UNIVERSITÄT ZU LÜBECK

From the Institute of Electrical Engineering in Medicine
of the University of Lübeck
Director: Prof. Dr. Georg Schildbach

MPC-Based Vehicle Trajectory Tracking Using Machine Learning for Parameter Optimization and Fault Detection

Dissertation
for Fulfillment of
Requirements
for the Doctoral Degree
of the University of Lübeck

from the Department of Computer Science and Technical Engineering

Submitted by

Toni Lubiniecki

from Stralsund

Lübeck, 2024

First referee: Prof. Dr. Georg Schildbach

Second referee: Prof. Dr.-Ing. habil. Marcin Grzegorzek

Date of oral examination: 30 April 2025

Approved for printing. Lübeck, 14 May 2025

Abstract

This thesis explores advancements in trajectory tracking control and fault detection within automated vehicle systems, focusing on two main areas: developing a learning-based model predictive control algorithm to enhance tracking accuracy and evaluating various neural networks as fault detection systems for trajectory tracking controllers. Both parts are assessed in a high-fidelity simulation environment.

The first part presents two adaptive model predictive controllers that use vehicle information, trajectory data, and tracking information to adapt the vehicle model within the model predictive control system, compensating for lost tracking accuracy due to model mismatches. One approach employs a trajectory-dynamic lookup table, while the more advanced approach uses Gaussian process regression with clustering. A thorough simulation study on real-world racetracks with varying dynamics demonstrates that the advanced approach effectively manages condition changes, significantly improves tracking performance, handles unknown trajectories with similar improvements, and memorizes adapted behavior through clustering.

The second part evaluates the effectiveness of four types of neural networks as fault detection systems. These networks detect changes in the vehicle, environmental shifts, or discrepancies between the applied vehicle model and the real vehicle. Trained a priori through supervised learning, the networks use tracking information, controller outputs, and vehicle data. The evaluation distinguishes between known and unknown fault conditions. The results suggest that neural networks are generally suitable for fault detection systems. Differences in effectiveness among the network types are minor for known fault conditions but more significant for unknown conditions.

Integrating adaptive model predictive control and neural network-based fault detection systems shows promise for developing robust and fault-tolerant control systems, enhancing accuracy and maintaining operational integrity in dynamic environments for trajectory tracking.

Zusammenfassung

Diese Dissertation beschäftigt sich mit der Trajektorienfolgeregelung und Fehlererkennung in automatisierten Fahrzeugsystemen und konzentriert sich auf zwei Hauptbereiche: die Entwicklung eines lernbasierten, modellprädiktiven Regelalgorithmus zur Verbesserung der Regelungsgenauigkeit und die Bewertung verschiedener neuronaler Netzwerke als Fehlererkennungssysteme für Trajektorienfolgeregler. Beide Teile werden in einer präzisen Simulationsumgebung bewertet.

Der erste Teil präsentiert zwei adaptive, modellprädiktive Regler, die Fahrzeuginformationen, Trajektoriendaten und Regelungsinformationen nutzen, um das Fahrzeugmodell innerhalb des modellprädiktiven Reglers anzupassen und so verlorene Regelungsgenauigkeit aufgrund von Modellabweichungen zu kompensieren. Ein Ansatz verwendet eine trajektorien-dynamische Lookup-Table, während der fortschrittlichere Ansatz Gaußprozess-Regression mit Clustering einsetzt. Eine umfassende Simulationsstudie auf realen Rennstrecken mit unterschiedlichen Dynamiken zeigt, dass der fortschrittliche Ansatz effektiv auf Zustandsänderungen reagiert, die Regelungsgenauigkeit signifikant verbessert, unbekannte Trajektorien mit ähnlicher Performanz gehandhabt werden und durch Clustering auf angepasstes Verhalten später zurückgegriffen werden kann.

Der zweite Teil bewertet die Wirksamkeit von vier verschiedenen Architekturen neuronaler Netzwerke als Fehlererkennungssysteme. Diese Netzwerke erkennen Veränderungen im Fahrzeug, Umwelteinflüsse oder Abweichungen zwischen dem angewandten Fahrzeugmodell und dem realen Fahrzeug. Die Netzwerke werden a priori durch überwachtes Lernen trainiert und nutzen Fahrzeuginformationen, Trajektoriendaten und Regelungsinformationen. Die Evaluierung unterscheidet zwischen bekannten und unbekannten Fehlerzuständen. Die Ergebnisse deuten darauf hin, dass die untersuchten Architekturen neuronaler Netzwerke im Allgemeinen für Fehlererkennungssysteme geeignet sind. Unterschiede in der Effektivität zwischen den Netzwerktypen sind bei bekannten Fehlerzuständen gering, werden jedoch bei unbekannten Zuständen bedeutender.

Die Integration von adaptiver, modellprädiktiver Regelung und neuronalen netzwerkbasierten Fehlererkennungssystemen zeigt Potenzial für die Entwicklung robuster und fehlertoleranter Regelungssysteme, die die Genauigkeit erhöhen und die Funktionssicherheit in dynamischen Umgebungen für die Trajektorienfolgeregelung aufrechterhalten.

Contents

List of Abbreviations

ADAM	**ADA**ptive **M**oment estimation
ADF	**A**utomated **D**riving **F**unction
AMPC	**A**daptive **M**odel **P**redictive **C**ontrol
ANN	**A**rtificial **N**eural **N**etwork
AV	**A**utonomous **V**ehicle
DBM	**D**ynamic **B**icycle **M**odel
CoG	**C**enter **O**f **G**ravity
CNN	**C**onvolutional **N**eural **N**etwork
CM4SL	**C**ar**M**aker **F**or **S**imu**L**ink
FIFO	**F**irst **I**n **F**irst **O**ut
GP	**G**aussian **P**rocess
GPR	**G**aussian **P**rocess **R**egression
GRU	**G**ated **R**ecurrent **U**nit
LiDAR	**Li**ght **D**etection **A**nd **R**anging
LSTM	**L**ong **S**hort-**T**erm **M**emory
LQR	**L**inear **Q**uadratic **R**egulator
MPC	**M**odel **P**redictive **C**ontrol
NIST	**N**ational **I**nstitute of **S**tandards and **T**echnology
OCP	**O**ptimal **C**ontrol **P**roblem
ODD	**O**perational **D**esign **D**omain
PHM	**P**rognostics and **H**ealth **M**anagement
PSO	**P**article **S**warm **O**ptimization
ReLU	**Re**ctified **L**inear **U**nit
RL	**R**einforcement **L**earning
RNN	**R**ecurrent **N**eural **N**etwork
RPP	**R**apid **P**rototyping **P**latform
SAE	**S**ociety of **A**utomotive **E**ngineers
SE	**S**quared **E**xponential
SGD	**S**tochastic **G**radient **D**escent
SNMPC	**S**tochastic **N**onlinear **M**odel **P**redictive **C**ontrol
SQP	**S**equential **Q**uadratic **P**rogramming

Chapter 1

Introduction

The field of trajectory tracking control for autonomous vehicles (AVs) has seen significant research aimed at enabling precise vehicle path following. Various control methods, including pure pursuit, PID control, linear quadratic regulator (LQR), and model predictive control (MPC), have been extensively explored by both industry and academia. These approaches, especially MPC, have been pivotal in advancing trajectory tracking in autonomous driving, as detailed in comprehensive reviews and specific applications by Yao et al. [1], Du et al. [2], Shen et al. [3], and Lima et al. [4].

