes
h_rc	Roll-centre height	m	Yes
C_down	Aerodynamic downforce coefficient	$1/\mathrm{m^2}$	Yes
$C_\mathrm{down\_split}$	Aerodynamic downforce distribution	–	Yes
C_d	Aerodynamic drag coefficient	$1/\mathrm{m^2}$	Yes
c_0	Rolling resistance coefficient	–	Yes
α	Driving force split factor	–	No
β	Braking force split factor	–	Yes
ϵ_ϕ	Roll stiffness ratio	–	Yes
Min wheel load	Minimum vertical load per wheel	N	No
p_{cx1}, p_{dx1}, p_{dx2}, p_{kx1}, p_{kx3}, p_{cy1}, p_{dy1}, p_{dy2}, p_{ey1}, p_{ky1}, p_{ky2}, r_{cy1}, r_{by1}, r_{by2}	MF96 tyre parameters (front/rear)	–	Yes

Parameter availability and bounds. A key aspect of training the NVM is the availability and bounding of parameters. Quantities that are not treated as trainable, such as vehicle mass, dimensions, or inertial properties, are generally available with high accuracy and can be considered fixed. In contrast, tyre parameters and other suspension-related quantities are typically difficult to measure directly, and must therefore be identified from data. Proper bounding of these parameters is essential, as the same

kinematic behaviour of the states $(x, y, \psi, v_x, v_y, \Omega, \delta_f)$ can be reproduced by different underlying force distributions. For example, the parameter ϵ_ϕ, which defines part of the distribution of lateral load transfer between front and rear axles, has a significant impact: if it is incorrectly bounded (*e.g.,* biased towards the front while the real vehicle is rear-biased), the model may still achieve a good fit of the observable states while generating incorrect tyre forces. If tyre-force data are unavailable, such discrepancies would remain undetected. While this limitation is acceptable when only state trajectories are of interest, replicating the correct tyre-level behaviour requires bounds that reflect at least the dominant physical tendencies of the system. This consideration motivated the use of high-fidelity simulator data in this thesis, as noted earlier. The additional signals available during validation enabled us to ensure that the identified model captured not only the observable vehicle states but also the underlying tyre forces with sufficient accuracy.

The steering actuator parameters τ_δ and $\dot{\delta}_{f_{\max}}$ were not treated as trainable for similar reasons. Since no telemetry was available from autonomous driving experiments, we could not identify these parameters from data. As a result, actuator dynamics had to be estimated from known specifications of one available autonomous steering system. In principle, these parameters could be learned from data, but doing so would require telemetry recorded during automated operation to ensure that the identified values reflect the actual actuation behaviour.

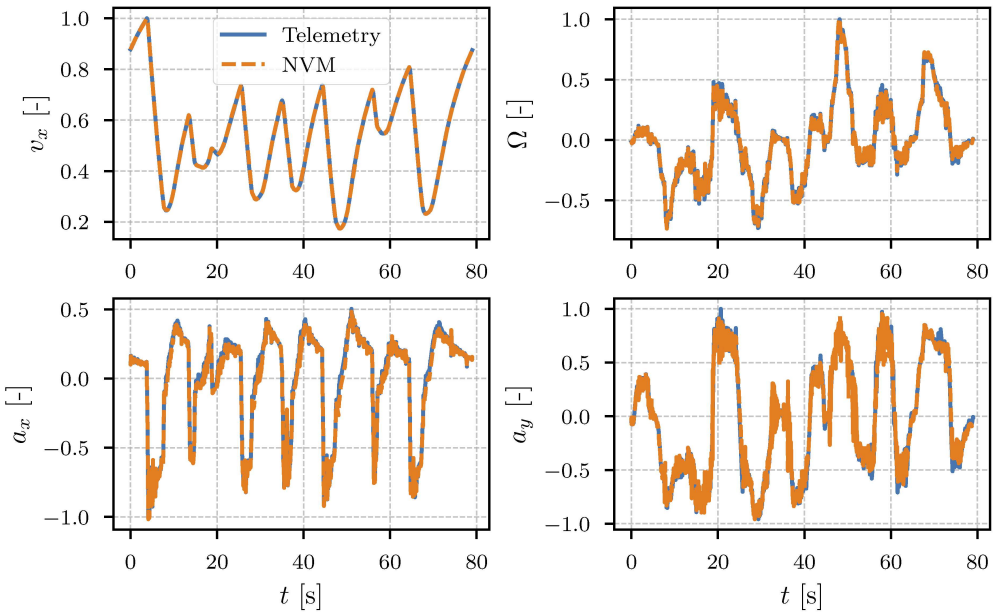


Figure 5.6: Normalized profiles of vehicle longitudinal speed v_x, yaw rate Ω, longitudinal acceleration a_x, and lateral acceleration a_y: comparison between the NVM and simulated telemetry.

5.3.3 Comparative Identification Results

The NVM was identified using the dataset introduced in the previous subsection, collected from a high-fidelity driving simulator with professional sim-racing drivers operating a detailed multi-body vehicle model. All signals were normalized to comply with confidentiality agreements. The identification relied on gradient-based optimization of the trainable parameters within the bounds discussed earlier, while geometric quantities such as axle distances and track widths were kept fixed.

The dataset used for identification included the vehicle states $[x, y, \psi, v_x, v_y, \Omega, \delta_f]$, which were directly matched in the cost function, and the control inputs $[a_x, \delta_{f_{\text{ctrl}}}]$. In addition, the wheel rotational speeds $[\omega_{\text{fl}}, \omega_{\text{fr}}, \omega_{\text{rl}}, \omega_{\text{rr}}]$ were included in the cost function exclusively during training. Although wheel rates are not explicit states of the model, they

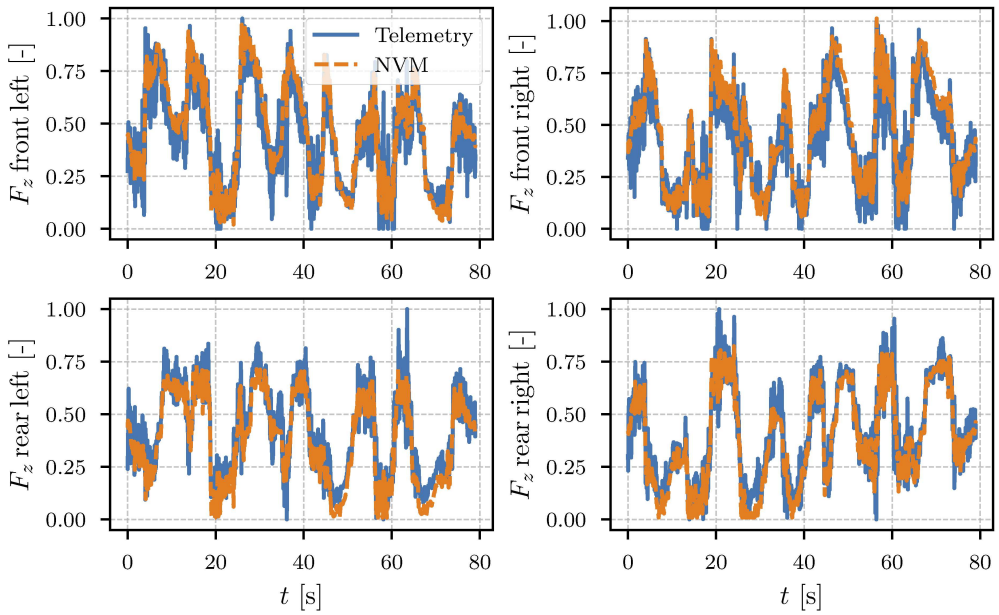


Figure 5.7: Normalized vertical wheel loads F_z on each wheel: comparison between the NVM and simulated telemetry.

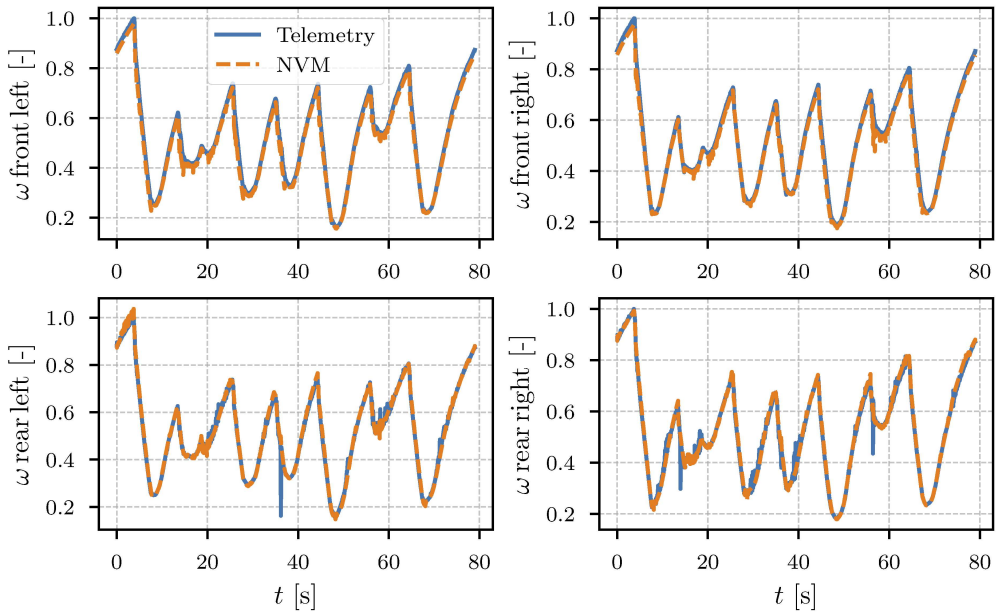


Figure 5.8: Normalized wheel rotational speeds: comparison between the NVM and simulated telemetry.

can be computed internally and provide valuable information for improving the estimation of longitudinal slips. Their use is not strictly necessary—comparable accuracy can be achieved without them—but they are standard signals available on most production vehicles via the CAN bus and were therefore considered practical to include. The detailed training procedure is described in Section 6.5.1 and is not repeated here.

The validation of the NVM was carried out by simulating the model over the entire recorded telemetry, initializing it only once at the start of the lap and subsequently feeding it the measured control inputs. To emulate the behaviour permitted in the OCP-based validation of the double-track model, small corrective actions were applied through longitudinal and lateral PID controllers. These acted as lightweight feedback terms, compensating for minor residual discrepancies between model and data. This setup

demonstrates that, given minimal corrective inputs, the NVM can accurately reproduce the full telemetry sequence, confirming its capability to capture the vehicle's dynamics over long horizons with high fidelity.

Figures 5.6, 5.7, and 5.8 show the comparison between the NVM predictions and the simulated telemetry for the main vehicle states and wheel-related quantities. The structure of these figures mirrors that of the double-track validation, enabling a direct visual comparison between the two models.

The results show that the NVM closely reproduces the dynamics observed in the simulated telemetry. Longitudinal speed, yaw rate, and accelerations follow the reference profiles with good accuracy. Vertical wheel loads and wheel rotational speeds are also well captured, despite the reduced order of the model and the smaller set of trainable parameters. The reduced identification effort and full differentiability make it a practical alternative to the double-track model when rapid deployment or frequent re-identification is required.

Chapter 6

Learning to Race from Minimal Telemetry Data

Abstract

This chapter presents the data-driven pipeline that enables the *Artificial Race Driver* (ARD) to identify vehicle-specific dynamic models and controllers from minimal real-world telemetry data. By exploiting only two laps of high-speed driving recorded by a human driver, the approach systematically identifies the kineto-dynamical vehicle model required for minimum-time *Economic Nonlinear Model Predictive Control* (E-NMPC) planning, the *Physics-Informed Steering Neural Network* (PhS-NN) used for low-level lateral control, and the *Neural Vehicle Model* (NVM) employed for closed-loop validation and feedback controller tuning. Through targeted preprocessing, structured identification procedures, and systematic neural network training, ARD rapidly transitions from no prior knowledge of the vehicle to near-optimal autonomous performance.

Contents

6.1 Introduction . . . 73
6.2 Data Acquisition and Preprocessing . . . 74
6.2.1 Two-Lap Telemetry Collection . . . 76
6.2.2 Signal Preprocessing: Filtering and Corrections . . . 76
6.2.3 Computation of Derived Quantities . . . 77
6.3 Identification of the Kineto-Dynamical Model for E-NMPC . . . 77
6.3.1 Construction of the g-g-v Diagram . . . 78
6.3.2 Extraction of Lateral Velocity Characteristics . . . 80
6.3.3 Identification of First-Order Dynamics . . . 80
6.4 Low-Level Controller Identification . . . 81
6.4.1 Training the PhS-NN Lateral Controller . . . 81
6.4.2 Considerations for Longitudinal Control Identification . . . 82
6.5 Closed-Loop Identification . . . 83
6.5.1 Training the Neural Vehicle Model . . . 83
6.5.2 Closed-Loop Identification and Corrective Control . . . 85
6.6 On Error Detection and Interpretability of Physics-Informed Models 87

6.1 Introduction

Driving a vehicle consistently at its handling limits requires accurate knowledge of its dynamic behaviour. Such knowledge is typically encapsulated within detailed vehicle models—ranging from physics-based to fully data-driven—that enable precise motion planning and control. However, acquiring high-fidelity models often demands extensive measurements, expert calibration, and significant experimental effort, as discussed in Chapter 5. In practice, several key parameters, such as tyre characteristics, remain uncertain due to the difficulty of measuring forces directly, limiting the confidence of purely simulation-based approaches.

Previous work by Piccinini [152] demonstrated minimum-time manoeuvre planning using high-fidelity vehicle models validated in simulation. While this method yielded excellent simulated performance, its direct transfer to real vehicles remains challenging. Relying on pre-validated models introduces sensitivity to model mismatch and unmodelled

real-world effects, which can degrade both performance and safety. Moreover, such models are rarely reusable across vehicles, as each new platform demands a complete identification and validation cycle.

To address these limitations, this chapter presents a data-driven identification pipeline that learns directly from minimal telemetry data, reducing dependence on detailed pre-validated models. The approach relies on just two laps of high-speed driving performed by a human driver, from which it extracts the essential dynamic information required for minimum-time motion planning and control.

The dataset includes a concise yet informative set of signals: vehicle position (x, y), yaw angle ψ, longitudinal and lateral velocities (v_x, v_y), yaw rate (Ω), accelerations (a_x, a_y), trajectory curvature (ρ), commanded and measured steering angles $(\delta_{\text{ctrl}}, \delta)$, and wheel rotational speeds $(\omega_{\text{fl}}, \omega_{\text{fr}}, \omega_{\text{rl}}, \omega_{\text{rr}})$.

From these signals, the pipeline identifies the key components of the complete ARD framework: the kineto-dynamical model, and its constraints, used for E-NMPC-based trajectory planning; the PhS-NN lateral controller; and the NVM employed for closed-loop validation and feedback controller tuning.

Although some quantities, such as the lateral velocity v_y, are noisy and difficult to measure directly, their estimation remains critical for achieving maximum performance. Accurate knowledge of lateral dynamics is indispensable when operating near the vehicle's physical limits, and such measurements are feasible with instrumentation commonly available on modern test vehicles. The remaining signals are either already available on most vehicles or can be easily measured/computed from other quantities (*e.g.,* curvature).

In summary, this chapter presents a data-efficient learning pipeline that enables ARD to reach near-optimal autonomous performance with minimal prior information. By combining structured neural identification with real-world telemetry, the method provides a practical and scalable route toward rapid adaptation across different vehicles and racing environments. Figure 6.1 shows a schematic overview of the entire identification process.

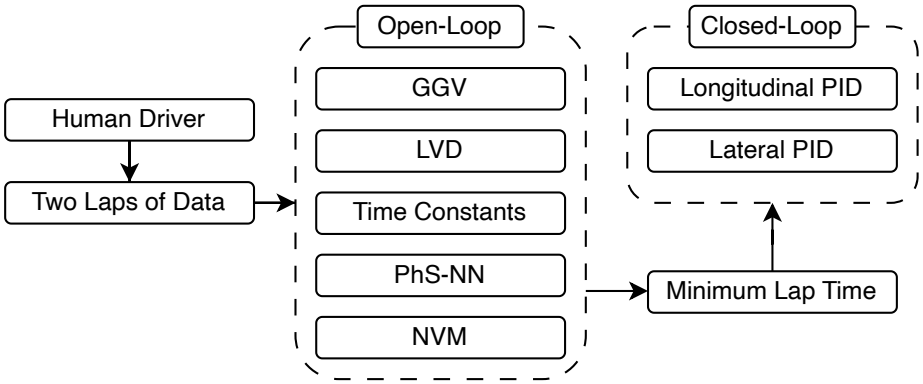


Figure 6.1: Overview of the entire identification pipeline. From just two laps of human driven data, we first identify in open-loop the components needed by the kineto-dynamical model, the low-level lateral controller, and the digital twin vehicle model. Then, we compute the minimum lap time and proceed with the identification of the closed-loop corrective controllers.

6.2 Data Acquisition and Preprocessing

Accurate identification relies not only on the quality of the learning algorithms but also on the integrity of the data on which they operate. This section details the acquisition and preparation of the telemetry used for model and controller identification. It first describes the two-lap data collection procedure adopted to obtain a minimal yet representative dataset, then outlines the filtering and correction steps, and finally explains the computation of derived quantities required for subsequent learning stages. The resulting processed telemetry, part of which illustrated in Figure 6.2, forms the basis for all identification and validation tasks presented in this chapter.

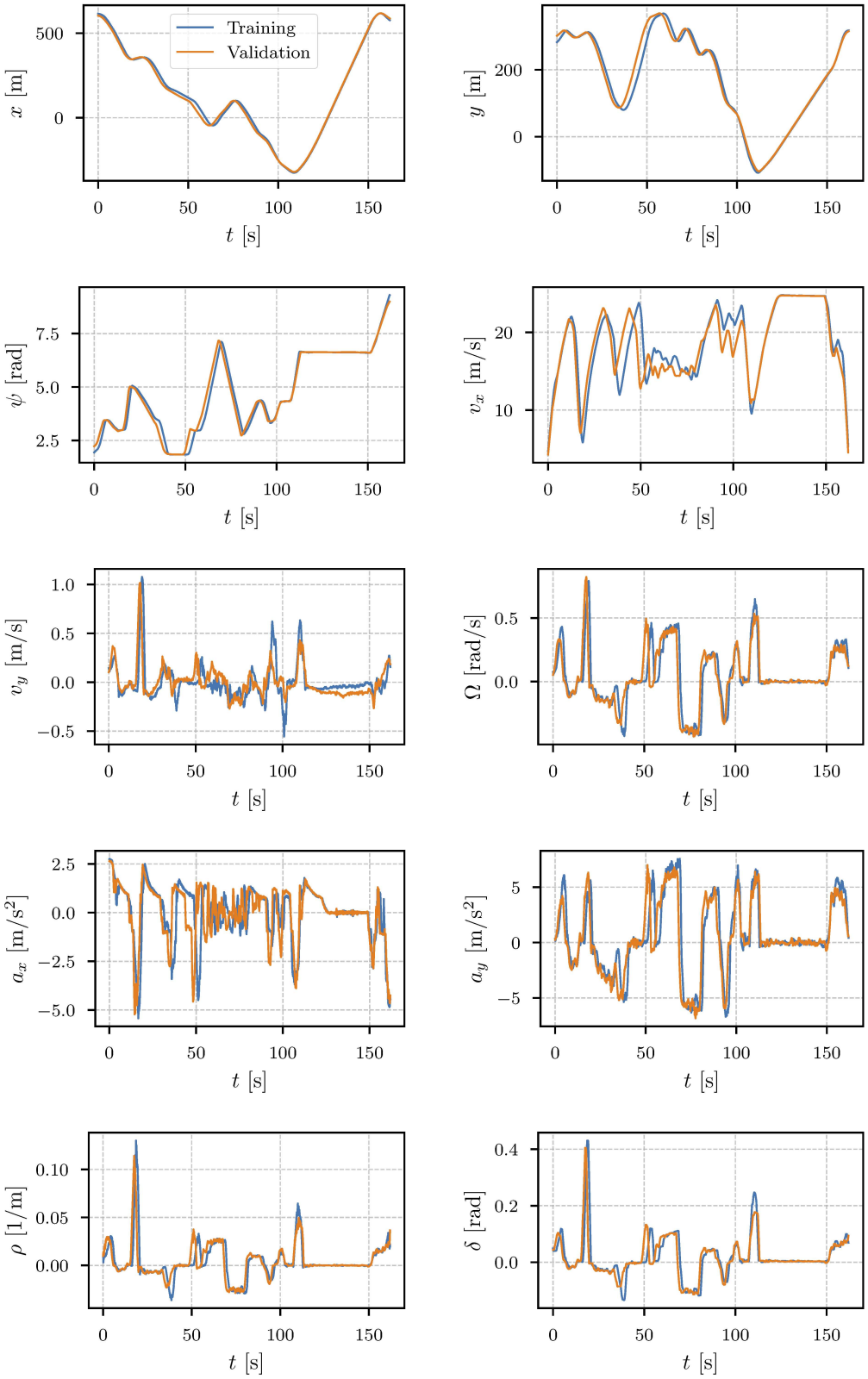


Figure 6.2: Subset of the real-world telemetry signals (post-processed) for the identification of the autonomous Volkswagen eCrafter (training in blue, validation in orange). From top to bottom and left to right: position (x, y), yaw angle ψ, longitudinal and lateral velocities v_x and v_y, yaw rate Ω, longitudinal and lateral accelerations a_x and a_y, transient curvature ρ, and steering angle δ.

Disclaimer: The identification results presented in this chapter refer to the real-world data obtained with an autonomous *Volkswagen eCrafter* platform. Due to non-disclosure agreements, the racecar model data cannot be publicly shown.

6.2.1 Two-Lap Telemetry Collection

The core strength of the proposed identification pipeline lies in its simplicity and minimal data requirements. The data collection process consists of recording two laps driven at the limits of a human driver's capabilities. This requires only a driver, a sensor-equipped vehicle, and a racetrack—the absolute minimum infrastructure for autonomous racing.

In practice, the driver completes two laps around the circuit at the highest achievable pace. Ideally, telemetry should be collected from "launched laps", meaning that the vehicle enters the lap already at speed rather than starting from rest, although this is not strictly necessary.

To enhance the generalization capability of the identification process, the two laps should differ slightly in configuration or route. Most circuits support multiple layouts, making such variation straightforward. Alternatively, the circuit can be driven in reverse direction or along a slightly different racing line. This variation allows the system to capture a broader range of the vehicle's dynamic behaviour.

It is also beneficial for the driver to adopt an aggressive driving style. Aggressive manoeuvres explore the performance envelope more thoroughly, providing richer and more informative data for system identification.

Despite these advantages, the quality of the resulting model inevitably depends on the driver's ability to approach the vehicle's true handling limits. An inexperienced driver who does not fully exploit the available performance will produce less informative telemetry, limiting identification accuracy. However, known physical characteristics of vehicle dynamics can mitigate this issue. For example, aerodynamic effects typically yield a g-g-v diagram with an expanding envelope as speed increases[1]. This property enables reasonable extrapolation of the vehicle's potential performance beyond the directly observed data, since even professional drivers rarely achieve optimal combined accelerations at very high speeds.

Another practical limitation concerns the characteristics of the chosen training track. A fast, straightforward circuit may not sufficiently expose the vehicle's cornering capabilities, while a slow and technical layout might not represent high-speed dynamics. Track selection therefore has a marked influence on the generalization capability of the identified model. Nonetheless, given that only two laps are required, it is both feasible and advisable to retrain and fine-tune the model for each circuit on which ARD operates. This process typically takes less than one hour and offers substantial performance gains relative to the minimal time investment.

6.2.2 Signal Preprocessing: Filtering and Corrections

Following data collection, the recorded telemetry undergoes preprocessing steps to ensure data quality and consistency. Initially, all signals are filtered using a Savitzky-Golay filter, which effectively reduces noise in measurements without significantly altering accurate data. This filtering particularly benefits noisy signals such as accelerations, providing cleaner data for subsequent identification procedures.

Next, signals must be referenced to the vehicle's *Centre of Gravity* (CoG). If data are already referenced correctly, no action is required. However, for sensor placements not coincident with the CoG, as is the case with the *Universität der Bundeswehr München*

[1] Except for the longitudinal acceleration, which tends to decrease due to powertrain saturation and resistive forces.

(UniBW) vehicles, corrections must be applied. These corrections are performed as follows:

$$\begin{aligned} v_x &\leftarrow v_x + \Omega \cdot y_{\text{shift}} \\ v_y &\leftarrow v_y - \Omega \cdot x_{\text{shift}} \\ a_x &\leftarrow a_x + \dot{\Omega} \cdot y_{\text{shift}} + \Omega^2 \cdot x_{\text{shift}} \\ a_y &\leftarrow a_y - \dot{\Omega} \cdot x_{\text{shift}} + \Omega^2 \cdot y_{\text{shift}} \end{aligned} \tag{6.1}$$

where x_{shift} and y_{shift} represent the distances from the sensor location to the CoG along the vehicle axes, and $\dot{\Omega}$ is the yaw acceleration computed as a numerical gradient of yaw rate Ω.

Finally, a qualitative comparison between corrected measurements and approximate reference signals is performed to validate data integrity, ensuring the corrected and filtered telemetry accurately reflects the vehicle's physical behaviour.

6.2.3 Computation of Derived Quantities

Following the preprocessing stage, certain additional derived quantities required for subsequent model identification must be computed. One critical derived quantity, typically not directly measured, is the transient curvature ρ, which is computed using the following relations:

$$\begin{aligned} \dot{v}_x &= a_x + \Omega \cdot v_y \\ \dot{v}_y &= a_y - \Omega \cdot v_x \\ \dot{\beta} &= \frac{\dot{v}_y v_x - \dot{v} x v_y}{v^2} \\ \rho &= \frac{\dot{\beta}}{v} + \frac{\Omega}{v} \end{aligned} \tag{6.2}$$

where $v = \sqrt{v_x^2 + v_y^2}$ is the total vehicle speed, $\dot{\beta}$ is the rate of change of the vehicle's sideslip angle, and Ω is the yaw rate. These computations are carried out directly on the preprocessed telemetry data to yield the transient curvature used in later identification stages.

6.3 Identification of the Kineto-Dynamical Model for E-NMPC

The kineto-dynamical model forms the core of the minimum-time motion planning strategy employed by ARD. To briefly reiterate, the *Optimal Control Problem* (OCP) solved within the E-NMPC scheme can be expressed as:

$$\min_{\boldsymbol{u}(\zeta)} \quad \Phi(\zeta_f) + \Phi(\zeta_i) + \int_{\zeta_i}^{\zeta_f} \mathcal{L}(\boldsymbol{x}(\zeta), \boldsymbol{u}(\zeta))\, d\zeta \tag{6.3a}$$

$$\text{s.t.} \quad \frac{d\boldsymbol{x}(\zeta)}{d\zeta} = \boldsymbol{f}(\boldsymbol{x}(\zeta), \boldsymbol{u}(\zeta)), \quad \text{kineto-dynamical model,} \tag{6.3b}$$

$$\boldsymbol{b}(\boldsymbol{x}(\zeta_i)) = 0, \quad \text{initial boundary conditions,} \tag{6.3c}$$

$$n(\zeta) \in \mathcal{R}(\zeta), \quad \text{track boundaries,} \tag{6.3d}$$

$$(\boldsymbol{x}(\zeta), \boldsymbol{u}(\zeta)) \in \mathcal{G}(\cdot), \quad \text{g-g-v envelope} \tag{6.3e}$$

where the kineto-dynamical model $\boldsymbol{f}(\cdot)$ is described by the following set of equations:

$$\begin{aligned} \dot{v}_x(t) &= a_x(t), \\ \tau_{v_y} \dot{v}_y(t) + v_y(t) &= \mathcal{F}_{v_y}(a_y(t), v_x(t), a_x(t)), \\ \tau_\Omega \dot{\Omega}(t) + \Omega(t) &= \Omega_{\text{ctrl}}(t), \\ \tau_{a_x} \dot{a}_x(t) + a_x(t) &= a_{x_{\text{ctrl}}}(t). \end{aligned} \tag{6.4}$$

In this section, we identify the essential parameters and characteristics required by the kineto-dynamical model, specifically focusing on:

- Construction of the g-g-v diagram, encapsulating the vehicle's achievable combined acceleration limits as a function of v_x;
- Extraction of lateral velocity characteristics ($\mathcal{F}_{v_y}$);
- Identification of the first-order time constants (τ_{v_y}, τ_Ω, τ_{a_x}) representing the dynamic response of lateral velocity, yaw rate, and longitudinal acceleration, respectively.

Each of these identification steps will be demonstrated using the real-world telemetry data partly shown in Figure 6.2.

6.3.1 Construction of the g-g-v Diagram

The g-g-v diagram encapsulates the vehicle's combined acceleration limits as a function of v_x, and it is essential for defining dynamic constraints within the kineto-dynamical E-NMPC model. As described in Sections 3.2.3.2 and 3.2.3.3, the g-g-v diagram comprises three main components: a convex polytope representing combined accelerations, nonlinear pure longitudinal acceleration limits, and nonlinear pure lateral acceleration limits. Although an additional combined braking limit could theoretically be included, as done in past work, real-world telemetry rarely contains such extreme conditions, rendering this additional constraint unnecessary.

The procedure for constructing the g-g-v diagram from telemetry data is semi-automated and can be summarized as follows:

1. **Data Loading and Visual Inspection:** Telemetry data are initially loaded and visually inspected to assess quality and completeness. At this stage, symmetry with respect to lateral acceleration (a_y) is assumed, and data points are mirrored accordingly—a valid approximation for most vehicles.
2. **Identification of Non-Uniform Data Distribution:** Telemetry data are typically unevenly distributed across different speed regions. Dense, informative data are often available at medium speeds, where both combined braking and acceleration conditions occur, whereas data at very low speeds (*e.g.,* vehicle launch or corner exit) and at very high speeds (*e.g.,* on long straights) are sparse or limited to near-pure longitudinal motion.
3. **Extrapolation and Data Filling:** To ensure full coverage of the g-g-v space, the dataset is extended at both low and high speeds. At low speeds, an optimistic extrapolation is applied, as the associated dynamics are mild and pose minimal risk. At high speeds, a conservative extrapolation is preferred to account for potential stability degradation. Two threshold speeds—defining the low- and high-speed extrapolation regions—are specified interactively by the user. Data points below the lower threshold are repeated from the threshold down to zero speed, the same is then done for the upper threshold up to the maximum speed. The result is a denser g-g-v.
4. **Identification of Pure Acceleration Limits:** The boundaries of pure throttle, braking, and lateral acceleration are identified by projecting the 3D data onto 2D planes and extracting the outer boundaries using an alpha-shape algorithm. The alpha parameter is tuned interactively to accurately capture the envelope of the data without being excessively conservative or optimistic.
5. **Boundary Smoothing with B-Splines:** Once the limits are identified, cubic smoothing B-Splines are applied to reduce local irregularities and discontinuities. This smoothing improves the convergence of the E-NMPC solver by smoothing sharp non-convexities in the identified boundaries.
6. **Convex Polytope Fit:** A convex polytope is then fitted to the filled-in dataset to represent the vehicle's combined acceleration capabilities. This is achieved using a 3D convex hull algorithm applied to the full set of points.

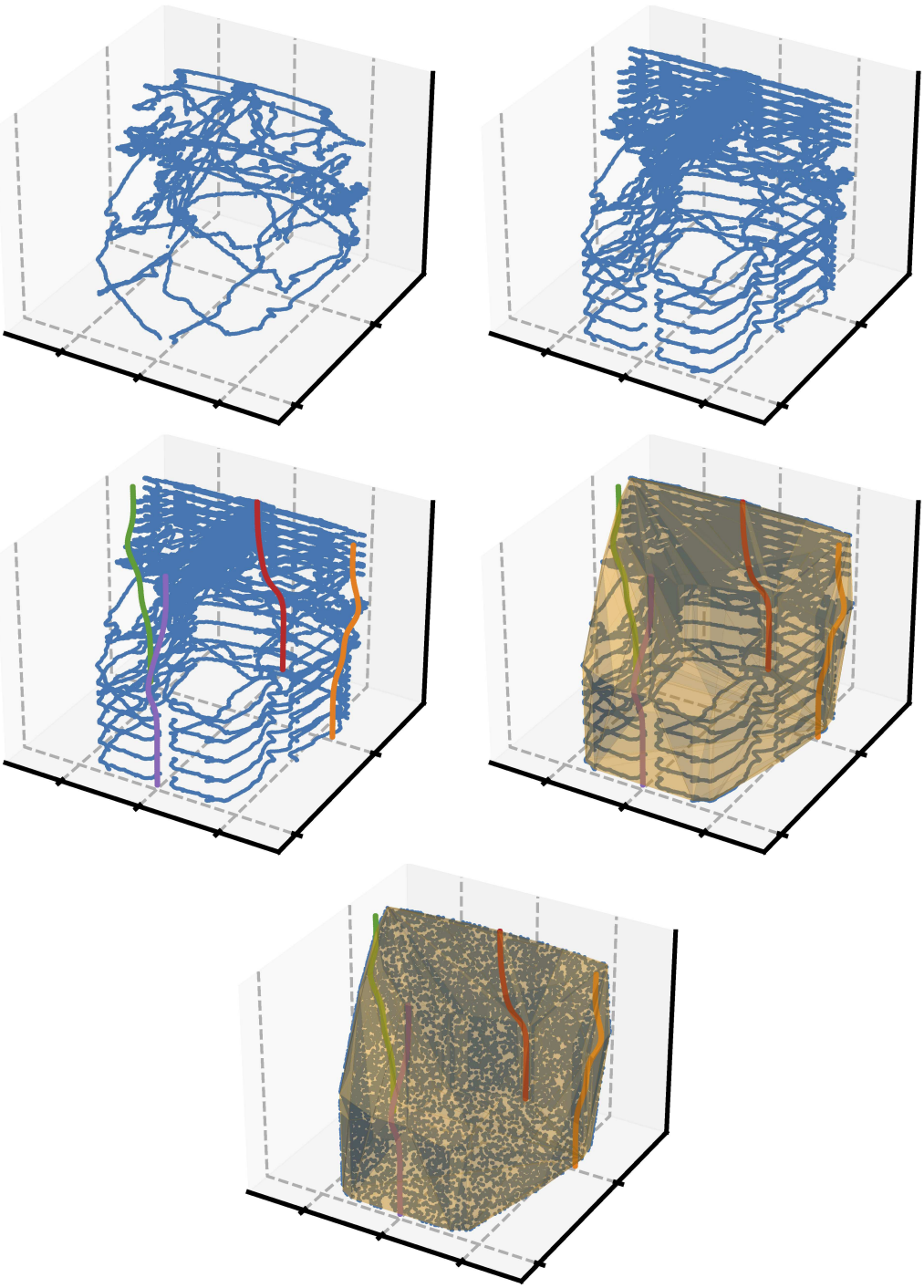


Figure 6.3: Illustration of the g-g-v diagram identification process. The figure is organized as a 3×2 table. Top row: raw telemetry data and the filled-in dataset after low- and high-speed extrapolation. Middle row: identification of pure acceleration limits and fitted convex polytope, shown incrementally over the data points. Bottom row: final combined view including the convex polytope, identified boundaries, and a cloud of points sampled along the resulting constraint surface. Axis labels and tick marks are omitted for clarity, as the plots are intended to illustrate the evolution of the constraint shape rather than provide quantitative results. Together, these plots summarize the complete construction workflow from raw data to the final g-g-v representation used by the E-NMPC model.

To illustrate the overall procedure, Figure 6.3 presents the key steps in the construction of the g-g-v diagram. The figure is organized as a 3×2 table: the first row shows the raw telemetry data and the filled-in dataset after low- and high-speed extrapolation; the second row depicts the identification of pure acceleration limits and the fitted convex polytope, both plotted incrementally over the data points; the final row shows the resulting g-g-v surface obtained by sampling the identified limits. Together, these visualizations summarize the complete process from raw data to the final g-g-v representation used in the E-NMPC model.

6.3.2 Extraction of Lateral Velocity Characteristics

The lateral velocity characteristics of the vehicle are critical for accurately reproducing lateral dynamics within the kineto-dynamical model. The corresponding lateral velocity dynamics follow the formulation introduced in Paragraph 3.2.2, repeated here for clarity:

$$\begin{aligned} \mathcal{F}_{v_y}(a_y, v_x, a_x) = \quad & f_1(v_x)\,(1 + b_1 a_x + b_2 a_x^2)\,a_y \\ & + f_3(v_x)\,(1 + b_3 a_x + b_4 a_x^2)\,a_y^3 \\ & + f_5(v_x)\,(1 + b_5 a_x + b_6 a_x^2)\,a_y^5. \end{aligned} \tag{6.5}$$

The parameters of the *Lateral Velocity Diagram* (LVD) model are identified directly from telemetry data containing lateral acceleration (a_y), longitudinal velocity (v_x), longitudinal acceleration (a_x), and lateral velocity (v_y). The dataset is divided into separate training and validation laps, and augmented by exploiting the symmetry with respect to the lateral acceleration. An adaptive fitting procedure determines the optimal discretization of v_x, evaluating candidate levels through nonlinear least-squares optimization and selecting the one that minimizes the validation error. The final model is then validated against the independent dataset to assess its predictive accuracy and generalization.

Figure 6.4 shows the validation results for the Volkswagen eCrafter, confirming that the LVD model effectively captures the vehicle's lateral velocity. Local discrepancies are visible in certain portions of the dataset, such as shifts observed during straight-line segments. These inconsistencies originate from imperfections in the measured data rather than from the model itself. A deeper analysis of such effects, and of how physics-informed models can reveal underlying measurement errors, is presented later in Section 6.6.

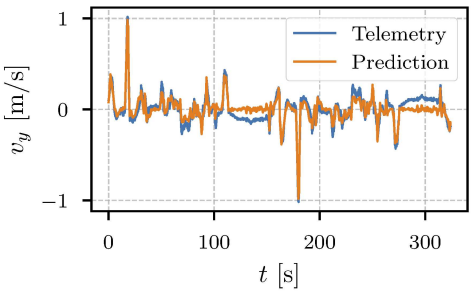


Figure 6.4: Validation results of the lateral velocity characteristics using the LVD model.

6.3.3 Identification of First-Order Dynamics

The dynamic response of key vehicle states—longitudinal acceleration a_x, lateral velocity v_y, and yaw rate Ω—is modelled in the kineto-dynamical formulation using first-order dynamics of the form:

$$\tau \dot{x}(t) + x(t) = x_{\text{ctrl}}(t), \tag{6.6}$$

where τ denotes the characteristic time constant of the response, $x(t)$ is the measured state, and $x_{\text{ctrl}}(t)$ represents the corresponding control action.

The identification of the time constant τ is carried out independently for each state by analysing the frequency content of the telemetry data. The power spectral density is computed using the Welch method to identify the dominant frequency components. The cutoff frequency, corresponding to a $-3\,\mathrm{dB}$ power drop from the peak, defines the effective bandwidth of the system. The time constant is then obtained as

$$\tau = \frac{1}{2\pi f_{\mathrm{cutoff}}}, \tag{6.7}$$

where f_{cutoff} is the identified cutoff frequency.

Because the actual control input signals are not available in the telemetry dataset, direct estimation of τ is not possible through classical system identification. Instead, the validation is performed through simulation of the identified first-order dynamics. The derivative $\dot{x}$ is obtained numerically, and the corresponding input signal is reconstructed using:

$$u_{\mathrm{est}} = \tau \dot{x} + x. \tag{6.8}$$

The estimated control input is clipped within the observed operating range, and the resulting simulated response is compared to the measured telemetry. This procedure provides a practical and consistent way to validate the identified time constants despite the absence of direct actuation data.

This analysis ensures that the simplified first-order representations of a_x, v_y, and Ω capture the relevant system dynamics while remaining lightweight enough for real-time use within the kineto-dynamical E-NMPC framework.

Figure 6.5 shows the results obtained from the identified time constants.

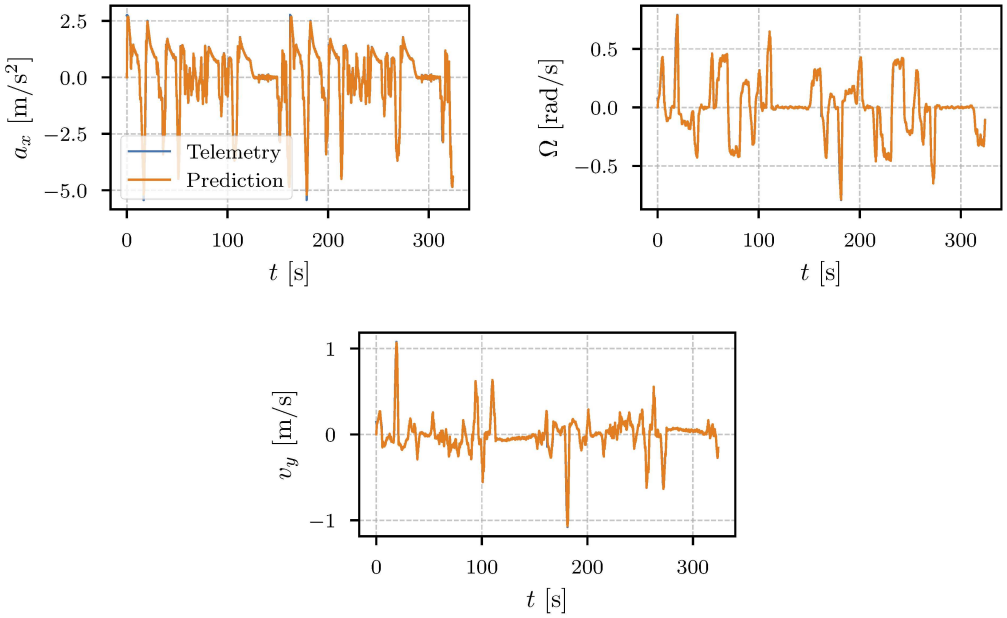


Figure 6.5: Validation results for the time constants' identification.