However, the performance of these control algorithms heavily depends on the accuracy of their underlying models. These models often face challenges from dynamic parameters such as weight distribution, tire behavior, and environmental factors like road friction and crosswinds. Furthermore, even with accurate models, the ability to detect and react to anomalies in tracking behavior in real time is crucial for ensuring the reliability and safety of these systems.

In the context of the operational design domain (ODD) of AVs, the importance of this research is highlighted by the need for AVs to operate safely across a wide range of conditions. The ODD encompasses diverse and dynamic scenarios, including different road types, traffic patterns, and environmental conditions. The ability of AVs to navigate these complexities hinges on the adaptability and robustness of their control systems. There are multiple reasons why learning-based control and fault detection are critical.

Variability in Driving Conditions Real-world driving conditions are highly variable and unpredictable. Traditional control methods, which rely on static models, struggle to maintain precision and reliability across all scenarios. Learning-based control, such as adaptive model predictive control (AMPC), allows the system to adapt in real time to changing conditions like road friction, tire wear, and weight distribution. This adaptability is crucial for AVs to handle the full spectrum of their ODD effectively.

Enhanced Safety and Reliability As AVs approach SAE Level 5 automation, the complexity of their control systems increases. This complexity makes it difficult to anticipate and model every possible fault condition using traditional methods. Neural network-based fault detection systems offer a solution by learning from vast amounts of data to identify subtle patterns indicative of

faults. This capability ensures that the AV can detect and respond to issues promptly, preventing potential failures and enhancing overall safety.

Scalability and Continuous Learning The integration of machine learning into control and fault detection systems allows for continuous improvement. Unlike traditional methods, which may require extensive retraining or manual adjustments, learning-based approaches can adapt to new data and conditions with minimal intervention. This continuous learning is vital for maintaining high performance and reliability as AVs encounter new and unforeseen scenarios within their ODD.

Operational Integrity in Diverse Environments The diverse environments encompassed by the ODD – urban, suburban, rural, and highway settings – each present unique challenges. Learning-based control systems can leverage data from these varied environments to refine their models and improve performance. Fault detection systems, similarly, can be trained to recognize environment-specific anomalies, ensuring consistent operational integrity.

The overall goal of this dissertation is to enhance the safety, reliability, and adaptability of trajectory tracking control systems in AVs by developing advanced learning-based control methods and fault detection mechanisms. Specifically, this research aims to: (**i**) develop an AMPC that can dynamically adjust to varying driving conditions, ensuring precise trajectory tracking, and (**ii**) explore the use of neural networks for real-time fault detection to identify anomalies in the trajectory tracking control system.

This thesis introduces two AMPC approaches. The initial approach utilizes a trajectory-dynamic lookup table to adjust the vehicle model's yaw intensification. Building on this, the second, more advanced approach incorporates Gaussian process regression (GPR) and clustering to enhance model adaptability, error correction, and memorization of adapted behavior. These approaches leverage trajectory data and tracking behavior to dynamically adjust the vehicle model parameters within the MPC framework.

Adaptive control is essential for navigating the complexities of real-world driving conditions, where unpredictability is common. The adaptability of these systems is vital for practical deployment, ensuring operational integrity and safety in dynamic environments. The field of learning-based MPC focuses on improving computational efficiency and prediction accuracy while exploring various machine learning algorithms to better adapt to real-world driving scenarios.

The second focus of this thesis addresses the increasing complexity of control systems. This part emphasizes the need for advanced fault detection mechanisms to minimize risks and enhance system reliability. Traditional fault diagnosis methods struggle to keep pace with the complexities of fully autonomous systems [5]. This study explores the potential of neural network architectures to detect faults based on anomalies in tracking performance, controller outputs, and vehicle ego motion, without isolating or identifying the specific source of the fault.

Neural network-based fault detection systems provide a nuanced understanding of control system behaviors by identifying subtle patterns indicative of faults, a capability not fully realized with traditional methods. The integration of neural networks as fault detection systems is a burgeoning field focused on developing scalable models that can continuously learn and adapt to new data without extensive retraining typically required. This approach addresses both the challenge of data dependency and the need for suitable architectures.

A potential future unified framework could significantly enhance the safety, reliability, and efficiency of AVs by developing advanced computational techniques for adaptive control and fault detection, ensuring safe operation across their entire ODD and enabling AVs to effectively handle real-world uncertainties. In summary, this thesis details the development of sophisticated adaptive model predictive control techniques and evaluates neural network architectures for fault detection in trajectory tracking control systems, paving the way for future innovations in autonomous vehicle safety and reliability.

1.1 Research landscape

Before outlining the thesis and presenting its contributions, a brief overview of the research landscape on learning-based MPC and neural network-based fault detection is provided.

Learning-based MPC is a dynamic field that integrates advanced machine learning techniques to enhance control systems in various applications, including AVs, robotics, and aerospace. As outlined in [6], current research activity in this area can be broadly categorized into three main directions:

Learning the System Dynamics The accuracy of the system model is crucial for MPC performance. Current research focuses on developing methods to automatically improve these models using data-driven techniques. Learning-based MPC involves the automatic adjustment of the system model, either during operation or between different operational instances. Techniques such as Gaussian process regression, neural networks, deep learning, and reinforcement learning are prominent in this area (e.g., [7], [8]).

Learning the Controller Design This area focuses on improving the components of the MPC problem formulation, such as cost functions, constraints, and terminal components, to optimize closed-loop performance. Techniques such as Bayesian optimization and inverse optimal control are employed to enhance these components, aiming to achieve better outcomes for the underlying tasks (e.g., [9], [10]).

MPC for Safe Learning Ensuring safety during the learning process is critical, especially in applications involving physical systems. MPC techniques are used to guarantee that learning-based controllers operate safely, maintaining system stability and performance under various conditions (e.g., [11], [12]).

Neural network-based fault detection is a rapidly evolving field that leverages advanced machine learning techniques to identify, diagnose, and predict faults in various systems. Key areas of research activity in this domain include:

Deep Learning Techniques for Fault Detection Deep learning, particularly convolutional neural networks (CNN) and recurrent neural networks (RNN), is widely used for fault detection due to their ability to handle large datasets and complex patterns. CNNs are effective in extracting spatial features from data, while RNNs are well-suited for sequential data, making them ideal for temporal fault detection tasks in systems like machinery and networks (e.g., [13], [14]).

Autoencoders for Anomaly Detection Autoencoders are neural networks used to learn compressed representations of data and are particularly effective for unsupervised anomaly detection. By training on normal operating data, autoencoders can identify anomalies as deviations from the learned normal patterns based on reconstruction errors, making them valuable for detecting unexpected system behaviors without labeled fault data (e.g., [15], [16]).

Hybrid Models Combining neural networks with other machine learning techniques or domain-specific models enhances fault detection performance by leveraging the strengths of multiple approaches. For instance, integrating neural networks with statistical methods or rule-based systems can improve detection accuracy and robustness, particularly in complex industrial environments (e.g., [17], [18]).

Transfer Learning for Fault Detection Transfer learning involves leveraging pre-trained models on related tasks to improve fault detection performance, especially useful when labeled data is limited. This approach allows models to apply knowledge from similar domains, reducing the need for extensive training data and accelerating the deployment of effective fault detection systems in new environments (e.g., [19], [20]).

Explainable AI in Fault Detection Understanding the decision-making process of neural networks is crucial for practical applications. Research efforts aim to make neural network-based fault detection more interpretable by developing methods that explain how models arrive at their predictions, ensuring transparency and trustworthiness in critical applications such as healthcare and autonomous systems (e.g., [21], [22]).

Due to their pivotal roles in enhancing the precision, reliability, and safety of autonomous systems, the AMPC approaches in this thesis are situated in the area of *Learning the System Dynamics*, while the exploration of neural networks for fault detection systems falls within the realm of *Deep Learning Techniques for Fault Detection*. The ability to dynamically adjust system models using advanced machine learning techniques is crucial for ensuring the optimal performance of MPC across a variety of conditions and applications. This adaptability

is particularly important in the ever-evolving landscape of AVs, where operational environments can change rapidly and unpredictably.

The integration of deep learning techniques for fault detection offers a powerful approach to identifying and addressing system anomalies in real-time. By leveraging the capabilities of neural networks to handle large datasets and complex patterns, it becomes possible to predict faults with a high degree of accuracy. This not only enhances the operational safety of these systems but also extends their longevity and efficiency by preemptively addressing potential issues before they lead to significant failures.

Combining these research areas allows for the development of robust, intelligent systems capable of self-optimization and resilient fault management, which are essential for the future of autonomous technology.

1.2 Outline and contribution

The following subsections outline this thesis and contextualize its contributions within the existing literature.

1.2.1 Simulation environment

Chapter 2 introduces the simulation environment used in this thesis, incorporating Matlab/Simulink and IPG CarMaker. The chapter outlines the integration of the GRAMPC model predictive control framework and provides implementation details of the automated driving function (ADF) for trajectory tracking. Additionally, it explains the definition and generation of trajectories from real-world closed-loop circuits.

1.2.2 Trajectory tracking control

Chapter 3 introduces the baseline MPC used for the subsequent learning-based approaches and fault detection systems. It addresses tracking errors and the performance measures that enable the comparison of different trajectory tracking controllers. This chapter also details the vehicle model employed in this thesis, including the dynamic bicycle model and the nonlinear tire model, and validates the model using simulation results. Furthermore, it outlines the adaptation of the MPC formulation to accommodate spatially discrete trajectories and provides implementation details within the GRAMPC framework.

1.2.3 Learning-based MPC for trajectory tracking

Chapters 4 and 5 introduce two approaches for learning-based MPC for trajectory tracking. The first approach has already been published in [23], while the second approach, representing an advancement of the first, has been published in [24].

Outline

Chapter 4 presents an AMPC designed to adjust the vehicle model's yaw intensification to compensate for lost tracking accuracy caused by model errors and changing conditions. The chapter begins with a discussion of existing literature on learning-based trajectory tracking control for automated vehicles and emphasizes the importance of robust models and adaptive control algorithms. The concept of yaw intensification in the context of steering behavior is introduced, followed by a detailed description of the learning approach used in the AMPC to continuously compensate for the mismatch between the applied vehicle model and the real-world vehicle, thereby improving tracking accuracy. A thorough simulation study is conducted to evaluate the performance of the AMPC under various conditions, including its ability to repeatedly adapt and handle unknown trajectories.

Chapter 5 presents an enhanced version of the AMPC with an online learning algorithm based on Gaussian process regression (GPR) and clustering. The chapter begins with a review of relevant literature on MPC combined with GPR. Subsequently, GPR is introduced with a detailed explanation of its components, including prior and posterior distributions, handling noise, and optimizing hyperparameters. Furthermore, the data clustering technique k-means is discussed as a means to group observation data into a dictionary for the GPRs posterior, enabling the memorization of adapted behavior. This is followed by a detailed description of the learning approach. A thorough simulation study, featuring the same experiments as the first approach, is conducted to evaluate performance and compare both approaches. The advanced approach proves to be superior to its predecessor in all aspects. Additional experiments are conducted to evaluate the capability of the approach to memorize adjusted behavior.

Literature

Learning-based MPC for trajectory tracking involves integrating advanced machine learning techniques with MPC to enhance the control system's ability to manage dynamic and uncertain environments. These integrations are particularly significant in fields such as autonomous vehicles, robotics, and aerospace, where precision in trajectory tracking is crucial. Below is an overview of key areas in the literature:

Integration of Machine Learning Techniques with MPC *Gaussian Process Regression* is widely used to model uncertainties in system dynamics, providing probabilistic predictions that enhance the MPCs ability to handle unknown disturbances and model inaccuracies. As early as 2003, Kocijan et al. [25] demonstrated the feasibility of applying Gaussian processes to model predictive control. However, additional efforts are required before this approach can be widely applied in engineering practice, as evidenced by their demonstration on the control of a pH process benchmark. Recently, Maiworm [26] explored

Gaussian processes and their applications in control theory. Moreover, non-engineering domains also benefit; for example, Ortmann et al. [27] utilized an MPC with GPR that accounts for periodic insulin sensitivity to enhance blood glucose control by predicting future effects.

Neural Networks and Deep Learning are applied for their strong function approximation capabilities to predict system behaviors and optimal control actions. Deep learning models can handle high-dimensional data and complex relationships beyond the scope of traditional controllers. Salzmann et al. [28] present a framework to efficiently integrate large, complex neural network architectures as dynamic models within the CasADi model predictive control pipeline.

Reinforcement Learning techniques are being integrated with MPC to optimize control policies based on learned experiences. This approach is particularly useful in environments where models cannot be easily formulated or require adaptation over time. Assael et al. [29] introduce a data-efficient, model-based reinforcement learning algorithm that learns closed-loop policies in continuous state and action spaces directly from image pixels using deep dynamical models.

Adaptive and Robust MPC Learning-based MPC systems are increasingly capable of *Real-Time Adaptation* to changes in their operating environment, significantly improving accuracy and system responsiveness. Since classic GPR, which scales with $\mathcal{O}(n^3)$, is not suitable for large datasets, Quinonero-Candela et al. [30] introduced a unifying framework for sparse approximations to Gaussian processes for regression in 2005. Bijl et al. [31] generalize the sparse online GPR techniques, allowing for online updating of both the fully independent training conditional and the partially independent training conditional approximations.