6.4 Low-Level Controller Identification

6.4.1 Training the PhS-NN Lateral Controller

The PhS-NN serves as the low-level lateral controller within the ARD framework. Its training process follows a streamlined and repeatable pipeline designed to ensure adaptability across different vehicles while maintaining minimal setup effort.

The network is instantiated from a set of configurable parameters defined in an external configuration file, allowing its structure—for instance, the number of channels used for processing lateral acceleration—to be easily adapted without modifying the core training code. The training relies on two laps of telemetry data, one used for learning and the other for validation. These contain the relevant vehicle signals, including longitudinal and lateral accelerations, speed, transient curvature, and commanded steering angle. To avoid directional bias, the dataset is symmetrically augmented so that left- and right-hand corners contribute equally to the learning process.

The PhS-NN is then trained using NNodely [179], a neural network modelling, training, and deployment package developed by our group. NNodely provides a dedicated framework for training physics-informed networks and includes direct support for ONNX export. After training, the model is validated on the second lap using standard error metrics and visual inspection of the predicted versus measured steering responses.

Once validated, the trained PhS-NN is exported to ONNX format for seamless deployment in the C++ implementation of ARD. This workflow allows the same training pipeline to be applied to any vehicle with minimal adjustments, ensuring a consistent process from data collection to real-time execution.

Figure 6.6 illustrates representative validation results for the PhS-NN controller.

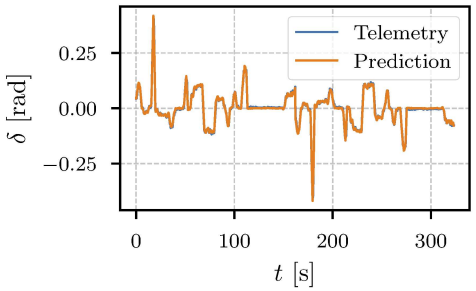


Figure 6.6: Validation results for the PhS-NN lateral controller, showing predicted and actual steering responses.

On the choice of steering angle reference. The steering angle used for training can be defined either at the wheels or at the steering wheel, depending on the requirements of the target application. If the vehicle accepts the wheel angle directly as control input, it is preferable to use this signal to avoid potential delays and dynamics introduced by the steering system. Conversely, if the vehicle only accepts the steering wheel angle as input, it is advisable to train the PhS-NN using that signal. In such cases, the network implicitly learns the steering system dynamics, including any delays, and accounts for them in the resulting control action.

6.4.2 Considerations for Longitudinal Control Identification

In earlier work [152], Piccinini primarily employed two vehicle models: a sedan and a racecar. Both models accepted as control inputs the steering wheel angle and a pedal position command (throttle or brake). In these settings, an additional longitudinal controller was required to convert the velocity profile planned by the E-NMPC into a desired pedal position, complementing the lateral controller described previously.

In this thesis, while the same racecar simulation model is occasionally used, the focus is strongly oriented towards real-world deployment using the UniBW test vehicles. These vehicles accept as inputs the desired longitudinal acceleration or velocity. As a result, there is no need to design or identify a dedicated longitudinal controller for real-world experiments, since the UniBW vehicles already incorporate a proprietary internal controller that translates desired acceleration/velocity into actuator commands.

It is important to emphasize, however, that devising a longitudinal controller compatible with the presented identification approach is not inherently difficult. Two straightforward alternatives can be envisioned:

1. **Physics-Informed Neural Network Controller:** A longitudinal controller could be developed following the same principles as the PhS-NN lateral controller. This would involve training a neural network to predict the inverse longitudinal dynamics from telemetry data, followed by the identification of a feedback *Proportional-Integral-Derivative* (PID) term as described later in this chapter.
2. **Gain-Scheduling PID Controller:** The gain-scheduling PID controller used by Piccinini could be re-identified using the same closed-loop identification approach detailed later in this chapter. This method is already implemented within the framework, albeit for slightly different purposes, and would readily extend to longitudinal control.

In simulation, we learn a PID controller to correct the desired longitudinal acceleration to better follow the desired longitudinal velocity. This controller is not used in real-world experiments as the vehicles already provide such a controller.

6.5 Closed-Loop Identification

Up to this point, all identification procedures have been performed in an open-loop fashion. This means that no feedback policy has been learned, and the identified models and controllers have been derived solely by replicating the behaviour observed in the recorded telemetry data. With only two laps available, the best achievable outcome is to reproduce the data as accurately as possible while avoiding overfitting. However, it is not possible to interact with the data: the identified controllers cannot be applied to the vehicle to observe how the resulting trajectories deviate from the planned ones.

This limitation is significant, as proper closed-loop analysis requires the ability to execute a control policy on the system, measure the resulting deviations from the planned trajectories, and iteratively refine the controller. Performing additional tests to collect such closed-loop error data would require more laps driven autonomously in open loop at the handling limits, which is not only costly and time-consuming, but also poses safety concerns.

A more efficient approach is to identify a sufficiently accurate vehicle model and use it to perform closed-loop analysis in simulation. By doing so, we can study how the planner and the open-loop controllers behave in a closed-loop setting, quantify the residual errors, and identify corrective feedback controllers without the need for further real-world testing. In our framework, these corrective controllers take the form of PIDs (optionally gain-scheduling if needed).

6.5.1 Training the Neural Vehicle Model

To enable closed-loop analysis, a NVM of the vehicle is identified from telemetry data collected over two laps of high-speed driving. The detailed structure of the model was presented in Chapter 5 and is not repeated here. The present section focuses on the training methodology and its integration within the overall identification pipeline.

The available telemetry includes position (x, y), yaw angle (ψ), longitudinal and lateral velocities (v_x, v_y), yaw rate (Ω), longitudinal acceleration (a_x), both the commanded and measured steering angles (δ_{ctrl}, δ), and the wheel rotational speeds ($\omega_{\text{fl}}, \omega_{\text{fr}}, \omega_{\text{rl}}, \omega_{\text{rr}}$). The model accepts the commanded longitudinal acceleration and front-wheel steering angle as inputs, while the predicted outputs include the vehicle states and the rotational speeds of the four wheels ($\omega_{\text{fl}}, \omega_{\text{fr}}, \omega_{\text{rl}}, \omega_{\text{rr}}$), among other things. The wheel rates are used in the cost function exclusively during training to improve the physical consistency of the learned dynamics, but are not required during inference. As such, they act as an auxiliary supervision signal that can easily be obtained from standard on-board sensors.

The telemetry is segmented into overlapping windows of one second to improve learning. Formally, given a discrete signal $x = [x_0, x_1, \ldots, x_n]$, the windowing process produces a matrix

$$X = \begin{bmatrix} x_0 & x_1 & \cdots & x_k \\ x_1 & x_2 & \cdots & x_{k+1} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n-k} & x_{n-k+1} & \cdots & x_n \end{bmatrix},$$

where each row corresponds to one temporal window of length $k+1$.

For each window, the model is initialized at the measured vehicle state and integrated forward in time using a differentiable numerical scheme, typically a first-order Euler method. The loss function penalizes deviations in the main dynamic states $[x, y, \psi, v_x, v_y, \Omega, \delta_f]$, as well as auxiliary quantities such as wheel speeds when available, which substantially enhance the prediction of longitudinal slips.

One lap is used for training and another for validation. To ensure balanced coverage of left- and right-hand manoeuvres, data are mirrored about the vehicle's symmetry axis, exploiting the inherent symmetry of most vehicles. Model validation includes both short- and long-horizon simulations: in the latter, the model predicts the complete lap starting from the initial measured state. While drift is expected over long horizons, the test provides a clear measure of model stability and its ability to reproduce global vehicle behaviour.

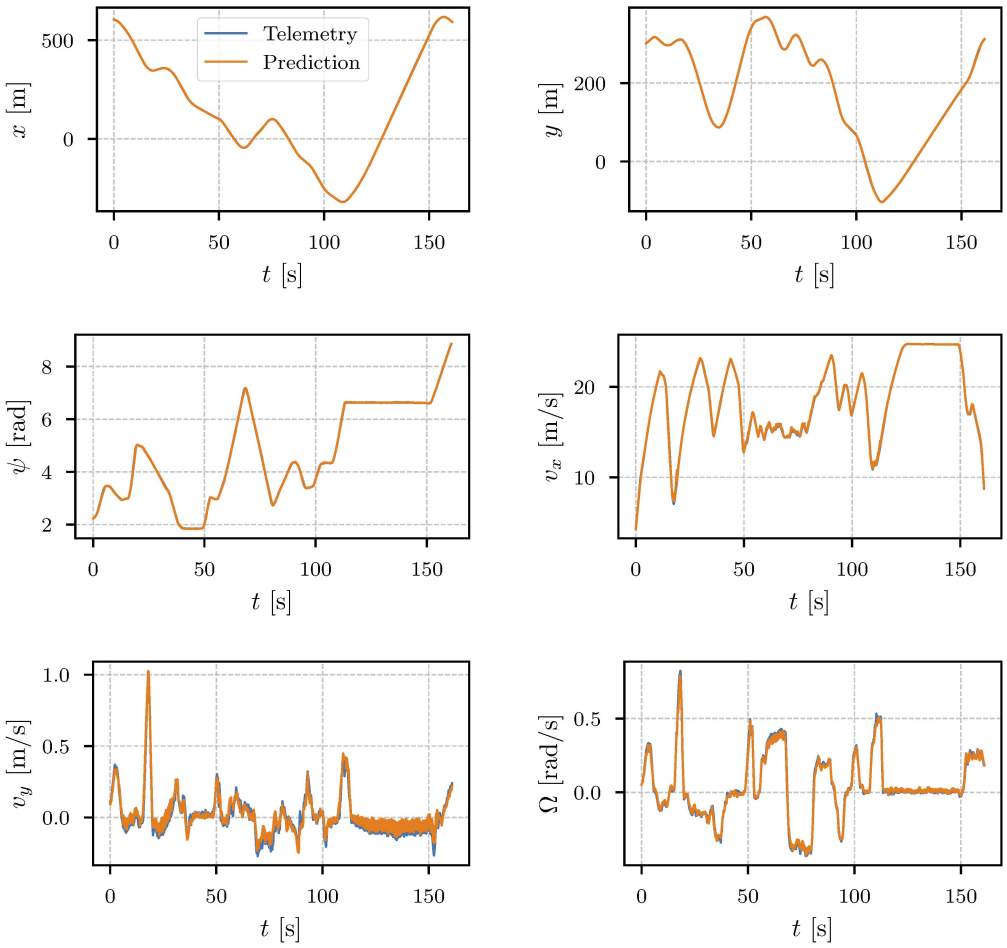


Figure 6.7: Part of the validation results for the NVM showing short-horizon predictions over a complete lap. Each prediction window spans 1 s, after which the model state is reset to the ground truth before initiating the next prediction segment.

After validation, the trained NVM is exported in ONNX format and deployed within the ARD closed-loop simulation framework. This ensures that the same model used during training can operate in real time on the hardware platform employed for both simulation and on-vehicle testing, guaranteeing consistency and reproducibility across all evaluation stages.

Figures 6.7 and 6.9 show part of the validation results for the NVM, including both short-horizon and long-horizon predictions. Moreover, Figure 6.8 shows a zoom over the short-horizon validation resluts to highlight the local errors. The short-horizon case corresponds to 1 s prediction windows, where the model state is periodically reset to the ground truth, whereas the long-horizon case spans the entire lap without intermediate resets. Once again, the discrepancies we can see in the lateral velocity are mainly due to errors in the measured signal, not in the model, which will be discussed in Section 6.6.

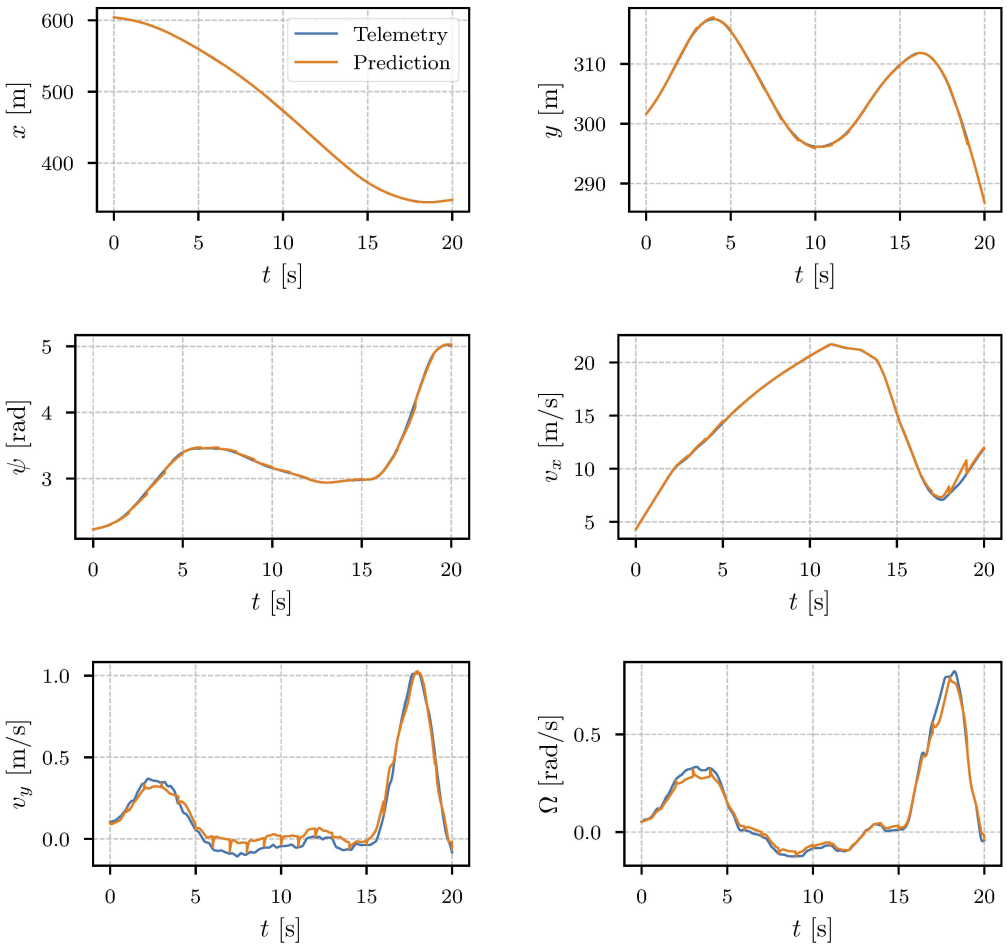


Figure 6.8: Zoom over part of the validation results for the NVM showing short-horizon predictions over a complete lap. Each prediction window spans 1 s, after which the model state is reset to the ground truth before initiating the next prediction segment.

6.5.2 Closed-Loop Identification and Corrective Control

While the open-loop PhS-NN lateral controller provides accurate feedforward steering actions, its lack of feedback correction limits its ability to compensate for unmodelled dynamics, disturbances, and residual identification errors. To analyse and improve closed-loop behaviour, the controller must therefore be embedded within a differentiable simulation framework.

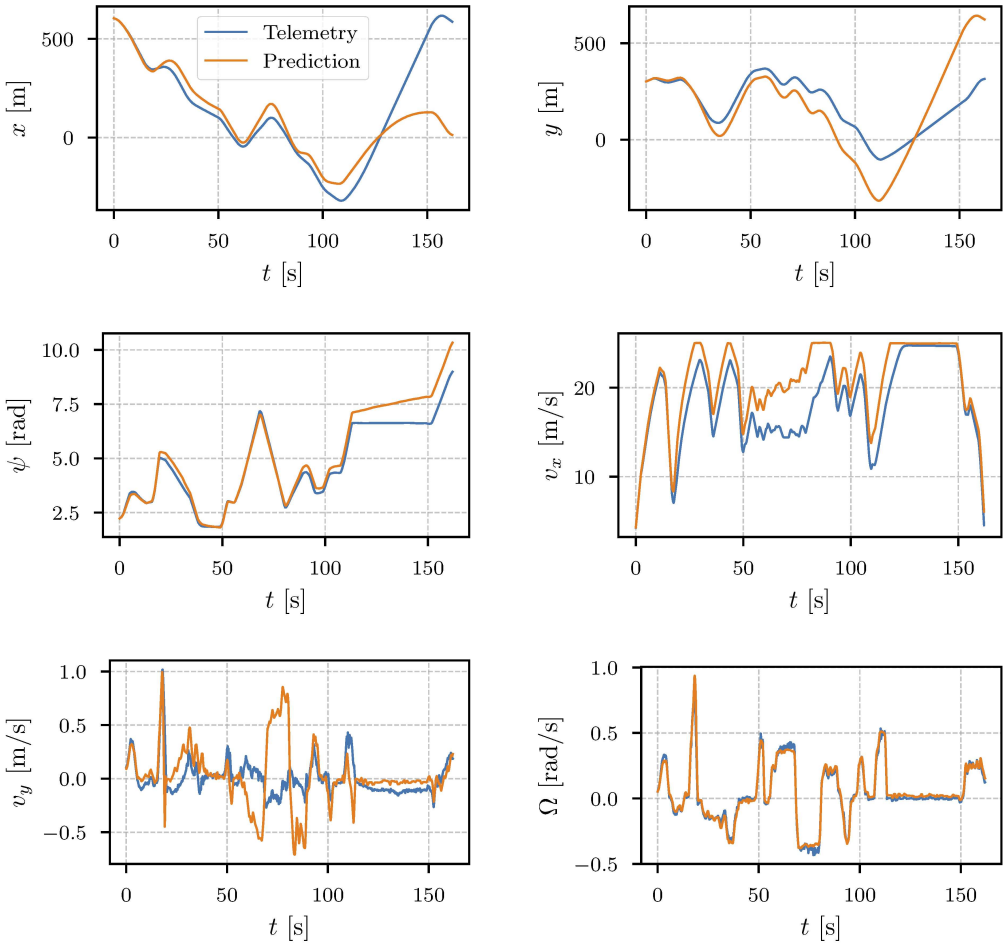


Figure 6.9: Part of the validation results for the NVM showing long-horizon prediction over an entire lap. We can see that the prediction is accurate for the first $\approx 20\,s$ of the horizon, and then degrades as expected.

Running the full E-NMPC in closed loop would represent the ideal setup, since this is the intended operational framework. However, such an approach is computationally prohibitive and non-differentiable[2], preventing end-to-end optimization. Instead, as shown in Figure 6.1, we use a precomputed *Minimum Lap Time* (MLT) trajectory obtained by solving the same E-NMPC problem over the entire circuit with cyclical boundary conditions. Fixing the reference trajectory removes the need to re-run the planner during identification while retaining realistic optimal driving conditions. The open-loop steering commands corresponding to this trajectory are precomputed, allowing differentiable closed-loop tracking simulations using the NVM.

Residual tracking errors are compensated by introducing corrective feedback controllers in the form of PIDs for both lateral and longitudinal dynamics. These controllers are implemented as differentiable PyTorch modules with anti-windup and filtered derivative terms, jointly optimized together with the NVM in a single computational graph:

$$\text{PID} \rightarrow \text{Neural Vehicle Model} \rightarrow \text{Integration Step} \rightarrow \text{PID} \rightarrow \dots$$

This setup preserves full differentiability across the entire simulation loop, enabling efficient gradient-based optimization. Typically, convergence is achieved within a few minutes of training.

[2]Techniques to differentiate an MPC exist, but at the moment of writing they are not feasible for this application.

Including a longitudinal PID is essential, as differences between the kineto-dynamical model and the NVM can otherwise lead to time desynchronization between the simulated and reference trajectories. Both PIDs can also be gain-scheduling, with each of the five gains—k_p, k_i, k_d, k_{aw}, and n—expressed as small MLPs scheduled by the longitudinal speed v_x. The resulting gain profiles can then be approximated by splines for real-time deployment.

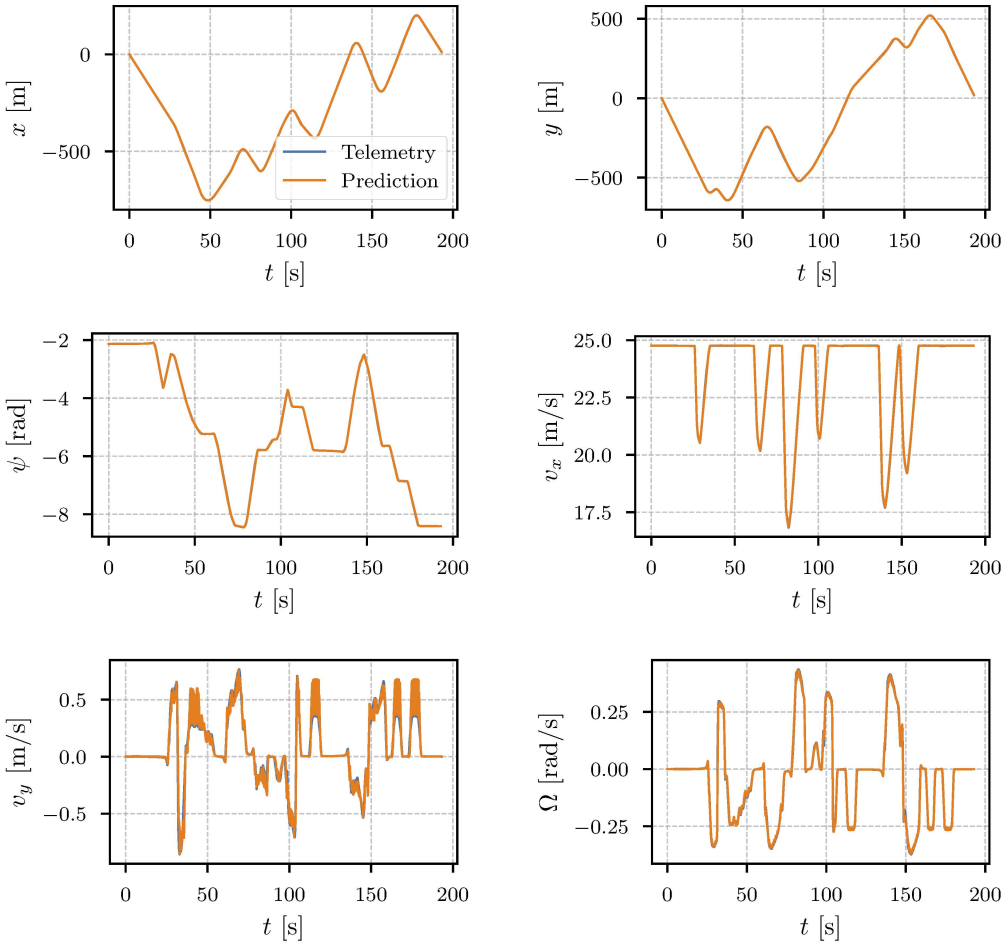


Figure 6.10: Part of the validation results for the NVM with corrective PID controllers, showing short-horizon prediction over an entire lap. Each prediction window spans 1 s, after which the model state is reset to the ground truth before initiating the next prediction segment.

Figures 6.10 and 6.11 show part of the validation results for the NVM, including both short-horizon and long-horizon predictions. The short-horizon case corresponds to 1 s prediction windows, where the model state is periodically reset to the ground truth, whereas the long-horizon case spans the entire lap without intermediate resets. In particular, we can see that with the addition of the corrective controllers, the NVM can now accurately predict the trajectory over basically the entire telemetry even in the long-horizon case.

6.6 On Error Detection and Interpretability of Physics-Informed Models

During the development of the NVM, an unexpected yet valuable property of the physics-informed approach emerged: its ability to reveal inconsistencies in the dataset that would likely remain unnoticed when using black-box data-driven models.

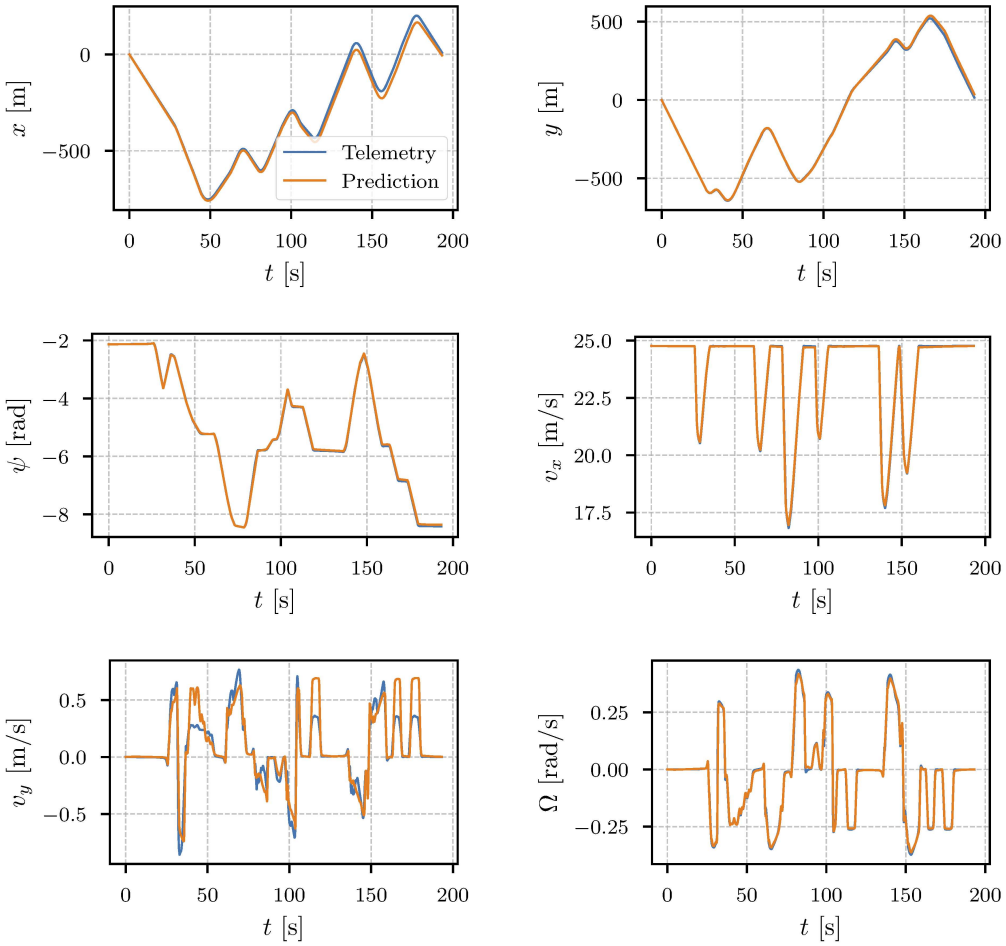


Figure 6.11: Part of the validation results for the NVM with corrective PID controllers, showing long-horizon prediction over an entire lap. Contrary to the results without corrective feedback terms, we can now see that the model successfully predicts the vehicle motion for almost the entire telemetry duration.

The dataset was collected with the UniBW Volkswagen eCrafter, a large electric van. At first inspection, the telemetry appeared consistent with expectations for the vehicle and driving conditions. The data were reported to be expressed with respect to the centre of the rear axle, and a corresponding shift correction was therefore applied. Once corrected, however, unusually high sideslip angles and lateral velocities were observed—values more typical of high-performance vehicles. Although the driving style during data collection was aggressive, such magnitudes were surprising but not implausible given the eCrafter's high CoG and the high curvature of the track.

During identification, the estimated lateral velocity systematically underestimated the measured signal. At that stage, model parameters were still unconstrained, and a generic neural compensation term was introduced into the lateral velocity dynamics as an experiment. This partially reduced the discrepancy but failed to eliminate it. When the generic term was removed and the parameters were constrained to physically meaningful ranges, the mismatch became more apparent.

This behaviour suggested that the reference point of the velocity measurement might have been misinterpreted. Repeating the analysis under the assumption that the signals were already expressed at the CoG partly resolved the issue, as the correction $v_y^{\text{corr}} = v_y - \Omega\, x_{\text{shift}}$ increases the lateral velocity when the reference point lies behind the CoG. Assuming an incorrect negative longitudinal shift would therefore inflate v_y. Subsequent discussions with the vehicle engineer confirmed that, following a vehicle

update, the shift correction had indeed not been applied—the measurements were still referred to the sensor location rather than the rear axle centre.

Further analysis indicated that the longitudinal offset required to match one of the v_y peaks in the telemetry to the one predicted by the NVM was approximately 62 cm. The engineer later verified that the sensor was located about 56 cm from the CoG, with an additional lateral offset.

A final observation concerned a drift in the measured v_y signal, especially on straights, where it remained up to 10% of its cornering peak. The model consistently refused to reproduce this behaviour, since sustaining a high lateral velocity during straight-line motion is physically implausible. A similar drift was later found in data from another UniBW test vehicle, a Volkswagen eGolf. Further investigation, in agreement with the vehicle engineer, suggested that the issue likely originated from the employed Kalman filter or its input sensors, though the precise cause remains under examination.

Interestingly, the LVD identification also failed to replicate these artefacts, confirming that the physics-informed structure naturally rejects unphysical trends. This outcome highlights a crucial benefit of physics-informed modelling: rather than overfitting to spurious data, such models expose inconsistencies that can be systematically investigated and resolved. Collaboration with the vehicle engineer was essential in verifying these findings.

This episode demonstrates that physics-informed models not only enhance generalization and robustness, but also improve interpretability and trustworthiness. In data-limited contexts such as the two-lap learning framework, this property is particularly valuable: by refusing to model unphysical behaviour, physics-informed models act as a natural safeguard against erroneous conclusions that might otherwise pass unnoticed in black-box data-driven approaches.

Part II—Simulation and Real-World Testing

Chapter 7

Test Setup

Abstract

This chapter presents the experimental setup adopted to evaluate the *Artificial Race Driver* (ARD) in simulation and real-world conditions. Two circuits are considered: the *Universität der Bundeswehr München* (UniBW) test track, where both simulation and real-world tests are performed, and the Circuit de Barcelona–Catalunya, a well-known and balanced racetrack used as a simulation benchmark. Two vehicles are evaluated: a Volkswagen eCrafter, tested both in simulation and on track, and a high-performance racecar analysed in simulation only. The chapter describes the main characteristics and challenges of the selected tracks and vehicles, and provides an overview of the experimental setup. Details on software architecture are omitted, as they were already discussed in Chapter 2.

Contents

7.1 Introduction 91
7.2 Chosen Tracks 92
7.2.1 UniBW Test Track 92
7.2.2 Circuit de Barcelona—Catalunya 92
7.2.3 UniBW Short Test Track 92
7.3 Test Vehicles 93
7.3.1 Volkswagen eCrafter 94
7.3.2 Racecar 94
7.4 Test Setup 94

7.1 Introduction

Part II presents a comprehensive evaluation of ARD in both simulation and real-world conditions. The aim is to demonstrate that ARD, when provided with models and controllers identified from only two laps of human-driven data, as described in Chapter 6, can achieve competitive performance in a variety of driving scenarios. This part consolidates the theoretical developments introduced in Part I into a set of structured experiments designed to assess the system's capabilities, limitations, and practical applicability.

The evaluation is carried out on two circuits with distinct characteristics: the UniBW test track, a short and technical circuit where both simulation and real-world tests are performed, and the Circuit de Barcelona—Catalunya, an internationally recognized racetrack with a balanced layout that serves as a benchmark for simulation studies. Two vehicles are considered: a Volkswagen eCrafter, tested in both simulation and on track, and a high-performance racecar, analysed in simulation only. The real-world tests were conducted on a slightly shortened and smoothed variant of the UniBW layout, introduced for safety reasons while preserving the circuit's overall character.

The use of both simulation and real-world experiments serves complementary purposes. Simulation enables extensive testing under controlled conditions, including the analysis of extreme or unsafe scenarios that cannot be explored on track. Real-world experiments, on the other hand, validate the overall framework in realistic conditions and provide a practical benchmark. In particular, the inclusion of at least one identical test case in both simulation and real-world settings allows for a direct comparison of sim-to-real discrepancies, quantifying the impact of modelling errors, environmental variability, and identification inaccuracies.

7.2 Chosen Tracks

Two circuits are used for the evaluation of ARD: the UniBW test track and the Circuit de Barcelona—Catalunya. These tracks were selected to provide complementary testing conditions, combining the practicality of real-world access with the relevance of a well-known international racing circuit. Their different layouts and physical characteristics allow for a diverse assessment of vehicle dynamics and trajectory planning performance. Qualitative layouts of both tracks are shown in Figure 7.1, providing quick visual reference for the subsequent discussions.

In the case of the UniBW circuit, two configurations are considered. The full layout is used for simulation studies, while a slightly shortened and smoothed version was adopted for the real-world experiments. The latter modification was introduced to ensure safe testing conditions given the presence of fixed obstacles and the limited accuracy of the track boundary alignment. A detailed description of the real-world layout and the corresponding adjustments is provided in Section 7.2.3.

7.2.1 UniBW Test Track

The UniBW test track is a short, technical circuit located at the Universität der Bundeswehr München. It features a compact layout with a mixture of tight corners and short straights, providing a challenging environment for vehicle control and trajectory planning. The track surface is representative of typical flat road conditions, with limited grip and local surface imperfections that can affect vehicle behaviour. In simulation, the full layout is employed, whereas the real-world tests are performed on a shortened variant introduced for safety reasons.

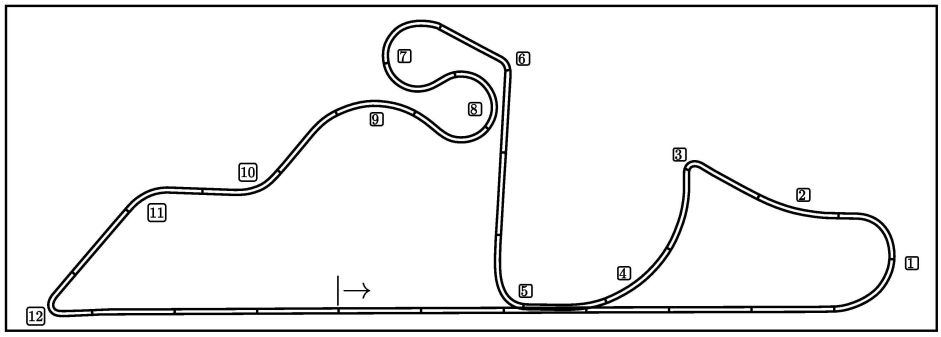
7.2.2 Circuit de Barcelona—Catalunya

The Circuit de Barcelona—Catalunya is an internationally recognized racetrack located in Montmeló, Spain. It is widely used for professional motorsport events, including Formula 1 and MotoGP, and is well known for its balanced combination of high-speed straights, medium-speed corners, and technical low-speed sections. The circuit length and layout make it an ideal benchmark for evaluating the performance of minimum-time planners and controllers in a realistic but purely simulated high-performance context. Importantly, the circuit can be approximated as two-dimensional for the purposes of trajectory planning, which aligns with the scope of this work. While 3D effects such as elevation changes exist, their influence is limited compared to more topographically complex tracks. For this reason, the circuit allows for a credible evaluation of ARD in simulation without overemphasizing contributions outside the focus of this thesis (namely, ARD's 3D extension).

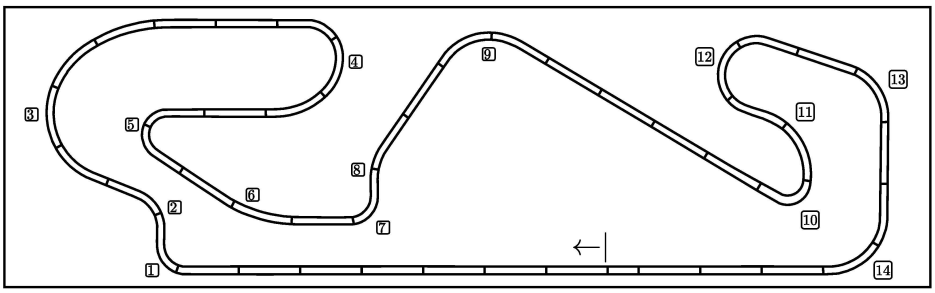
7.2.3 UniBW Short Test Track

For the real-world experiments, a shortened and slightly modified variant of the UniBW test track was adopted. The adaptation was motivated by safety considerations arising during on-track deployment. As detailed in Chapter 9, the alignment between the real and the measured track boundaries was affected by positional offsets, which required additional safety margins to prevent driving off track. While most corners remained unaffected, two sections were modified to ensure sufficient clearance from surrounding obstacles. Corner 3 was bypassed entirely due to the proximity of a structural pole that could not be safely avoided within the available margin. In contrast, Corner 12 was preserved but locally smoothed to reduce the risk of running close to the outer boundary.

(a) UniBW track



(b) Catalunya track



(c) UniBW Short track

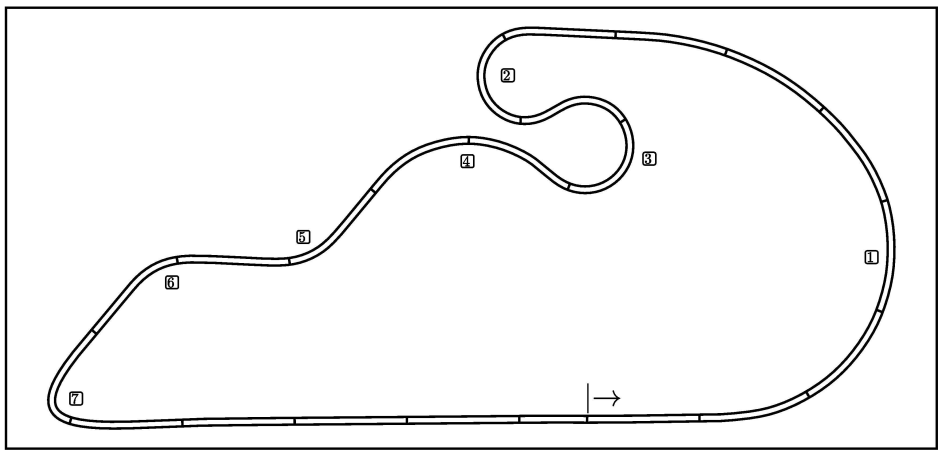


Figure 7.1: Layouts of the UniBW, Catalunya, and UniBW Short tracks with corner numbering. Along the track, perpendicular lines are placed at regular 100 m intervals. These layouts serve as quick reference for the following results discussions.

7.3 Test Vehicles

Two vehicles are considered for the evaluation of ARD: a Volkswagen eCrafter and a high-performance two-seat racecar. The eCrafter is used in both simulation and real-world testing, whereas the racecar is analysed in simulation only. These vehicles span a wide range of dynamic characteristics, from a heavy commercial van with a top speed of 90 km/h to a road-legal racecar capable of 340 km/h. Both platforms rely on the two-lap identification strategy described in Chapter 6, allowing ARD to be evaluated consistently under minimal data requirements despite the large differences in vehicle dynamics.

In simulation, both vehicles include physical actuator limits, but are free from real-

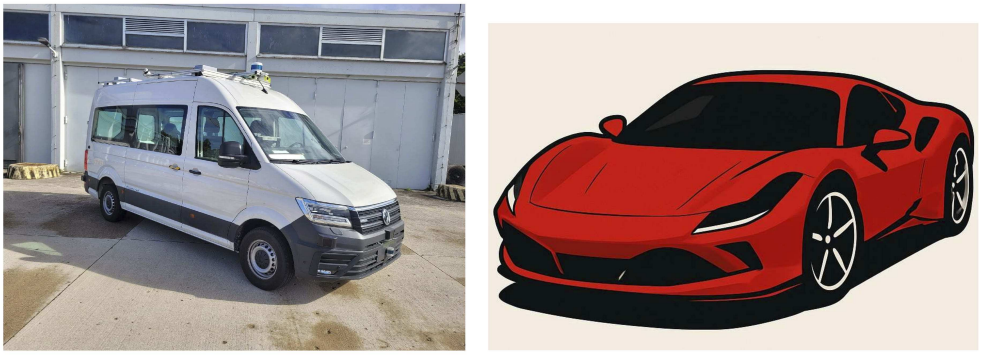


Figure 7.2: Vehicles considered in this thesis. Volkswagen eCrafter (left) used for both simulation and real-world testing, and a high-performance racecar (right) used in simulation. For confidentiality reasons we cannot show an actual picture of the racecar, the presented sketch (AI generated) is purely qualitative to make evident the type of considered vehicle.

world actuation constraints such as safety system interventions. In contrast, the real eCrafter imposes strict conditions: if the steering rate exceeds internal thresholds, or if the *Electronic Stability Programme* (ESP), *Anti-lock Braking System* (ABS), or rollover protection intervene beyond a certain unknown limit, the automation is immediately deactivated, and the lap is terminated. These safety constraints fundamentally differ from actuator saturation, as a single infeasible control action irreversibly ends the experiment. It is important to note that these security thresholds are unknown and therefore posed a significant challenge for the real-world deployment.

7.3.1 Volkswagen eCrafter

The Volkswagen eCrafter is a fully electric commercial van with a top speed of approximately 90 km/h, high mass, high centre of gravity, and moderate power and acceleration. Although its dynamic performance is limited, the actuation system is relatively responsive, allowing for effective low-speed manoeuvring. In real-world testing, however, any violation of steering-rate, ESP, or ABS thresholds leads to immediate deactivation of automation, ending the lap. This requirement forces ARD to generate trajectories that are not only time-efficient but also conservative enough to guarantee feasible execution under these constraints. The vehicle is shown in Figure 7.2.