These systems demonstrate enhanced *Robustness* to both internal disturbances and external environmental changes, which is critical for applications like autonomous driving on unpredictable roads. Lu et al. [32] present an adaptive fault-tolerant control approach for AVs that addresses actuator or sensor faults to improve driving safety. They develop a learning-based stochastic model predictive control strategy that incorporates the vehicle's real dynamics to achieve accurate trajectory tracking. Kabzan et al. [33] utilize GPR to account for residual model uncertainty and achieve safe driving behavior, while Arany [34] uses GPR to enhance vehicle safety control under variable friction road conditions.

Application-Specific Developments In the automotive industry, and specifically for *Autonomous Vehicles*, learning-based MPC is employed to manage complex scenarios involving dynamic obstacles and varying road conditions. Lui et al. [35], [36] present a Gaussian process-based MPC that includes approximate uncertainty propagation and safety constraints to ensure that the ego vehicle overtakes obstacles without collision. Hewig et al. [37] propose an MPC approach for vehicle control that integrates a nominal system with an additive

nonlinear part of the dynamics modeled as a GP, aiming to enhance both performance and safety over a nominal controller.

For *Robots* operating in unstructured environments, learning-based MPC aids in navigating and manipulating objects with high precision, adapting to new tasks through online learning. Ostafew et al. [38] propose a learning-based MPC algorithm for an autonomous mobile robot, designed to reduce path-tracking errors over repeated traversals along a reference path. Their algorithm employs a simple a priori vehicle model and a learned disturbance model.

In *Aerospace*, the technology is utilized to manage the highly nonlinear dynamics of spacecraft and aircraft, where traditional models often fall short. Cao et al. [39] propose an approach in which the dynamic models of an unmanned quadrotor are obtained purely from operational data, using probabilistic Gaussian process models.

Another area of development is improving computational efficiency through model simplification. Efforts are ongoing to simplify the computational models used in learning-based MPC, making faster computations feasible on real-time systems. Additionally, the use of specialized hardware, such as GPUs and FPGAs, is becoming increasingly common to handle the computational demands of these sophisticated control systems.

Contribution

In the context of online model parameter tuning during vehicle trajectory tracking, Lin et al. [40] and Kebbati et al. [41] have developed approaches for online tuning and optimizing parameters, primarily tailored for specific maneuvers such as the standard double lane change. However, these methods lack generalization to unknown trajectories and fail to continuously adapt to new instances of condition changes. Similarly, the approaches by Vaskov et al. [42], [43], which aim to estimate tire parameters online, are constrained by the same limitations. Therefore, the proposed approaches aim to fill this gap by developing concepts for online tuning and optimizing parameters that are independent of the maneuver or conditions and operate continuously. The contribution of this thesis lies in establishing two innovations: (**i**) the approaches exhibit a generic nature, enhancing trajectory tracking even for unknown trajectories, and (**ii**) the approaches demonstrate the capability to continuously adapt to new instances of condition changes, offering robustness in dynamic environments.

The approaches presented by Kocijan et al. [25], Hewing et al. [37], Jain et al. [44], and Kabzan et al. [33] focus not on trajectory tracking tasks but on improving lap times around a closed-loop circuit along a reference trajectory. Consequently, they contend with just one condition or state of vehicle model error to adapt to. However, these approaches yield significantly improved results by utilizing GPR. The second approach in this thesis employs GPR to further enhance overall performance and its ability to consistently adapt to new condition changes. Distinguishing it from existing literature, this approach introduces a

novel feature through the application of clustering: (**iii**) the capability to memorize experienced condition changes and react accordingly when encountered again.

1.2.4 Fault detection for trajectory tracking controllers

Chapters 6 and 7 explore the effectiveness of various neural network types as fault detection systems for trajectory tracking controllers.

Outline

Chapter 6 takes a comprehensive look at the utilization of neural networks as fault detection systems for trajectory tracking controllers, focusing on non-learning-based approaches. It assesses four types of neural networks: Artificial Neural Networks (ANN), Recurrent Neural Networks (RNN), Long Short-Term Memory (LSTM) networks, and Gated Recurrent Unit (GRU) networks, examining their efficacy as fault detection systems. The chapter is structured to include a literature review, descriptions of the neural networks, training data preparation, and a thorough simulation study. This study aims to assess and compare the capabilities of these networks in identifying both known and unknown fault conditions.

Chapter 7 explores the influence of the proposed AMPC as the underlying control approach on the effectiveness of neural networks as fault detection systems. Through a series of experiments conducted across various trajectories and closed-loop circuits, this chapter evaluates the impact on the performance of the different types of neural networks previously assessed.

Literature

The field of fault detection in AVs is rapidly evolving, with a focus on enhancing safety, reliability, and performance. Research directions in this domain include:

Sensor Fusion and Data Integration Fault detection systems in AVs increasingly rely on the integration of data from diverse sensors like cameras, radar, and LiDAR. Researchers are working on methods to fuse this data effectively, ensuring robust fault detection even under sensor failure or environmental uncertainties. This involves developing algorithms that can make reliable decisions based on incomplete or noisy data from multiple sensors. A good starting point is provided by Yeong et al. [45], who offer an end-to-end review of the hardware and software methods required for sensor fusion object detection. They highlight some of the challenges in the sensor fusion field and propose possible future research directions for automated driving systems. Pan et al. [46] propose a deep learning-based real-time data fusion network for intelligent vehicles, incorporating fault diagnosis and fault tolerance mechanisms that achieve state-of-the-art results in speed and accuracy. This network can

accurately detect the location of the target even when some sensors are out of focus or out of order.

Machine Learning and Artificial Intelligence The application of artificial intelligence and machine learning techniques is a dominant trend in fault detection for AVs. These techniques include deep learning models for predictive maintenance, anomaly detection, and real-time decision-making. The goal is to train models that can predict and diagnose faults before they affect the vehicle's operation, thereby preventing potential failures. For instance, Ren et al. [47] propose a general fault detection method using deep learning techniques to learn patterns of faults reflected in the dynamic model of an AV. Al-Zeyadi et al. [48] built a diagnostic system for intelligent vehicles to estimate required services, also based on deep learning techniques, to predict a wide range of faults with their Deep Symptoms-Based Model.

Prognostics and Health Management This research area focuses on predicting the health and operational capability of AV components over time. By using historical data and real-time performance, PHM systems aim to predict when a component might fail and suggest preventive measures. This approach aids in scheduling maintenance and replacements in a timely manner, thus avoiding unexpected downtimes. In 2011, Das et al. [49] discussed the essential steps involved in building an effective PHM system. More recently, Hadraoui et al. [50] proposed an approach to diagnose and predict the health condition of induction motors used in electric vehicle powertrain applications using machine learning techniques.

Further areas include cybersecurity, hardware redundancy, robust design, and regulatory and standard development. All these elements play a crucial role in advancing the state of the art in fault detection technologies, aiming to make AVs safer and more reliable as they become an integral part of our transportation systems.

Contribution

This thesis addresses various contributions in the field of fault detection and monitoring for controlled systems, some of which already focus on trajectory tracking control or apply neural networks. Biddle et al. [5] and Alsuwian et al. [51] realize fault detection via sensor monitoring. Dai et al. [52] and Salsbury et al. [53] benchmark control outputs to detect abnormal behavior. Geng et al. [54], [55], and Asadi et al. [56] propose approaches to detect fault conditions based on trajectory tracking. Loquasto et al. [57] apply a neural network to classify disturbances or changes in the plant dynamics of an MPC.

This study aims to evaluate an approach that incorporates all of these ideas, utilizing different types of neural networks as fault detection systems for trajectory tracking controllers. Disturbances and changes in plant dynamics are represented as fault conditions. Benchmarking control outputs, monitoring sensors (vehicle's ego motion), and tracking behavior serve as the selected features

for the neural networks. A thorough simulation study aims to: (**i**) evaluate the networks' ability to detect known incorrect behavior using a training dataset that already includes data from such fault conditions, (**ii**) assess their capability to detect unknown fault conditions, and (**iii**) examine how the effectiveness is influenced when the trajectory tracking controller adopts a learning-based approach.

1.2.5 Conclusion and outlook

Chapter 8 provides a brief summary of this thesis as well as concluding remarks. Furthermore, an outlook on possible future research is given.

1.3 Publications

The work in this thesis is based on the following publications:

- **T. Lubiniecki** and G. Schildbach, "Adaptive MPC for trajectory tracking with online adaption of the vehicle model's yaw intensification", 2023 European Control Conference (ECC), Bucharest, Romania, 2023, pp. 1-6, [23].
- **T. Lubiniecki** and G. Schildbach, "Trajectory Tracking MPC with Online Model Adaptation using Gaussian Process Regression and Clustering", 2023 IEEE 26th International Conference on Intelligent Transportation Systems (ITSC), Bilbao, Spain, 2023, pp. 154-159, [24].

Additional work in collaboration with colleagues has been prepared during my doctorate. This work is related to learning-based model predictive control but has not been included in this thesis.

- N. Panagiotopoulos, **T. Lubiniecki**, A. Turnwald, and, N. Baldauf, "Robust Machine Learning Systems For Dependable Space Applications", European Data Handling and Data Processing Conference – EDHPC 2023, Juan-Les-Pins, France, 2023, pp. 1-5, [58].
- N. Baldauf, **T. Lubiniecki**, A. Turnwald, K. Lakatos, and N. Panagiotopoulos, "Learning-based motion control of a rover on unknown ground", 12th International Conference on Guidance, Navigation and Control Systems (GNC), Sopot, Poland, 2023, [59].

Chapter 2

Simulation environment

The simulation environment utilized in this thesis consists of two main components: Matlab/Simulink as a programming and numeric computing platform and IPG CarMaker as a vehicle dynamics simulation tool. This chapter introduces the applied components as well as the model predictive control framework GRAMPC. Furthermore, implementation details of the necessary automated driving function to enable trajectory tracking are presented.

2.1 Overview

A schematic illustration of the complete simulation environment is shown in Figure 2.1. Matlab/Simulink and IPG CarMaker are connected to each other via the CarMaker For Simulink (CM4SL) interface provided by IPG CarMaker.

The automated driving function is fully implemented in Matlab/Simulink. The MPC framework GRAMPC, which is written in C code, is integrated as a Matlab MEX function. This enables its utilization as a standard Simulink blockset within the environment.

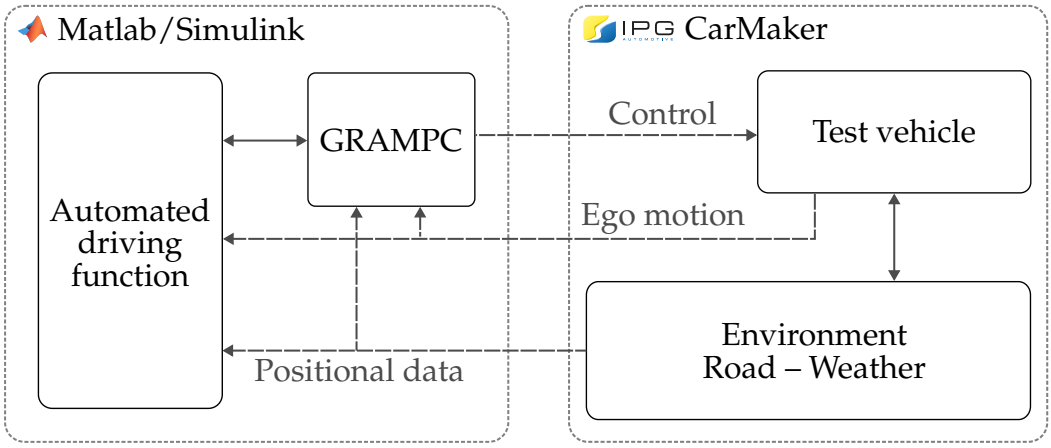


FIGURE 2.1: Schematic illustration of the simulation environment featuring Matlab/Simulink and IPG CarMaker.

2.1.1 Vehicle dynamics simulation

The seamless interaction between Matlab/Simulink and IPG CarMaker provides easy access to all simulation data and full control of the complete simulation environment as well as the test vehicle. This enables the development environment of this thesis to be highly automated regarding its scenario variation and evaluation.

The high-fidelity vehicle dynamics simulation of IPG CarMaker allows development close to a real-world vehicle. The base controller developed in this thesis can be executed on a rapid prototyping platform (RPP). Multiple test vehicles with different powertrain concepts are provided by IPG CarMaker. The simulation studies in this thesis utilize only vehicles with electric drives.

2.1.2 Vehicle motion control interface

The default vehicle motion control interface provided by IPG CarMaker is divided into longitudinal and lateral control. Longitudinal control consists of the pedal positions for the accelerator and the brake, while lateral control consists only of the steering wheel angle. The controller in this thesis provides its control outputs based on single-track conventions, specifically a desired acceleration $a_{x,ctrl}$ for longitudinal control and a desired steering angle $\delta_{v,ctrl}$ for lateral control.

To connect the controller to the vehicle, an additional interface is implemented. The lateral control interface is relatively easy to implement, utilizing only a steering ratio table for the relation between the steering wheel angle and the average steering angle on the front axle.

The tables for longitudinal control use current velocity and desired acceleration as breakpoints, with corresponding pedal positions as the table data. These tables, which cover both positive and negative longitudinal acceleration, are derived from empirical data collected during multiple test runs. This ensures accurate descriptions of pedal positions across the entire range of velocities and accelerations.

2.1.3 GRAMPC framework

As described in [60], the GRAMPC framework is software for nonlinear MPC that can efficiently control nonlinear and highly dynamic systems with sampling times in the (sub)millisecond range. The revised and latest version by Englert et al. [60] is based on the MPC toolbox GRAMPC [61], originally developed for nonlinear systems with pure input constraints.

The underlying optimization algorithm of the revised version is based on an augmented Lagrangian formulation, combined with a real-time gradient method, and features tailored line search and multiplier update strategies optimized for time and memory efficiency. Although sequential quadratic programming (SQP) and interior point methods typically offer superior convergence speed and accuracy, the augmented Lagrangian framework can rapidly provide a sub-optimal solution at low computational costs, which is crucial for real-time

applications and embedded optimization. Sub-optimal solution strategies are frequently utilized in real-time model predictive control [62], [63], [64].