7.3.2 Racecar

The second platform is a high-performance two-seat racecar, road legal yet capable of approximately 340 km/h. Its low mass, powerful engine, and advanced suspension and braking systems provide significantly higher performance than the eCrafter. This vehicle is analysed exclusively in simulation, where it serves as a reference for ARD under extreme dynamic conditions. No photograph of this vehicle is included for confidentiality reasons, but its characteristics alone highlight the contrast with the eCrafter and demonstrate ARD's ability to generalize across very different vehicle dynamics. A qualitative representation of this vehicle can be found in Figure 7.2.

7.4 Test Setup

All simulations and real-world tests are executed on the same computational platform to ensure consistent performance across experiments. ARD is run on a 16-inch Apple MacBook Pro equipped with an M2 Max processor and 32 GB of RAM, which provides sufficient resources to execute the high-level planner, the low-level controllers, the vehicle model, and the visualizations in real-time. All simulation experiments, including trajectory planning, vehicle control, and closed-loop evaluations, are performed on this laptop without the need for additional hardware.

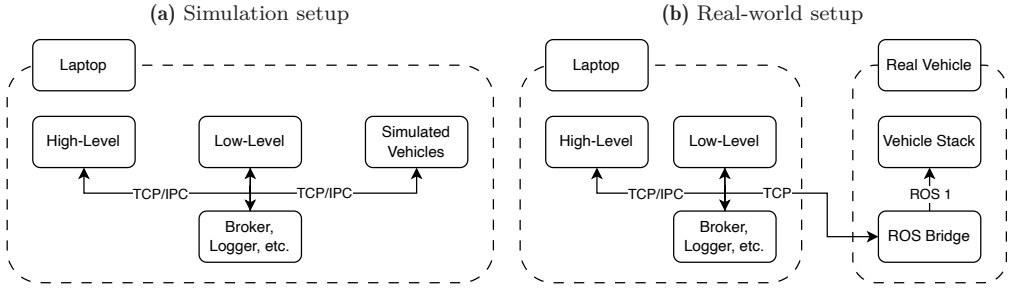


Figure 7.3: Schematic overview of the experimental setup for simulation and real-world tests. In simulation the entire stack runs locally on the laptop. During real-world tests, no vehicle is simulated, and communication with the real vehicle is handled via a custom ROS bridge.

For real-world tests, the high-level motion planner and low-level controllers are likewise executed on the same laptop, which is connected to the vehicle through a ROS-based interface. A ROS bridge runs on the vehicle, handling communication with on-board sensors, actuators, and interface modules. All perception and actuation signals, as well as safety-related messages, are processed directly on the vehicle, while the planning and control stack runs remotely on the laptop.

Communication between modules uses local TCP or inter-process communication during simulations, ensuring low-latency data exchange. For real-world deployment, the laptop connects to the vehicle via wired Ethernet using TCP. This arrangement provides both the throughput and robustness required for real-time operation while maintaining the flexibility to run the complete ARD stack on a portable computational platform.

Using the same hardware and software environment across all tests ensures that differences in performance between simulation and real-world experiments arise from the vehicle and environment rather than from computational constraints or architectural changes. A schematic overview of the setup is shown in Figure 7.3.

Chapter 8

Evaluation in Simulation

Abstract

This chapter presents a comprehensive evaluation of the *Artificial Race Driver* (ARD) in simulation, where controlled conditions make it possible to test extreme scenarios that would be unsafe or impractical in real-world experiments. Offline *Minimum Lap Time* (MLT) trajectories are first computed for each track and vehicle combination using the kineto-dynamical planner, providing theoretical lower bounds for lap times. ARD is then assessed against these baselines in terms of lap times and utilization of the vehicle's dynamic capabilities. In addition, its response to non-ideal operating conditions, such as dropped messages and sensor noise, is analysed to assess the system's behaviour under realistic disturbances. The chapter also shows how ARD's behaviour can be biased towards different cornering approaches, such as preferring early or late apex trajectories, by tuning specific planner parameters. These results demonstrate that ARD achieves near-optimal performance while adapting to different cornering styles and vehicle models.

Contents

8.1 Introduction 96
8.2 MLT vs MPC 97
8.2.1 Volkswagen eCrafter 98
8.2.2 Racecar 98
8.3 MPC vs CL-SYNC 102
8.3.1 Volkswagen eCrafter 103
8.3.2 Racecar 103
8.4 CL-SYNC vs CL-ASYNC 104
8.4.1 Volkswagen eCrafter 104
8.4.2 Racecar 104
8.5 Computation Times 105
8.6 Evaluation under Non-Ideal Conditions 114
8.7 Driving Style Biasing 116

8.1 Introduction

Simulation is a fundamental tool for evaluating ARD as it enables testing under controlled and repeatable conditions, without the constraints, risks, or costs associated with real-world experiments. The complete ARD stack runs in real-time on the same hardware used for experimental tests. This ensures that simulation results are directly comparable to real-world ones, while allowing tests at extreme operating conditions, such as high speeds and near-limit manoeuvres, that may be unsafe to reproduce on track.

Two distinct vehicles are considered: the Volkswagen eCrafter, a commercial electric van with a top speed of approximately 90 km/h, and a high-performance racecar capable of 340 km/h. These vehicles span a broad range of dynamic characteristics, from the low-power, high-mass eCrafter to the lightweight, high-power racecar. The same two-lap learning strategy is used to adapt ARD to each vehicle, enabling consistent comparisons across a wide dynamic envelope. It is important to stress that ARD never drives the vehicle during this learning process. The two-lap strategy consists of identifying high-level models, open loop controllers, and a *Neural Vehicle Model* (NVM) from real data, followed by small closed-loop corrections on steering and acceleration with the learned model,

using the MLT trajectory as reference. No online tuning of the controllers is performed. This makes the subsequent evaluations effectively zero-shot, and any transfer to new tracks a genuine test of generalization.

For each vehicle and track combination, four types of solutions are computed:

- **MLT**: offline trajectories obtained with the kineto-dynamical planner, serving as theoretical lower bounds.
- **MPC**: the same optimization problem solved online with a receding horizon, validating the planner formulation.
- **CL-SYNC**: closed-loop execution with a driven vehicle model and synchronous evaluation, demonstrating that ARD can control a vehicle near the limits without prior online training.
- **CL-ASYNC**: closed-loop execution in asynchronous mode, introducing realistic delays as in the real-world deployment, showing readiness for experimental validation.

This progression provides a cohesive evaluation path. Comparing MLT to MPC validates the planner's ability to approach the theoretical optimum. Comparing MPC to CL-SYNC demonstrates that ARD's hierarchical structure, despite relying only on minimal corrective tuning, can stably control a vehicle model near the limits. Finally, comparing CL-SYNC to CL-ASYNC highlights the effects of real-time execution and asynchronous delays, bridging the gap to real-world deployment. In addition, the results for the eCrafter are reported both on a track used for training and on a previously unseen validation track. Instead, the results obtained with the racecar are all obtained on unseen tracks, as the track used for training cannot be disclosed. These choices demonstrate ARD's ability to transfer knowledge across circuits.

The remainder of this chapter is organized accordingly. Section 8.2 presents the comparison between MLT and MPC. Section 8.3 analyses the differences between MPC and CL-SYNC. Section 8.4 reports the effect of asynchronous execution. Section 8.5 discusses the computation times achieved by the proposed formulation and implementation. Section 8.6 evaluates the system behaviour under non-ideal operating conditions, including dropped messages and sensor noise. Finally, Section 8.7 demonstrates how ARD's behaviour can be biased towards different cornering approaches, such as favouring early or late apex trajectories.

All simulations share the same initial conditions: the vehicle starts 10 m before the finish line at a speed of 1 m/s. Results are reported from the moment the vehicle first crosses the finish line to the completion of the lap, ensuring consistent comparisons across vehicles, tracks, and solution types. Each result is visualized in a single figure with a fixed 4×2 layout. The first row shows the track with the trajectories, followed by longitudinal velocity v_x, sideslip β, longitudinal and lateral accelerations a_x and a_y, yaw rate Ω, and the g-g diagram. Lap times are reported in Table 8.1.

Table 8.1: Lap times of MLT, MPC, CL-SYNC, and CL-ASYNC for each vehicle and track.

Vehicle	Track	t_{MLT} [s]	t_{MPC} [s]	$t_{\text{CL-SYNC}}$ [s]	$t_{\text{CL-ASYNC}}$ [s]
eCrafter	UniBW	146.038	146.139	149.566	149.988
	Catalunya	197.382	197.458	202.640	203.561
Racecar	UniBW	86.682	86.748	87.875	88.917
	Catalunya	111.046	111.182	112.803	113.894

8.2 MLT vs MPC

This section compares the offline MLT solutions with the online MPC trajectories. The objective is to validate that the receding-horizon formulation achieves performance close

to the theoretical optimum, both on the track used for training and on a previously unseen circuit used only for validation. Lap times are summarized in Table 8.1.

8.2.1 Volkswagen eCrafter

For the eCrafter, the MLT is solved once over the full lap with strict initial conditions and free final conditions. The MPC instead uses a 200 m horizon with 200 mesh points. To prioritize continuity between successive solutions, a cost term penalizes state deviations from the first metres of the previous trajectory. In addition, strict initial conditions are enforced at each step on all states, while soft final conditions on n and ξ gently bias the solution towards the centreline, providing a limited preview of the upcoming track. The long horizon and fine discretization increase computation times but yield a solution closer to the MLT optimum. For real-time applications, both horizon length and mesh density will be reduced.

UniBW track (training). Figure 8.1 shows the comparison on the *Universität der Bundeswehr München* (UniBW) circuit, where ARD was trained. The MPC trajectory closely follows the MLT solution, with only minor deviations in corner approach and exit. These differences stem from the finite-horizon formulation: while the MLT problem has full knowledge of the track and can optimize each entry and exit with respect to the remaining track, the MPC can only plan within its limited horizon. This occasionally leads to suboptimal corner placement and a small loss in lap time. As a result, the MPC is about 0.101 s slower than the offline benchmark. Nonetheless, the state profiles confirm the overall similarity, and the g-g diagram shows that the MPC reaches combined acceleration levels comparable to the theoretical optimum.

Catalunya track (validation). On the unseen Catalunya circuit (Figure 8.2), the outcome is similar. The lap time difference is slightly smaller, around 0.076 s, and the state profiles together with the g-g diagram confirm that MPC produces a trajectory very close to the MLT. In this case the discrepancy is even less pronounced than on the UniBW track. This can be explained by the characteristics of the Catalunya circuit. Although it combines sharp corners with long straights, it is comparatively less demanding for a slow vehicle such as the eCrafter. With a top speed limited to 90 km/h and a narrow g-g envelope, the van can maintain relatively high speeds throughout most of the lap. This is evident from the velocity profile. Consequently, the reduced MPC horizon does not cause significant differences in cornering strategy compared to the MLT. Interestingly, we can clearly see the effects of the reduced horizon. The Catalunya track has several corners right after long straights. If we focus on corners 1, 4, and 10, we can clearly see that the MPC planned a different trajectory, only to realize it needed to change it due to the upcoming corner. This is evident from the lateral acceleration profile right before the entry to those corners.

8.2.2 Racecar

For the racecar, the MLT is again solved once over the full lap with strict initial conditions and free final conditions. The MPC reduces the horizon to 300 m, with 300 mesh points. As with the eCrafter, a cost term penalizes state deviations from the first metres of the previous solution to maintain continuity. Strict initial conditions are enforced at each step on all states, and soft final conditions on n and ξ provide a limited preview of the track ahead. The fine and long mesh increases computation times, but ensures we get a solution more similar to the MLT one. For real-time applications the mesh will be reduced both in length and number of points.

UniBW track (validation). On the unseen UniBW circuit (Figure 8.3), the MPC produces competitive lap times. Interestingly, the gap with the MLT decreases compared

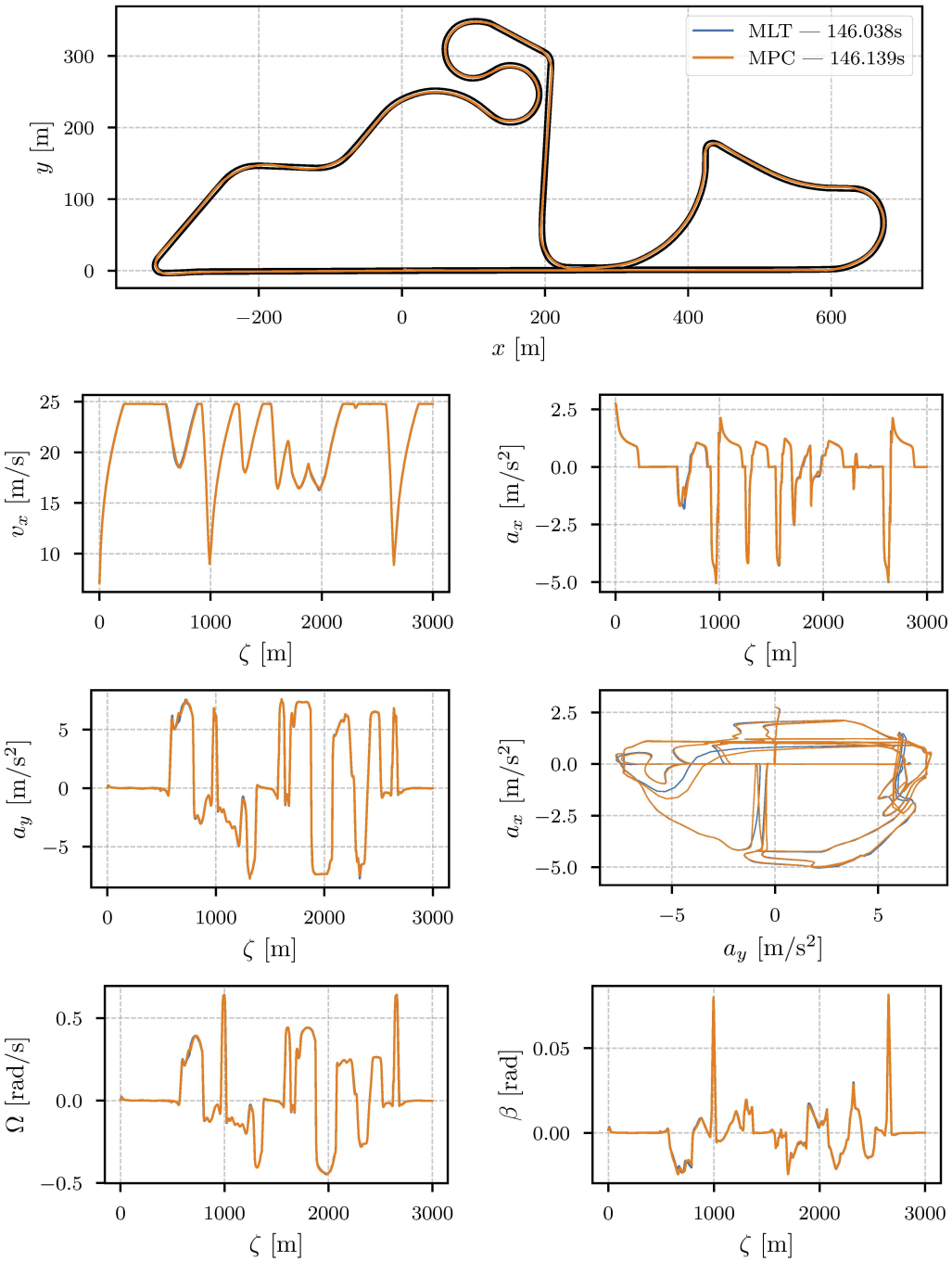


Figure 8.1: Telemetry for the eCrafter on UniBW: MLT vs MPC. At the top, the racetrack with the two trajectories, and the legend with the solution names and lap times. Below it, the longitudinal velocity v_x, the longitudinal acceleration a_x, the lateral acceleration a_y, the g-g diagram, the yaw rate Ω, and finally the side slip angle β. All plots, except for the trajectories, are plotted against the curvilinear abscissa ζ.

to Catalunya. This is the opposite trend observed with the eCrafter. For such a high-performance vehicle, running at comparatively lower velocities places operation in a more manoeuvrable region of the g-g envelope, with larger feasible combined and pure longitudinal accelerations. As a result, differences in cornering approach are less critical, and the lap time difference reduces to about 0.066 s.

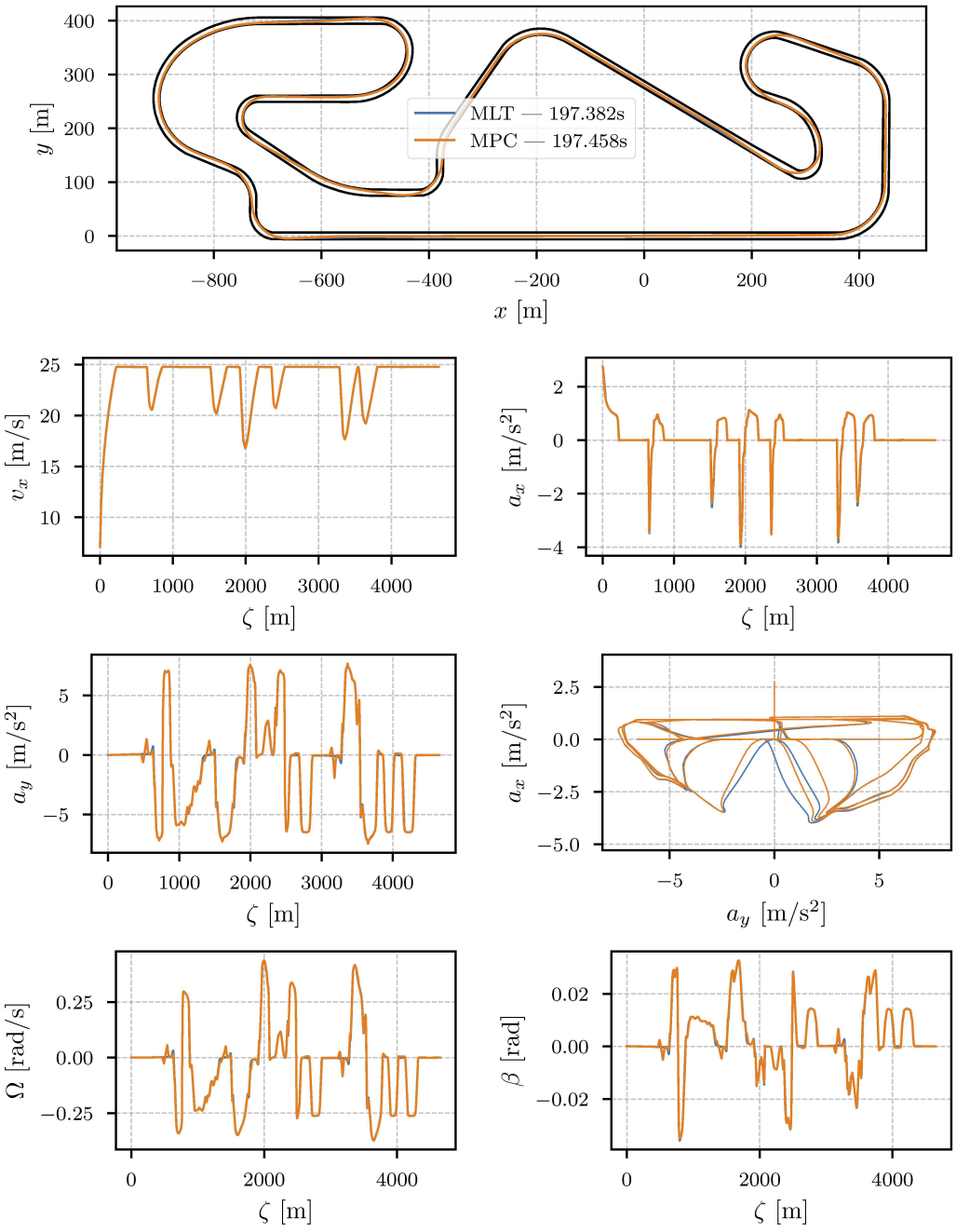


Figure 8.2: Telemetry for the eCrafter on Catalunya: MLT vs MPC. At the top, the racetrack with the two trajectories, and the legend with the solution names and lap times. Below it, the longitudinal velocity v_x, the longitudinal acceleration a_x, the lateral acceleration a_y, the g-g diagram, the yaw rate Ω, and finally the side slip angle β. All plots, except for the trajectories, are plotted against the curvilinear abscissa ζ.

Catalunya track (validation). For the racecar on the Catalunya track (Figure 8.4), another unseen track, the MPC achieves near-optimal performance. As with the eCrafter, the state profiles closely match those of the MLT solution, with only small differences in corner approach and exit. The g-g diagram confirms that the MPC also operates at the dynamic limits, similarly to the offline optimum. The lap times differ only slightly, with the MPC being about 0.136 s slower.

Overall, these results show that the receding-horizon MPC formulation achieves

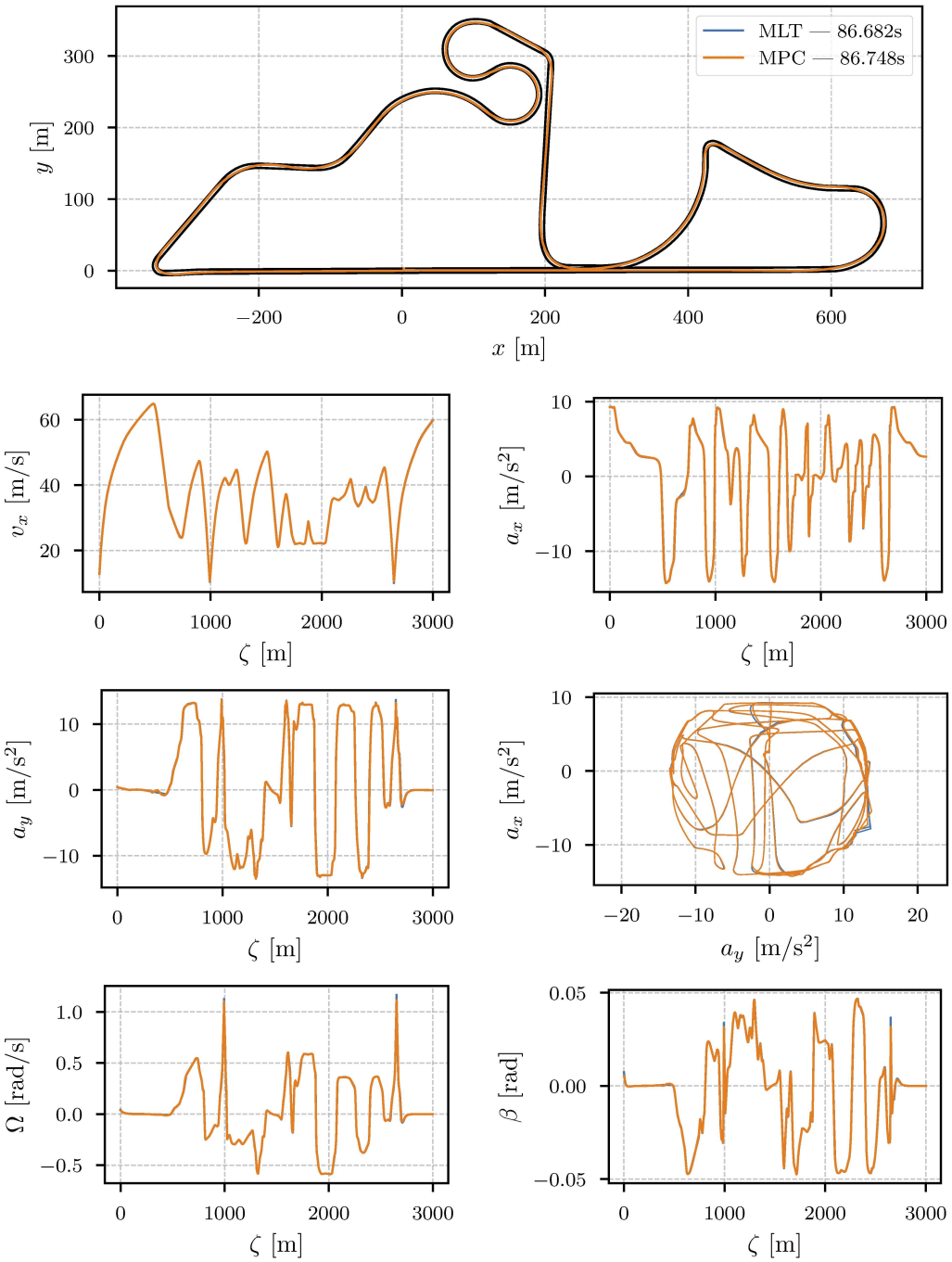


Figure 8.3: Telemetry for the racecar on UniBW: MLT vs MPC. At the top, the racetrack with the two trajectories, and the legend with the solution names and lap times. Below it, the longitudinal velocity v_x, the longitudinal acceleration a_x, the lateral acceleration a_y, the g-g diagram, the yaw rate Ω, and finally the side slip angle β. All plots, except for the trajectories, are plotted against the curvilinear abscissa ζ.

performance very close to the offline MLT benchmark on both vehicles and both circuits. The lap time differences remain well below 0.2 s, and the state profiles together with the g-g diagrams confirm that the MPC consistently exploits the available dynamic envelope.

Although the racecar model is technically capable of speeds up to 340 km/h, we can see that it only reaches about 250 km/h in our simulations. This is a limitation of learning directly from human-driven data. In the dataset the human driver did not exceed speeds over 250 km/h, and we therefore had to assume that was the maximum vehicle speed.

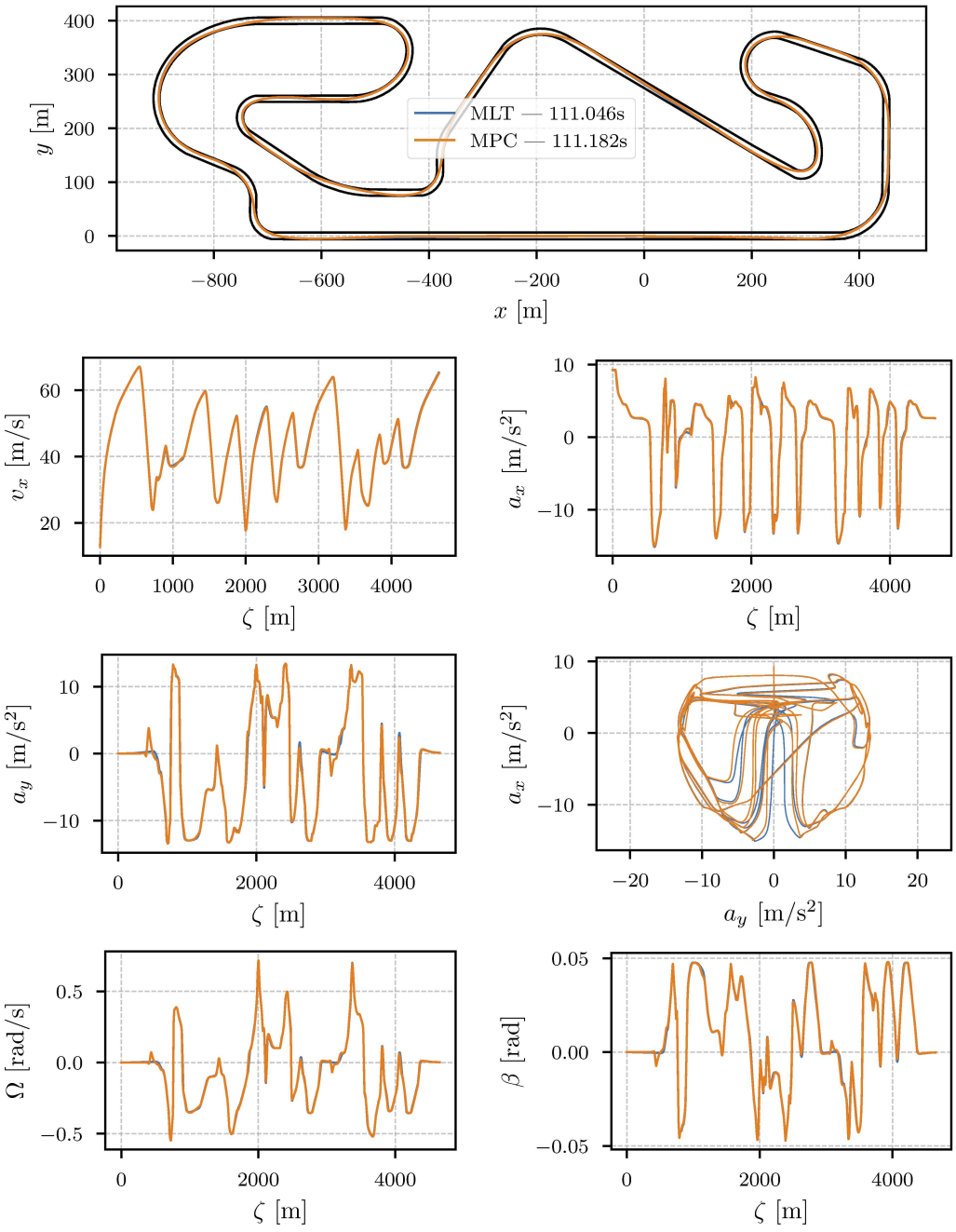


Figure 8.4: Telemetry for the racecar on Catalunya: MLT vs MPC. At the top, the racetrack with the two trajectories, and the legend with the solution names and lap times. Below it, the longitudinal velocity v_x, the longitudinal acceleration a_x, the lateral acceleration a_y, the g-g diagram, the yaw rate Ω, and finally the side slip angle β. All plots, except for the trajectories, are plotted against the curvilinear abscissa ζ.

Having established that the MPC can be competitive with the theoretical optimum, the next section examines its performance when coupled in closed loop with a driven vehicle model.

8.3 MPC vs CL-SYNC

This section compares the trajectories obtained with the online MPC formulation to those generated in closed loop with the driven vehicle model and synchronous execution

(CL-SYNC). The aim is to quantify the impact of model mismatch when the planner and controllers are coupled with a realistic representation of the vehicle dynamics. Lap times are summarized in Table 8.1.

8.3.1 Volkswagen eCrafter

For the eCrafter, CL-SYNC uses a lighter setup to accelerate computation: the horizon is shortened to 100 m with 30 mesh points. Continuity penalties and boundary conditions are retained as in the MPC case. In particular, strict initial conditions are imposed on n and ξ from the current vehicle state, while the remaining state conditions come from the previous MPC solution (interpolated to be at the current time). The shorter horizon reduces the available preview and may degrade solution quality, leading to slightly higher lap times.

UniBW track (training). Figure 8.5 shows the comparison on the UniBW circuit. The CL-SYNC trajectory remains close to the MPC one, though with visible differences in braking and acceleration phases. The lap time increases by about 3.417 s, mainly due to this effect. The g–g diagram highlights that the combined braking regions are utilized less. This behaviour is likely caused by the increased sideslip angle during cornering, which triggers replanning at the high level and corrective actions at the low level.

Catalunya track (validation). On the unseen Catalunya circuit (Figure 8.6), the lap time difference is larger, around 5.182 s. The most evident deviations occur in corners 4 and 12, which are notoriously demanding. Here, the low-level controllers, not trained for this circuit, struggle to track the planned trajectory, resulting in a wider path through those corners. Interestingly, the wider paths keep the sideslip angle lower. This decrease in performance is therefore expected when transferring to a new track.

8.3.2 Racecar

For the racecar, CL-SYNC also adopts a reduced discretization, with a 200 m horizon and 100 mesh points. As with the eCrafter, continuity penalties, strict initial conditions, and soft final conditions are applied. In addition, a small relaxation of the polytope constraint is introduced to improve convergence, which is deemed acceptable since the vehicle is successfully evaluated in closed loop with the driven model.

UniBW track (validation). On the UniBW circuit (Figure 8.7), CL-SYNC again produces competitive results, with a lap time difference of about 1.127 s. The main effect is a reduced ability of the low-level controllers to follow the planned trajectory.

Catalunya track (validation). For the Catalunya circuit (Figure 8.8), the lap time gap between MPC and CL-SYNC is modest, about 1.621 s. The trajectories overlap closely, with the main differences appearing again due to low-level tracking errors.

In summary, moving from the MPC to CL-SYNC introduces a modest performance loss, with lap times increasing by one to five seconds depending on the vehicle and track. The main differences arise in braking and cornering phases, where the low-level controllers and model mismatch limit the exploitation of combined acceleration. Despite this, the overall trajectories remain close to those of the MPC, confirming that ARD retains competitive performance when coupled with a driven vehicle model. The next section investigates the additional effects of asynchronous execution, reflecting the conditions encountered in real-world tests.

On the differences between the eCrafter and the racecar One might notice that, despite the racecar exhibiting far more challenging dynamics, ARD achieves better performance on it—showing three to five times smaller lap-time differences when going

from the MPC to CL-SYNC. This is not a coincidence but stems from how the low-level controllers were trained for the two vehicles. During identification, variations in the corrective PID controller outputs are penalized to prevent unwanted oscillations in the corrective action, which is especially important for stable execution in asynchronous and real-world environments. However, when a model is restricted to simulation only, as in the case of the racecar, the controllers can be tuned more aggressively since they operate exclusively on the simulated vehicle. In contrast, the eCrafter controllers were trained more conservatively to increase the chances of smoother behaviour when deployed on the real vehicle. This was done to prevent unwanted activations of the vehicle safety systems. Allowing a more aggressive tuning would likely improve performance in simulation.

8.4 CL-SYNC vs CL-ASYNC

This section compares closed-loop simulations with synchronous execution (CL-SYNC) against their asynchronous counterparts (CL-ASYNC). The asynchronous mode reflects the real-time execution conditions encountered in the real vehicle, where delays may occur between planning and actuation. Lap times are summarized in Table 8.1.

8.4.1 Volkswagen eCrafter

For the eCrafter, CL-ASYNC inherits the same setup as CL-SYNC but adds a small penalty on longitudinal jerk and yaw acceleration. This additional terms improve convergence under asynchronous execution by slightly reducing acceleration and steering aggressiveness.

UniBW track (training). On the UniBW circuit (Figure 8.9), the synchronous and asynchronous runs appear very similar, yet the lap time increases slightly in the asynchronous case (a difference of about 0.422 s). This is an expected outcome due to the system not waiting for a new MPC solution at each planning step, but using whatever is available. The overall trajectories remain consistent, but this subtle timing effect ultimately produced a marginal lap time disadvantage.

Catalunya track (validation). On the Catalunya circuit (Figure 8.10), the asynchronous run is slower, with a lap time increase of about 0.921 s. The trajectories are largely similar, yet small differences in braking and acceleration accumulate to a noticeable gap. This confirms that delay introduces a slight penalty relative to the synchronous case.

8.4.2 Racecar

For the racecar, CL-ASYNC inherits the CL-SYNC setup, and adds another small relaxation on the controls constraints and a penalization on the maximum speed along the centreline, longitudinal jerk, and yaw acceleration.

UniBW track (validation). On the UniBW circuit (Figure 8.11), the asynchronous run is slower by about 1.042 s. We can see that the longitudinal PID controller struggles to keep the desired longitudinal velocity. This is evident from the longitudinal velocity signal.

Catalunya track (validation). For the Catalunya circuit (Figure 8.12), the asynchronous run is slower by about 1.091 s. We can see that the longitudinal PID controller struggles again to keep the desired longitudinal velocity, oscillating due to delays. This is evident from the longitudinal acceleration signal, which spikes negatively. Despite this, ARD completes the lap successfully, further confirming its capacity to generalize across tracks, even under real-time execution constraints and controller oscillations. It is also worth noting that the resulting trajectories can differ locally, particularly within corners. In these cases, ARD autonomously replans and adjusts its path to minimise the deviation

from the optimal solution. This behaviour is consistent with earlier observations in previous works, highlighting the system's ability to adapt its trajectory online in response to local disturbances.

In summary, asynchronous execution produces lap times that are broadly comparable to the synchronous case, with differences typically within a second. These results highlight both the feasibility of real-time execution and the importance of carefully tuning the low-level controllers for operation under asynchronous conditions.

8.5 Computation Times

Computation times for offline or purely simulated solutions (*i.e.,* MLT, MPC, and CL-SYNC) are of limited relevance. Real-time execution does not constrain these modes. Instead, analysing the planner timing under asynchronous closed-loop operation (CL-ASYNC) provides valuable insight into the practical feasibility of the *Economic Nonlinear Model Predictive Control* (E-NMPC) formulation.

In the CL-ASYNC configuration, the low-level controllers and vehicle dynamics easily run within their respective time budgets of 10 ms, whereas the planner is allowed to exceed its 50 ms cycle time. Consequently, the planner timing statistics directly indicate whether the E-NMPC can sustain real-time operation when deployed on hardware comparable to that used in simulation.

Table 8.2 summarizes the solver statistics for both vehicles and tracks. Each entry reports the mean, standard deviation, median, minimum/maximum computation times, and convergence rates. These timings also include overhead such as guess generation and logging. We remind that these results were obtained on a 16-inch Apple MacBook Pro equipped with an M2 Max processor and 32 GB of RAM.

Table 8.2: Planner computation statistics for CL-ASYNC runs on each vehicle and track. All timings are reported in milliseconds.

Vehicle	Track	μ	σ	Min	Max	Median	Conv. [%]
eCrafter	UniBW	15.259	25.346	6.623	1049.293	10.433	99.9
	Catalunya	13.695	8.338	8.478	355.033	11.622	100.0
Racecar	UniBW	31.346	26.270	13.682	513.135	24.822	99.9
	Catalunya	29.733	28.752	10.331	521.818	23.947	100.0

Given that the cycle time of the high-level planner is 50 ms, it can be seen that the solver operates comfortably within this limit, both in terms of mean and median computation times. However, occasional spikes up to approximately 1 s are observed. These rare events represent the main difference between successful and failed laps, as they correspond to periods in which the vehicle executes for up to one second without receiving a new plan. In such cases, the chosen horizon length and the quality of the low-level controllers play a decisive role in maintaining stability.

Although this behaviour did not compromise the results presented, future developments will include forced timeouts of the E-NMPC to ensure strict real-time operation. This feature was not yet implemented due to convergence issues in the current formulation.

The convergence rates reported in Table 8.2 are remarkably high given the strong non-linearities of the problem, confirming the robustness of both the formulation and the Pins solver. Finally, the higher computation times observed for the racecar model are primarily due to its denser discretization mesh, while the differences between tracks arise from their geometry. In particular, the UniBW circuit features two corners with very high curvature and a constant narrow width of 6 m, which significantly increase the complexity of the optimization problem—a challenge for any planner.

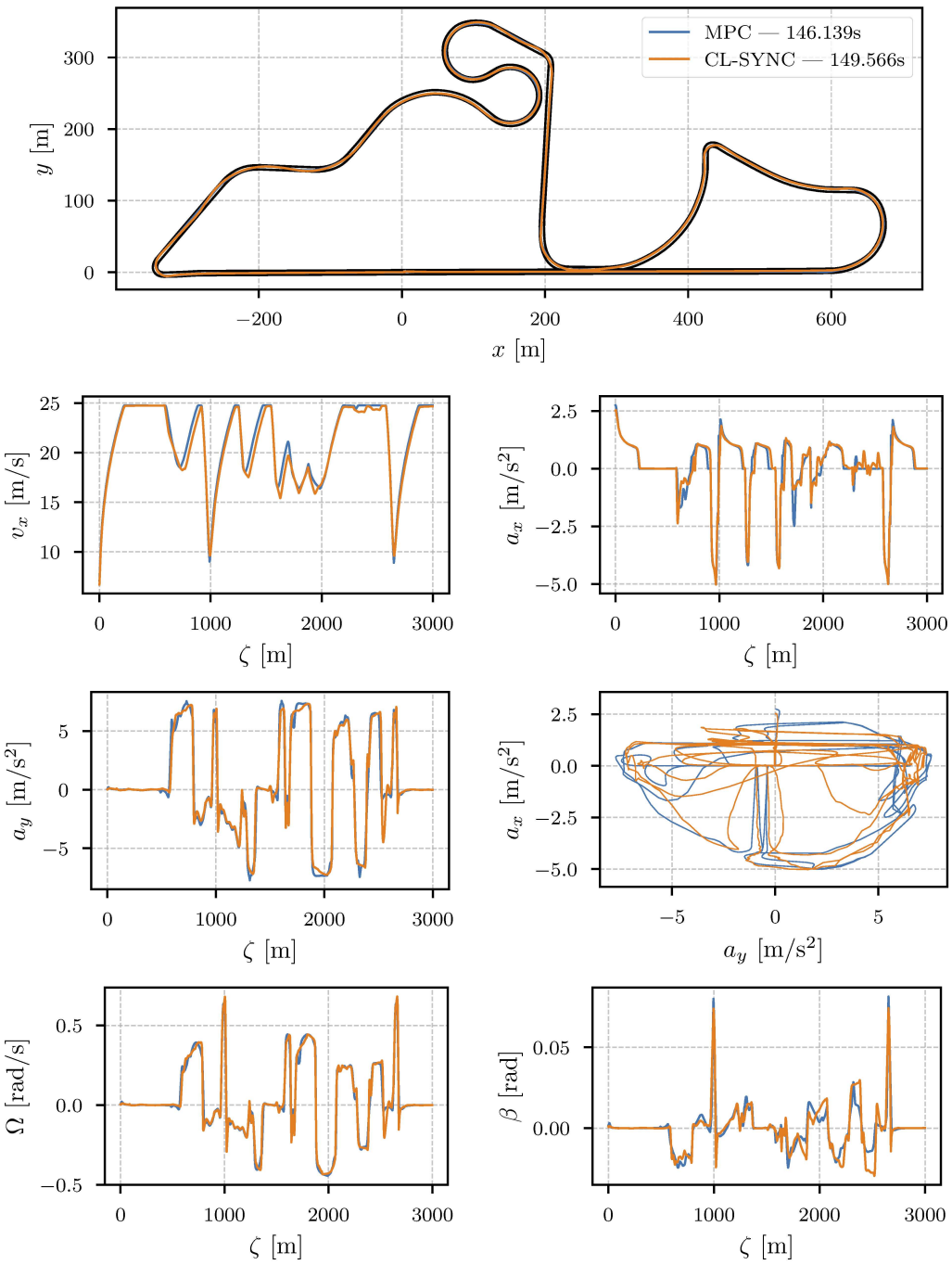


Figure 8.5: Telemetry for the eCrafter on UniBW: MPC vs CL-SYNC. At the top, the racetrack with the two trajectories, and the legend with the solution names and lap times. Below it, the longitudinal velocity v_x, the longitudinal acceleration a_x, the lateral acceleration a_y, the g-g diagram, the yaw rate Ω, and finally the side slip angle β. All plots, except for the trajectories, are plotted against the curvilinear abscissa ζ.