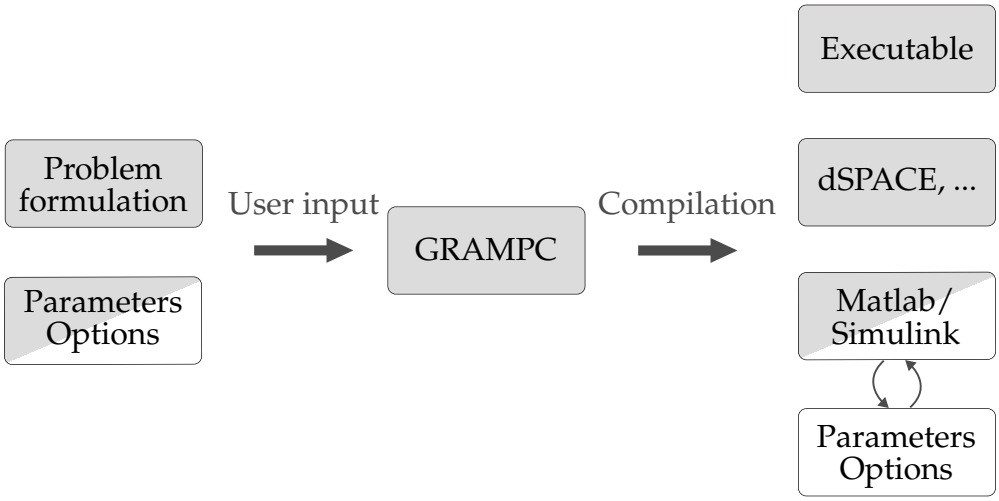


FIGURE 2.2: General structure of GRAMPC (gray: C code, white: Matlab) and build process as described in [60].

Due to its implementation in plain C code and not relying on external libraries, the GRAMPC framework is portable and executable on different operating systems and hardware platforms. This allows for easy integration into C++, Matlab/Simulink, or embedded systems. Figure 2.2 illustrates the general structure of the GRAMPC framework and its build process.

The decision to use GRAMPC as the MPC framework stems from its ability to run as a real-time application in an embedded system. This enables the underlying trajectory tracking MPC to be (partially) utilized for project work with real-world test vehicles on RPPs.

2.2 Automated driving function

This section introduces the automated driving function (ADF) utilized within the simulation environment. The goal of this ADF is to periodically provide a feasible trajectory to the tracking controller. In a real-world automated driving system, such a trajectory is typically derived from sensory data (such as cameras, LiDAR[1], radar), map data (including high-definition environment maps), and ego motion data (information on the vehicle's own motion and dynamics), computed by a trajectory planning layer [65]. For the development of a learning-based model predictive controller for trajectory tracking, the origin of the provided trajectory is irrelevant. Consequently, the ADF simulates a planning layer with a predefined and offline generated full closed-loop circuit trajectory. Therefore, the implementation of a complete planning layer, including sensory components, is unnecessary.

[1]Light Detection and Ranging – a method for determining ranges by targeting an object or surface with a laser and measuring the time for the reflected light to return to the receiver.

2.2.1 Trajectory definition

In this thesis, a trajectory is defined by a fixed number of n nodes. Each node contains a specified set of information as shown in Table 2.1. This information describes a path in a two-dimensional space with temporal dependency.

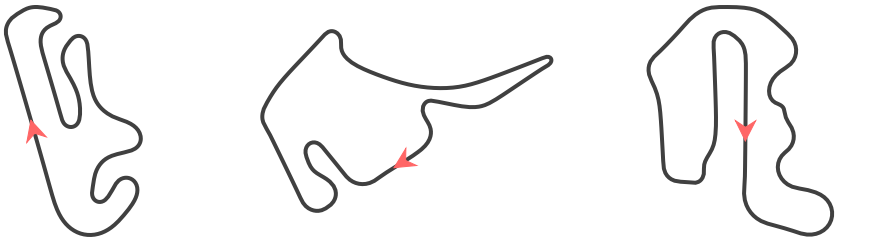
Information	Unit	Description
$x_{t,n}$	m	Position in the x-direction
$y_{t,n}$	m	Position in the y-direction
$s_{t,n}$	m	Cumulative distance to the 1^{st} node
$\psi_{t,n}$	rad	Heading angle
$\kappa_{t,n}$	m^{-1}	Curvature
$v_{t,n}$	$\mathrm{m\,s}^{-1}$	Velocity
$a_{t,n}$	$\mathrm{m\,s}^{-2}$	Acceleration

TABLE 2.1: Information contained by a single node as one of n nodes in a trajectory.

The positional information $x_{t,n}$ and $y_{t,n}$ are based on a global reference frame. The heading angle $\psi_{t,n}$ is determined using the two-argument variant of the arc-tangent function. The velocity $v_{t,n}$ and acceleration $a_{t,n}$ are based on the vehicle's coordinate system along the longitudinal axis.

2.2.2 Real-world closed-loop circuits

To circumvent the need for simulating a complete trajectory planning layer of an automated driving system, predefined and offline generated complete circuit trajectories are utilized within the simulation environment. All trajectories in this thesis originate from real-world closed-loop circuits, as depicted in Figure 2.3.

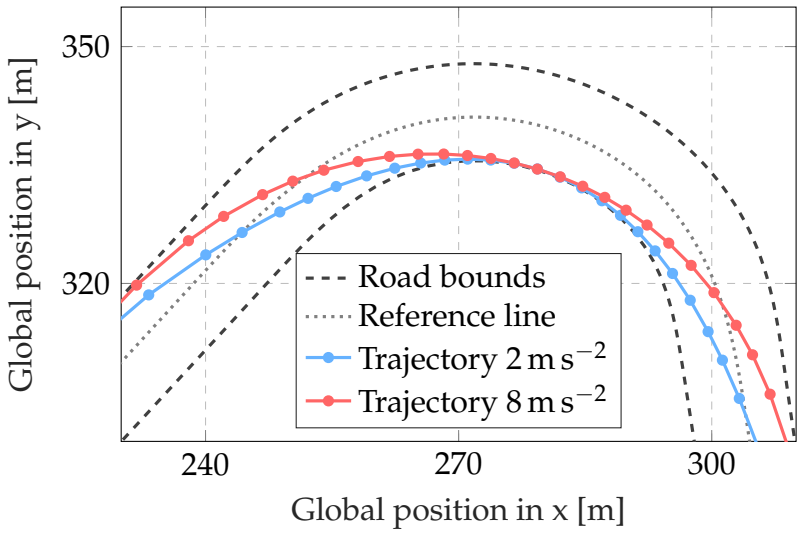


(A) Algarve Inter. Circuit 4.653 km (B) Hockenheimring 4.574 km (C) Thunderhill Raceway 4.454 km

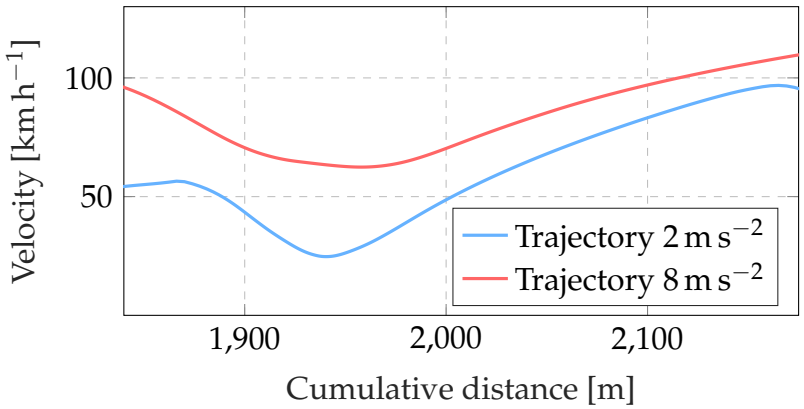
FIGURE 2.3: Three real-world closed-loop circuits whose track boundaries are used to generate complete circuit trajectories.