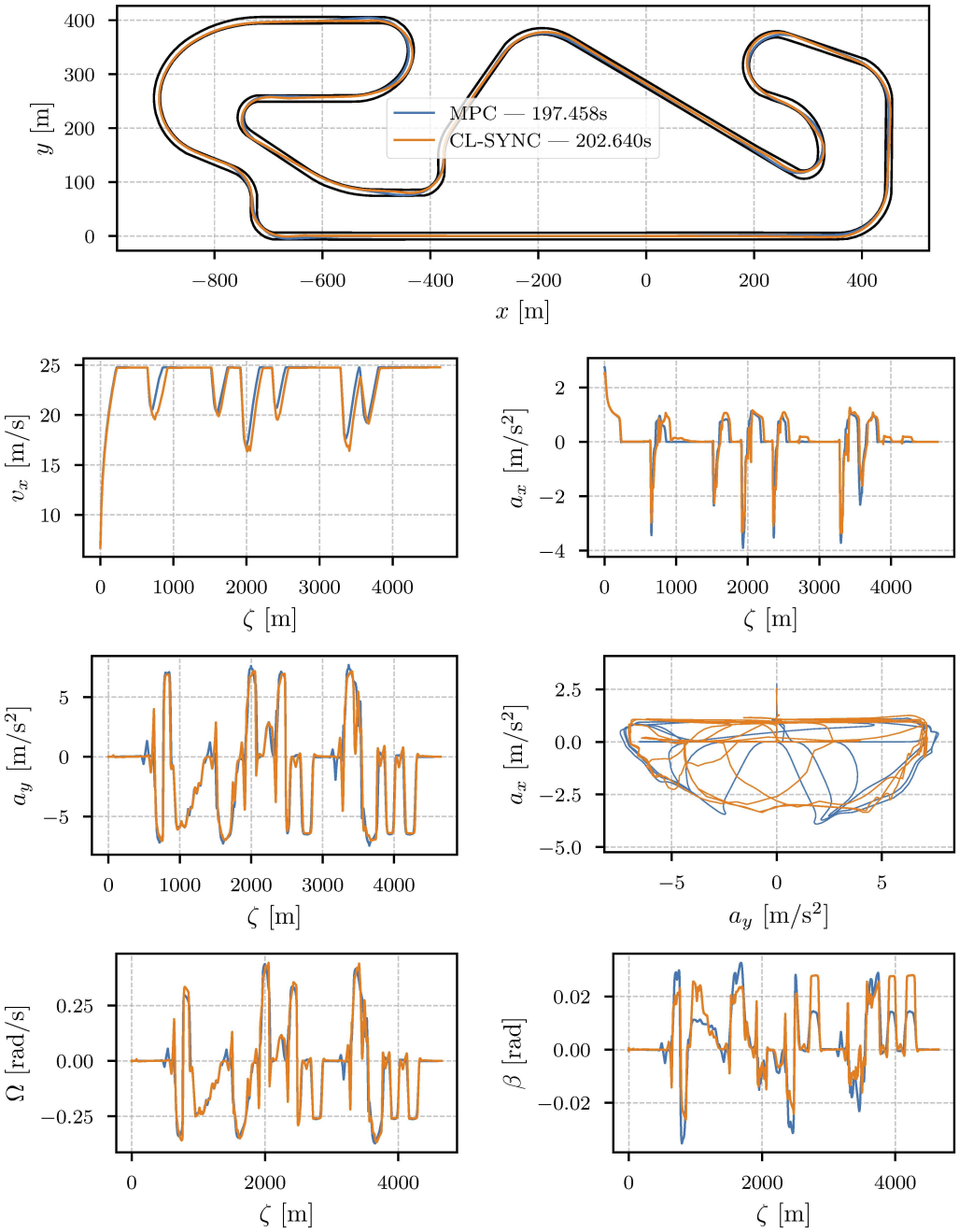


Figure 8.6: Telemetry for the eCrafter on Catalunya: MPC vs CL-SYNC. At the top, the racetrack with the two trajectories, and the legend with the solution names and lap times. Below it, the longitudinal velocity v_x, the longitudinal acceleration a_x, the lateral acceleration a_y, the g-g diagram, the yaw rate Ω, and finally the side slip angle β. All plots, except for the trajectories, are plotted against the curvilinear abscissa ζ.

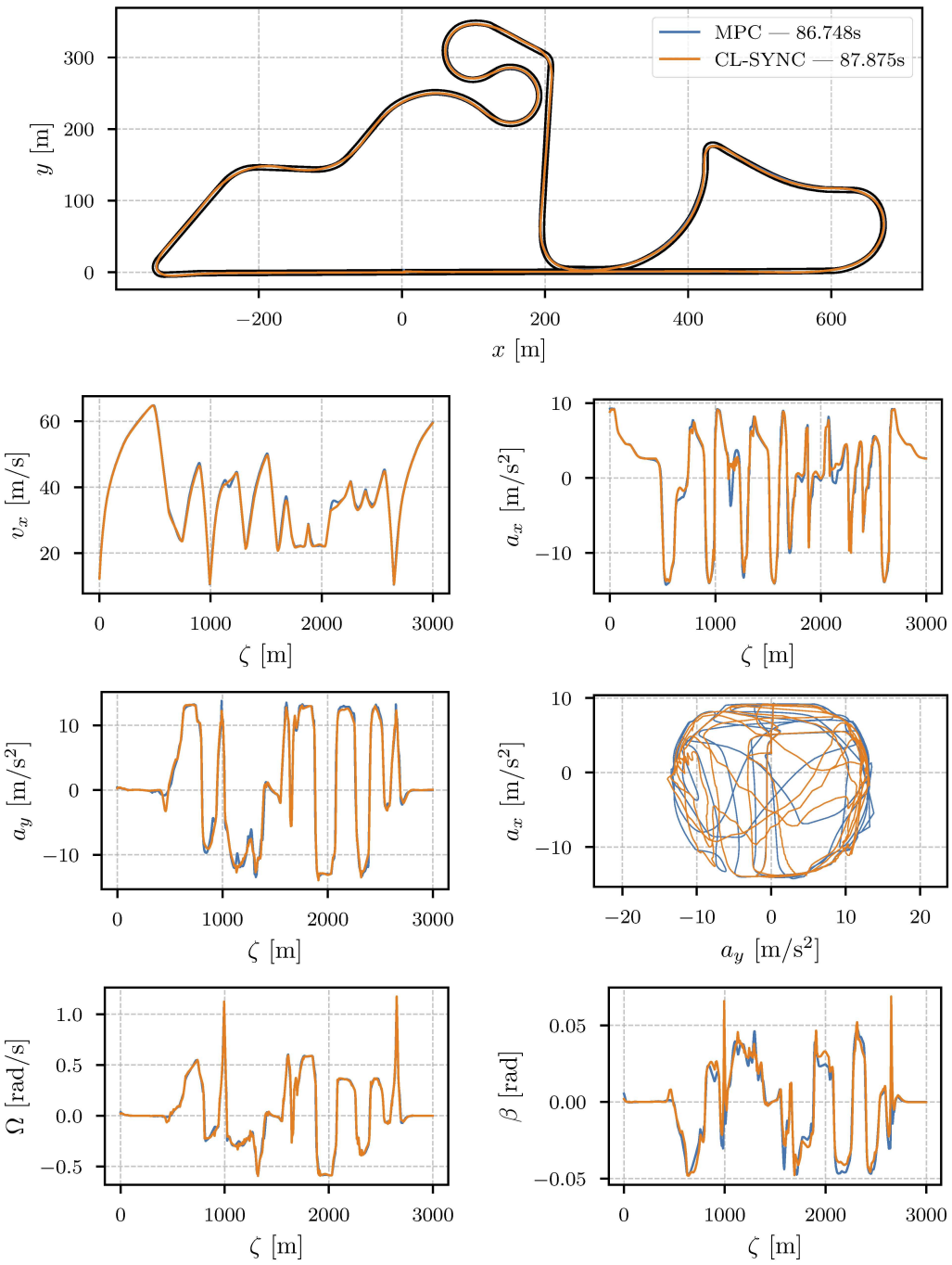


Figure 8.7: Telemetry for the racecar on UniBW: MPC vs CL-SYNC. At the top, the racetrack with the two trajectories, and the legend with the solution names and lap times. Below it, the longitudinal velocity v_x, the longitudinal acceleration a_x, the lateral acceleration a_y, the g-g diagram, the yaw rate Ω, and finally the side slip angle β. All plots, except for the trajectories, are plotted against the curvilinear abscissa ζ.

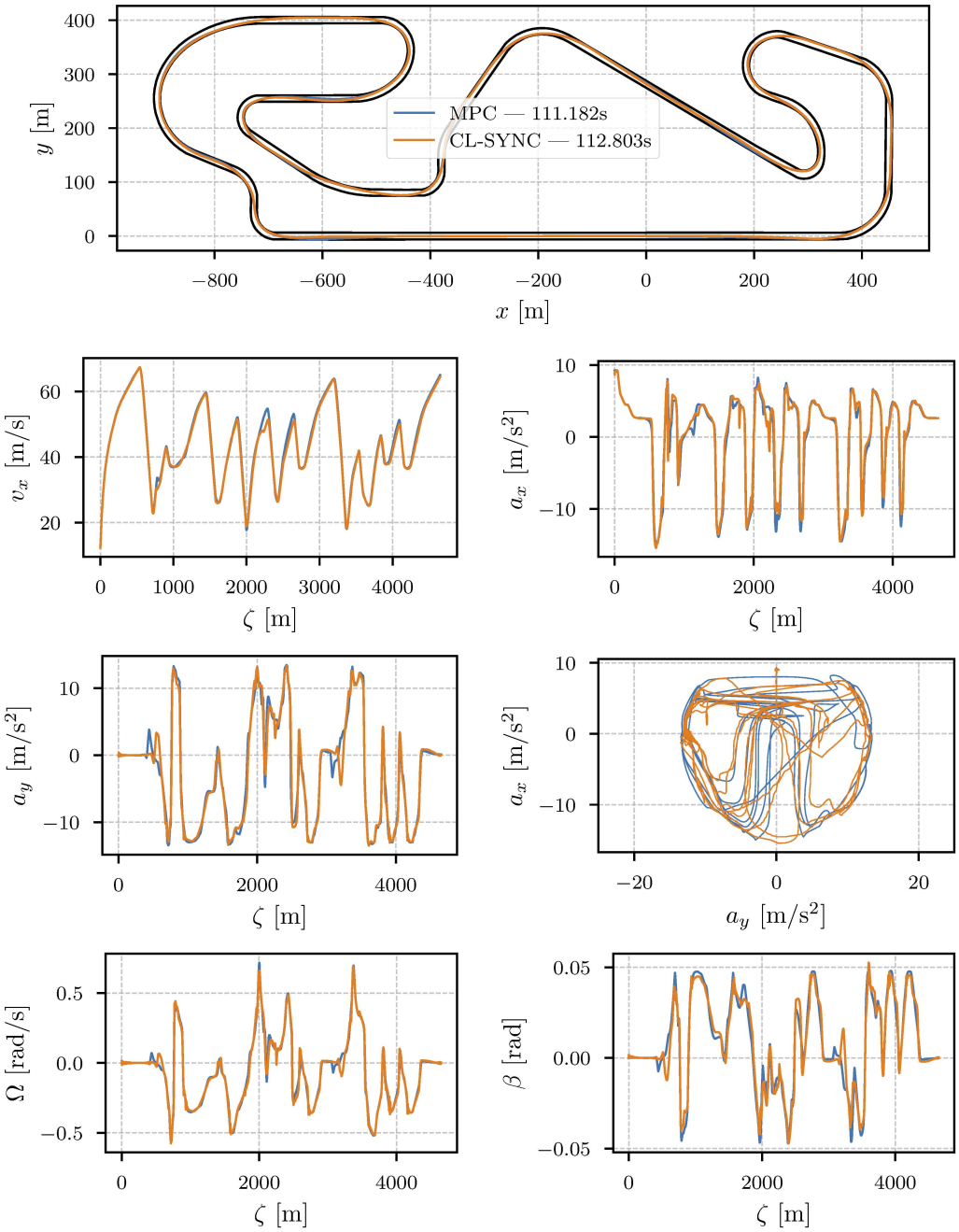


Figure 8.8: Telemetry for the racecar on Catalunya: MPC vs CL-SYNC. At the top, the racetrack with the two trajectories, and the legend with the solution names and lap times. Below it, the longitudinal velocity v_x, the longitudinal acceleration a_x, the lateral acceleration a_y, the g-g diagram, the yaw rate Ω, and finally the side slip angle β. All plots, except for the trajectories, are plotted against the curvilinear abscissa ζ.

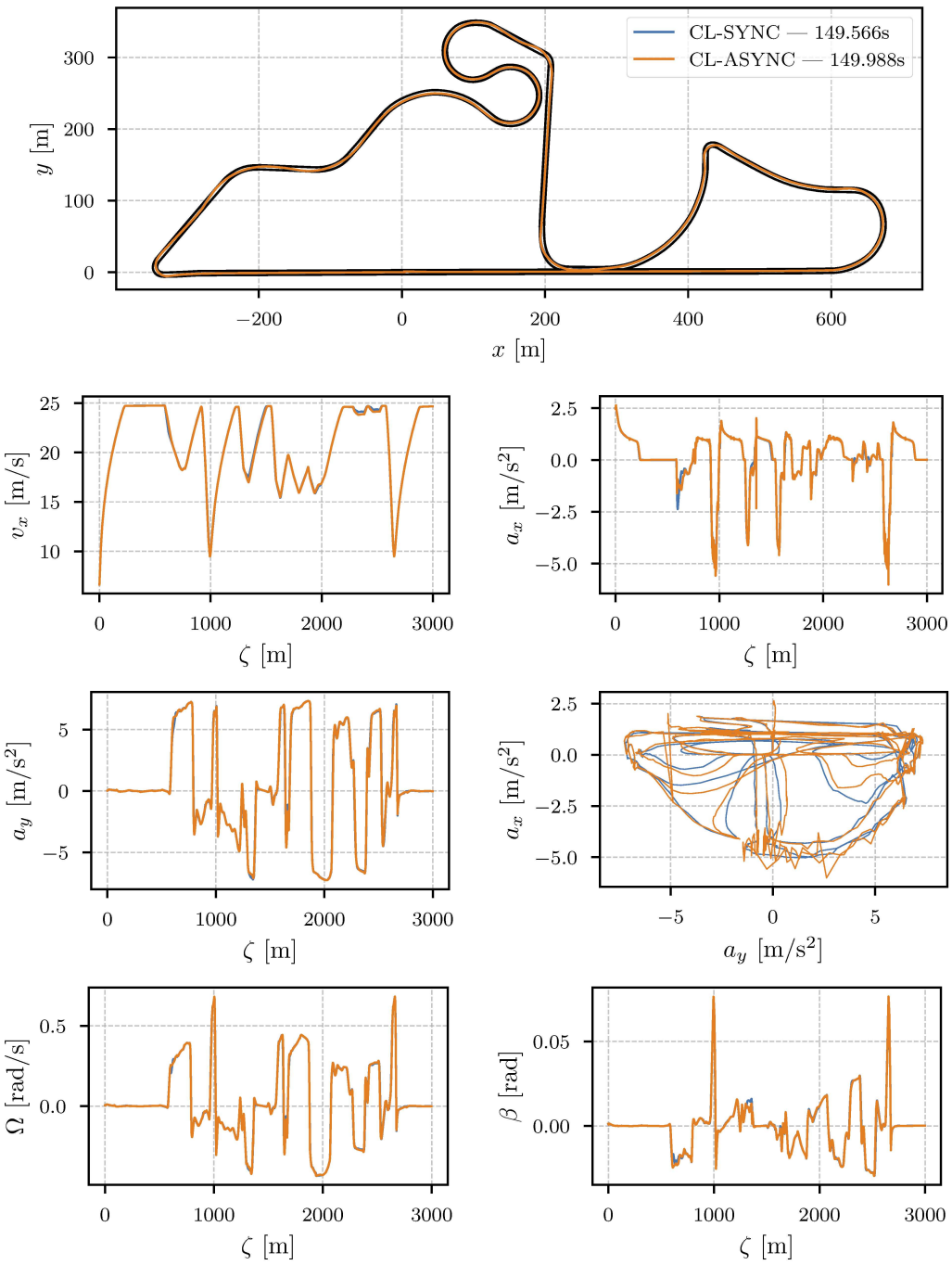


Figure 8.9: Telemetry for the eCrafter on UniBW: CL-SYNC vs CL-ASYNC. At the top, the racetrack with the two trajectories, and the legend with the solution names and lap times. Below it, the longitudinal velocity v_x, the longitudinal acceleration a_x, the lateral acceleration a_y, the g-g diagram, the yaw rate Ω, and finally the side slip angle β. All plots, except for the trajectories, are plotted against the curvilinear abscissa ζ.

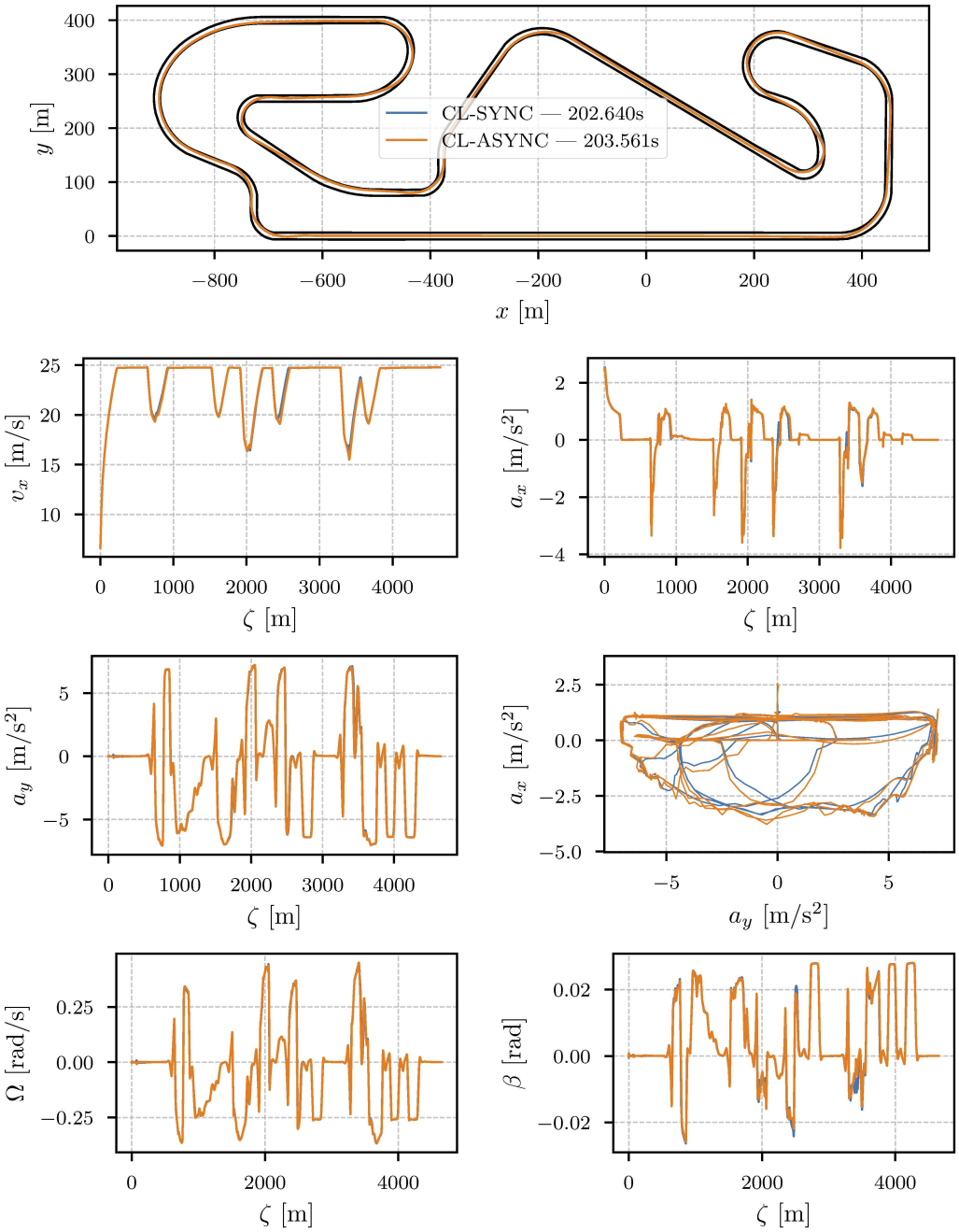


Figure 8.10: Telemetry for the eCrafter on Catalunya: CL-SYNC vs CL-ASYNC. At the top, the racetrack with the two trajectories, and the legend with the solution names and lap times. Below it, the longitudinal velocity v_x, the longitudinal acceleration a_x, the lateral acceleration a_y, the g-g diagram, the yaw rate Ω, and finally the side slip angle β. All plots, except for the trajectories, are plotted against the curvilinear abscissa ζ.

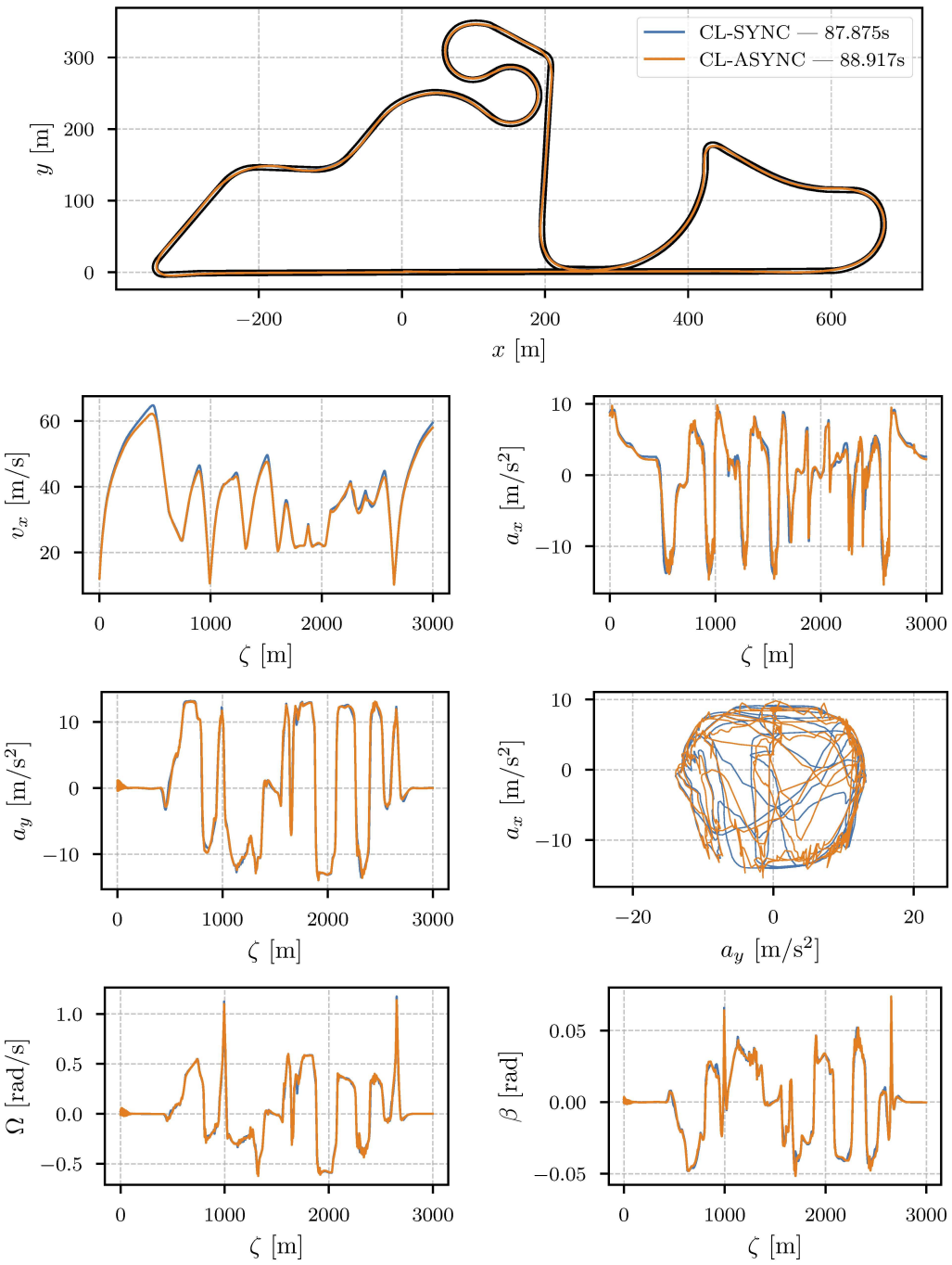


Figure 8.11: Telemetry for the racecar on UniBW: CL-SYNC vs CL-ASYNC. At the top, the racetrack with the two trajectories, and the legend with the solution names and lap times. Below it, the longitudinal velocity v_x, the longitudinal acceleration a_x, the lateral acceleration a_y, the g-g diagram, the yaw rate Ω, and finally the side slip angle β. All plots, except for the trajectories, are plotted against the curvilinear abscissa ζ.

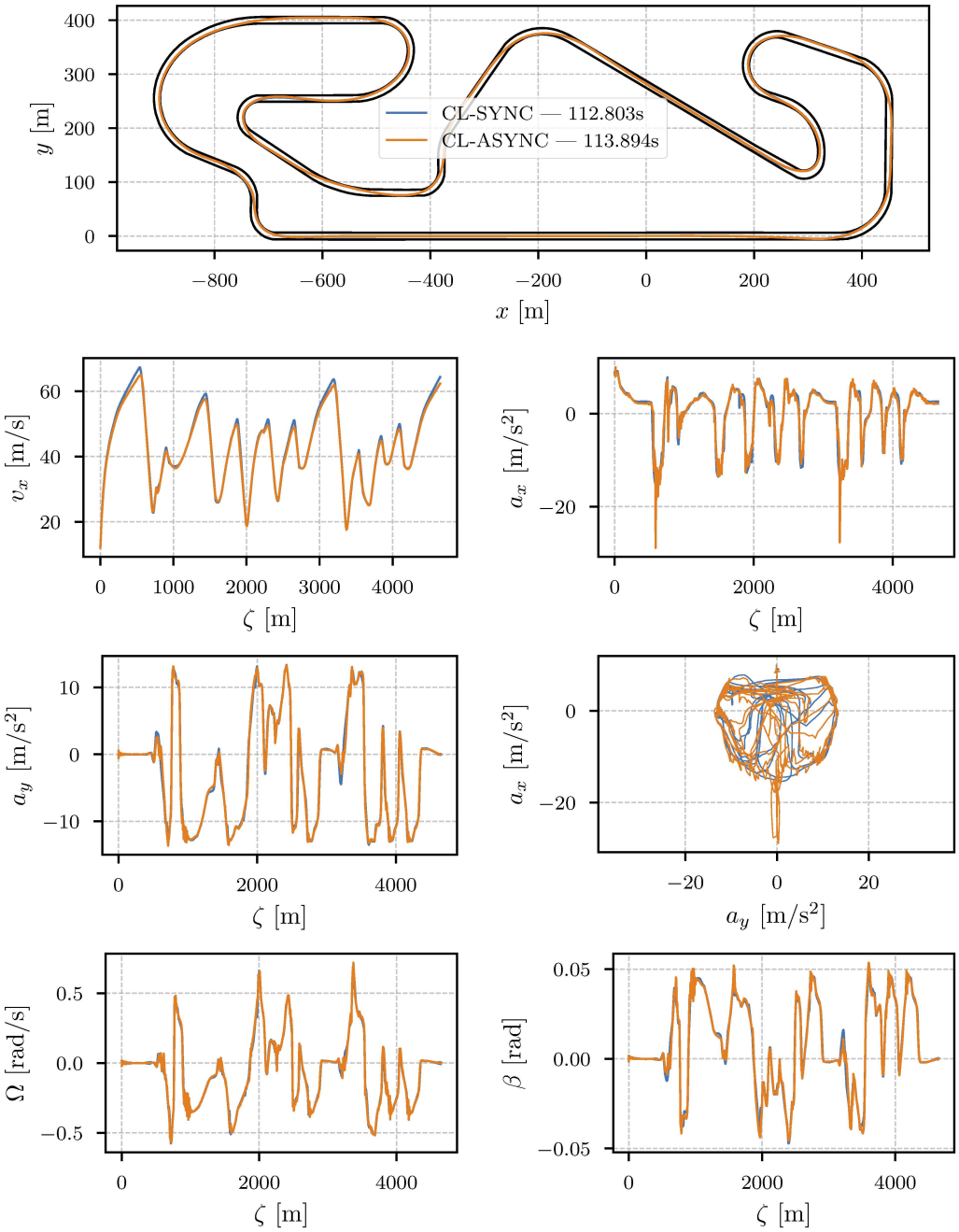


Figure 8.12: Telemetry for the racecar on Catalunya: CL-SYNC vs CL-ASYNC. At the top, the racetrack with the two trajectories, and the legend with the solution names and lap times. Below it, the longitudinal velocity v_x, the longitudinal acceleration a_x, the lateral acceleration a_y, the g-g diagram, the yaw rate Ω, and finally the side slip angle β. All plots, except for the trajectories, are plotted against the curvilinear abscissa ζ.

8.6 Evaluation under Non-Ideal Conditions

To assess the behaviour of ARD under realistic communication and sensing conditions, a set of tests was conducted in asynchronous mode with intentionally degraded data transmission and measurement quality. These experiments aimed to observe how the system would react when facing packet losses and sensor noise.

The tests were performed using the same setup adopted for the real-world experiments, in which the vehicle model runs as a ROS node and communicates with the planner and controller through the ROS bridge. To favour convergence, the g-g-v and control constraints were slightly relaxed compared to the CL-ASYNC configuration. The same relaxation was used in the real-world tests, where it proved effective and safe.

Three packet loss levels were considered: 0%, 25%, and 50%, meaning that on average no message was lost, one in four messages was lost, and half of the messages were lost, respectively. Independent white noise was also added to the main state variables, scaled by a noise multiplier taking values 0.0, 1.0, and 2.0. The perturbations were defined as

$$\begin{aligned} x &\mathrel{+}= \mathcal{N}(0, 0.025)\,\nu & v_y &\mathrel{+}= \mathcal{N}(0, 0.025)\,\nu \\ y &\mathrel{+}= \mathcal{N}(0, 0.025)\,\nu & \Omega &\mathrel{+}= \mathcal{N}(0, 0.005)\,\nu \\ \psi &\mathrel{+}= \mathcal{N}(0, 0.010)\,\nu & a_x &\mathrel{+}= \mathcal{N}(0, 0.100)\,\nu \\ v_x &\mathrel{+}= \mathcal{N}(0, 0.250)\,\nu & a_y &\mathrel{+}= \mathcal{N}(0, 0.100)\,\nu \end{aligned} \tag{8.1}$$

where ν denotes the selected noise multiplier.

Packet Loss [%]	Noise Multiplier ν		
	0.0	1.0	2.0
0	150.746	150.709	152.188
25	150.631	151.122	152.495
50	150.553	150.834	153.741*

Table 8.3: Lap times [s] for varying packet loss and noise. The asterisk marks the case in which the vehicle briefly exceeded the track limits.

Table 8.3 summarizes the obtained lap times for all tested configurations. Overall, ARD maintained stable operation and completed full laps in all cases. The lap-time variations remain within a few seconds across the tested range, confirming that the framework can handle moderate message losses and noisy signals without degradation in performance. Only the most demanding configuration (50% loss and noise multiplier 2.0) briefly exceeded the track boundaries, marked with an asterisk in the table, indicating proximity to the system's operational limit.

These findings show that ARD remains functional and effective even under non-ideal conditions similar to those encountered in real-world deployment. Figure 8.13 shows the results obtained during the harshest setting: packet loss at 50 % and noise multiplier set to 2.

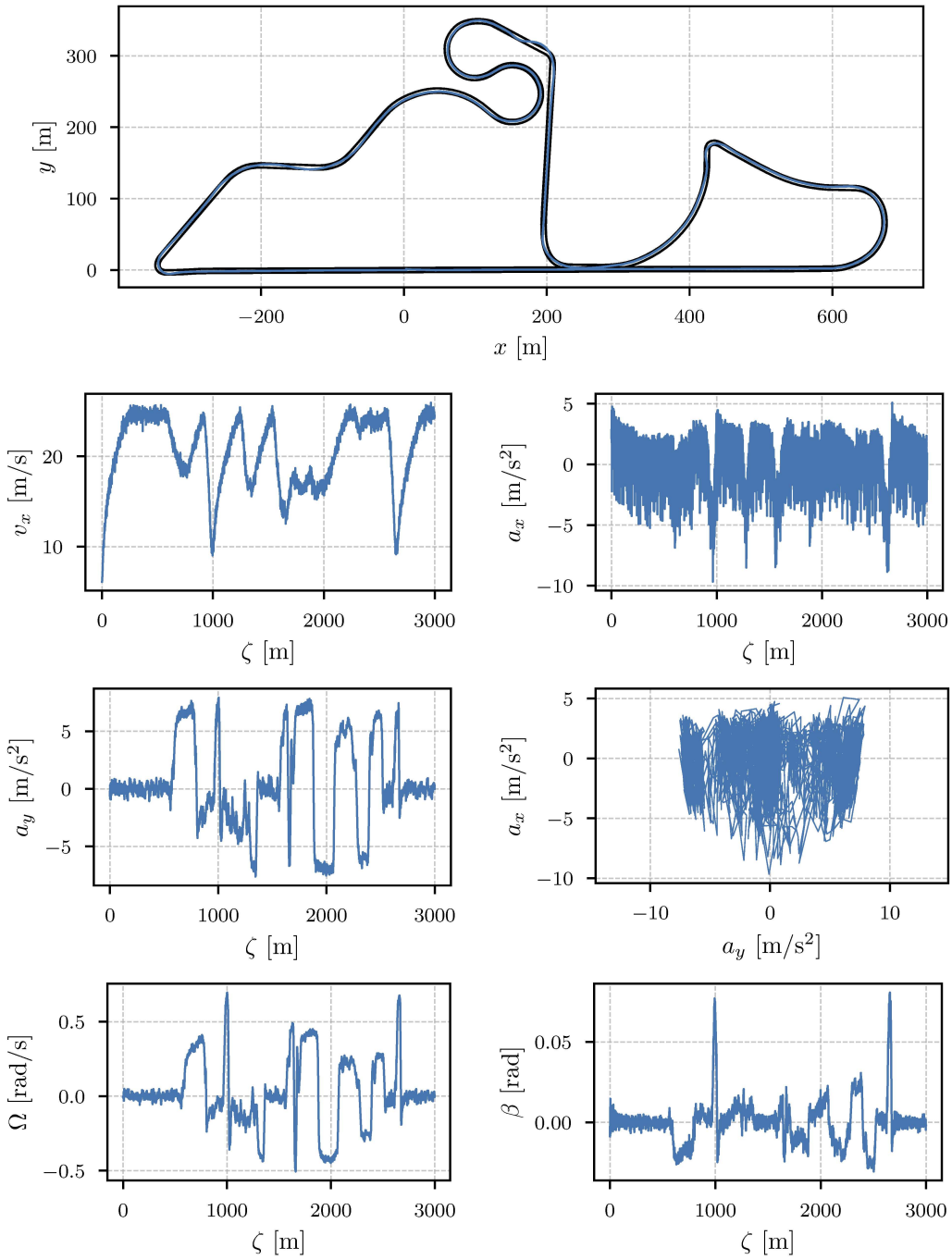


Figure 8.13: Telemetry for the eCrafter with 50 % packet-loss and noise multiplier set to 2. At the top, the racetrack with the trajectory. Below it, the longitudinal velocity v_x, the longitudinal acceleration a_x, the lateral acceleration a_y, the g-g diagram, the yaw rate Ω, and finally the side slip angle β. All plots, except for the trajectory, are plotted against the curvilinear abscissa ζ. We can see how the system exits the track on corner 6, showing that we are close to the limits of how much disturbance the system can handle.

8.7 Driving Style Biasing

In addition to generating minimum-time manoeuvres, ARD can adapt its driving style online by modifying the cost terms of its E-NMPC. In particular, two elements influence the behaviour of the planner: a terminal cost acting on the lateral displacement n and relative yaw angle ξ, which removes the dependency on full-lap solutions, and a term that maximizes the vehicle's exit speed at the end of the horizon. By tuning the corresponding weight W_{v_x} of the exit speed maximization, ARD can be biased from an early- to a late-apex driving style.

The following experiments were conducted on the Catalunya racetrack, shown in Figure 8.14, using the old identification approach on the high-fidelity double-track racecar model. The results were obtained with the E-NMPC closing the loop on itself, therefore without driving the actual vehicle model. This allows us to focus on biasing the E-NMPC solution itself, without additional effects stemming from closed-loop execution.

The planning horizon was set to $L = 300$ m and replanning occurred every 50 ms. The minimum-time weight was fixed to $W_t = 2.0$, while the exit-speed weight W_{v_x} was varied between 0.00 and 0.10 in steps of 0.01. Each configuration is denoted as $\text{MPC}_{W_{v_x}}$, for example $\text{MPC}_{0.00}$ represents the pure minimum-time solution. Table 8.4 summarizes the test parameters.

Throughout this Section, the *apex* is defined as the point along the vehicle trajectory where the minimum speed is reached. The *clipping point* refers to the location where the trajectory comes closest to the inner track boundary within a corner.

Table 8.4: MPC parameters for the driving-style biasing tests.

Parameter	Value
Planning horizon (L)	300 m
Replan time	50 ms
Minimum-time weight (W_t)	2.0
Exit-speed weight (W_{v_x})	[0.00, 0.01, …, 0.10]

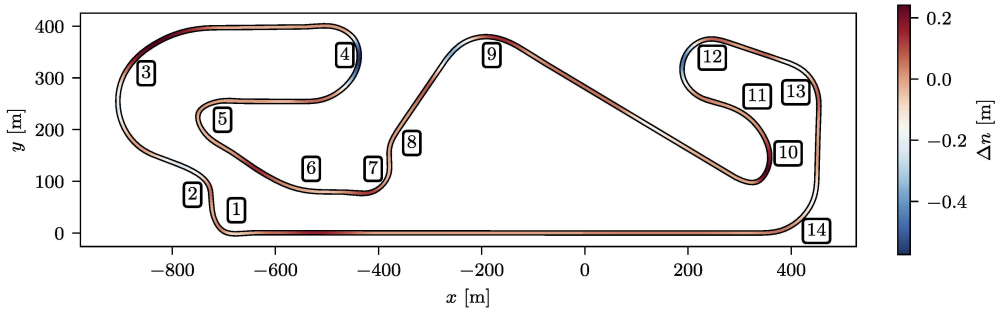
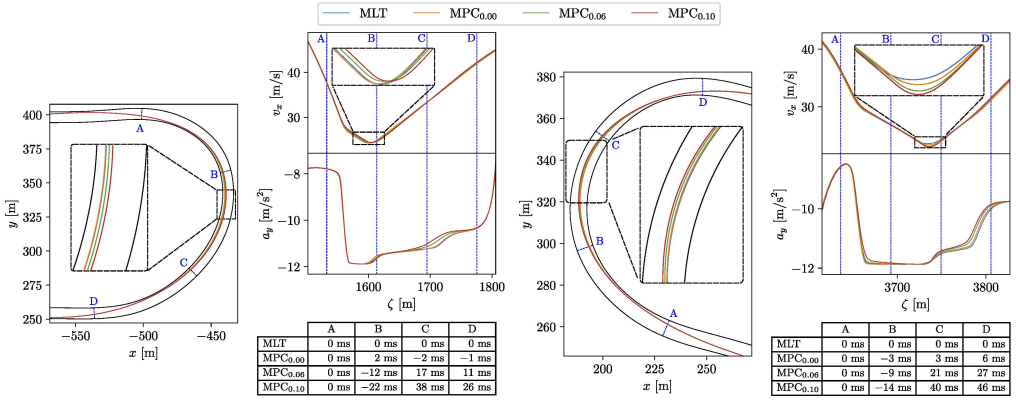


Figure 8.14: Difference between the lateral coordinates of $\text{MPC}_{0.00}$ and the best-performing configuration $\text{MPC}_{0.06}$ on the Catalunya racetrack. The trajectories exhibit systematic widening near corner exits, consistent with a late-apex style. Figure taken from our work [97] © 2025 IEEE.