These trajectories are generated through vehicle model-based optimization, aiming to minimize lap time around a circuit. The optimization problem is constrained by the track boundaries, desired vehicle dynamics, and physical limitations. With the exception of the desired maximum lateral acceleration $a_{y,max}$, all

trajectory variations are generated using the same set of vehicle model parameters and vehicle dynamic constraints. Figure 2.4 exemplifies how the cornering behavior and velocity profile change with varying the maximum lateral acceleration constraint $a_{y,max}$.



(A) Trajectory paths



(B) Velocity profiles

FIGURE 2.4: Illustrations of the resulting paths and velocity profiles for a single corner, varying the maximum lateral acceleration constraint $a_{y,max}$ to 2 m s^{-2} and 8 m s^{-2}.

Figure 2.4 also illustrates that the trajectories are spatially discretized based on curvature, with nodes closer together as curvature increases. The minimum and maximum discretization for the trajectories are set to $\Delta_{s,traj,min} = 0.5\,m$ and $\Delta_{s,traj,max} = 4.0\,m$, respectively.

2.2.3 Planning layer simulation

The structure of the internal planning layer of the ADF is illustrated in Figure 2.5. Here, an offline-generated trajectory $\mathcal{T}_{off}$ is selected and loaded by the

ADF at the beginning of a simulation run.[2] The vehicle's positional data, denoted as $\mathcal{P}_{\text{veh}}$, is provided by IPG CarMaker.

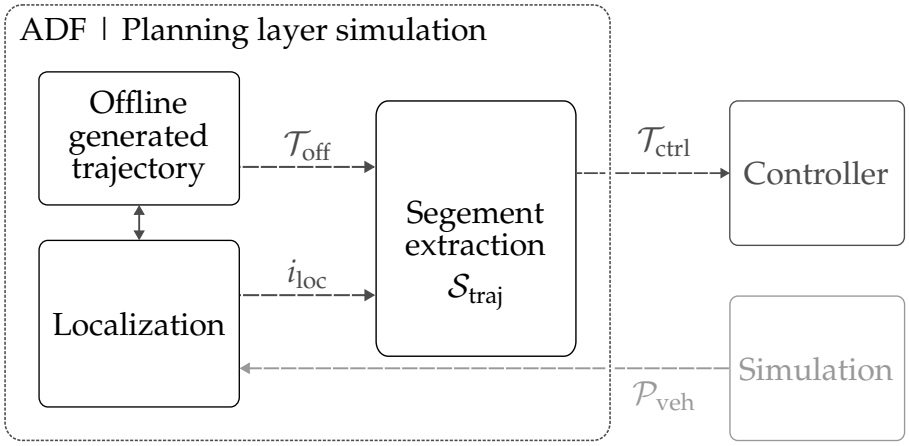


FIGURE 2.5: Illustration of the planning layer emulation as part of the automated driving function within the simulation environment.

The localization method determines the index i_{loc} of the nearest node from the offline trajectory to the current position of the vehicle. This node index and the offline trajectory are provided to the segment extraction method. Based on the node index and further desired trajectory settings $\mathcal{S}_{\text{traj}}$, such as the number of nodes or the minimum time horizon, the final control trajectory $\mathcal{T}_{\text{ctrl}}$ for the tracking controller is determined. This trajectory consists of original nodes from the offline trajectory, inheriting the dynamic spatial discretization applied during the optimization process mentioned in Section 2.2.2.

Figure 2.6 illustrates, as an example, how the control trajectory is extracted from the offline trajectory according to the vehicle's current position. Due to the need for interpolation between nodes at a later stage, the nearest node to the vehicle's current position is never designated as the first node of the control trajectory.

2.2.4 Fixed time discretization

The segment extraction method can also generate a fixed time discrete trajectory when provided with the ego motion data of the vehicle, a desired time horizon T_{hor}, and a delta time d_{t}. A fixed time discrete trajectory offers the advantage that reference points for the desired vehicle position are already established for each optimization step along the time horizon. In contrast, with a spatially discrete trajectory, the reference points must be additionally determined at each optimization step.

[2]The data from the selected offline trajectory is converted and forwarded to IPG CarMaker's scenario editor to create the corresponding road within the simulation environment.

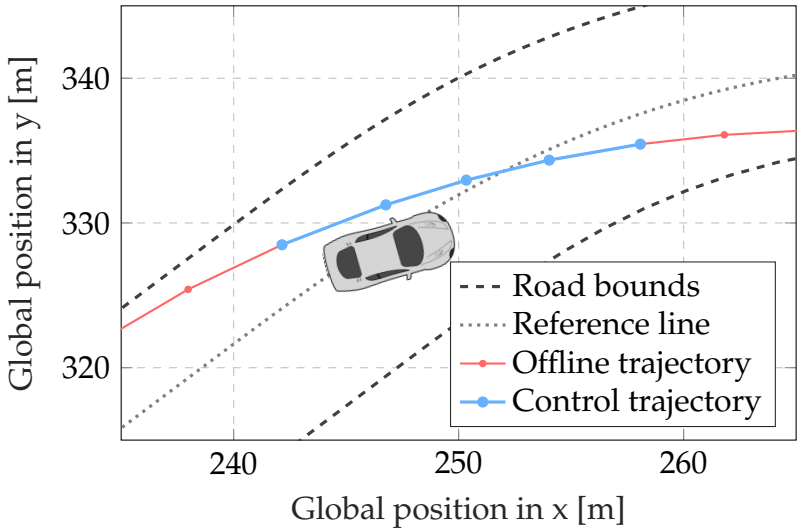


FIGURE 2.6: Visualization of the planning layer simulation providing a control trajectory based on the offline trajectory and the vehicle's current position.

The decision to utilize a spatially discrete trajectory for the controller in this thesis is influenced by the legacy of previous and ongoing work at the *e:fs TechHub GmbH*. Furthermore, the computational overhead associated with this approach is negligible for the scope of this thesis.