Corner 4 and 12 analysis. Figure 8.15 compares representative corners of the Catalunya circuit. Increasing W_{v_x} progressively shifts the apex of each turn towards the exit, producing wider paths and higher exit speeds. In Corner 4, for instance, $\text{MPC}_{0.06}$ and $\text{MPC}_{0.10}$ maintain greater speed within the turn and lower lateral accelerations a_y in

the exit phase, while the minimum-time solution ($W_{v_x} = 0$) follows an earlier apex and accelerates sooner. The same trend is visible in Corner 12, where higher W_{v_x} values yield higher exit velocities and lower combined accelerations. This behaviour aligns with the intuition that late-apex trajectories may offer improved stability and repeatability without increasing the overall lap time.



	A	B	C	D
MLT	0 ms	0 ms	0 ms	0 ms
$MPC_{0.00}$	0 ms	2 ms	−2 ms	−1 ms
$MPC_{0.06}$	0 ms	−12 ms	17 ms	11 ms
$MPC_{0.10}$	0 ms	−22 ms	38 ms	26 ms

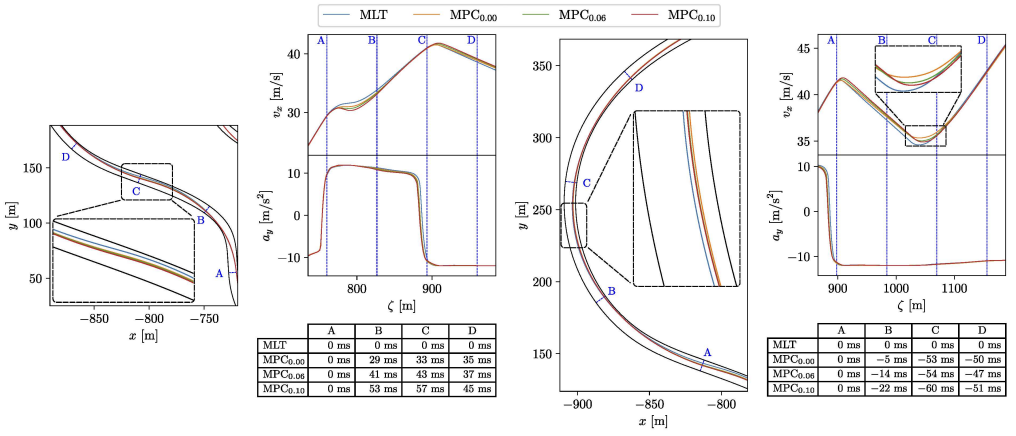
(a)

	A	B	C	D
MLT	0 ms	0 ms	0 ms	0 ms
$MPC_{0.00}$	0 ms	−3 ms	3 ms	6 ms
$MPC_{0.06}$	0 ms	−9 ms	21 ms	27 ms
$MPC_{0.10}$	0 ms	−14 ms	40 ms	46 ms

(b)

Figure 8.15: Effect of W_{v_x} on the planned trajectories in Corner 4 (left) and Corner 12 (right). Increasing W_{v_x} leads to later apexes, higher exit speeds, and lower combined accelerations. Figure taken from our work [97] © 2025 IEEE.

Corner 3 analysis. A similar effect is observed in Corner 3 (Figure 8.16). On the short preceding straight (segment B–C in Figure 8.16a), the $MPC_{W_{v_x}}$ solutions place the car progressively closer to the outer (left) margin as W_{v_x} increases, enabling later braking and higher entry speeds at point A (Figure 8.16b). All configurations exhibit two clipping points, and the apex (minimum speed) shifts towards the exit as W_{v_x} increases (late apex). In contrast, the MLT trajectory approaches the corner from the right margin, brakes earlier, and maintains a lower speed up to the acceleration point (reached earlier than the $MPC_{W_{v_x}}$; early apex). Notably, the MLT follows the widest overall path among all configurations across this section (straight and corner), which explains its local time advantage in the entry segment (see the table in Figure 8.16a). However, $MPC_{0.10}$ is the fastest through the corner itself (sector A–D in Figure 8.16b), showing that a late-apex trajectory can be locally time-optimal. Such behaviour is unattainable with conventional minimum-time E-NMPC, highlighting the benefit of our biasing formulation.



	A	B	C	D
MLT	0 ms	0 ms	0 ms	0 ms
$MPC_{0.00}$	0 ms	29 ms	33 ms	35 ms
$MPC_{0.06}$	0 ms	41 ms	43 ms	37 ms
$MPC_{0.10}$	0 ms	53 ms	57 ms	45 ms

(a)

	A	B	C	D
MLT	0 ms	0 ms	0 ms	0 ms
$MPC_{0.00}$	0 ms	−5 ms	−53 ms	−50 ms
$MPC_{0.06}$	0 ms	−14 ms	−54 ms	−47 ms
$MPC_{0.10}$	0 ms	−22 ms	−60 ms	−51 ms

(b)

Figure 8.16: Straight segment (left) and Corner 3 (right) of the Catalunya racetrack. Higher W_{v_x} values yield later braking points and a clear transition from early- to late-apex trajectories. Figure taken from our work [97] © 2025 IEEE.

Lap-time comparison. Table 8.5 summarizes the lap times and solver performance. The configuration $\text{MPC}_{0.06}$ achieves the best overall performance, completing the lap only 7 ms slower than the offline MLT solution, and faster than the pure minimum-time $\text{MPC}_{0.00}$. This confirms that moderate biasing towards higher exit speeds can improve global performance. The mean solve times remain below 30 ms, confirming that the approach is suitable for online execution.

Table 8.5: Comparison of lap times and mean solve times of the MLT and $\text{MPC}_{W_{v_x}}$ solutions.

	Lap time [s]	Mean solve time [ms]
MLT	113.535	–
$\text{MPC}_{0.00}$	113.556	28.923
$\text{MPC}_{0.06}$	113.542	27.682
$\text{MPC}_{0.10}$	113.552	27.552

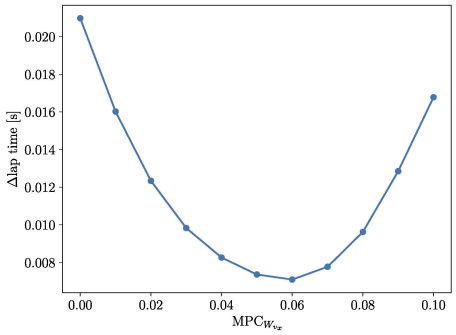
Sensitivity analyses. The effect of the W_{v_x} weight is summarized in Figure 8.17. Figure 8.17a shows that the lap-time difference with respect to the MLT follows a parabolic trend with a minimum at $W_{v_x} = 0.06$, indicating that moderate exit-speed maximization yields the best trade-off between exit speed maximization and global lap time. Figure 8.17b reports the *Curvature-weighted Mean Deviation* (MDK) of the lateral coordinate n and longitudinal speed v_x from the MLT solution, defined as

$$\text{MDK}(x) = \text{mean}\Big((x_{\text{MPC}_{W_{v_x}}} - x_{\text{MLT}}) \cdot \frac{\kappa}{\kappa_{\max}}\Big),$$

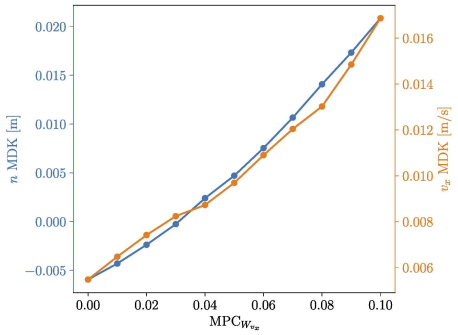
where κ is the centreline curvature and $\kappa_{\max}$ its maximum. Positive n deviations indicate wider paths in corners, while positive v_x deviations indicate higher speeds; both increase with W_{v_x}, confirming the systematic bias towards wider, faster, late-apex trajectories. Finally, Figure 8.17c shows the *Root-Mean-Squared Deviation* (RMSD) of v_x and a_x from the MLT: $\text{MPC}_{0.06}$ attains the lowest RMSD for both signals, explaining its overall lap-time advantage.

Overall, these results demonstrate that ARD can modulate its driving style online through a simple adjustment of the cost-function weights. By biasing the exit-speed term, ARD transitions smoothly from an early- to a late-apex style while maintaining real-time operation and near-optimal lap times. This adaptability not only highlights the flexibility of the proposed E-NMPC formulation, but also provides a foundation for future research on context-aware and human-inspired racing strategies.

(a) Lap-time difference between $\text{MPC}_{W_{v_x}}$ and the MLT solution. The minimum at $W_{v_x} = 0.06$ identifies the best compromise between minimum-time and exit-speed objectives.



(b) MDK of lateral position n and longitudinal velocity v_x with respect to the MLT. Higher W_{v_x} values yield wider and faster trajectories.



(c) RMSD of longitudinal velocity v_x and longitudinal acceleration a_x from the MLT profiles. $\text{MPC}_{0.06}$ shows the lowest RMSD, indicating the closest dynamic behaviour to the MLT.

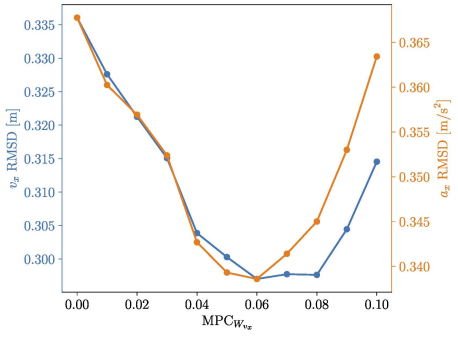


Figure 8.17: Sensitivity analyses of the $\text{MPC}_{W_{v_x}}$ solutions. (a) Lap-time difference; (b) MDK of n and v_x; (c) RMSD of v_x and a_x. Together, they show that a moderate exit-speed weight ($W_{v_x} = 0.06$) yields the best compromise between minimum-time and maximum-speed objectives. Figures taken from our work [97] © 2025 IEEE.

Chapter 9

Real-World Deployment

Abstract

This chapter documents the real-world deployment of the *Artificial Race Driver* (ARD) framework. The campaign aimed to demonstrate the feasibility and reliability of executing the complete planning and control pipeline on a real vehicle under realistic operating conditions. Five autonomous laps were performed at increasing levels of aggressiveness, followed by a human-driven lap recorded in the same session for comparison. The experiments confirm that ARD can operate autonomously and consistently outside simulation, approaching the vehicle's limits within the imposed safety constraints. The collected data enable a direct comparison with simulation and offer valuable insights into the correspondence between simulated and real behaviour.

Contents

9.1 Introduction 120
9.1.1 Experimental Constraints and Limitations 120
9.2 Conducted Experiments 121
9.3 Discussion 131

9.1 Introduction

The final stage in the development of the ARD framework was its deployment on a real vehicle. After extensive validation in simulation, the goal was to demonstrate that the same architecture could operate autonomously in a real environment, closing the gap between simulation and the real world. While Chapter 7 described the vehicle platform and experimental infrastructure in detail, this chapter focuses on the outcomes of the on-track campaign.

The testing campaign aimed to evaluate the complete planning–control pipeline under realistic operating conditions. The available track time amounted to approximately ten hours, half of which were spent on vehicle setup and troubleshooting. The results presented here were all obtained during the final test session, which lasted a little over one hour. As such, all runs were executed under comparable environmental conditions.

Five autonomous laps were performed at increasing levels of aggressiveness, progressively relaxing safety margins and virtual speed limits to explore the vehicle's dynamic capabilities. A human-driven lap was also recorded in the same session to provide a performance reference under identical conditions. These experiments confirm that ARD can operate autonomously and consistently on a real production vehicle. The collected data mark an important step towards systematic real-world validation and provide insight into the transfer of physics-driven, time-optimal planning and control strategies from simulation to practice.

9.1.1 Experimental Constraints and Limitations

The real-world deployment of the ARD framework was subject to several technical and organisational constraints that influenced the achievable performance and shaped the scope of the campaign. While these factors did not diminish the significance of the results, they provide essential context for their interpretation.

Limited track time. The total track time amounted to roughly ten hours, including vehicle preparation, calibration, and troubleshooting. Because of the tight booking schedule, repeated sessions were not possible, and a significant portion of the available time was devoted to resolving hardware and communication issues. As a result, only a limited number of complete autonomous runs could be executed.

Reference-line accuracy. The reference lines and track boundaries provided for planning exhibited alignment offsets with respect to the actual layout. To prevent off-track behaviour, the bounds were conservatively padded. The nominal track width was 6 m, while the vehicle, being 2.426 m wide, already occupied nearly half of it. After padding, the effective drivable corridor was reduced to about 4.787 m, constraining the achievable cornering speeds and limiting the use of combined accelerations.

Sensor quantization. The measured steering-angle signal exhibited coarse quantization, preventing an accurate identification of the steering dynamics. The corresponding identified dynamics and additional delay parameter were therefore estimated heuristically from observed responses rather than through formal system identification.

Steering actuation. The steering system introduced an effective delay of approximately 200 ms, which was only roughly estimated and therefore compensated only partially in the control pipeline. In addition, internal rate limits were enforced on the steering commands, the magnitude of which was not disclosed. These factors reduced the effective control capabilities of the steering controller.

Longitudinal control. The vehicle relied on a closed and undocumented longitudinal control module, which tracked a requested speed reference. The response of this module was noticeably slow when handling aggressive speed profiles, making late braking actions impractical. Combined with the presence of rough tarmac sections along the track—composed of discrete blocks rather than a continuous surface—this motivated the use of conservative braking points to avoid loss of stability or activation of safety systems.

Safety systems. All production safety features remained active, including *Anti-lock Braking System* (ABS), *Electronic Stability Programme* (ESP), rollover protection, and a steering-rate limiter. In addition, the vehicle engineer supervising the tests manually intervened whenever the trajectory approached the track boundaries or unexpected behaviour was observed. Each intervention, whether triggered by the safety systems or by the operator, immediately disabled the autonomous mode and terminated the run.

These constraints collectively explain the conservative driving conditions adopted during testing. Nevertheless, the experiments provided valuable feedback on the integration of the ARD framework with production-grade hardware and highlighted several key aspects to address in future campaigns.

9.2 Conducted Experiments

The final test session comprised six laps in total: five executed autonomously by ARD and one performed manually by a human driver. All runs were conducted consecutively within a single session lasting slightly over one hour, ensuring comparable environmental conditions and vehicle states. The track surface was partially wet, as can be seen in Figure 9.9. The autonomous laps were carried out at increasing levels of aggressiveness, progressively relaxing virtual speed limits and safety margins to explore the vehicle's dynamic potential within safe boundaries. Virtual speed limits were defined along the curvilinear coordinate of the track to regulate braking points and peak velocities, providing an additional safety layer in sections with higher curvature or rough surface conditions. Although the track is mostly 2D, various segments are paved with discrete road blocks rather than continuous asphalt, producing noticeable vertical oscillations at higher speeds.

Since these effects were not modelled, they necessitated conservative limits to prevent partial tyre unloading and unintended activation of the production safety systems.

The *Economic Nonlinear Model Predictive Control* (E-NMPC) was configured as follows. The horizon set to 100 m with 100 mesh points. Cycle time set to 100 Hz. Continuity penalties, strict initial conditions on n and ξ from the current vehicle state, while the remaining state conditions come from the previous MPC solution (interpolated to be at the current time). Soft final conditions on n and ξ to lightly bias toward the reference line. Penalization of high yaw acceleration, longitudinal jerk, and beta angle. Slight relaxation of g-g-v and control constraints, and penalization on maximum speed along the centreline. The higher cycle time and mesh density was introduced to ensure good solution resolution. A delay of 100 ms was added to the lateral controller to compensate the one found on the vehicle. The rest of the delay was already accounted in the estimated steering dynamics.

The penalizations on longitudinal jerk, yaw acceleration, beta angle, and maximum speed along the centreline are referred as "safety margins". They will be gradually relaxed to allow for more aggressive driving.

Table 9.1 summarizes the lap times for the different configurations.

Configuration	Lap time [s]
Autonomous – Slow	276.066
Autonomous – Moderate	179.805
Autonomous – Fast–Low	157.257
Autonomous – Fast–Mid	128.837
Autonomous – Fast–High	117.414
Simulation – Fast–High–Sim	115.686
Simulation – Fast–Max–Sim	98.237
Human (same session)	100.373

Table 9.1: Lap times recorded during the final test session. We remark that the human driver was not subject to the same safety limitations imposed on the autonomous system. He could exploit the full vehicle capabilities without being forced to stop if the safety systems engaged, he could use the entire track width, and the longitudinal and lateral actuation was not limited by the autonomous actuators. Additionally, he also partially cut corner 7.

Slow lap. The first autonomous lap was executed at low speed to confirm the correct operation of the full planning–control pipeline on the real vehicle. The run was conducted under strict speed limits and large safety margins, with a resulting lap time of 276.066 s. Despite the conservative conditions, this test validated the complete communication chain between the planner, controller, and vehicle, and confirmed that the system could maintain stable autonomous motion from start to finish. The recorded telemetry is shown in Figure 9.1.

Moderate lap. The second run raised the virtual speed limits while maintaining wide safety margins. This configuration served to verify that the system could handle higher velocities and stronger dynamic transients without compromising stability. The lap time dropped to 179.805 s, demonstrating that the framework could autonomously exploit a greater portion of the vehicle's available performance envelope. The recorded telemetry is shown in Figure 9.2.

Fast–Low lap. For the third lap, the speed limits were increased further, and the safety margins were slightly relaxed to allow for more aggressive driving. The lap was completed in 157.257 s, marking the first attempt at driving near the onset of the vehicle's dynamic constraints. The overall behaviour remained stable and repeatable. This run marked

the transition from verification-oriented testing to performance-oriented operation. The recorded telemetry is shown in Figure 9.3.

Fast–Mid lap. The fourth run continued the progressive relaxation of limits. The vehicle completed the lap in 128.837 s, showing clear signs of more time-optimal behaviour. The trajectory remained smooth, and the system executed the planned trajectories consistently. The recorded telemetry is shown in Figure 9.4.

Fast–High lap. The final autonomous lap pushed the configuration to the edge of what could be safely achieved with the available vehicle hardware and safety systems. The speed limits were raised to the maximum allowed by the track conditions, and the planner operated close to the boundaries of the conservative corridor. To prevent undesired disengagements of the vehicle's protection systems, two early brake points were introduced in the most critical sections (corners 2 and 7). Despite these precautions, the lap achieved a time of 117.414 s, representing the fastest autonomous performance recorded. The recorded telemetry is shown in Figure 9.5.

Simulation reproduction. To assess the correspondence between simulated and real behaviour, the same configuration used in the Fast–High run was reproduced in simulation (Fast–High–Sim). The resulting lap time of 115.686 s closely matches the real-world measurement. The telemetry overlays show strong agreement in yaw rate and lateral acceleration, while differences are observed in longitudinal velocity, longitudinal acceleration, and sideslip angle β. These discrepancies are mainly attributed to the limited performance of the vehicle's built-in longitudinal controller, which operated as a black box and required enforcing conservative braking points for safety. Consequently, the vehicle could not reach the same combined acceleration levels achieved in simulation. In addition, the *Neural Vehicle Model* (NVM) was identified under dry conditions, whereas the test track was partially wet, which may have further reduced available grip. Overall, the comparison provides further evidence of the accuracy of the validated NVM and of the consistency between simulated and experimental results. The comparison of the two laps is shown in Figure 9.6.

Human-driven reference. Immediately after the autonomous tests, a human driver completed a reference lap under identical track and environmental conditions, achieving a lap time of 100.373 s. This performance advantage is primarily due to the operational constraints applied to the autonomous system. The human driver could approach the true track limits without the conservative padding, was unaffected by the activation of safety systems, and wasn't limited by actuation limits. Although the driver attempted to follow the autonomous racing line—visibly marked on the track by tyre traces left during the autonomous runs, as can be seen in Figure 9.9—he also cut corner 7, gaining additional time. When the same safety limitations are removed in simulation, ARD achieves 98.237 s (Fast–Max–Sim), indicating that the system could approach or beat human performance if more time to fix the badly aligned track margins and identify the longitudinal controller delay was available. The comparison of the two laps is shown in Figure 9.7.

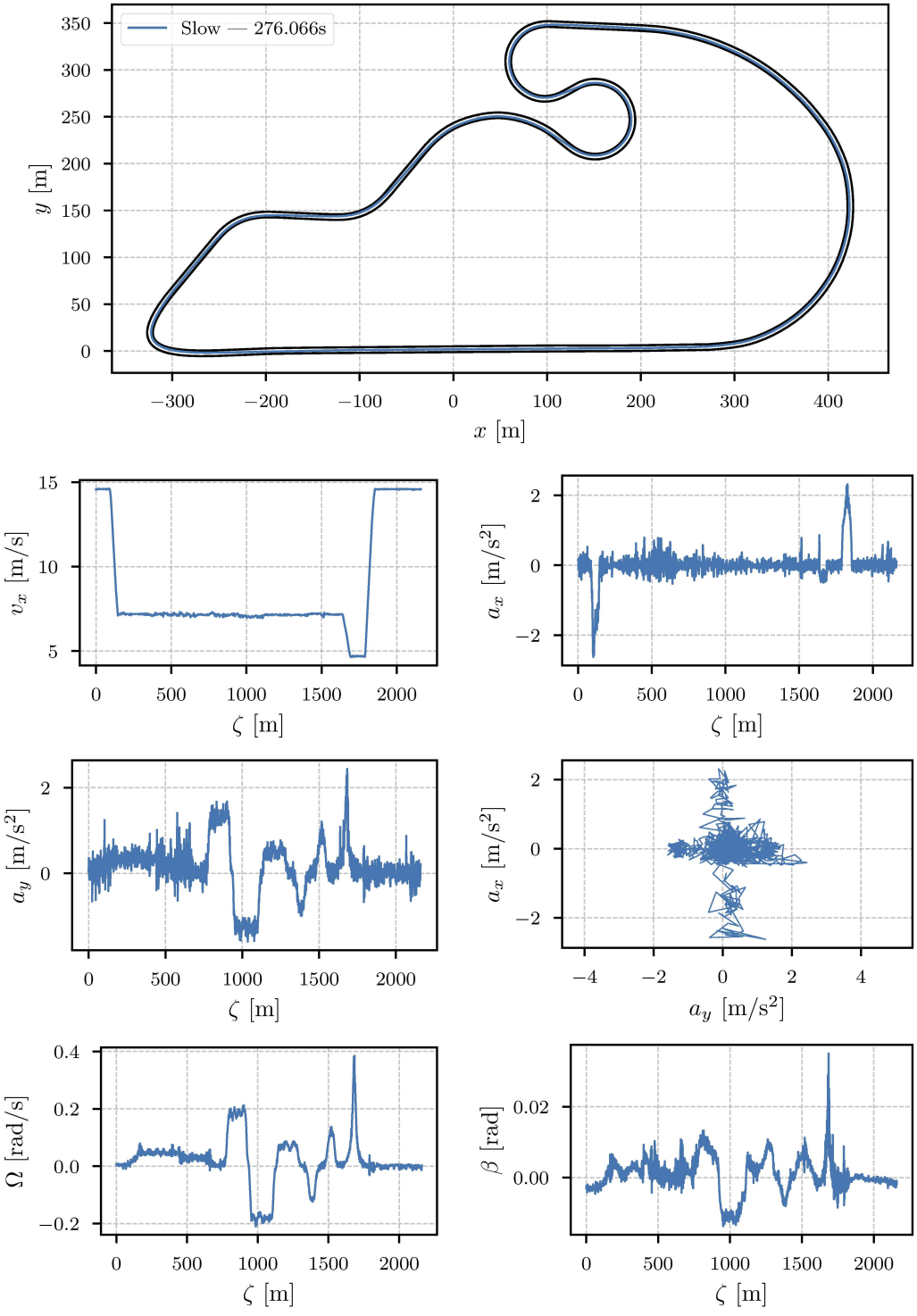


Figure 9.1: Real-world telemetry for the eCrafter on *Universität der Bundeswehr München* (UniBW): Slow. At the top, the racetrack with the trajectory, and the legend with the solution name and lap time. Below it, the longitudinal velocity v_x, the longitudinal acceleration a_x, the lateral acceleration a_y, the g-g diagram, the yaw rate Ω, and finally the side slip angle β. All plots, except for the trajectory, are plotted against the curvilinear abscissa ζ.

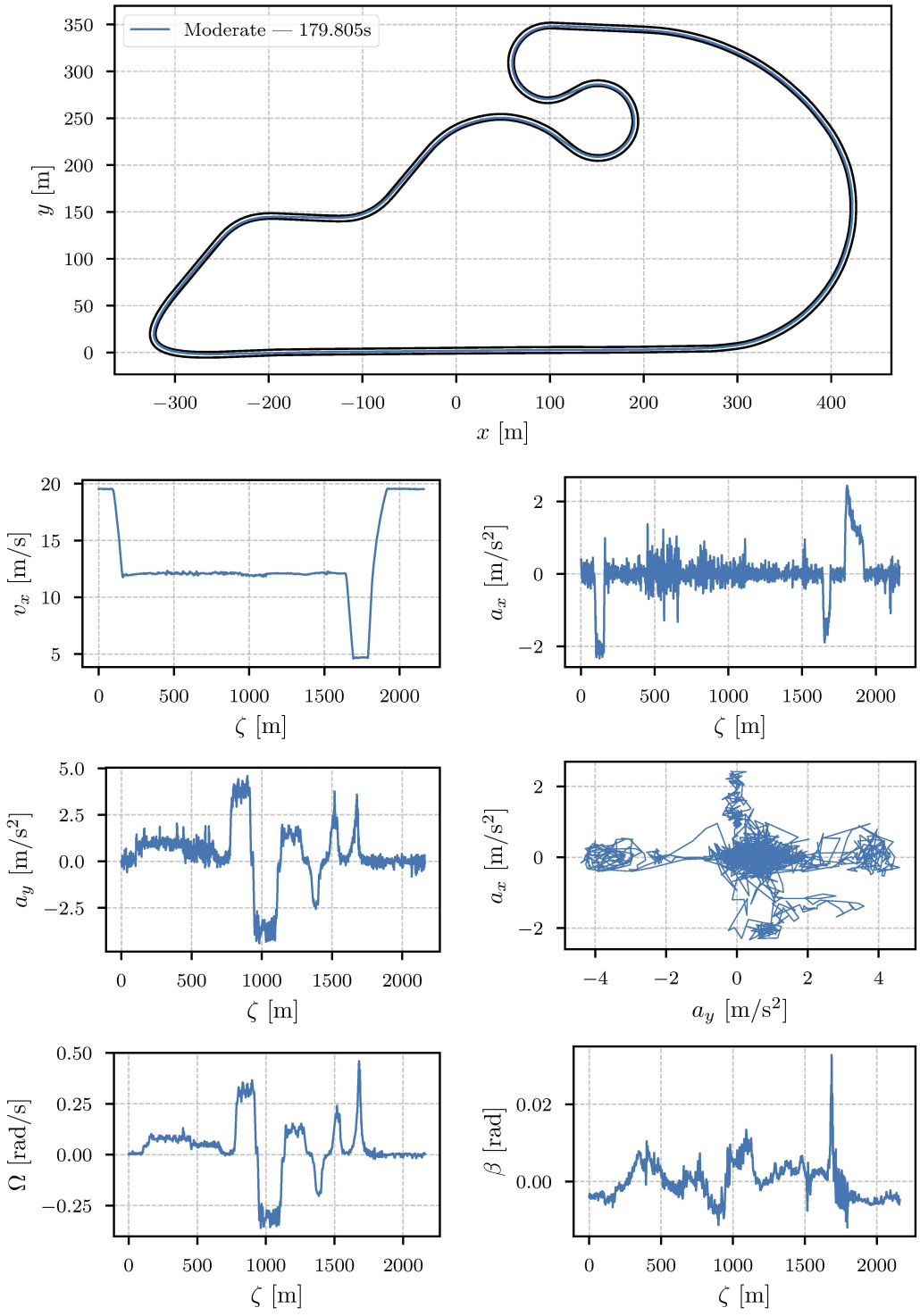


Figure 9.2: Real-world telemetry for the eCrafter on UniBW: Moderate. At the top, the racetrack with the trajectory, and the legend with the solution name and lap time. Below it, the longitudinal velocity v_x, the longitudinal acceleration a_x, the lateral acceleration a_y, the g-g diagram, the yaw rate Ω, and finally the side slip angle β. All plots, except for the trajectory, are plotted against the curvilinear abscissa ζ.

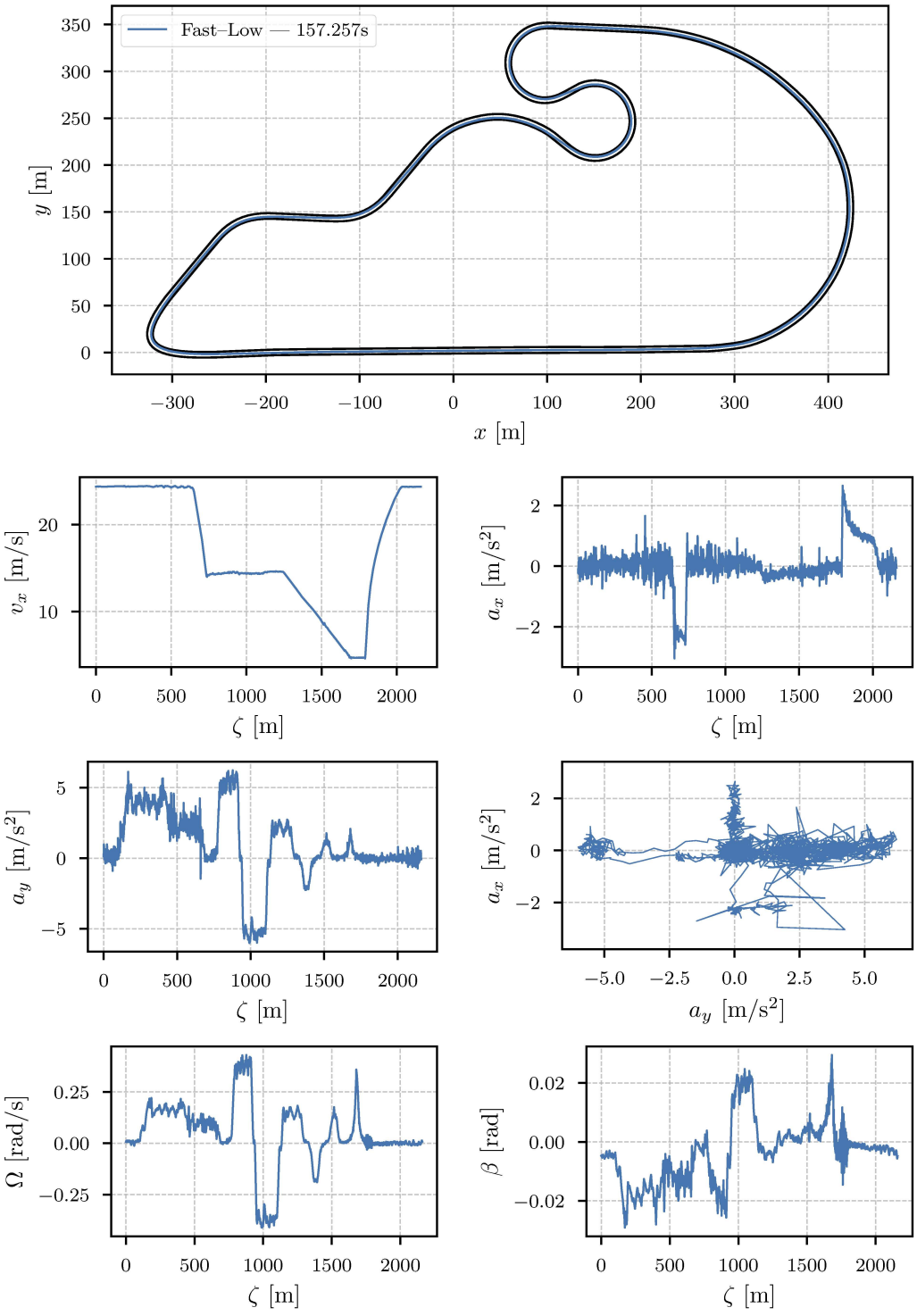


Figure 9.3: Real-world telemetry for the eCrafter on UniBW: Fast–Low. At the top, the racetrack with the trajectory, and the legend with the solution name and lap time. Below it, the longitudinal velocity v_x, the longitudinal acceleration a_x, the lateral acceleration a_y, the g-g diagram, the yaw rate Ω, and finally the side slip angle β. All plots, except for the trajectory, are plotted against the curvilinear abscissa ζ.

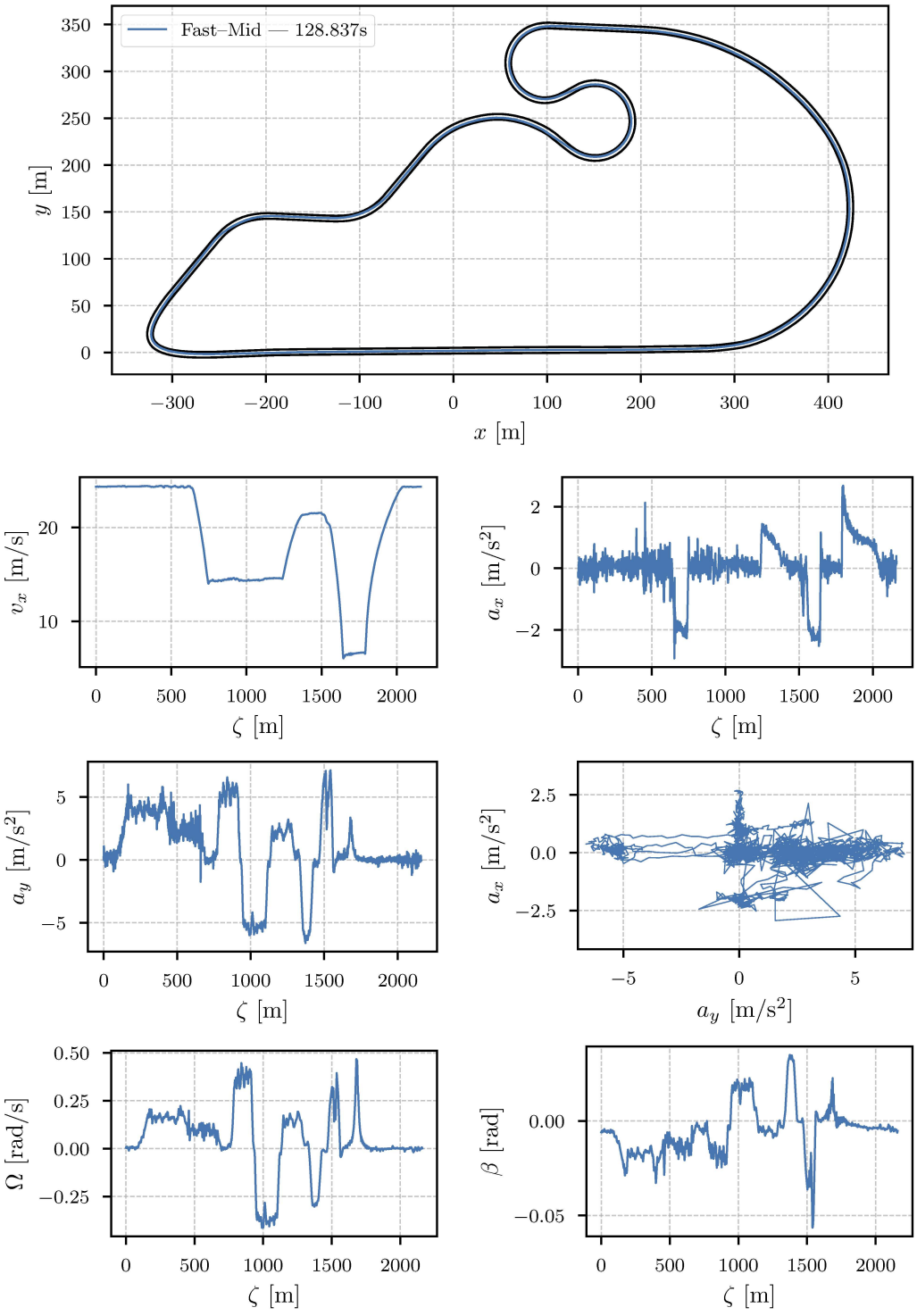


Figure 9.4: Real-world telemetry for the eCrafter on UniBW: Fast–Mid. At the top, the racetrack with the trajectory, and the legend with the solution name and lap time. Below it, the longitudinal velocity v_x, the longitudinal acceleration a_x, the lateral acceleration a_y, the g-g diagram, the yaw rate Ω, and finally the side slip angle β. All plots, except for the trajectory, are plotted against the curvilinear abscissa ζ.

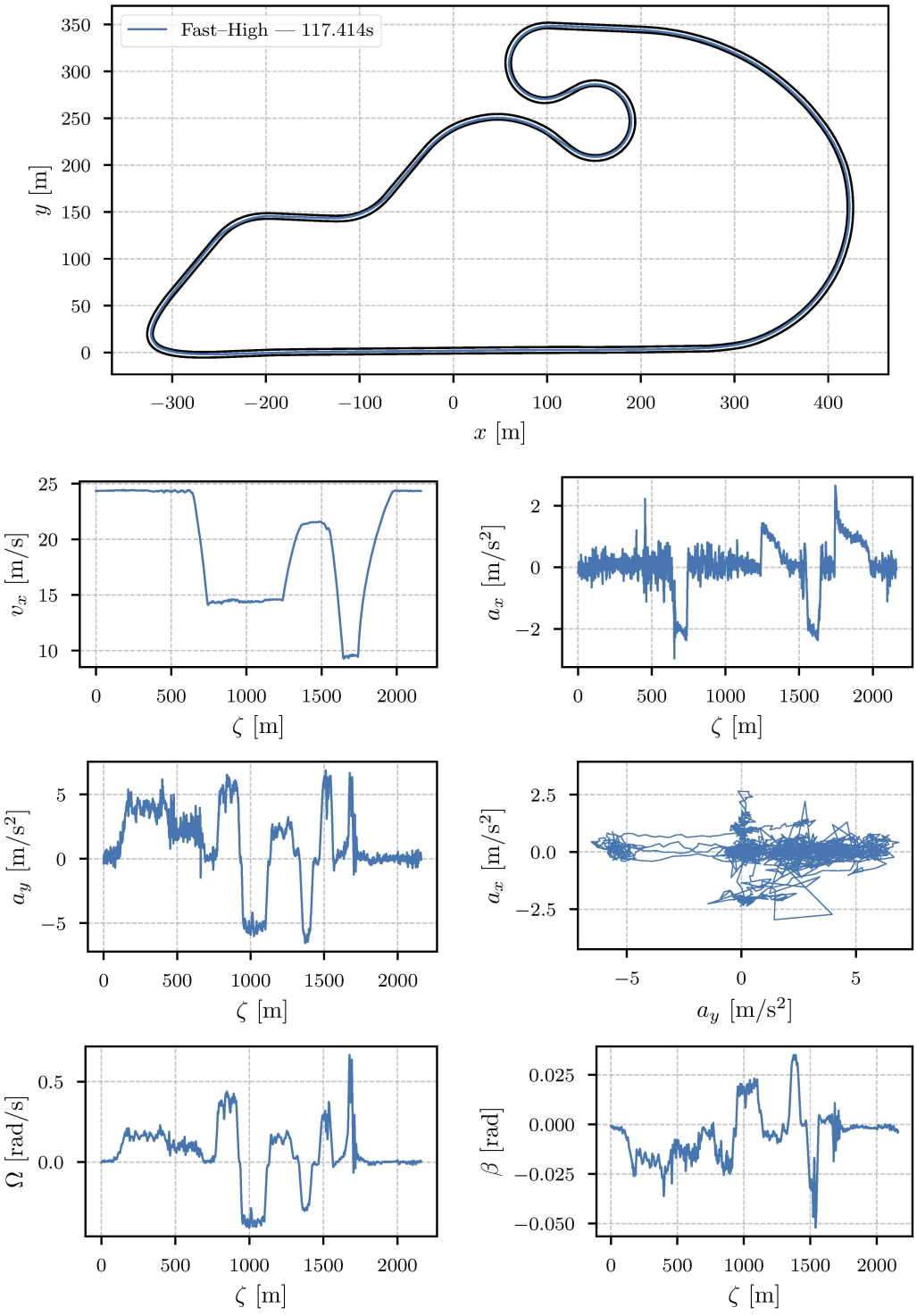


Figure 9.5: Real-world telemetry for the eCrafter on UniBW: Fast–High. At the top, the racetrack with the trajectory, and the legend with the solution name and lap time. Below it, the longitudinal velocity v_x, the longitudinal acceleration a_x, the lateral acceleration a_y, the g-g diagram, the yaw rate Ω, and finally the side slip angle β. All plots, except for the trajectory, are plotted against the curvilinear abscissa ζ.

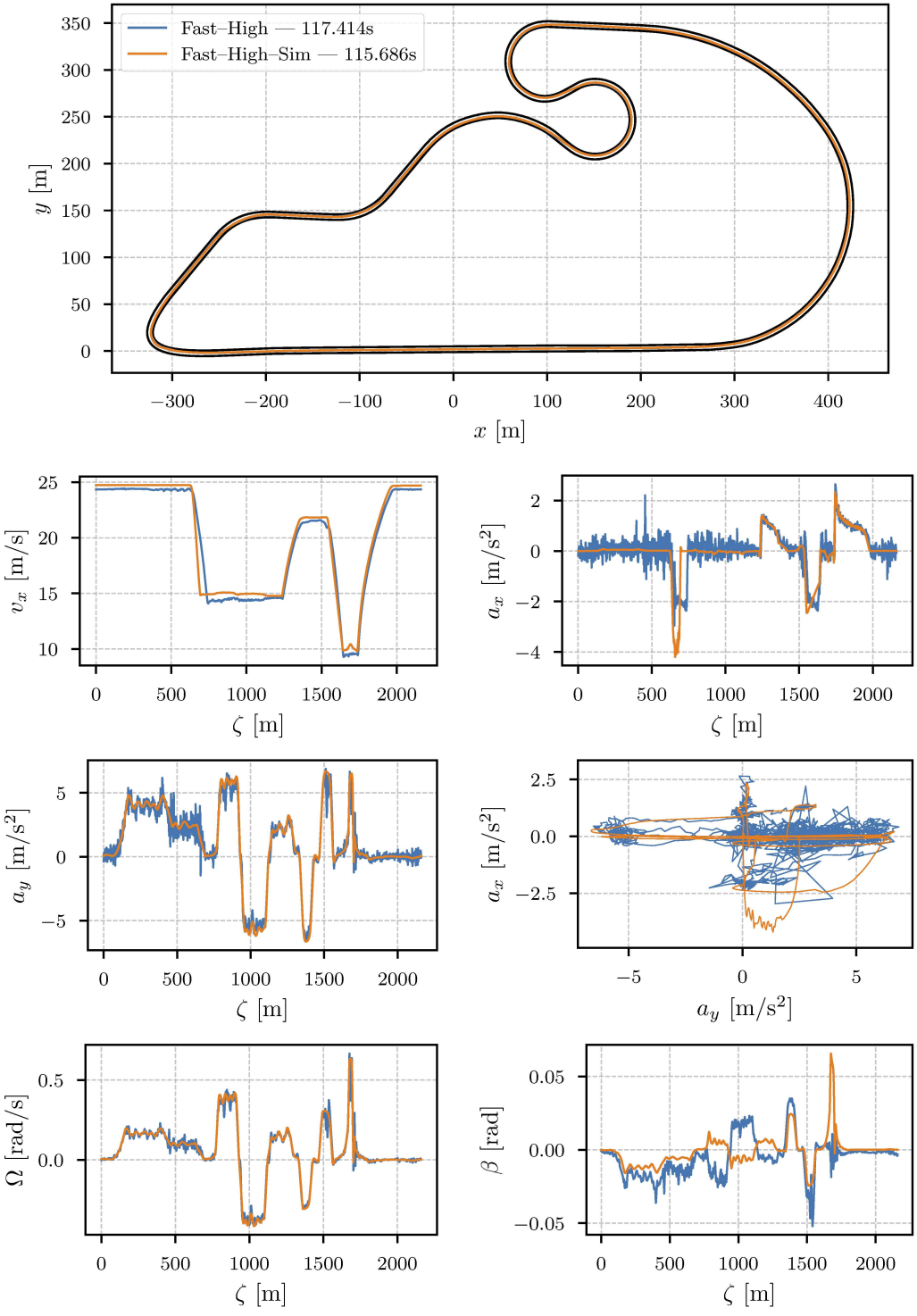


Figure 9.6: Real-world telemetry for the eCrafter on UniBW: Fast–High vs Fast–High–Sim. At the top, the racetrack with the trajectories, and the legend with the solution names and lap times. Below it, the longitudinal velocity v_x, the longitudinal acceleration a_x, the lateral acceleration a_y, the g-g diagram, the yaw rate Ω, and finally the side slip angle β. All plots, except for the trajectories, are plotted against the curvilinear abscissa ζ.

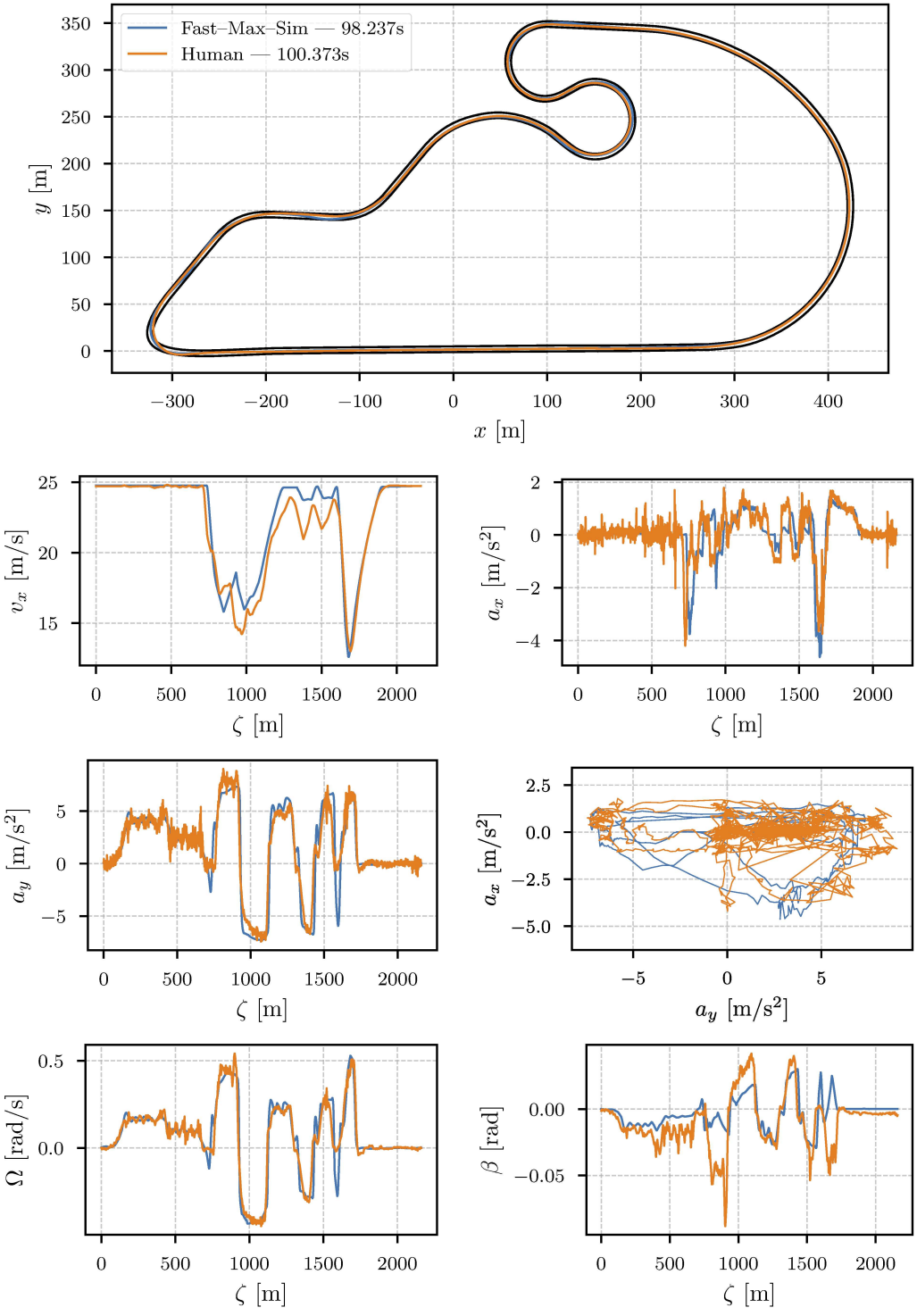


Figure 9.7: Real-world telemetry for the eCrafter on UniBW: Fast–Max–Sim vs Human. At the top, the racetrack with the trajectories, and the legend with the solution names and lap times. Below it, the longitudinal velocity v_x, the longitudinal acceleration a_x, the lateral acceleration a_y, the g-g diagram, the yaw rate Ω, and finally the side slip angle β. All plots, except for the trajectories, are plotted against the curvilinear abscissa ζ.

9.3 Discussion

To assess the consistency of the autonomous operation, the average absolute lateral deviation ($|\overline{\Delta n}|$) of each autonomous lap was computed with respect to the Fast–High trajectory. The resulting profiles, shown in Figure 9.8, confirm a high degree of repeatability across runs. Even the slower configurations follow nearly identical paths, which is expected given the conservative track padding applied for safety. Only local differences appear in corners 5 and 6, where as the speed increased the planned trajectories changed considerably. Overall, the trajectories demonstrate that ARD can reproduce complex manoeuvres with remarkable consistency under real-world conditions. Figure 9.9 further illustrates this repeatability, showing the tyre marks left on the track after multiple laps, which closely overlap along most of the circuit.

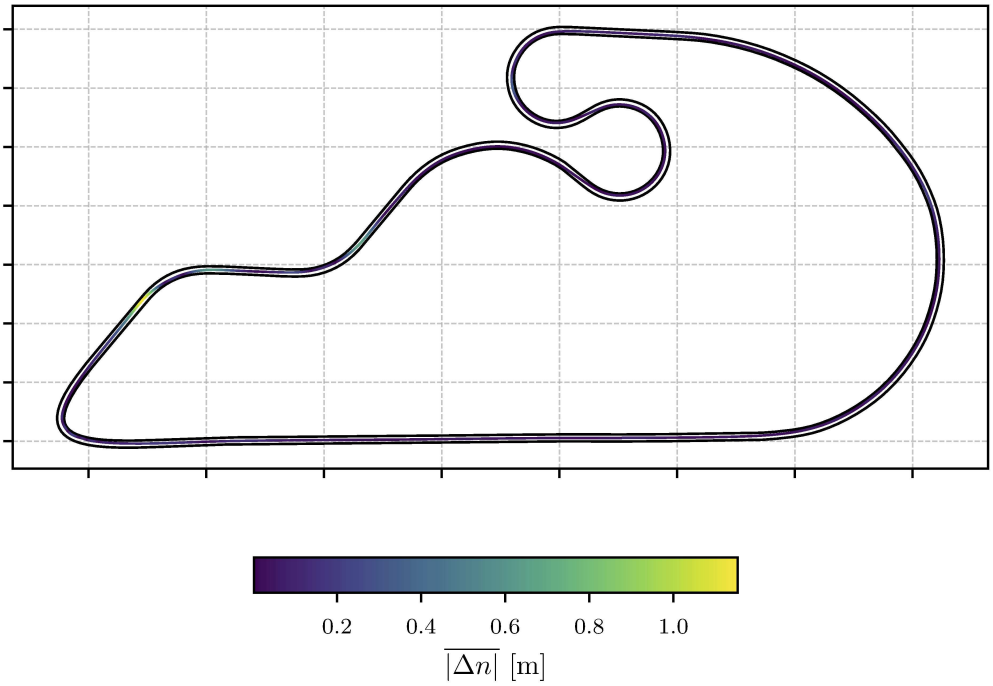


Figure 9.8: Average absolute lateral deviation of each autonomous lap computed with respect to the Fast–High trajectory. This confirms a high degree of repeatability across runs.

Beyond validating the framework's capabilities, the on-track campaign provided several important lessons for future developments. First, modelling the road for high-performance driving requires an accurate representation of the full three-dimensional geometry, including local surface roughness. Even moderate imperfections, such as the discrete road blocks of the UniBW track (Figure 9.9), can influence vehicle dynamics, especially at high speed, and should be incorporated in the simulation environment. Methods like the GripMap presented in [180] could be used to learn these effects online without explicit modelling.

Second, while the kineto-dynamical vehicle model proved capable of planning and executing realistic trajectories, its accuracy could be further enhanced by including a more detailed representations of the actuation systems and load transfers. Embedding these effects would improve planning on uneven surfaces and in situations where the vehicle approaches understeer or oversteer limits. Using the NVM directly in the E-NMPC could prove advantageous, although convergence may be an issue.

Third, both tyres and brakes experienced a noticeable increase in temperature during the tests, yet ARD maintained stable operation and consistent behaviour. Although

Figure 9.9: Tyre marks (top image) left on the track after multiple laps, which closely overlap along most of the circuit. Discrete road blocks (bottom image) which composed part of the track tarmac.

thermal effects were not modelled, incorporating them would significantly improve prediction accuracy over extended runs or under repeated high-intensity manoeuvres.

Finally, the campaign highlighted the importance of using vehicle platforms with sufficiently responsive actuation systems for autonomous racing. The production-grade actuation of the eCrafter imposed non-negligible delays and rate limits that restricted performance and complicated controller tuning. Nevertheless, achieving reliable autonomous operation under such conditions is a valuable outcome, as similar limitations are common in urban and industrial vehicles, where safety and comfort constraints dominate.

Overall, these experiments demonstrate that ARD can perform complete autonomous laps on a real vehicle with high consistency, while also revealing the key directions in which further refinement of both modelling and hardware can go.

Part III—Early Work in Shared Control

Chapter 10

The Artificial Race Coach

Abstract

This chapter introduces the *Artificial Race Coach* (ARC), a preliminary exploration into shared control between a human driver and the *Artificial Race Driver* (ARD). Unlike conventional autonomous systems that aim to replace human input, ARC is designed to assist and guide the driver in real-time, with the goal of improving driving performance rather than overriding intent. The system is implemented in a steer-by-wire simulator environment, but reconstructs the feel of a physically coupled steering system by summing both the simulated vehicle model's feedback torque and ARC's guidance torque. A novel control allocation strategy is proposed, based on a composite of fifth-order smoothstep functions, which enables fine-grained shaping of a guidance valley in the control input space. This formulation supports asymmetric feedback, dead zones, and smooth authority transitions, yielding a natural and interpretable interaction. ARC's architecture includes a graphical user interface for visual and auditory cues, as well as a torque feedback module for lateral guidance. While pedal vibration feedback was not implemented, it is identified as a promising future direction given its passive and non-distracting nature. The system was integrated into the existing ARD stack and tested informally with a human participant, demonstrating the feasibility and intuitiveness of the approach. ARC represents an early but promising step toward real-time machine-assisted driver coaching in high-performance contexts.

Contents

10.1 Introduction 134
10.2 From Autonomous Driving to Shared Control 135
10.2.1 Overview of Shared Control Paradigms 135
10.2.2 Control Interface Assumptions 135
10.3 ARC Architecture 136
10.4 Control Allocation through Smoothstep-Based Guidance 137
10.5 Preliminary Tests and Observations 138
10.6 Discussion and Outlook 139

10.1 Introduction

While ARD was conceived as a fully autonomous system, its capabilities as a high-performance motion planner and controller open opportunities that extend beyond full automation. The knowledge and precision embedded in ARD can be leveraged not only to drive the vehicle but also to assist and guide a human driver in real-time. This chapter explores that direction through the design and preliminary implementation of ARC.

ARC marks a shift from autonomy to collaboration. Rather than replacing the human driver, it establishes a shared control framework in which both agents act concurrently on the vehicle. This is particularly relevant in racing, where optimal actions are often unintuitive, difficult to execute, and highly sensitive to timing and consistency. By blending machine-generated guidance with human intent, ARC seeks to preserve driver agency while gently steering behaviour toward optimal control strategies.

In contrast to shared control systems developed for safety-critical interventions, ARC adopts a coaching-oriented perspective. Its purpose is not to prevent failure but to help the driver learn how to perform better. The interaction is designed to be interpretable and gradual, reflecting the instructor–student dynamic often used in aviation training.

The current implementation runs in a steer-by-wire simulator environment, with particular attention to preserving the natural feel of a mechanically coupled interface. Although physical control over longitudinal inputs is unavailable, alternative feedback modalities are introduced to maintain driver awareness without breaking immersion. Central to this implementation is a tunable control allocation law that determines how assistance is blended within the human–machine interaction.

This chapter presents the ARC concept, its integration with ARD, and the implementation choices that enable real-time collaborative driving. While the system has so far been tested only informally, the results demonstrate its feasibility and motivate further research into shared control for high-performance applications.

10.2 From Autonomous Driving to Shared Control

10.2.1 Overview of Shared Control Paradigms

Shared control encompasses interaction strategies in which both human and automation simultaneously contribute to the vehicle's motion. These approaches aim to combine the strengths of human intuition and situational awareness with the precision and consistency of automated control. Unlike fully autonomous operation or conventional driver assistance, shared control keeps the human in the loop, often with dynamically varying levels of authority.

Several metaphors have been proposed to conceptualize this interaction. Among them, the instructor–student paradigm is particularly relevant to ARC. Here, the automation acts as an experienced guide, assisting in the execution of manoeuvres while encouraging the human driver to learn and adapt. Depending on the context and the degree of disagreement between driver and system, the interaction may range from passive resistance to active correction.

Implementation-wise, shared control systems can be broadly classified as either *coupled* or *decoupled*. In coupled systems, both agents apply forces to the same physical interface (*e.g.,* the steering wheel), allowing direct, bidirectional haptic interaction. In decoupled systems, such as steer-by-wire configurations, driver and automation commands are fused algorithmically without a mechanical connection. Although ARC operates in a decoupled simulator setup, it was explicitly designed to reproduce the tactile qualities of a coupled interface through torque feedback.

A central element of any shared control system is the allocation of authority. In ARC, this allocation is governed by a tunable nonlinear function that continuously modulates assistance according to the divergence between driver input and optimal control action. This smooth feedback-driven approach enables interpretable and progressive interaction while avoiding abrupt takeovers or fixed blending ratios.

10.2.2 Control Interface Assumptions

The design of ARC assumes a steer-by-wire architecture, consistent with the configuration of our driving simulator. This setup allows independent torque commands to be applied to the steering wheel, fully decoupled from the mechanical steering mechanism. Although technically a decoupled system, ARC is deliberately designed to emulate the sensation of a mechanically coupled interface—an essential aspect of high-performance driving.

In racing, the driver's ability to feel the car's dynamic response through the steering wheel is critical. A purely decoupled control loop, in which the vehicle steering feedback is filtered, delayed, or removed, risks breaking this vital channel of communication. To mitigate this, ARC injects its guidance torque into the same feedback loop as the simulated vehicle model. The resulting steering torque, equal to the sum of ARC's feedback and the vehicle's physical response, effectively recreates the feel of a coupled system. This ensures that the driver remains engaged with the vehicle dynamics while receiving intuitive guidance.

For longitudinal control, the simulator does not support actuation of the throttle or brake pedals. Consequently, ARC provides feedback in this domain through visual and auditory cues instead of physical actuation. Although these modalities convey information effectively, they can be cognitively demanding and potentially distracting, particularly in a racing context. A promising future extension is the use of haptic pedal feedback—such as vibration motors—which could deliver passive and unobtrusive guidance in line with ARC's coaching philosophy.

10.3 ARC Architecture

ARC operates as a shared-control layer built on top of the existing ARD stack. The core ARD components—including the high-level planner, low-level controller, and vehicle model—remain unchanged. To ensure that ARD generates valid trajectories regardless of the driver's skill or consistency, the initial state used for planning is constrained to match the current vehicle state exactly. This boundary condition, applied at each planning cycle, enables ARD to replan trajectories that remain feasible from the driver's current pose and velocity. As a result, the guidance remains valid even when the driver deviates significantly from ideal conditions. Figure 10.1 shows the conceptual overview of ARC's architecture.

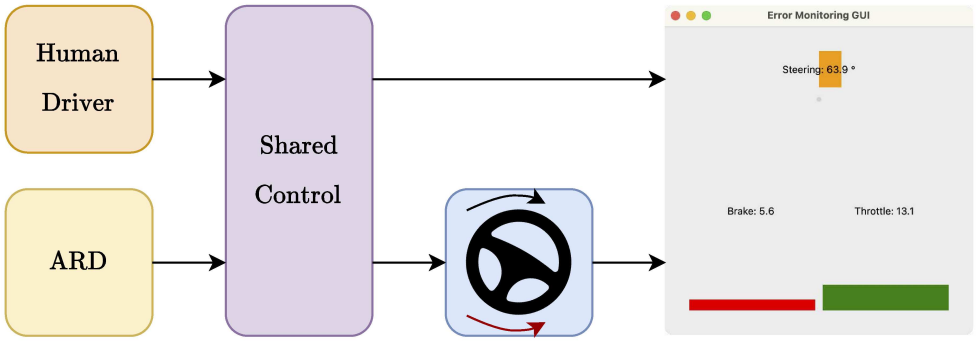


Figure 10.1: Conceptual overview of ARC within the ARD framework. Both ARD and the human driver provide inputs to a shared-control block governed by a smoothstep-based allocation law. The resulting commands actuate the steering wheel, while an error-monitoring interface delivers visual and auditory feedback to the driver.

ARC runs in real-time, operating concurrently with the human driver. At each control step, inputs from both the driver and ARD are compared. The difference is used to compute a guidance signal that informs the driver how to align with the optimal trajectory. Unlike traditional applications of ARD, which rely on emulated driver interfaces, ARC employs dedicated agent interfaces to inject its low-level commands directly into the simulator's control and feedback subsystems. This enables precise, low-latency interaction between ARD and the vehicle model while bypassing unnecessary abstractions.

Feedback is delivered through two primary channels: haptic and visual. For lateral control, the computed feedback torque is applied directly to the steering wheel via the simulator's force-feedback interface. The total torque felt by the driver is the sum of the vehicle's dynamic response and ARC's corrective torque, recreating the sensation of a mechanically coupled steering system while retaining the flexibility of steer-by-wire.

In parallel, a GUI is displayed in front of the driver, mounted similarly to a cockpit dashboard. It visualizes the discrepancies between the driver's inputs and the corresponding ARD commands. Since lateral feedback is already conveyed haptically, the GUI focuses primarily on longitudinal control, displaying throttle and brake commands prominently. These cues help the driver understand the magnitude of the desired longitudinal actuation.

An auditory cue system complements the visual feedback. The sound's volume and pitch reflect the current mismatch in longitudinal input, reinforcing the GUI information

without requiring constant visual attention. While the visual and auditory channels are functional, they are cognitively demanding and considered temporary solutions. Future iterations will focus on haptic feedback at the pedals, such as vibration motors, offering passive and unobtrusive guidance consistent with ARC's coaching philosophy.

Together, these components form a closed-loop system that continuously monitors, compares, and feeds back driver input relative to an optimal baseline—all operating in real-time as the human navigates the circuit.

10.4 Control Allocation through Smoothstep-Based Guidance

The allocation of control authority between the driver and the automation is central to any shared control system. In ARC, this allocation is governed by a fully parametric, piecewise-smooth function constructed by stitching together multiple fifth-order polynomial segments. This formulation enables the shaping of a nonlinear guidance valley in the input space, mapping the difference between the driver's and ARD's control commands to a feedback strength coefficient α.

Each smoothstep segment is defined as a fifth-order polynomial of the form

$$s(x) = ax^5 + bx^4 + cx^3 + dx^2 + ex + f,$$

where the coefficients a, b, c, d, e, f are determined by boundary conditions on the function value, first derivative, and second derivative at both ends of the interval. Specifically, the following constraints are imposed at points x_0 and x_1:

$$\begin{aligned} s(x_0) &= y_0, & s(x_1) &= y_1, \\ s'(x_0) &= 0, & s'(x_1) &= 0, \\ s''(x_0) &= 0, & s''(x_1) &= 0. \end{aligned}$$

These constraints ensure that each segment is flat and with zero curvature at its endpoints, allowing multiple segments to be joined without discontinuities.

A complete guidance function is constructed by chaining four such segments across a set of threshold points $x_0, x_1, \ldots, x_7$, with corresponding output levels $y_0, y_1, \ldots, y_4$. The resulting allocation function $\alpha(x)$ is defined as

$$\alpha(x) = \begin{cases} y_0, & x < x_0, \\ s_1(x), & x_0 \le x \le x_1, \\ y_1, & x_1 < x < x_2, \\ s_2(x), & x_2 \le x \le x_3, \\ y_2, & x_3 < x < x_4, \\ s_3(x), & x_4 \le x \le x_5, \\ y_3, & x_5 < x < x_6, \\ s_4(x), & x_6 \le x \le x_7, \\ y_4, & x > x_7, \end{cases}$$

where each $s_i(x)$ is a fifth-order polynomial constructed using the boundary constraints previously discussed. Figure 10.2 shows an example of this smoothstep-based allocation function.

This formulation provides full control over the shape of the allocation profile. Asymmetries, dead zones, saturation levels, and slope transitions can all be specified directly through the choice of (x_i, y_i) points. The function thus enables subtle, progressive guidance when driver and automation are aligned, and stronger corrective feedback when their inputs diverge.

In the lateral domain, this allocation value directly modulates the torque applied to the steering wheel. The total steering torque τ_{tot} felt by the driver is computed as

$$\tau_{\text{tot}} = \tau_{\text{veh}} + \alpha(x)\,\tau_{\max},$$

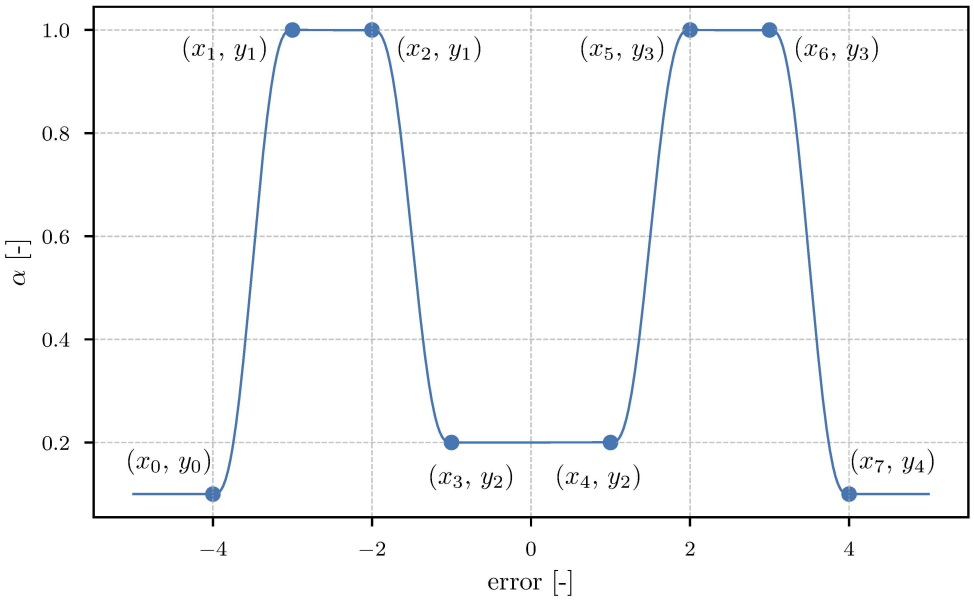


Figure 10.2: Example of a smoothstep-based allocation function constructed by joining four fifth-order smoothstep functions. Each segment is fully tunable, enabling asymmetric guidance profiles and dead zones. Given an input error, for example the error in steering angle, the function outputs the allocation value α to give to the corrective feedback.

where τ_{veh} is the steering torque generated by the vehicle dynamics, $\tau_{\max}$ is the maximum corrective torque ARC is allowed to apply, and $\alpha(x)$ is the allocation coefficient derived from the smoothstep-based guidance function. This additive formulation preserves the physical feel of the vehicle response while superimposing an interpretable, progressive assistance torque.

For longitudinal inputs, the same allocation principle could be utilized to compute the desired corrective effort, but it is not used currently due to the inability to act on the pedals directly. Instead, the mismatch is conveyed directly to the driver through the GUI and auditory cues, as described in Section 10.3.

10.5 Preliminary Tests and Observations

To validate the basic functionality and interaction quality of the ARC system, an informal test session was conducted with a single human subject in the driving simulator. The participant was already familiar with both the simulator environment and the vehicle model. The session began with several warm-up laps, both with and without ARC, to allow the driver to acclimate to the shared-control behaviour and to the baseline manual driving conditions.

After familiarization, two laps were recorded for comparison: one manually driven without ARC (pure human control) and one under shared control with ARC active. The results, shown in Figure 10.3, indicate a qualitative improvement in lap performance with the assistance active. The lap time decreased by 3.438 s, primarily due to higher average speeds and better braking and throttle application at corner entry and exit. Given the preliminary nature of the test and the limited sample size, these results should be interpreted qualitatively.

The driver reported that the lateral guidance provided by the steering torque was helpful in understanding the intended cornering line, while the longitudinal cues improved awareness of the desired braking and acceleration points, as well as overall sustained throttle actions. However, the visual and auditory feedback for throttle and brake were found to be distracting when precise control was required. The subject noted that a

haptic feedback channel—such as pedal vibration or direct pedal actuation—would likely provide clearer and less intrusive longitudinal guidance, analogous to the steering torque feedback.

Overall, this preliminary test confirmed that the ARC system can operate in closed loop with a human driver, providing intuitive and effective guidance. While no statistically significant conclusions can yet be drawn regarding performance gains or learning effects, the session demonstrated the system's feasibility and highlighted promising directions for more structured user studies and hardware-in-the-loop experiments.

10.6 Discussion and Outlook

Although ARC was conceived in the context of autonomous racing, its design principles extend beyond high-performance driving. The coaching-oriented shared control paradigm explored in this chapter offers a compelling alternative to conventional driver-assistance systems, particularly during the transition toward higher levels of autonomy.

Contemporary level 2 and level 3 systems typically operate autonomously for extended periods [181], requiring only passive driver supervision. In practice, however, such passivity often leads to disengagement and delayed reactions when the system requests intervention. As noted in [144], human reaction times in these scenarios are typically insufficient to mitigate emerging hazards—especially when the vehicle is already operating near its limits.

ARC challenges this paradigm by maintaining continuous driver engagement. Rather than alternating control authority between the human and the automation, ARC promotes parallel guidance in which both agents remain active at all times. Through structured and interpretable feedback, the driver stays within the control loop while being supported in learning how to respond to dynamic situations more effectively.

The racing environment provides a demanding testbed for this concept. It exposes the system to extreme conditions—high accelerations, rapid transitions, and at-limit vehicle dynamics—that are difficult to reproduce in conventional driving. Developing shared control strategies under such conditions pushes the limits of responsiveness, interpretability, and driver–automation cooperation. Insights gained in this domain may transfer to safety-critical applications such as emergency collision avoidance in urban traffic, where decisive and cooperative action is essential.

At a broader level, ARC also addresses questions about the role of automation in skill acquisition. By supporting rather than overriding the human driver, it enables real-time coaching, personalized adaptation, and a more gradual path toward autonomy. This approach aligns with applications where trust, transparency, and human performance remain fundamental. Moreover, the learning aspect does not only apply to driving faster, but may also lead to fuel-efficient shared driving strategies.

Future work should address several current limitations: the absence of physical feedback in the longitudinal channel, the lack of formal evaluation across multiple participants, and the need for structured measures of performance and learning. Nevertheless, ARC demonstrates the feasibility of a real-time, coaching-oriented shared control framework and provides a promising foundation for continued research into collaborative driving interfaces.

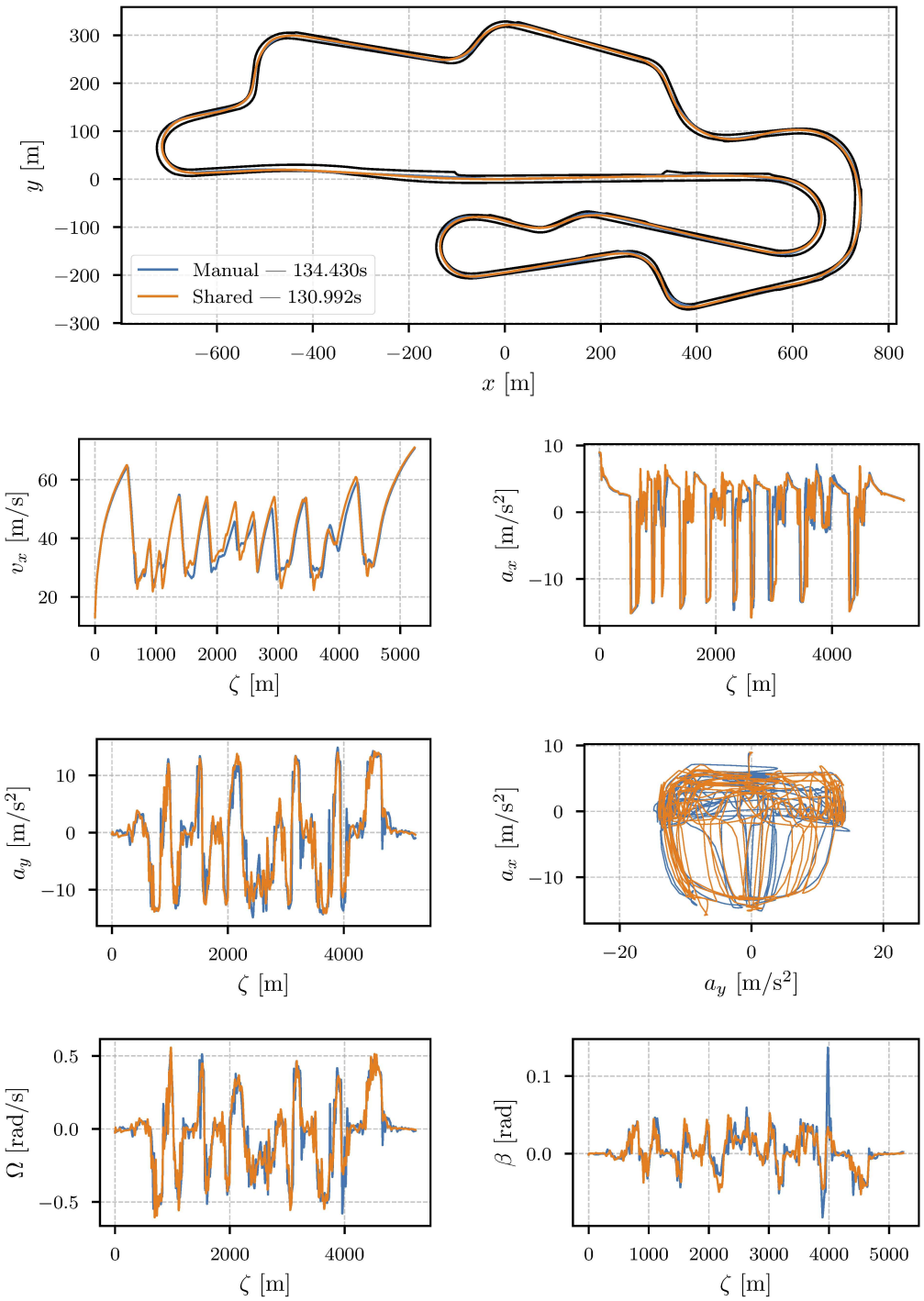


Figure 10.3: Comparison between a human-driven lap with and without shared control via ARC on the Mugello circuit with the high-fidelity double track model and the old identification procedure. At the top, the racetrack with the trajectories along with the legend and the lap times. Below it, the longitudinal velocity v_x, the longitudinal acceleration a_x, the lateral acceleration a_y, the g-g diagram, the yaw rate Ω, and finally the side slip angle β. All plots, except for the trajectories, are plotted against the curvilinear abscissa ζ.

Conclusions

Chapter 11

Conclusions

Contents

11.1 Summary of Contributions 142
11.2 Reflections on Autonomous Racing and Human-In-The-Loop Systems 143
11.3 Future Research Directions 144

11.1 Summary of Contributions

This thesis presented the development and evaluation of the *Artificial Race Driver* (ARD), a fully modular framework for autonomous racing. ARD is designed to operate seamlessly in both simulation and real-world environments, supporting a variety of vehicles and deployment conditions.

The framework is structured into three main components: a high-level motion planner, a low-level controller, and a vehicle model (or physical platform). Each component was engineered for real-time operation and rapid adaptation, while maintaining a clear separation of concerns and ease of integration.

The main contributions of this thesis are summarised below.

- **Design and Implementation of ARD.** A complete autonomous racing framework was developed, encompassing all required modules for motion planning, control, and simulation. The architecture was conceived to maximise modularity and portability, allowing the same software stack to operate across vehicles, tracks, and experimental conditions with minimal reconfiguration. This unified design represents a significant step towards reproducible, cross-platform research in autonomous racing.
- **Motion Planning via *Economic Nonlinear Model Predictive Control* (E-NMPC).** The high-level planner was formulated as an E-NMPC problem specifically tailored for racing. It integrates a novel g-g-v constraint formulation to guarantee dynamic feasibility while preserving computational efficiency. The planner can generate minimum-time trajectories in real time and was validated on multiple vehicles and circuits.
- **Physics-Informed Neural Lateral Control.** A novel *Physics-Informed Steering Neural Network* (PhS-NN) was introduced for lateral control. This controller learns a structured inverse model of the vehicle's dynamics, embedding physical priors such as quasi steady-state relationships while learning the transient dynamics from data. This approach combines interpretability with strong generalisation across operating conditions.
- **Neural Single-Track Vehicle Model.** A lightweight neural vehicle model was developed as a trainable and differentiable alternative to classical physics-based formulations. Despite retaining the compact single-track-like structure, the proposed model captures highly nonlinear effects such as tyre saturation, load transfer, and differential behaviour without increasing the number of states. It matches the predictive accuracy of a high-fidelity double-track model while drastically reducing the effort required for identification. Furthermore, it can be fine-tuned both offline and online, enabling continuous adaptation to changing conditions such as tyre wear or temperature variations.
- **Two-Lap Learning Strategy.** A semi-automated identification pipeline was developed to extract key vehicle and controller parameters from minimal human-driven data. Using only two laps of telemetry, this strategy enables rapid deployment of ARD on

new platforms and tracks, offering a practical path towards self-adaptive autonomous racing systems.

- **Simulation and Real-World Validation.** The complete ARD stack was validated through extensive experiments in both simulation and on-track testing. Multiple vehicle configurations and racing circuits were employed, including real-world deployment on the *Universität der Bundeswehr München* (UniBW) test track with an electric Volkswagen eCrafter. Additionally, simulation tests were performed to asses the system performance under non-ideal conditions, such as dropped messages and measurement noise. The results demonstrate that ARD can drive close to the handling limits—even in non-ideal conditions—and the consistency between simulated and real-world performance, confirming the accuracy of the validated *Neural Vehicle Model* (NVM) and the capabilities of ARD.
- **Exploration of Shared Control.** Preliminary work on the *Artificial Race Coach* (ARC) extended the ARD framework towards human–machine collaboration. The system implements steer-by-wire shared control and multimodal haptic and visual feedback, demonstrating interesting preliminary results in cooperative manoeuvre execution.

11.2 Reflections on Autonomous Racing and Human-In-The-Loop Systems

The research presented in this thesis demonstrates how autonomous racing can serve as a valuable testing ground for the development of high-performance, real-time planning and control systems. Racing poses stringent constraints on dynamic feasibility, reaction time, and control authority, pushing algorithms and architectures to their limits. As such, it offers a unique opportunity to explore and validate strategies that may later transfer to more general autonomous driving applications.

In developing ARD, the focus was not solely on performance, but also on adaptability and deployability. These aspects become increasingly relevant when considering real-world scenarios, where exact models are unavailable, sensor noise is present, and vehicle characteristics may vary significantly. The successful deployment of ARD across simulated and real platforms, with minimal telemetry and minimal model knowledge, highlights the practical relevance of the framework beyond the racing domain.

At the same time, this work also emphasized the importance of keeping the human in the loop. While full autonomy has been the long-standing goal in many domains, current automotive systems still rely on frequent handovers and partial driver supervision. These transitional modes, often observed in so-called level 2 and level 3 systems, are known to suffer from reduced situational awareness and delayed reaction times on the part of the human driver.

ARC, although still in its early stages, was conceived to address this issue. Instead of relegating the human to a passive monitoring role, ARC promotes continuous engagement through shared control. By guiding the driver with intuitive haptic and visual cues, ARC preserves human agency while leveraging autonomous guidance. Preliminary tests suggest that such an approach could foster better collaboration, improve safety, and reduce cognitive load in demanding driving situations.

In hindsight, the coexistence of ARD and ARC within this thesis reflects a broader shift in the field of autonomy. Rather than pursuing autonomy as an all-or-nothing property, there is growing recognition of the value of collaborative systems, where humans and machines work together, each contributing their strengths. Autonomous racing and shared control may seem like contrasting paradigms, but this work shows that they can coexist and even reinforce one another.

11.3 Future Research Directions

While the contributions of this thesis advance the state of autonomous racing and human-in-the-loop control, several research directions remain open and warrant further investigation.

- **Unified Control Architectures.** In the current ARD framework, longitudinal and lateral control were handled by distinct modules. This reflects common engineering practice but neglects the intrinsic coupling between the two. Future research should investigate unified control architectures capable of exploiting this coupling, for example through multidimensional physics-informed learning or full vehicle dynamics inversion, to achieve more precise and responsive behaviour near handling limits.
- **Real-Time Model Adaptation.** The two-lap learning strategy proved effective for rapid offline identification, but it remains a static process. Extending this approach to support continuous or online adaptation would enable ARD to maintain high performance under varying conditions such as tyre wear, temperature changes, or evolving road friction. This capability would be key to achieving self-adaptive autonomous vehicles. Techniques like the GripMap [180] will be explored.
- **Full-Stack Learning with Safety Guarantees.** This thesis integrated learning selectively within individual components (*e.g.,* lateral control, vehicle dynamics). A promising direction is to extend this paradigm to the entire control stack, making all modules—including the E-NMPC—differentiable and trainable end to end. If implemented using the same physics-informed paradigm we already use, it will preserve transparency, interpretability, and verifiable safety guarantees.
- **Neural Vehicle Model in the E-NMPC.** Another direction is to embed the learned neural vehicle model directly within the planning scheme, replacing the kineto-dynamical model. This could enhance planning accuracy and simplify the low-level controllers by internalising part of the vehicle behaviour within the optimisation process. This in turn would solve current limitations, such as the absence of steering-specific dynamics and transient understeer/oversteer behaviour. These effects were non-neglible during on-track experiments.
- **Systematic Evaluation Across Tracks and Vehicles.** Although ARD was validated on several vehicles and circuits, a broader and more systematic evaluation campaign is essential. This includes testing on tracks with varied topography and surface properties, and employing vehicles without strict actuation limits to explore higher dynamic envelopes. Establishing structured benchmarks across platforms would provide valuable insights into generalisability and scalability.
- **Human-Centric Evaluation of Shared Control.** The ARC framework remains an early exploration of shared control. Its effectiveness should be rigorously assessed through controlled user studies focusing on trust, learnability, stability, and fatigue. Further work should also analyse the closed-loop stability of the combined human–machine system, especially when operating near handling limits. Integrating physiological or behavioural signals (*e.g.,* gaze, grip force) could allow real-time personalisation of assistance.
- **Multimodal Feedback Strategies.** While the current implementation of ARC provides visual and haptic guidance, future systems could incorporate pedal-based feedback. This modality could deliver more natural and less intrusive guidance, reducing visual and cognitive load. Adapting the feedback strategy to the driver's skill level and task context could further enhance comfort, performance, and acceptance.
- **Transfer to Urban and Mixed-Traffic Scenarios.** The principles underpinning this work—aggressive yet feasible motion planning, fast adaptation, and collaborative

control—are relevant beyond racing. Applying these methods to urban or mixed-traffic settings poses a challenging but valuable step towards bridging high-performance and safety-critical autonomy.

- **Fuel-Efficient Driving Strategies.** The online planning capabilities of ARD could be repurposed for ecological driving. By modifying the planning model and cost function, the framework could be used to minimise energy consumption or emissions rather than lap time. This shown in past works from our group [182], [183]. This approach could naturally extend to ARC, enabling it to coach human drivers towards more fuel-efficient driving styles.

Appendices

Appendix A

Driving Simulator

Abstract

This appendix documents the in-house driving simulator developed to support the training, testing, and validation of the *Artificial Race Driver* (ARD). The simulator was designed from scratch to meet the specific requirements of real-time autonomous driving research, with a strong focus on modularity, cross-platform compatibility, and low-latency operation. It supports a wide range of configurations, from simple headless execution on a laptop to immersive, human-in-the-loop testing in the simulator room. The entire system is developed internally, allowing full access to the codebase and seamless integration with the broader ARD framework. This appendix provides a structured overview of the simulator's hardware setup, software architecture, real-time control and logging infrastructure, and the visualization tools built around it.

Contents

A.1 Introduction 147
A.2 Hardware and Peripheral Setup 148
A.3 Custom Software Stack 149
A.3.1 Architecture Overview 149
A.3.2 Core Services 150
A.3.3 System Management and Hardware Integration 153
A.3.4 Simulation Backend 156
A.3.5 Development Utilities 158

A.1 Introduction

The development of a flexible, robust, and real-time capable driving simulator was a foundational component of this work. Over the course of a year, a fully custom simulator was implemented to support the rapid prototyping, training, and evaluation of ARD and, later, the *Artificial Race Coach* (ARC). While several commercial and academic solutions exist, none met the specific requirements of low-latency control, modular system integration, and cross-platform deployment that were essential for the goals of this project.

The simulator was designed from the ground up to accommodate a wide range of use cases, from headless testing on a laptop to immersive human-in-the-loop experiments in a dedicated simulator room. The hardware setup of the simulator room was developed in cooperation with Matteo Larcher, who defined the layout and selection of peripherals. Although hardware was not the focus of this work, it is included here for completeness.

The core of the system is a modular and portable software stack built around modern middleware and messaging standards. It supports both synchronous and asynchronous execution, distributed operation across multiple machines, and real-time communication with planning and control software. A custom logging system, a set of visualization tools, and numerous development utilities were implemented to streamline everyday use and reduce integration overhead. The simulator is actively used for both algorithm development and experimental validation.

The remainder of this appendix is structured as follows. Section A.2 summarizes the hardware setup. Then, Section A.3 presents the software architecture.

A.2 Hardware and Peripheral Setup

Although the simulator was primarily designed as a modular and platform-agnostic software tool, a dedicated simulator room was developed to support immersive and reproducible testing in both human-in-the-loop and hardware-in-the-loop configurations. This physical setup represents the most complete deployment of the simulator stack, demonstrating its capabilities in a fully distributed, low-latency environment. Figure A.1 shows our driving simulator room.

Figure A.1: Our driving simulator room.

The hardware infrastructure was designed in collaboration with Matteo Larcher, who led the selection and arrangement of the simulator room components. The simulator room features the following equipment:

- Three 4K laser projectors, aligned with a 210-degree cylindrical projection screen, providing seamless wraparound visuals.
- A 3 *Degrees of Freedom* (DoF) motion platform, supporting a full cockpit setup. Allowed motions are roll, pitch, and heave.
- A complete racing control set mounted on the platform:
  - Force-feedback steering wheel with buttons, switches, and gear shift paddles;
  - Sequential gear shifter (push-pull lever type);
  - Throttle, brake, and clutch pedals;
  - Handbrake lever.
- A 5.1 surround sound system for immersive audio feedback.
- Three workstation-class PCs, each responsible for rendering one projector's view.
- One real-time computer dedicated to time-critical tasks such as vehicle dynamics simulation. This machine is equipped with CAN interfaces to support hardware-in-the-loop (HiL) testing with physical automotive components.

- One general-purpose workstation for development, user applications, and supervisory tasks.
- One Intel NUC acting as a central manager node to coordinate startup and shutdown of the distributed system.
- One Ethernet switch that defines an isolated subnetwork to reduce communication latency and enable external devices (*e.g.,* laptops) to interface seamlessly with the simulator.
- Two Raspberry Pi 4 boards:
  - One acting as an NTP server and communication broker;
  - One dedicated to real-time logging.

This configuration supports a wide range of experimental scenarios, from immersive human-in-the-loop testing to hardware-in-the-loop validation of embedded components. Tight control over communication, synchronization, and feedback enables closed-loop tests in realistic conditions. However, it is important to note that the simulator room is only one of several supported use cases. The software stack introduced in the next section is fully cross-platform (where possible) and can be executed on single-machine setups without requiring specialized hardware.

A.3 Custom Software Stack

A.3.1 Architecture Overview

The driving simulator is built as a distributed, modular system structured around a central communication broker. All components interact via a publish/subscribe communication model, enabling loose coupling, dynamic configuration, and flexible deployment across a range of hardware setups. While the simulator can be executed on a single machine, it was designed from the start to support multi-node, real-time setups such as the simulator room. Figure A.2 shows a schematic overview of the architecture in the specific case of the simulator room.

At the core of the system lies the *broker*, a lightweight message-forwarding proxy that facilitates all communication between processes. Each component—whether software module, hardware interface, or user application—connects only to the broker, which abstracts the physical network topology. This design reduces configuration complexity and enables seamless substitution or duplication of nodes. The broker can be deployed on any machine; in the simulator room, it runs on a dedicated Raspberry Pi 4.

The communication structure is entirely based on the publish/subscribe paradigm, implemented over ZeroMQ sockets. Messages are either encoded using FlatBuffers—to minimize serialization overhead—or sent as plain JSON. The former is used by the core simulator modules to ensure low-latency operation, while the latter is supported for convenience and interoperability with external tools.

To support persistent logging and real-time telemetry, a dedicated *logger* module subscribes to a special push/pull channel exposed by the broker. This mechanism allows for multiple logger instances to run in parallel—even across different machines—without duplicating logged messages. If the incoming message is already in JSON format, it is logged directly to a MongoDB database. Otherwise, the message is automatically converted from its FlatBuffers representation into JSON before logging. This conversion infrastructure is automatically generated for each defined interface, streamlining development and ensuring consistency.

In addition to persistent logging, the logger can optionally relay all received JSON messages to PlotJuggler for real-time visualization. This functionality required a minor modification to the PlotJuggler project to support dynamic topic updates, which was contributed upstream and is now part of the official release.

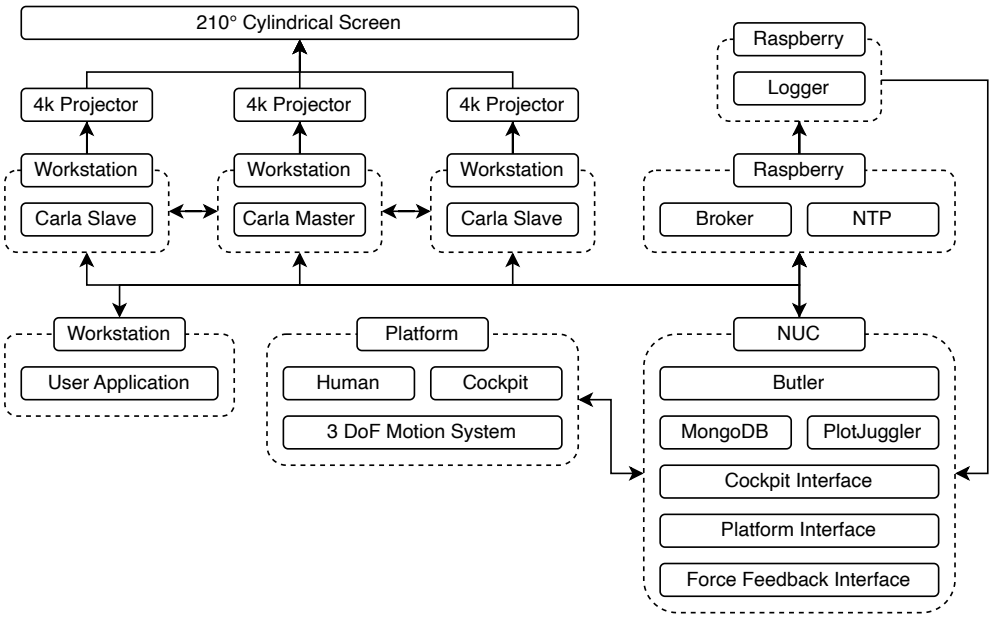


Figure A.2: Schematic overview of the architecture of the driving simulator in the specific case of the simluator room.

All functional modules—including the cockpit interface, motion platform control, and force feedback system—communicate exclusively via the broker. The same applies to the simulation backend based on CARLA, and to any user application such as ARD, which acts as a client of the simulator infrastructure.

To enable reproducible closed-loop experiments and accurate time-stamping, all nodes in the simulator room synchronize their clocks via an internal NTP server, ensuring sub-millisecond clock drift. While PTP would offer higher precision, the current setup was deemed sufficient for all use cases encountered during development.

The simulator software supports both synchronous and asynchronous execution modes. When CARLA is run locally on the development workstation, it operates in synchronous mode, enabling time-deterministic simulations. When deployed across the three projector-connected machines in the simulator room, CARLA runs three independent simulations synchronized in real-time. In this configuration, asynchronous communication is preferred to ensure responsive performance, despite significant optimization of CARLA's internals.

A.3.2 Core Services

A.3.2.1 Communication Broker

A centralized communication broker was chosen to simplify configuration, improve modularity, and enable robust introspection. While mesh-like topologies are common in academic systems due to their initial convenience, they quickly become difficult to scale, reconfigure, or monitor. The broker-based star topology adopted here avoids these limitations by decoupling components: each module only needs to know how to reach the broker, not its peers. This enables late joining, hot-plugging, and easier debugging, at the cost of minimal additional latency.

The broker itself is intentionally kept as simple as possible. As the central point of message forwarding, it is expected to run continuously and reliably for long durations—potentially hours or even days. To ensure maximum stability and portability, the broker is implemented using ZeroMQ's built-in proxy functionality, which provides a robust, event-driven message forwarding service with minimal dependencies.

Two configurations of the proxy are supported:

- **Daemon mode:** This is the default for deployment. The broker runs as a background process with no interactive features, forwarding all messages received on its XSUB (extended subscriber) socket to its XPUB (extended publisher) socket. This mode is stable, fast, and suitable for unattended operation.
- **Steerable mode:** In this mode, the broker runs interactively and exposes a control socket, allowing the user to inspect, pause, resume, or shut down message forwarding. This mode is primarily useful for debugging or temporary setups.

Regardless of mode, the broker also exposes a PUSH socket to which all messages are forwarded. This *capture channel* enables real-time logging by any number of downstream consumers (see Section A.3.2.2). Unlike the PUB/SUB mechanism—which delivers a copy of each message to every subscriber—the PUSH/PULL pattern guarantees exclusive delivery: once a message is consumed by a client, it is no longer available to others. This makes it ideal for parallel, distributed logging where duplicate entries must be avoided.

For performance reasons, the broker uses both TCP and IPC endpoints. This dual-stack setup enables seamless switching between local and distributed deployments. If no subscribers are present on a given endpoint (*e.g.,* TCP), messages are simply not emitted, resulting in negligible overhead. This redundancy allows flexibility in deployment: while IPC is typically faster for intra-process communication, TCP loopback can outperform it on some platforms.

The simplicity of the broker is deliberate. By relying on ZeroMQ's native mechanisms and avoiding any internal message parsing or processing, the broker remains highly stable and easy to maintain. Its sole responsibility is to forward messages, not interpret them. Higher-level logic—such as interface verification, schema matching, or logging—is handled elsewhere in the stack.

A.3.2.2 Distributed Logging Infrastructure

Robust logging is fundamental to the development and validation of autonomous driving systems. Without reliable data capture and inspection, it becomes difficult to identify faults, validate hypotheses, or iterate on algorithms. The simulator therefore includes a custom logging module designed to operate in real-time, with minimal latency and maximum configurability. This makes it possible to monitor messages as they flow through the system and verify their correctness during runtime, without waiting for post-processing or conversion steps.

The logger is designed around a few core goals: reliability, real-time capability, and flexibility. At its core, each logger instance listens to a dedicated PUSH socket exposed by the broker, receiving a continuous stream of messages captured by the broker's internal forwarding mechanism. Each message is either directly stored or converted to JSON on the fly before being written to a MongoDB database. This decision depends on the format: messages already in JSON (*e.g.,* from user applications or external tools) are logged as-is; messages using the simulator's internal FlatBuffers interfaces are automatically converted via a generated schema.

To meet performance and scalability requirements, each logger instance can be configured with a number of parallel worker threads. In a single-machine setup, a single logger can handle a high message rate simply by increasing the number of workers. In distributed setups, multiple logger instances can be launched on different machines—ZeroMQ's PUSH/PULL mechanism ensures that each message is consumed only once, avoiding duplication.

Each worker runs in a dedicated thread. It continuously consumes messages, converts them if needed, and appends them to a buffer. Periodically, based on an internal timing check, the buffer is flushed to the database in batches. This approach avoids per-message database writes while keeping delay bounded. The design was inspired by the Japanese bamboo water fountain: messages are accumulated in memory and then flushed once the buffer "tips".

The FlatBuffers-to-JSON conversion is efficient in most cases, but poses a challenge when dealing with large messages. For example, the ego vehicle interface used in the simulator encodes hundreds of individual fields (*e.g.,* actuator states and sensor signals), resulting in over a thousand scalar values. Logging such messages at high frequency (*e.g.,* 1000 Hz) stresses both CPU and memory bandwidth. The logger mitigates this through parallel processing and an efficient queueing strategy based on ZeroMQ's internal buffering.

However, ZeroMQ's in-memory queues are not bounded, and excessive memory pressure can lead to message accumulation and process termination due to out-of-memory conditions—especially on constrained devices such as the Raspberry Pi 4. Rather than introducing a custom file-backed queue system, the logger simply relies on the host operating system's swap mechanism as a fallback. This adds latency under extreme conditions but ensures that no messages are lost and avoids the complexity of implementing and maintaining custom queueing logic.

Both MongoDB logging and real-time telemetry are optional. When enabled, the logger can stream all JSON messages to PlotJuggler via a dedicated channel. This allows the user to visualize any data flowing through the simulator stack in real-time. The visualization layer supports dynamic topic discovery and was enabled by a small upstream contribution to the PlotJuggler project.

Overall, the logger provides a robust and flexible foundation for high-frequency data capture, with enough performance to operate in real-time even on modest hardware. On the simulator room's Raspberry Pi 4, the logger has been run continuously for extended periods with no reliability issues.

A.3.2.3 Interface Generation and Messaging

All communication between simulator modules is based on a structured, publish-only messaging model built on top of ZeroMQ. Messages are sent as multipart ZMQ messages, each consisting of four components in strict order:

- **Topic:** A UTF-8 string defining the subject of the message.
- **Payload:** A binary blob representing the serialized contents of a FlatBuffers schema.
- **Timestamp:** A string-encoded timestamp.
- **Cycle count:** A string-encoded counter representing the message index.

The simulator supports multiple FlatBuffers schemas, each representing a specific interface (*e.g.,* cockpit state, vehicle state). Once a new schema is defined and pushed to the repository, a GitHub Actions pipeline automatically generates:

- C++ and Python bindings for the new interface;
- A JSON conversion utility for the logger;
- A standardized topic string definition, ensuring consistent usage across all modules.

This workflow ensures that new interfaces are immediately usable and consistent across the entire stack. The build pipeline is extendable to other languages supported by `flatc`, although only C++ and Python are used at present.

A core design constraint governs all messaging within the simulator: every interface must be an *output interface*. No node is allowed to request data or publish input-specific contracts. Instead, each module simply publishes its own state and relevant data. Consumers are responsible for composing the inputs they need by subscribing to the appropriate messages. This discipline enforces strict decoupling between modules, making the system resilient to partial failures, easier to debug, and safer to extend.

While this model introduces additional development effort—particularly when coordination between modules is required—it avoids the pitfalls of implicit coupling and ad-hoc data dependencies. In practice, this approach has proven highly effective,

especially in collaborative or long-lived research environments where software evolves rapidly and unpredictably.

As an example, the cockpit interface simply publishes sensor readings such as steering angle, and pedal states. It defines no inputs and maintains no knowledge of downstream consumers. A vehicle model requiring driver inputs will subscribe to the cockpit messages and extract the relevant fields. More complex behaviours may require combining multiple messages from different sources, but these compositions are handled externally by the consumer, never by the publisher.

This strict separation of concerns—coupled with automated interface generation and flexible topic-based routing—ensures that the simulator remains maintainable, testable, and robust as it grows.

A minimal example of a FlatBuffers schema used in the simulator is shown below. The example defines a simple Cockpit interface that might be used to publish steering and pedal positions:

Listing A.1: Minimal example of a `Cockpit.fbs` interface.

```
namespace sim;

table Cockpit {
  timestamp: ulong;
  cycle: ulong;

  steering_angle: float;
  throttle: float;
  brake: float;
  clutch: float;

  handbrake: bool;
}

root_type Cockpit;
```

Once this file is added to the repository, the build system automatically generates the C++ and Python bindings, the JSON conversion logic for the logger, and the corresponding topic definition for consistent use across the simulator.

Each message is tagged with a topic name derived from the schema itself. By convention, the topic is constructed automatically as a combination of the schema namespace and root type (*e.g.,* `sim.Cockpit`). All top-level tables are considered valid message types and receive their own topic, meaning that even nested substructures can be published independently if needed. This naming scheme ensures uniqueness and consistency across modules, reducing the chance of mismatches or ambiguity in message handling.

A.3.3 System Management and Hardware Integration

A.3.3.1 Master Node: Centralized Orchestration for the Simulator Room

In the simulator room setup, a dedicated *master node* is used to manage the deployment and coordination of all distributed components. This role is fulfilled by an Intel NUC running a custom control interface named Butler. While the simulator software is inherently modular and can be launched manually or via scripts, the complexity of the multi-machine deployment in the simulator room motivated the development of a higher-level orchestration tool.

Butler is implemented as a graphical interface in PowerShell, providing a simple and centralized way to manage the state of the simulator. It allows the operator to perform a range of actions across the simulator room network without needing to manually access each machine.

Specifically, Butler provides the following capabilities:

- Power on, shut down, or restart the three projector-connected PCs via network commands.

- Install new versions of the customized CARLA simulator to those PCs.
- Start, stop, or restart the CARLA server instances (one per PC).
- Start, stop, or restart a complete simulation session, including launching `carla-control` and any associated modules.
- Manage core simulator services such as the `broker`, `logger`, `cockpit`, `force feedback`, and `motion platform` controllers.

Although originally developed to support the specific requirements of the CARLA-based deployment, Butler was designed with generality in mind. Its architecture allows it to be extended to support other simulation backends, such as CarMaker, or future internal tools.

A new version of Butler is currently under development. This improved version is written in Python using the PySide framework, offering a more portable and extensible architecture. Unlike the current PowerShell-based implementation—which is tightly coupled to Windows—the new version is designed to support multiple operating systems and more powerful scripting workflows. It introduces a lightweight client service on each machine, enabling real-time status monitoring, distributed control, and bidirectional communication. A redesigned graphical interface is also being developed to accompany this version.

The master node is not required for deployments outside the simulator room. On a single-machine setup, or in simpler multi-node configurations, all components can be launched independently or scripted using lightweight launchers. However, in the full-scale room configuration, the presence of a central orchestrator greatly improves usability and reliability during operation.

A.3.3.2 Cockpit Interface

The cockpit interface is used exclusively in the simulator room, where it serves as the bridge between the physical driving controls and the rest of the simulator stack. It collects all relevant driver input signals—such as steering angle, throttle, brake, clutch, and gear position—and publishes them to the broker using a standardized simulator message.

While conceptually simple, the implementation was complicated by the fragmented nature of the cockpit hardware. Each component—steering wheel, pedals, sequential shifter, handbrake—originates from a different manufacturer, with no unified interface. Most of these devices expose themselves only as raw game controllers, without meaningful abstraction. Worse, several vendors provide full functionality only under Windows, with limited or no cross-platform support.

In particular, the steering wheel assembly required using Windows-specific APIs to access its full input state, as its HID reports lacked standard mapping. The device could not be accessed reliably using typical input libraries on other platforms, making a Windows-based implementation the only practical option. Similar limitations affected other peripherals as well (see Section A.3.3.3 for further discussion).

The cockpit interface runs as a standalone application on Windows. It queries the controller states at runtime, and publishes the processed data to the simulator broker. It has no input channels—consistent with the simulator's strict publish-only design philosophy—and serves only as a source of driver state information.

Force feedback is handled separately due to its entirely different implementation (see Section A.3.3.4). The cockpit interface is concerned only with acquiring input states from the physical hardware and making them available to other modules in the simulation loop.

A.3.3.3 Motion Platform Control

The motion platform used in the simulator room provides 3 DoF and is capable of reproducing pitch, roll, and heave motions for immersive driver feedback. It is controlled

via a dedicated software module that acts as the interface between the simulator and the platform hardware.

The platform is supplied by a third-party vendor and comes with a proprietary software suite, including a set of graphical interfaces and a built-in motion cueing engine. Although no formal documentation was provided, the APIs are relatively clean and well documented. The interface exposes both control and feedback channels, making it possible to operate the platform directly or through the vendor's motion algorithms.

While the APIs are technically cross-platform, key features—including the vendor's GUI and proprietary cueing engine—are only available on Windows. For this reason, the platform interface is implemented and ran under Windows in the simulator room. This configuration allows the user to switch seamlessly between the vendor's motion cueing and a custom internal one, depending on the testing needs.

The vendor-provided GUI offers a range of real-time tools, including a virtual cockpit dashboard and a set of live tuning parameters for the motion dynamics. These are accessible only when the proprietary cueing is used. When running custom motion algorithms, these features are disabled, and access to platform feedback becomes significantly slower—on the order of several milliseconds per query—rendering continuous readout impractical.

The control application serves a dual role:

- It **publishes** the current platform state when using the vendor's cueing engine (*e.g.,* platform pose, system status).
- It **subscribes** to the ego vehicle state, interprets it, and forwards the resulting motion commands to the platform.

Users can select between cueing strategies at startup time. When the vendor's motion engine is active, the interface merely acts as a bridge between simulator signals and the vendor's software. When custom cueing is enabled, the interface applies internal motion algorithms and writes directly to the platform's low-level control API. In both modes, communication with the rest of the simulator stack follows the same publish/subscribe architecture used system-wide.

This structure allows the platform to be integrated flexibly within the simulator stack while retaining access to vendor tooling when needed. It also opens the possibility of gradually transitioning to fully custom motion cueing in the future, if required.

A.3.3.4 Force Feedback Control

The force feedback system, responsible for generating steering torque at the simulator wheel, was by far the most constrained and technically demanding hardware integration in the simulator room. The wheel assembly is a commercial device that offers full access to its force feedback motor only through a proprietary driver stack, available exclusively on Windows. After extensive experimentation, a limited configuration was found to run under Linux, but only in a default-safe mode with a fixed torque cap of 5 Nm and additional damping. This configuration is intended as a failsafe or low-risk testing mode and does not allow meaningful control.

Full torque control requires interfacing with the device via DirectX and its legacy DirectInput API, a poorly documented and cumbersome interface. After multiple unsuccessful attempts, integration was made possible by adapting an open-source example project that demonstrated constant-force feedback via DirectInput:

- https://github.com/walbourn/directx-sdk-samples-reworked/tree/main/DirectInput/FFConst

Using this as a foundation, a custom application was developed to control the steering torque applied to the wheel. The application subscribes to multiple data streams:

- Ego vehicle state, to extract physical quantities such as steering torque;

- Autonomous agent low-level output, which provides the intended steering angle;
- Cockpit steering angle, to measure the user's current input.

The force feedback algorithm partitions the available torque budget into two components:

1. A *vehicle feedback component*, which reproduces the physical steering effort required by the simulated dynamics;
2. An *ARC guidance component*, which communicates the autonomous agent's intended steering action.

The ARC torque is computed using the shared control strategy described in Chapter 10, based on the angular error between the user's input and the agent's desired steering angle. This error is passed through a parametric smoothstep function to determine the final proportion of torque to be allocated to the user. The two torques are then summed and applied as a single motor command to the wheel.

The application is listener-only: it subscribes to relevant simulator messages, computes the appropriate force response, and sends commands directly to the motor controller via the DirectInput API. It publishes no data to the rest of the simulator stack, in accordance with the simulator's strict separation between control and sensing.

Despite the technical and platform limitations, this implementation enables a realistic and dynamic shared control interaction between human and autonomous agent, suitable for ARC-style experiments as well as immersive human-in-the-loop testing.

A.3.4 Simulation Backend

A.3.4.1 Custom Carla Extensions

While the core of the simulator is designed to be simulator-agnostic, several significant extensions were developed for the CARLA simulator to support real-time, multi-machine deployment in the simulator room. These modifications span vehicle integration, terrain interaction, rendering performance, and map and asset design. The result is a highly adapted version of CARLA capable of running dense, immersive driving scenarios at high frame rates. The code of our CARLA version is available at https://github.com/driving-simulator-unitn/carla.

Custom Vehicle Models. Two vehicle modelling approaches were implemented to suit different use cases:

- An *internal vehicle model*, integrated directly into CARLA's C++ source code, allows for high-fidelity simulation without introducing external communication delays.
- An *external vehicle model*, interfaced via ZeroMQ, enables synchronization between CARLA and an externally simulated ego vehicle (*e.g.,* using a validated physics model). CARLA listens for state updates and visually replicates the motion of the external vehicle in real-time.

Terrain Server Integration. To support external vehicle models, an in-engine *terrain server* was also implemented. This module is part of the ZeroMQ communication interface inside CARLA and provides environment feedback for external use. When enabled, the terrain server monitors points of interest included in the incoming ego vehicle messages. For each point of interest, it computes contact information with the environment (*e.g.,* ground penetration, surface material, and collision state) and publishes the result as a dedicated topic. External dynamics modules can then subscribe to this topic to obtain real-time terrain feedback, closing the loop between CARLA visuals and physically accurate simulation.

Rendering Optimization via Spectator Camera. To improve rendering performance, the simulator avoids CARLA's standard camera sensors and instead uses the built-in spectator camera to render the driver's point of view. This approach eliminates the overhead associated with image generation and sensor callbacks, significantly reducing GPU and CPU load. As a result, the system is capable of running at 120 Hz and 1080p resolution on all three projectors in the simulator room. In more demanding conditions—such as dense traffic scenarios involving hundreds of vehicles—the system maintains a stable 60 Hz, which is sufficient for immersive operation. The spectator camera is controlled via Python when using CARLA's default vehicle model, and via C++ when using internal or external custom models to minimize latency.

Custom Maps and Assets. A set of custom maps and 3D assets were developed to support autonomous racing scenarios. These include high-fidelity racetracks such as the Mugello Circuit, which were used extensively for evaluating ARD. In addition to track geometry, custom vehicle objects (meshes and physical bodies) were created to match the appearance and dimensions of high-performance racecars. These assets are implemented at the Unreal Engine level and are separate from the internal vehicle dynamics.

Traffic Scenario Extensions. Finally, custom traffic teleporting strategies were implemented to ensure consistent vehicle density around the ego vehicle. The implementation was carried out by Nicola Mengarda, a master thesis student, as part of a collaborative effort. He also contributed to the development of the custom vehicle model interfaces and the racetrack maps.

These extensions significantly improve the usability, realism, and performance of CARLA in a real-time, human-in-the-loop simulator setting. Synchronization across machines is handled separately and discussed in the next subsection.

A.3.4.2 Multi-Machine Carla Coordination

CARLA does not natively support multiscreen or multi-projector setups. While the simulator can technically be configured to render across multiple monitors on a single machine, doing so requires expensive and restrictive warp-and-blend licences. In our case, only single-projector-single-computer licences were available, which made a standard unified rendering configuration infeasible.

Although Unreal Engine offers mechanisms to support multiscreen projection (*e.g.,* nDisplay), these features are difficult to integrate into an existing large-scale codebase like CARLA. The version used in this work (CARLA 0.9.15) is based on Unreal Engine 4.26, which lacks many of the multiscreen rendering improvements introduced in later versions. Upgrading was not a viable option due to CARLA's known instability and reduced feature set in more recent releases.

To overcome these limitations, a custom multi-machine coordination system was developed. The solution consists of three independent CARLA server instances—one per projector-connected PC—running in parallel and coordinated by a central master process. The roles are as follows:

- One instance acts as the *master*, responsible for managing simulation state and synchronization.
- The remaining two instances act as *slaves*, rendering the left and right viewpoints using kinematically driven entities.

Each server spawns an identical simulation scene, including the ego vehicle and any required traffic agents. On the slave machines, all vehicles are configured as kinematic actors: their states are not updated via physics but are instead directly overwritten with new poses received from the master. This reduces computational cost and avoids simulation drift.

The simulation loop proceeds as follows:

1. The master server simulates the environment, including ego vehicle dynamics (via CARLA or external models), traffic logic, and sensor data.
2. After each simulation step, the master serializes the relevant state—ego pose, traffic poses, and any shared environment state—and sends it to all slave machines.
3. The slaves receive the update, apply the poses to their local actors, render the current frame, and send back a lightweight acknowledgment ("I'm done").
4. While waiting for slave acknowledgments, the master performs any remaining per-frame tasks, such as sensor processing or logging, then proceeds to the next frame.

Communication is implemented using ZeroMQ's Python interface (pyzmq), with data transmitted as pickled Python objects over a minimal PUB/SUB socket pair. The system is designed for low-latency operation: messages are compact, and the slaves perform no simulation logic beyond kinematic updates. The master does not track individual slaves explicitly; instead, it counts received acknowledgments to determine when to proceed.

This architecture is inherently scalable. Since the data is serialized once and published to all subscribers, additional slaves—such as rearview or auxiliary monitors—can be added with minimal effort. The current implementation anticipates this by decoupling the master's main loop from the rendering-specific tasks, making expansion straightforward.

Importantly, the entire system operates in a *synchronous and deterministic* manner. Each simulation frame is advanced only when all slaves have completed rendering and acknowledged receipt of the update. This ensures consistent timing and reproducibility across all machines, frame by frame. As long as the slave machines are able to render faster than the target frame rate, the system maintains real-time operation without sacrificing synchronization guarantees. If not, the system still remains synchronous, but it will start to visibly lag.

Overall, this coordination framework enables high-performance, multi-machine rendering for immersive environments using an engine that does not natively support it, while remaining simple and robust.

A.3.5 Development Utilities

To support real-time performance and developer ergonomics, a set of lightweight utility components were developed alongside the simulator infrastructure. These modules address recurring limitations in C++17's standard library, add robustness to third-party tools, and provide fast and flexible building blocks for messaging and timing. Each utility is designed for speed and clarity, with minimal abstractions and no unnecessary overhead. In high-frequency loops—where 1 ms cycle times are common—such components must behave predictably and efficiently, often operating silently in the background.

STL-Compatible Generic Queue. C++17's standard library provides several queue-like containers, but lacks a unified interface for performance-tuned circular queues or single-value queues. A custom generic queue was implemented to fill this gap. It is designed as a partially specialized template that supports three modes:

- `SINGLE` holds a single value, with no internal buffer—ideal for overwriting with the latest message while exposing a queue-like API.
- `FIFO` implements a first-in, first-out queue with preallocated memory.
- `LIFO` implements a stack (last-in, first-out), also preallocated.

This utility is used extensively throughout the simulator to manage message queues.

Fixed-Rate Timer. A fixed-rate timer class was adapted from a utility originally developed by Matteo Larcher. It repeatedly executes a user-provided lambda at a specified frequency. Two operating modes are available:

- *Eco mode:* Uses `std::this_thread::sleep_for()` to delay between executions. Its precision depends on the host OS—Linux typically achieves 1–3 ms, while Windows struggles to go below 15 ms.
- *Precise mode:* Uses a busy-wait loop to continuously monitor elapsed time. This ensures maximum timing precision, typically within a few microseconds over tens of minutes of operation, at the cost of increased CPU load.

This timer is particularly useful in vehicle dynamics modules and other simulation components where exact timing is critical and jitter must be minimized.

TOML Value Resolver. The simulator uses the toml++ library for configuration parsing. While powerful, the library heavily relies on `std::optional` and manual chaining, making deep key access verbose and error-prone. A small wrapper was implemented to improve usability. It resolves nested keys, throws clear errors when a key is missing, and reports the full key chain leading to the failure. This greatly simplifies configuration logic and provides meaningful diagnostics when debugging simulation parameters. This utility could potentially be contributed upstream as a quality-of-life improvement.

ZMQ Multipart Message Wrapper. A simple utility wraps ZeroMQ multipart messages using a standard vector interface. This improves readability and provides a consistent API for assembling and parsing messages across the simulator, particularly when using FlatBuffers or raw byte payloads. Preallocation is also employed here.

ZMQ Receive Queue. This component implements a background receiver thread that listens on a user-provided ZMQ socket and queues incoming messages in a custom queue (typically FIFO or SINGLE). It uses condition variables for efficient wake-up and non-blocking operation, making messages available as soon as they arrive. Template specialization ensures no type-conversion overhead is introduced. This class is widely used in modules such as the cockpit, and external vehicle interfaces.

ZMQ Topic Queue. A specialized version of the receive queue, the topic queue is designed for SUB sockets. It internally manages a map from topic strings to queues of received messages. Topics are discovered dynamically at runtime, allowing for flexible subscription patterns. When retrieving a message, the user specifies the desired topic, and the most recent message for that topic is returned and popped. This is particularly useful when subscribing to all topics and routing messages by source without hardcoding topic names or polling multiple sockets.

All queues in the system follow a strict ownership model: once a message is retrieved, it is removed from the queue. This ensures predictable memory usage and avoids accidental duplication of time-sensitive data. The design philosophy across all utilities remains the same: avoid hidden complexity, minimize runtime overhead, and provide just enough abstraction to make real-time simulation maintainable and work reliably.

References

[1] Badue, C. et al., "Self-driving cars: A survey," *Expert Systems with Applications*, vol. 165, p. 113 816, 2021. DOI: https://doi.org/10.1016/j.eswa.2020.113816.

[2] Liu, L. et al., "Computing systems for autonomous driving: State of the art and challenges," *IEEE Internet of Things Journal*, vol. 8, no. 8, pp. 6469–6486, 2021. DOI: 10.1109/JIOT.2020.3043716.

[3] Yurtsever, E., Lambert, J., Carballo, A., and Takeda, K., "A survey of autonomous driving: Common practices and emerging technologies," *IEEE Access*, vol. 8, pp. 58 443–58 469, 2020. DOI: 10.1109/access.2020.2983149.

[4] Zhao, J. et al., "Autonomous driving system: A comprehensive survey," *Expert Systems with Applications*, May 15, 2024. DOI: 10.1016/j.eswa.2023.122836.

[5] European Commission, *Road safety: Commission welcomes agreement on new EU rules to help save lives*, Press release, 2019. [Online]. Available: https://ec.europa.eu/commission/presscorner/detail/en/IP_19_1793.

[6] European Climate, Infrastructure and Environment Executive Agency, *EU road safety: Towards vision zero*, 2022. DOI: 10.2840/701809. [Online]. Available: https://cinea.ec.europa.eu/system/files/2023-02/H2020%20Transport-Road%20Safety%202022-web.pdf.

[7] Betz, J. et al., "Autonomous vehicles on the edge: A survey on autonomous vehicle racing," *IEEE Open Journal of Intelligent Transportation Systems*, vol. 3, pp. 458–488, 2022. DOI: 10.1109/OJITS.2022.3181510.

[8] *Formula ata driverless*. [Online]. Available: https://www.formula-ata.it/formula-driverless/.

[9] *F1tenth*. [Online]. Available: https://f1tenth.org/about.html.

[10] *Roborace*. [Online]. Available: https://roborace.com.

[11] *Indy autonomous challenge*. [Online]. Available: https://www.indyautonomouschallenge.com.

[12] *Abudhabi autonomous racing league*. [Online]. Available: https://a2rl.io/autonomous-car-race.

[13] Grigorescu, S., Trasnea, B., Cocias, T., and Macesanu, G., "A survey of deep learning techniques for autonomous driving," *Journal of Field Robotics*, vol. 37, no. 3, pp. 362–386, 2020. DOI: https://doi.org/10.1002/rob.21918. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.21918.

[14] Wen, L.-H. and Jo, K.-H., "Deep learning-based perception systems for autonomous driving: A comprehensive survey," *Neurocomputing*, vol. 489, pp. 255–270, 2022. DOI: https://doi.org/10.1016/j.neucom.2021.08.155.

[15] Aradi, S., "Survey of deep reinforcement learning for motion planning of autonomous vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 2, pp. 740–759, 2022. DOI: 10.1109/TITS.2020.3024655.

[16] Mozaffari, S., Al-Jarrah, O. Y., Dianati, M., Jennings, P., and Mouzakitis, A., "Deep learning-based vehicle behavior prediction for autonomous driving applications: A review," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 1, pp. 33–47, 2022. DOI: 10.1109/TITS.2020.3012034.

[17] Kuutti, S., Bowden, R., Jin, Y., Barber, P., and Fallah, S., "A survey of deep learning applications to autonomous vehicle control," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 2, pp. 712–733, 2021. DOI: 10.1109/TITS.2019.2962338.

[18] Chib, P. S. and Singh, P., "Recent advancements in end-to-end autonomous driving using deep learning: A survey," *IEEE Transactions on Intelligent Vehicles*, pp. 1–18, 2023. DOI: 10.1109/TIV.2023.3318070.

[19] Hermansdorfer, L., Trauth, R., Betz, J., and Lienkamp, M., "End-to-end neural network for vehicle dynamics modeling," in *2020 6th IEEE Congress on Information Science and Technology (CiSt)*, 2020, pp. 407–412. DOI: 10.1109/CiSt49399.2021.9357196.

[20] Nie, X., Min, C., Pan, Y., Li, K., and Li, Z., "Deep-neural-network-based modelling of longitudinal-lateral dynamics to predict the vehicle states for autonomous driving," *Sensors*, vol. 22, no. 5, 2022. DOI: 10.3390/s22052013.

[21] Rokonuzzaman, M., Mohajer, N., Nahavandi, S., and Mohamed, S., "Model predictive control with learned vehicle dynamics for autonomous vehicle path tracking," *IEEE Access*, vol. 9, pp. 128 233–128 249, 2021. DOI: 10.1109/ACCESS.2021.3112560.

[22] Spielberg, N. A., Brown, M., Kapania, N. R., Kegelman, J. C., and Gerdes, J. C., "Neural network vehicle models for high-performance automated driving," *Science Robotics*, vol. 4, no. 28, eaaw1975, 2019. DOI: 10.1126/scirobotics.aaw1975.

[23] Weihmayr, D., Birkner, C., Marzbani, H., and Jazar, R. N., "Data-driven vehicle dynamics: Leveraging SINDy for optimization-based vehicular motion planning," *IEEE Access*, 2025. DOI: 10.1109/ACCESS.2025.3594892.

[24] Piccinini, M., Larcher, M., Pagot, E., Piscini, D., Pasquato, L., and Biral, F., "A predictive neural hierarchical framework for on-line time-optimal motion planning and control of black-box vehicle models," *Vehicle System Dynamics*, vol. 61, no. 1, pp. 83–110, 2023. DOI: 10.1080/00423114.2022.2035776. eprint: https://doi.org/10.1080/00423114.2022.2035776.

[25] Piccinini, M., Taddei, S., Larcher, M., Piazza, M., and Biral, F., "A physics-driven artificial agent for online time-optimal vehicle motion planning and control," *IEEE Access*, 2023. DOI: 10.1109/ACCESS.2023.3274836.

[26] Fuchs, F., Song, Y., Kaufmann, E., Scaramuzza, D., and Dürr, P., "Super-human performance in gran turismo sport using deep reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4257–4264, 2021. DOI: 10.1109/LRA.2021.3064284.

[27] Wurman, P. et al., "Outracing champion gran turismo drivers with deep reinforcement learning," *Nature*, vol. 602, pp. 223–228, Feb. 2022. DOI: 10.1038/s41586-021-04357-7.

[28] IEEE Spectrum, *Tesla's autopilot depends on a deluge of data*, 2022. [Online]. Available: https://spectrum.ieee.org/tesla-autopilot-data-deluge.

[29] Kuznietsov, A., Gyevnar, B., Wang, C., Peters, S., and Albrecht, S. V., "Explainable AI for safe and trustworthy autonomous driving: A systematic review," *IEEE Transactions on Intelligent Transportation Systems*, Dec. 2024. DOI: 10.1109/TITS.2024.3474469.

[30] Limebeer, D. J. N. and Massaro, M., *Dynamics and Optimal Control of Road Vehicles*. Oxford University Press, Sep. 2018. DOI: 10.1093/oso/9780198825715.001.0001.

[31] Massaro, M. and Limebeer, D. J. N., "Minimum-lap-time optimisation and simulation," *Vehicle System Dynamics*, vol. 59, no. 7, pp. 1069–1113, 2021. DOI: 10.1080/00423114.2021.1910718. eprint: https://doi.org/10.1080/00423114.2021.1910718.

[32] Gabiccini, M., Bartali, L., and Guiggiani, M., "Analysis of driving styles of a gp2 car via minimum lap-time direct trajectory optimization," *Multibody System Dynamics*, vol. 53, pp. 1–29, Sep. 2021. DOI: 10.1007/s11044-021-09789-7.

[33] Perantoni, G. and Limebeer, D. J., "Optimal control for a formula one car with variable parameters," *Vehicle System Dynamics*, vol. 52, no. 5, pp. 653–678, 2014. DOI: 10.1080/00423114.2014.889315.

[34] Duhr, P., Sandeep, A., Cerofolini, A., and Onder, C. H., "Convex performance envelope for minimum lap time energy management of race cars," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 8, pp. 8280–8295, 2022. DOI: 10.1109/TVT.2022.3172473.

[35] Sedlacek, T., Odenthal, D., and Wollherr, D., "Minimum-time optimal control for battery electric vehicles with four wheel-independent drives considering electrical overloading," *Vehicle System Dynamics*, vol. 60, no. 2, pp. 491–515, 2022. DOI: 10.1080/00423114.2020.1823004. eprint: https://doi.org/10.1080/00423114.2020.1823004.

[36] Lot, R. and Bianco, N. D., "Lap time optimisation of a racing go-kart," *Vehicle System Dynamics*, vol. 54, no. 2, pp. 210–230, 2016. DOI: 10.1080/00423114.2015.1125514.

[37] Casanova, D., Sharp, R., and Symonds, P., "Minimum time manoeuvring: The significance of yaw inertia," *Vehicle System Dynamics*, vol. 34, no. 2, pp. 77–115, 2000. DOI: 10.1076/0042-3114(200008)34:2;1-G;FT077.

[38] Christ, F., Wischnewski, A., Heilmeier, A., and Lohmann, B., "Time-optimal trajectory planning for a race car considering variable tyre-road friction coefficients," *Vehicle System Dynamics*, vol. 59, no. 4, pp. 588–612, 2021. DOI: 10.1080/00423114.2019.1704804. eprint: https://doi.org/10.1080/00423114.2019.1704804.

[39] Heilmeier, A., Wischnewski, A., Hermansdorfer, L., Betz, J., Lienkamp, M., and Lohmann, B., "Minimum curvature trajectory planning and control for an autonomous race car," *Vehicle System Dynamics*, vol. 58, no. 10, pp. 1497–1527, 2020. DOI: 10.1080/00423114.2019.1631455. eprint: https://doi.org/10.1080/00423114.2019.1631455.

[40] Piccinini, M., Taddei, S., Pagot, E., Bertolazzi, E., and Biral, F., "How optimal is the minimum-time manoeuvre of an artificial race driver?" *Vehicle System Dynamics*, Sep. 24, 2024. DOI: 10.1080/00423114.2024.2407176.

[41] Limebeer, D. J. and Rao, A. V., "Faster, higher, and greener: Vehicular optimal control," *IEEE Control Systems Magazine*, vol. 35, no. 2, pp. 36–56, 2015. DOI: 10.1109/MCS.2014.2384951.

[42] Paden, B., Čáp, M., Yong, S. Z., Yershov, D., and Frazzoli, E., "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Transactions on Intelligent Vehicles*, vol. 1, no. 1, pp. 33–55, 2016. DOI: 10.1109/TIV.2016.2578706.

[43] Sharp, R. S. and Peng, H., "Vehicle dynamics applications of optimal control theory," *Vehicle System Dynamics*, vol. 49, no. 7, pp. 1073–1111, 2011. DOI: 10.1080/00423114.2011.586707. eprint: https://doi.org/10.1080/00423114.2011.586707.

[44] Brayshaw, D. and Harrison, M., "A quasi steady state approach to race car lap simulation in order to understand the effects of racing line and centre of gravity location," *Proceedings of The Institution of Mechanical Engineers Part D-journal of Automobile Engineering*, vol. 219, pp. 725–739, Jun. 2005. DOI: 10.1243/095440705X11211.

[45] Gadola, M., Vetturi, D., Cambiaghi, D., and Manzo, L., "A tool for lap time simulation," *SAE Technical Papers*, Dec. 1996. DOI: 10.4271/962529.

[46] Metz, D. and Williams, D., "Near time-optimal control of racing vehicles," *Automatica*, vol. 25, no. 6, pp. 841–857, 1989. DOI: https://doi.org/10.1016/0005-1098(89)90052-6.

[47] Rice, R., *Measuring Car-driver Interaction with the G-g Diagram*. Society of Automotive Engineers, 1973.

[48] Veneri, M. and Massaro, M., "A free-trajectory quasi-steady-state optimal-control method for minimum lap-time of race vehicles," *Vehicle System Dynamics*, vol. 58, no. 6, pp. 933–954, 2020. DOI: 10.1080/00423114.2019.1608364.

[49] Piccinini, M., Taddei, S., Piazza, M., and Biral, F., "Impacts of g-g-v constraints formulations on online minimum-time vehicle trajectory planning," ser. 17th IFAC Symposium on Control of Transportation Systems CTS 2024, Jan. 1, 2024. DOI: 10.1016/j.ifacol.2024.07.323.

[50] Lovato, S. and Massaro, M., "A three-dimensional free-trajectory quasi-steady-state optimal-control method for minimum-lap-time of race vehicles," *Vehicle System Dynamics*, vol. 60, no. 5, pp. 1512–1530, 2022. DOI: 10.1080/00423114.2021.1878242. eprint: https://doi.org/10.1080/00423114.2021.1878242.

[51] Lovato, S. and Massaro, M., "Three-dimensional fixed-trajectory approaches to the minimum-lap time of road vehicles," *Vehicle System Dynamics*, vol. 60, no. 11, pp. 3650–3667, 2022. DOI: 10.1080/00423114.2021.1969024. eprint: https://doi.org/10.1080/00423114.2021.1969024.

[52] Rowold, M., Ögretmen, L., Kasolowsky, U., and Lohmann, B., "Online time-optimal trajectory planning on three-dimensional race tracks," in *2023 IEEE Intelligent Vehicles Symposium (IV)*, 2023, pp. 1–8. DOI: 10.1109/IV55152.2023.10186701.

[53] Werner, F., Sagmeister, S., Piccinini, M., and Betz, J., *A quasi-steady-state black box simulation approach for the generation of g-g-g-v diagrams*, Apr. 14, 2025. DOI: 10.48550/arXiv.2504.10225. arXiv: 2504.10225 [cs]. Accessed: Jun. 10, 2025. [Online]. Available: http://arxiv.org/abs/2504.10225.

[54] Piccinini, M., Taddei, S., Betz, J., and Biral, F., "Kineto-dynamical planning and accurate execution of minimum-time maneuvers on three-dimensional circuits," arXiv, Feb. 5, 2025. DOI: 10.48550/arXiv.2502.03454.

[55] Savaresi, S., Spelta, C., Ciotti, D., Sofia, M., Rosignoli, E., and Bina, E., "Virtual selection of the optimal gear-set in a race car," *International Journal of Vehicle Systems Modelling and Testing*, vol. 3, Jan. 2008. DOI: 10.1504/IJVSMT.2008.020618.

[56] Tremlett, A., Assadian, F., Purdy, D., Vaughan, N., Moore, A., and Halley, M., "Quasi-steady-state linearisation of the racing vehicle acceleration envelope: A limited slip differential example," *Vehicle System Dynamics*, vol. 52, no. 11, pp. 1416–1442, 2014. DOI: 10.1080/00423114.2014.943927. eprint: https://doi.org/10.1080/00423114.2014.943927.

[57] Massaro, M., Lovato, S., and Veneri, M., "An optimal control approach to the computation of g-g diagrams," *Vehicle System Dynamics*, vol. 0, no. 0, pp. 1–15, 2023. DOI: 10.1080/00423114.2023.2178467. eprint: https://doi.org/10.1080/00423114.2023.2178467.

[58] Biniewicz, J. and Pyrz, M., "A quasi-steady-state minimum lap time simulation of race motorcycles using experimental data," *Vehicle System Dynamics*, vol. 0, no. 0, pp. 1–23, 2023. DOI: 10.1080/00423114.2023.2170256. eprint: https://doi.org/10.1080/00423114.2023.2170256.

[59] Berntorp, K., Olofsson, B., Lundahl, K., and Nielsen, L., "Models and methodology for optimal trajectory generation in safety-critical road–vehicle manoeuvres," *Vehicle System Dynamics*, vol. 52, no. 10, pp. 1304–1332, 2014. DOI: 10.1080/00423114.2014.939094. eprint: https://doi.org/10.1080/00423114.2014.939094.

[60] Bianco, N. D., Roberto, L., and Gadola, M., "Minimum time optimal control simulation of a gp2 race car," *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, p. 095 440 701 772 815, Oct. 2017. DOI: 10.1177/0954407017728158.

[61] Limebeer, D., Perantoni, G., and Rao, A., "Optimal control of formula one car energy recovery systems," *International Journal of Control*, vol. 87, no. 10, pp. 2065–2080, 2014. DOI: 10.1080/00207179.2014.900705.

[62] Tavernini, D., Massaro, M., Velenis, E., Katzourakis, D. I., and Lot, R., "Minimum time cornering: The effect of road surface and car transmission layout," *Vehicle System Dynamics*, vol. 51, no. 10, pp. 1533–1547, 2013. DOI: 10.1080/00423114.2013.813557.

[63] Tremlett, A. et al., "Optimal control of motorsport differentials," *Vehicle System Dynamics*, vol. 53, no. 12, pp. 1772–1794, 2015. DOI: 10.1080/00423114.2015.1093150. eprint: https://doi.org/10.1080/00423114.2015.1093150.

[64] Kapania, N. R., Subosits, J., and Christian Gerdes, J., "A sequential two-step algorithm for fast generation of vehicle racing trajectories," *Journal of Dynamic Systems, Measurement, and Control*, vol. 138, no. 9, p. 091 005, Jun. 2016. DOI: 10.1115/1.4033311. eprint: https://asmedigitalcollection.asme.org/dynamicsystems/article-pdf/138/9/091005/6123103/ds _138_09_091005.pdf.

[65] Lenzo, B. and Rossi, V., "A simple mono-dimensional approach for lap time optimisation," *Applied Sciences*, vol. 10, no. 4, 2020. DOI: 10.3390/app10041498.

[66] Frego, M., Bertolazzi, E., Biral, F., Fontanelli, D., and Palopoli, L., "Semi-analytical minimum time solutions for a vehicle following clothoid-based trajectory subject to velocity constraints," in *2016 European Control Conference (ECC)*, 2016, pp. 2221–2227. DOI: 10.1109/ECC.2016.7810621.

[67] Frego, M., Bertolazzi, E., Biral, F., Fontanelli, D., and Palopoli, L., "Semi-analytical minimum time solutions with velocity constraints for trajectory following of vehicles," *Automatica*, vol. 86, pp. 18–28, 2017. DOI: https://doi.org/10.1016/j.automatica.2017.08.020.

[68] Piazza, M., Piccinini, M., Taddei, S., Biral, F., and Berolazzi, E., "Real-time velocity profile optimization for time-optimal maneuvering with generic acceleration constraints," *IEEE Robotics and Automation Letters*, 2026. DOI: 10.1109/LRA.2025.3643297.

[69] Piccinini, M., Gottschalk, S., Gerdts, M., and Biral, F., "Computationally efficient minimum-time motion primitives for vehicle trajectory planning," *IEEE Open Journal of Intelligent Transportation Systems*, vol. 5, pp. 642–655, 2024. DOI: 10.1109/OJITS.2024.3476540. Accessed: Jun. 12, 2025. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10711857.

[70] González, D., Pérez, J., Milanés, V., and Nashashibi, F., "A review of motion planning techniques for automated vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 4, pp. 1135–1145, 2016. DOI: 10.1109/TITS.2015.2498841.

[71] Katrakazas, C., Quddus, M., Chen, W.-H., and Deka, L., "Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions," *Transportation Research Part C: Emerging Technologies*, vol. 60, pp. 416–442, 2015. DOI: https://doi.org/10.1016/j.trc.2015.09.011.

[72] Grüne, L. and Pannek, J., *Nonlinear Model Predictive Control: Theory and Algorithms*. Springer Publishing Company, Incorporated, 2013.

[73] Rawlings, J., Mayne, D., and Diehl, M., *Model Predictive Control: Theory, Computation, and Design*. Nob Hill Publishing, 2017.

[74] Faulwasser, T., Grüne, L., and Müller, M. A., *Economic Nonlinear Model Predictive Control*. 2018.

[75] Liniger, A., Domahidi, A., and Morari, M., "Optimization-based autonomous racing of 1:43 scale rc cars," *Optimal Control Applications and Methods*, vol. 36, pp. 628–647, Sep. 2015. DOI: 10.1002/oca.2123.

[76] Novi, T., Liniger, A., Capitani, R., and Annicchiarico, C., "Real-time control for at-limit handling driving on a predefined path," *Vehicle System Dynamics*, vol. 58, no. 7, pp. 1007–1036, 2020. DOI: 10.1080/00423114.2019.1605081.

[77] Spielberg, N. A., Brown, M., and Gerdes, J. C., "Neural network model predictive motion control applied to automated driving with unknown friction," *IEEE Transactions on Control Systems Technology*, vol. 30, no. 5, pp. 1934–1945, 2022. DOI: 10.1109/TCST.2021.3130225.

[78] Subosits, J. K. and Gerdes, J. C., "From the racetrack to the road: Real-time trajectory replanning for autonomous driving," *IEEE Transactions on Intelligent Vehicles*, vol. 4, no. 2, pp. 309–320, 2019.

[79] Vázquez, J. L., Brühlmeier, M., Liniger, A., Rupenyan, A., and Lygeros, J., "Optimization-based hierarchical motion planning for autonomous racing," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 2397–2403. DOI: 10.1109/IROS45743.2020.9341731.

[80] Wischnewski, A., Herrmann, T., Werner, F., and Lohmann, B., "A tube-mpc approach to autonomous multi-vehicle racing on high-speed ovals," *IEEE Transactions on Intelligent Vehicles*, pp. 1–1, 2022. DOI: 10.1109/TIV.2022.3169986.

[81] Li, J.-T., Chen, C.-K., and Ren, H., "Time-optimal trajectory planning and tracking for autonomous vehicles," *Sensors*, Jan. 2024, Publisher: Multidisciplinary Digital Publishing Institute. DOI: 10.3390/s24113281.

[82] Altché, F., Polack, P., and de La Fortelle, A., "High-speed trajectory planning for autonomous vehicles using a simple dynamic model," in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, 2017, pp. 1–7.

[83] Verschueren, R., Zanon, M., Quirynen, R., and Diehl, M., "Time-optimal race car driving using an online exact hessian based nonlinear mpc algorithm," in *2016 European Control Conference (ECC)*, 2016, pp. 141–147.

[84] Kegelman, J. C., Harbott, L. K., and Gerdes, J. C., "Insights into vehicle trajectories at the handling limits: Analysing open data from race car drivers," *Vehicle System Dynamics*, vol. 55, no. 2, pp. 191–207, 2017. DOI: 10.1080/00423114.2016.1249893. eprint: https://doi.org/10.1080/00423114.2016.1249893.

[85] Löckel, S., Peters, J., and Vliet, P. van, "A probabilistic framework for imitating human race driver behavior," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2086–2093, 2020.

[86] Altché, F., Polack, P., and de La Fortelle, A., "A simple dynamic model for aggressive, near-limits trajectory planning," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, 2017, pp. 141–147.

[87] Montani, M., Ronchi, L., Capitani, R., and Annicchiarico, C., "A hierarchical autonomous driver for a racing car: Real-time planning and tracking of the trajectory," *Energies*, vol. 14, no. 19, 2021. DOI: 10.3390/en14196008. [Online]. Available: https://www.mdpi.com/1996-1073/14/19/6008.

[88] Betz, J. et al., "Tum autonomous motorsport: An autonomous racing software for the indy autonomous challenge," *Journal of Field Robotics*, vol. 40, no. 4, pp. 783–809, 2023. DOI: https://doi.org/10.1002/rob.22153. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.22153.

[89] Hewing, L., Wabersich, K. P., Menner, M., and Zeilinger, M. N., "Learning-based model predictive control: Toward safe learning in control," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 3, no. 1, pp. 269–296, 2020. DOI: 10.1146/annurev-control-090419-075625. eprint: https://doi.org/10.1146/annurev-control-090419-075625.

[90] Hewing, L., Liniger, A., and Zeilinger, M. N., "Cautious nmpc with gaussian process dynamics for autonomous miniature race cars," in *2018 European Control Conference (ECC)*, 2018, pp. 1341–1348.

[91] Kabzan, J., Hewing, L., Liniger, A., and Zeilinger, M. N., "Learning-based model predictive control for autonomous racing," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3363–3370, 2019.

[92] Costa, G., Pinho, J., Botto, M. A., and Lima, P. U., "Online learning of mpc for autonomous racing," *Robotics and Autonomous Systems*, vol. 167, p. 104 469, 2023. DOI: https://doi.org/10.1016/j.robot.2023.104469.

[93] Pinho, J., Costa, G., Lima, P. U., and Ayala Botto, M., "Learning-based model predictive control for autonomous racing," *World Electric Vehicle Journal*, vol. 14, no. 7, 2023. DOI: 10.3390/wevj14070163. [Online]. Available: https://www.mdpi.com/2032-6653/14/7/163.

[94] Rosolia, U. and Borrelli, F., "Learning how to autonomously race a car: A predictive control approach," *IEEE Transactions on Control Systems Technology*, pp. 1–7, 2019.

[95] Rosolia, U., Carvalho, A., and Borrelli, F., "Autonomous racing using learning model predictive control," in *2017 American Control Conference (ACC)*, 2017, pp. 5115–5120. DOI: 10.23919/ACC.2017.7963748.

[96] Pagot, E., Piccinini, M., and Biral, F., "Real-time optimal control of an autonomous rc car with minimum-time maneuvers and a novel kineto-dynamical model," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 2390–2396. DOI: 10.1109/IROS45743.2020.9340640.

[97] Taddei, S., Piccinini, M., and Biral, F., "Biasing the driving style of an artificial race driver for online time-optimal maneuver planning*," in *2025 IEEE Intelligent Vehicles Symposium (IV)*. DOI: 10.1109/IV64158.2025.11097381.

[98] Subosits J. Gerdes, C., "Impacts of model fidelity on trajectory optimization for autonomous vehicles in extreme maneuvers," *IEEE Transactions on Intelligent Vehicles*, vol. 6, no. 3, pp. 546–558, 2021. DOI: 10.1109/TIV.2021.3051325.

[99] Kebbati, Y., Ait-Oufroukh, N., Ichalal, D., and Vigneron, V., "Lateral control for autonomous wheeled vehicles: A technical review," *Asian Journal of Control*, vol. n/a, no. n/a, DOI: https://doi.org/10.1002/asjc.2980.

[100] Rokonuzzaman, M., Mohajer, N., Nahavandi, S., and Mohamed, S., "Review and performance evaluation of path tracking controllers of autonomous vehicles," *IET Intelligent Transport Systems*, vol. 15, no. 5, pp. 646–670, 2021. DOI: https://doi.org/10.1049/itr2.12051. eprint: https://ietresearch.onlinelibrary.wiley.com/doi/pdf/10.1049/itr2.12051.

[101] Yao, Q., Tian, Y., Wang, Q., and Wang, S., "Control strategies on path tracking for autonomous vehicle: State of the art and future challenges," *IEEE Access*, vol. 8, pp. 161 211–161 222, 2020. DOI: 10.1109/ACCESS.2020.3020075.

[102] Berntorp, K. and Magnusson, F., "Hierarchical predictive control for ground-vehicle maneuvering," in *2015 American Control Conference (ACC)*, 2015, pp. 2771–2776. DOI: 10.1109/ACC.2015.7171154.

[103] Zarrouki, B., Nunes, J., and Betz, J., "R²nmpc: A real-time reduced robustified nonlinear model predictive control with ellipsoidal uncertainty sets for autonomous vehicle motion control," *IFAC-PapersOnLine*, 2024, 8th IFAC Conference on Nonlinear Model Predictive Control NMPC 2024. DOI: https://doi.org/10.1016/j.ifacol.2024.09.048.

[104] Zarrouki, B., Wang, C., and Betz, J., "A stochastic nonlinear model predictive control with an uncertainty propagation horizon for autonomous vehicle motion control," in *2024 American Control Conference (ACC)*, 2024. DOI: 10.23919/ACC60939.2024.10645032.

[105] Devineau, G., Polack, P., Altché, F., and Moutarde, F., "Coupled longitudinal and lateral control of a vehicle using deep learning," in *2018 IEEE International Conference on Intelligent Transportation Systems (ITSC)*, Nov. 2018, pp. 642–649. DOI: 10.1109/ITSC.2018.8570020.

[106] Lio, M. D., Bortoluzzi, D., and Papini, G. P. R., "Modelling longitudinal vehicle dynamics with neural networks," *Vehicle System Dynamics*, vol. 0, no. 0, pp. 1–19, 2019. DOI: 10.1080/00423114.2019.1638947.

[107] Da Lio, M., Donà, R., Papini, G. P. R., Biral, F., and Svensson, H., "A mental simulation approach for learning neural-network predictive control (in self-driving cars)," *IEEE Access*, vol. 8, pp. 192 041–192 064, 2020. DOI: 10.1109/ACCESS.2020.3032780.

[108] Piscini, D., Pagot, E., Valenti, G., and Biral, F., "Experimental comparison of trajectory control and planning algorithms for autonomous vehicles," *IECON 2019 - 45th Annual Conference of the IEEE Industrial Electronics Society*, vol. 1, pp. 5217–5222, 2019.

[109] Sukhil, V. and Behl, M., "Adaptive lookahead pure-pursuit for autonomous racing," *CoRR*, vol. abs/2111.08873, 2021. arXiv: 2111.08873. [Online]. Available: https://arxiv.org/abs/2111.08873.

[110] Biral, F., Bertolazzi, E., Bortoluzzi, D., and Bosetti, P., "Development and testing of an autonomous driving module for critical driving conditions," in *ASME International Mechanical Engineering Congress and Exposition*, vol. 17: Transportation Systems, 2008, pp. 361–370. DOI: 10.1115/IMECE2008-68487.

[111] Kapania, N. R. and Gerdes, J. C., "Design of a feedback-feedforward steering controller for accurate path tracking and stability at the limits of handling," *Vehicle System Dynamics*, vol. 53, no. 12, pp. 1687–1704, 2015. DOI: 10.1080/00423114.2015.1055279. eprint: https://doi.org/10.1080/00423114.2015.1055279.

[112] Ghignone, E., Baumann, N., and Magno, M., "TC-driver: A trajectory conditioned reinforcement learning approach to zero-shot autonomous racing," *Field Robotics*, vol. 3, no. 1, pp. 637–651, Jan. 2023. DOI: 10.55417/fr.2023020.

[113] Kapania, N. R. and Gerdes, J. C., "Learning at the racetrack: Data-driven methods to improve racing performance over multiple laps," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 8, pp. 8232–8242, 2020. DOI: 10.1109/TVT.2020.2998065.

[114] Spielberg, N. A., Templer, M., Subosits, J., and Gerdes, J. C., "Learning policies for automated racing using vehicle model gradients," *IEEE Open Journal of Intelligent Transportation Systems*, vol. 4, pp. 130–142, 2023. DOI: 10.1109/OJITS.2023.3237977.

[115] Le Mero, L., Yi, D., Dianati, M., and Mouzakitis, A., "A survey on imitation learning techniques for end-to-end autonomous vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 9, pp. 14 128–14 147, 2022. DOI: 10.1109/TITS.2022.3144867.

[116] Sun, X., Zhou, M., Zhuang, Z., Yang, S., Betz, J., and Mangharam, R., "A benchmark comparison of imitation learning-based control policies for autonomous racing," in *2023 IEEE Intelligent Vehicles Symposium (IV)*, 2023, pp. 1–5. DOI: 10.1109/IV55152.2023.10186780.

[117] Cao, S., Joa, E., and Borrelli, F., *A simple approach to constraint-aware imitation learning with application to autonomous racing*, Aug. 27, 2025. DOI: 10.48550/arXiv.2503.07737.

[118] Perot, E., Jaritz, M., Toromanoff, M., and De Charette, R., "End-to-end driving in a realistic racing game with deep reinforcement learning," in *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2017, pp. 474–475. DOI: 10.1109/CVPRW.2017.64.

[119] Güçkıran, K. and Bolat, B., "Autonomous car racing in simulation environment using deep reinforcement learning," in *2019 Innovations in Intelligent Systems and Applications Conference (ASYU)*, 2019, pp. 1–6. DOI: 10.1109/ASYU48272.2019.8946332.

[120] Evans, B. D., Engelbrecht, H. A., and Jordaan, H. W., *High-speed autonomous racing using trajectory-aided deep reinforcement learning*, 2023. arXiv: 2306.07003 [cs.RO].

[121] Cai, P., Wang, H., Huang, H., Liu, Y., and Liu, M., "Vision-based autonomous car racing using deep imitative reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 7262–7269, 2021. DOI: 10.1109/LRA.2021.3097345.

[122] Ju, S., Vliet, P. van, Arenz, O., and Peters, J., "Digital twin of a driver-in-the-loop race car simulation with contextual reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 8, no. 7, pp. 4107–4114, 2023. DOI: 10.1109/LRA.2023.3279618.

[123] Omeiza, D., Webb, H., Jirotka, M., and Kunze, L., "Explanations in autonomous driving: A survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 8, pp. 10 142–10 162, 2022. DOI: 10.1109/TITS.2021.3122865.

[124] Raissi, M., Perdikaris, P., and Karniadakis, G., "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, vol. 378, pp. 686–707, 2019. DOI: https://doi.org/10.1016/j.jcp.2018.10.045.

[125] Tan, C., Cai, Y., Wang, H., Sun, X., and Chen, L., "Vehicle state estimation combining physics-informed neural network and unscented kalman filtering on manifolds," *Sensors*, vol. 23, no. 15, 2023. DOI: 10.3390/s23156665.

[126] Acosta, M. and Kanarachos, S., "Tire lateral force estimation and grip potential identification using neural networks, extended kalman filter, and recursive least squares," vol. 30, no. 11, 2018.

[127] IEEE Spectrum, *Here's how to make deep learning more sustainable*, 2022. [Online]. Available: https://spectrum.ieee.org/deep-learning-sustainability.

[128] Zhang, Y., Tiňo, P., Leonardis, A., and Tang, K., "A survey on neural network interpretability," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 5, no. 5, pp. 726–742, 2021. DOI: 10.1109/TETCI.2021.3100641.

[129] Lio, M. D., Piccinini, M., and Biral, F., "Robust and sample-efficient estimation of vehicle lateral velocity using neural networks with explainable structure informed by kinematic principles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 12, pp. 13 670–13 684, 2023. DOI: 10.1109/TITS.2023.3303776.

[130] Pagot, E., Piccinini, M., Bertolazzi, E., and Biral, F., "Fast planning and tracking of complex autonomous parking maneuvers with optimal control and pseudo-neural networks," *IEEE Access*, vol. 11, pp. 124 163–124 180, 2023. DOI: 10.1109/ACCESS.2023.3330431.

[131] Plebe, A., Da Lio, M., and Bortoluzzi, D., "On reliable neural network sensorimotor control in autonomous vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 2, pp. 711–722, 2020. DOI: 10.1109/TITS.2019.2896375.

[132] Park, K., Han, S. H., Lee, H., and Kwahk, J., "Shared steering control: How strong and how prompt should the intervention be for a better driving experience?" *International Journal of Industrial Ergonomics*, vol. 86, p. 103 213, Nov. 2021. DOI: 10.1016/j.ergon.2021.103213. Accessed: May 17, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0169814121001311.

[133] Alirezaei, M., Corno, M., Katzourakis, D., Ghaffari, A., and Kazemi, R., "A Robust Steering Assistance System for Road Departure Avoidance," *IEEE Transactions on Vehicular Technology*, vol. 61, no. 5, pp. 1953–1960, Jun. 2012, Conference Name: IEEE Transactions on Vehicular Technology. DOI: 10.1109/TVT.2012.2191001. Accessed: May 6, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6172254.

[134] Anderson, S. J., Karumanchi, S. B., Iagnemma, K., and Walker, J. M., "The intelligent copilot: A constraint-based approach to shared-adaptive control of ground vehicles," *IEEE Intelligent Transportation Systems Magazine*, vol. 5, no. 2, pp. 45–54, 2013, Conference Name: IEEE Intelligent Transportation Systems Magazine. DOI: 10.1109/MITS.2013.2247796. Accessed: May 8, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6507273.

[135] Anderson, S. J., Peters, S. C., Pilutti, T. E., and Iagnemma, K., "Design and Development of an OptimalControl-Based Framework for Trajectory Planning, Threat Assessment, and SemiAutonomous Control of Passenger Vehicles in Hazard Avoidance Scenarios," 2009. Accessed: May 9, 2024. [Online]. Available: http://www.isrr2009.ethz.ch/doc/isrr2009_proceedings/isrr2009_0045.pdf.

[136] Benloucif, A., Nguyen, A.-T., Sentouh, C., and Popieul, J.-C., "Cooperative Trajectory Planning for Haptic Shared Control Between Driver and Automation in Highway Driving," *IEEE Transactions on Industrial Electronics*, vol. 66, no. 12, pp. 9846–9857, Dec. 2019, Conference Name: IEEE Transactions on Industrial Electronics. DOI: 10.1109/TIE.2019.2893864. Accessed: May 7, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8624569.

[137] Li, P., Gao, Z., Pu, D., and Wang, N., "Human-Machine Cooperative Steering Control Based on Non -cooperative Nash Game," en, *International Journal of Automotive Technology*, Apr. 2024. DOI: 10.1007/s12239-024-00048-0. Accessed: Apr. 30, 2024. [Online]. Available: https://doi.org/10.1007/s12239-024-00048-0.

[138] Wang, Z., Gong, X., Li, X., Li, X., and Huang, J., "Human-Machine Shared Steering Control Under High-Speed Emergency Obstacle Avoidance Scenarios," en, *Transportation Research Record*, p. 03 611 981 231 203 221, Oct. 2023, Publisher: SAGE Publications Inc. DOI: 10.1177/03611981231203221. Accessed: May 13, 2024. [Online]. Available: https://doi.org/10.1177/03611981231203221.

[139] tpadmin, *Toyota Research Institute Bets Big In Vegas On 'Toyota Guardian' Autonomy*, en-US, Jan. 2019. Accessed: May 20, 2024. [Online]. Available: https://pressroom.toyota.com/toyota-research-institute-bets-big-in-vegas-on-toyota-guardian-autonomy/.

[140] Mars, F., Deroo, M., and Hoc, J.-M., "Analysis of Human-Machine Cooperation When Driving with Different Degrees of Haptic Shared Control," *IEEE Transactions on Haptics*, vol. 7, no. 3, pp. 324–333, Jul. 2014, Conference Name: IEEE Transactions on Haptics. DOI: 10.1109/TOH.2013.2295095. Accessed: Apr. 30, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6710125.

[141] Mosbach, S., Flad, M., and Hohmann, S., "Cooperative longitudinal driver assistance system based on shared control," in *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Oct. 2017, pp. 1776–1781. DOI: 10.1109/SMC.2017.8122873. Accessed: May 6, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8122873.

[142] Kaufman, R., Costa, J., and Kimani, E., *Effects of Multimodal Explanations for Autonomous Driving on Driving Performance, Cognitive Load, Expertise, Confidence , and Trust*, arXiv:2401.04206 [cs], Apr. 2024. DOI: 10.48550/arXiv.2401.04206. Accessed: Apr. 30, 2024. [Online]. Available: http://arxiv.org/abs/2401.04206.

[143] Marcano, M., Díaz, S., Pérez, J., and Irigoyen, E., "A Review of Shared Control for Automated Vehicles: Theory and Applications," *IEEE Transactions on Human-Machine Systems*, vol. 50, no. 6, pp. 475–491, Dec. 2020, Conference Name: IEEE Transactions on Human-Machine Systems. DOI: 10.1109/THMS.2020.3017748. Accessed: May 7, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9203972.

[144] Sonoda, K., Okada, K., Sato, K., Abe, G., and Wada, T., "Does Shared Mode Improve Steering and Vehicle Motions During Control Transition From Automated to Manual Driving in Real Passenger Car?" *IEEE Access*, vol. 10, pp. 85 880–85 890, 2022, Conference Name: IEEE Access. DOI: 10.1109/ACCESS.2022.3197885. Accessed: May 17, 2024. [Online]. Available: https://ieeexplore.ieee.org/document/9853213/figures#figures.

[145] Shi, Z., Chen, H., Qu, T., and Yu, S., "Human-Machine Cooperative Steering Control Considering Mitigating Human-Machine Conflict Based on Driver Trust," *IEEE Transactions on Human-Machine Systems*, vol. 52, no. 5, pp. 1036–1048, Oct. 2022, Conference Name: IEEE Transactions on Human-Machine Systems. DOI: 10.1109/THMS.2022.3190683. Accessed: Apr. 30, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9847600.

[146] Xie, J., Xu, X., Wang, F., Liu, Z., and Chen, L., "Coordination Control Strategy for Human-Machine Cooperative Steering of Intelligent Vehicles: A Reinforcement Learning Approach," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 11, pp. 21 163–21 177, Nov. 2022, Conference Name: IEEE Transactions on Intelligent Transportation Systems. DOI: 10.1109/TITS.2022.3187016. Accessed: Apr. 30, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9827998.

[147] Jiang, Y., Ding, Y., Zhang, X., Xu, X., and Huang, J., "A self-learning human-machine cooperative control method based on driver intention recognition," en, *CAAI Transactions on Intelligence Technology*, vol. n/a, no. n/a, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1049/cit2.12313. DOI: 10.1049/cit2.12313. Accessed: Apr. 30, 2024. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1049/cit2.12313.

[148] Anderson, S. J., Walker, J. M., and Iagnemma, K., "Experimental Performance Analysis of a Homotopy-Based Shared Autonomy Framework," *IEEE Transactions on Human-Machine Systems*, vol. 44, no. 2, pp. 190–199, Apr. 2014, Conference Name: IEEE Transactions on Human-Machine Systems. DOI: 10.1109/TSMC.2014.2298383. Accessed: May 8, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6766255.

[149] Zhou, Y., Huang, C., and Hang, P., "Game Theory-Based Interactive Control for Human-Machine Cooperative Driving," en, *Applied Sciences*, vol. 14, no. 6, p. 2441, Jan. 2024, Number: 6 Publisher: Multidisciplinary Digital Publishing Institute. DOI: 10.3390/app14062441. Accessed: Apr. 29, 2024. [Online]. Available: https://www.mdpi.com/2076-3417/14/6/2441.

[150] Guo, C. et al., "Cooperation between driver and automated driving system: Implementation and evaluation," *Transportation Research Part F: Traffic Psychology and Behaviour*, Special TRF issue: Driving simulation, vol. 61, pp. 314–325, Feb. 2019. DOI: 10.1016/j.trf.2017.04.006. Accessed: May 6, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1369847816306623.

[151] Li, W., Li, Q., Li, S. E., Li, R., Ren, Y., and Wang, W., "Indirect Shared Control Through Non-Zero Sum Differential Game for Cooperative Automated Driving," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 9, pp. 15 980–15 992, Sep. 2022, Conference Name: IEEE Transactions on Intelligent Transportation Systems. DOI: 10.1109/TITS.2022.3146895. Accessed: May 17, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9709196.

[152] Piccinini, M., "Artificial drivers for online time-optimal vehicle trajectory planning and control," Accepted: 2024-04-29T07:58:39Z Publisher: TRENTO, Ph.D. dissertation, Apr. 12, 2024. Accessed: Jul. 4, 2025. [Online]. Available: https://hdl.handle.net/11572/406350.

[153] "MongoDB: The world's leading modern database," MongoDB, Accessed: Oct. 1, 2025. [Online]. Available: https://www.mongodb.com/.

[154] "PlotJuggler," PlotJuggler, Accessed: Oct. 1, 2025. [Online]. Available: https://plotjuggler.io.

[155] "Spdlog," Accessed: Oct. 1, 2025. [Online]. Available: https://spdlog.readthedocs.io/en/latest/.

[156] "Rerun," Accessed: Oct. 1, 2025. [Online]. Available: https://rerun.io.

[157] "ROS: Home," Accessed: Jul. 9, 2025. [Online]. Available: https://www.ros.org/.

[158] "ZeroMQ," Accessed: Jul. 9, 2025. [Online]. Available: https://zeromq.org/.

[159] "FlatBuffers docs," Accessed: Jul. 9, 2025. [Online]. Available: https://flatbuffers.dev/.

[160] "JSON," Accessed: Oct. 1, 2025. [Online]. Available: https://www.json.org/json-en.html.

[161] Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., and Koltun, V., "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

[162] "The most powerful real-time 3d creation tool," Unreal Engine, Accessed: Oct. 27, 2025. [Online]. Available: https://www.unrealengine.com/en-US/home.

[163] *Network topology*, in *Wikipedia*, Mar. 24, 2025. Accessed: Jul. 9, 2025. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Network_topology&oldid=1282098432.

[164] "ROS 2 documentation — ROS 2 documentation: Kilted documentation," Accessed: Jul. 9, 2025. [Online]. Available: https://docs.ros.org/en/kilted/index.html.

[165] "Protocol buffers," Accessed: Oct. 1, 2025. [Online]. Available: https://protobuf.dev/.

[166] Lot, R. and Biral, F., "A curvilinear abscissa approach for the lap time optimization of racing vehicles," *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 7559–7565, 2014, 19th IFAC World Congress. DOI: https://doi.org/10.3182/20140824-6-ZA-1003.00868.

[167] Limebeer, D. J. N. and Perantoni, G., "Optimal Control of a Formula One Car on a Three-Dimensional Track—Part 2: Optimal Control," *Journal of Dynamic Systems, Measurement, and Control*, vol. 137, no. 5, p. 051 019, May 2015. DOI: 10.1115/1.4029466.

[168] D. J. N. Limebeer M. Bastin, E. W. and Fensham, H. G., "Optimal control of a nascar – specification race car," *Vehicle System Dynamics*, vol. 61, no. 5, pp. 1210–1235, 2023. DOI: 10.1080/00423114.2022.2067573.

[169] "ISO 8855:2011," ISO, Accessed: Oct. 1, 2025. [Online]. Available: https://www.iso.org/standard/51180.html.

[170] Biral, F., Bertolazzi, E., and Bosetti, P., "Notes on numerical methods for solving optimal control problems," *IEEJ Journal of Industry Applications*, vol. 5, pp. 154–166, Mar. 2016. DOI: 10.1541/ieejjia.5.154.

[171] "Maplesoft - software for mathematics, online learning, engineering," Accessed: Oct. 1, 2025. [Online]. Available: https://www.maplesoft.com/.

[172] Nelles, O., *Nonlinear System Identification: From Classical Approaches to Neural Networks and Fuzzy Models* (Engineering online library). Springer, 2001.

[173] Nelles, O., Fink, A., and Isermann, R., "Local linear model trees (lolimot) toolbox for nonlinear system identification," *IFAC Proceedings Volumes*, vol. 33, no. 15, pp. 845–850, 2000, 12th IFAC Symposium on System Identification (SYSID 2000), Santa Barbara, CA, USA, 21-23 June 2000. DOI: https://doi.org/10.1016/S1474-6670(17)39858-0.

[174] Abe, M., *Vehicle Handling Dynamics: Theory and Application*. Elsevier Science, 2009.

[175] Guiggiani, M., *The Science of Vehicle Dynamics: Handling, Braking, and Ride of Road and Race Cars*. Springer, 2018.

[176] Pacejka, H., *Tire and Vehicle Dynamics Ed. 3*. Elsevier Science, 2012.

[177] "PyTorch," Accessed: Jul. 25, 2025. [Online]. Available: https://pytorch.org/.

[178] Rajamani, R., *Vehicle Dynamics and Control* (Mechanical Engineering Series). Springer US, 2011.

[179] Papini, G. P. R., *Tonegas/nnodely*. Accessed: Jul. 30, 2025. [Online]. Available: https://github.com/tonegas/nnodely.

[180] Werner, F., Schwehn, A.-K., Lienkamp, M., and Betz, J., "GripMap: An efficient, spatially resolved constraint framework for offline and online trajectory planning in autonomous racing," in *2025 IEEE Intelligent Vehicles Symposium (IV)*, Jun. 2025. DOI: 10.1109/IV64158.2025.11097550.

[181] "Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles j3016_202104," Accessed: Oct. 19, 2025. [Online]. Available: https://www.sae.org/standards/j3016_202104-taxonomy-definitions-terms-related-driving-automation-systems-road-motor-vehicles.

[182] Valenti, G., Pagot, E., De Pascali, L., and Biral, F., "Battery aging- aware online optimal control: An energy management system for hybrid electric vehicles supported by a bio-inspired velocity prediction," *IEEE Access*, 2021. DOI: 10.1109/ACCESS.2021.3134471.

[183] De Pascali, L., Biral, F., and Onori, S., "Aging-aware optimal energy management control for a parallel hybrid vehicle based on electrochemical-degradation dynamics," *IEEE Transactions on Vehicular Technology (TVT)*, 2020. DOI: 10.1109/TVT.2020.3019241.