Chapter 3

Trajectory tracking control

This chapter introduces the MPC-based controller for the trajectory tracking task, which serves as the baseline controller for the learning-based approaches presented from Chapter 4 onwards. The structure of this chapter is as follows: initially, Section 3.1 introduces the tracking errors and the tracking performance measure. Section 3.2 derives and analyzes the vehicle and tire models applied within the MPC. Finally, Section 3.3 demonstrates how the optimal control problem (OCP) is adapted to the trajectory tracking task and provides implementation details regarding the GRAMPC framework.

3.1 Trajectory tracking

Trajectory tracking involves controlling a vehicle along a desired trajectory while minimizing tracking errors. The tracking errors considered in this thesis include lateral deviation, heading error, and velocity error. To compare different controllers, vehicles, or combinations thereof, a tracking performance metric is introduced. This metric, based on a fixed traveled distance, indicates how accurately a vehicle has traveled along a desired trajectory.

3.1.1 Tracking errors

The positional tracking errors, lateral deviation e_{lat} and heading error e_{ψ}, are illustrated in Figure 3.1. Here, the black dots represent the nodes of a spatially discrete trajectory. To calculate both errors, a reference point P_{ref} on the trajectory must be determined. This determination is made by finding the shortest vector from the vehicle's center of gravity (CoG) that is perpendicular to a tangent on the trajectory. The point where this tangent touches the trajectory is identified as the reference point.

Lateral deviation

The lateral deviation e_{lat} is the distance from the reference point P_{ref} to the vehicle's CoG, represented by the red vector in Figure 3.1. The convention regarding the sign of the lateral deviation in this thesis is as follows: a vehicle that travels left of the trajectory has a negative lateral deviation, whereas the lateral deviation is positive when a vehicle travels right of the trajectory.

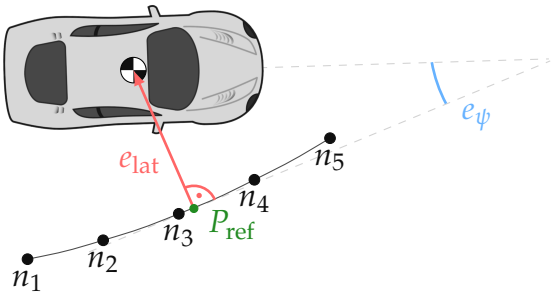


FIGURE 3.1: Illustration of the positional tracking errors: lateral deviation e_{lat} and heading error e_{ψ}. Both errors are calculated based on a determined reference point P_{ref} on the trajectory.

Heading error

The heading error e_{ψ} is the angular difference between the longitudinal axis of the vehicle and the tangent through the reference point on the trajectory, represented by the blue arc in Figure 3.1. Considering the heading error in isolation, without taking into account the lateral deviation, is not a reliable method for assessing tracking accuracy. For example, a vehicle could follow a trajectory without a heading error but still exhibit a significant lateral offset. Conversely, a vehicle might travel with minimal lateral deviation yet incur a substantial heading error, potentially due to the vehicle's current side slip angle.

Velocity difference

The velocity error e_{vx} is the difference between the vehicle's current velocity and the target velocity at the reference point on the trajectory. Because the baseline controller in this thesis operates on spatially discrete trajectories, velocity errors do not directly affect positional tracking accuracy. Theoretically, it is possible to follow a trajectory with a desired velocity offset because the determination of reference points during the optimization process is continuously based on the current position. In contrast, for a time-discrete trajectory, the reference points for each optimization step are predetermined by the velocity, i.e., the time between the nodes. Here, a significant velocity error influences the positional tracking accuracy.

3.1.2 Tracking performance

To compare the tracking accuracy of different controllers, vehicles, or combinations thereof, this thesis adopts the tracking accuracy introduced in [23] and [24]. This performance indicator $P_{\text{e,lat}}$ describes, as a percentage value, how often along a fixed traveled distance the magnitude of the lateral deviation e_{lat} remains within a threshold $e_{\text{lat,threshold}}$.

$$P_{\mathrm{e,lat}} = \frac{\int \chi(|e_{\mathrm{lat}}(t)| \leq e_{\mathrm{lat,threshold}})\, dt}{\int dt} \cdot 100\,\%, \tag{3.1}$$

where $\chi(\cdot)$ denotes the indicator function that returns 1 if the condition in its argument is true and 0 otherwise. The tracking performance is reset after completing a fixed traveled distance. In the simulations in this thesis, this fixed traveled distance always corresponds to the distance of a complete lap on the utilized closed-loop circuit.

The heading error e_ψ and velocity error e_{vx} are not considered for tracking performance. The absence of the heading error is due to the test vehicle's steering characteristic, which tends towards understeering. The trajectory dynamics and changing conditions are not expected to provoke oversteering or side slip angles. Consequently, positional accuracy implicitly reduces angular errors. Additionally, simulations during the development process have shown that the tracking controller consistently follows the desired velocity with an error of less than 0.5 % at all times. Since the condition changes are intended to influence lateral tracking accuracy, the velocity error is also excluded.

3.2 Applied vehicle model

The model predictive controller utilizes a dynamic bicycle model (DBM), also referred to as a single-track or two-wheel vehicle model, to simulate the motion of a real-world vehicle. The model is extended by a nonlinear tire model to enhance its overall accuracy. This combination of models is frequently employed for vehicle motion control, as demonstrated in [66] and [67], among others. Furthermore, this combination is known to offer a good balance between simplicity and accuracy in terms of computational cost and performance [68].

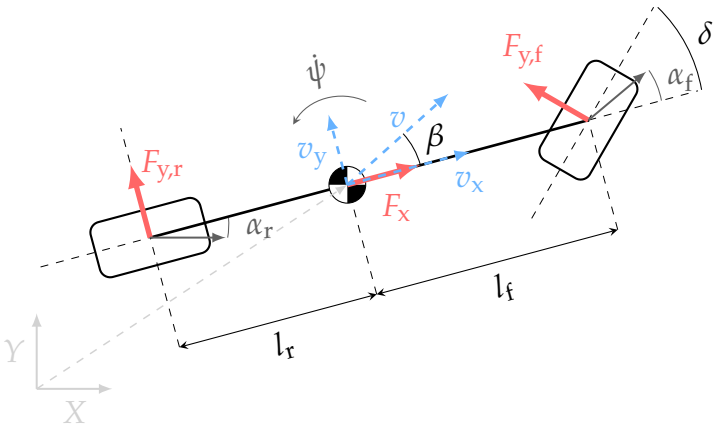


FIGURE 3.2: Illustration of the dynamic bicycle model. The red arrows denote forces, while the blue arrows denote velocities.

3.2.1 Dynamic bicycle model

The dynamic bicycle model considers the vehicle as a rigid body with mass m and moment of inertia J_z around the vehicle's vertical axis. The center of gravity between the front and rear axle is denoted by l_f and l_r, respectively. While only in-plane motion is considered, roll and pitch dynamics, and therefore dynamic tire load changes, are neglected. Figure 3.2 provides a detailed illustration of the bicycle model, where the red vectors represent forces and the blue vectors represent velocities. The steering angle is denoted as δ, the tire slip angles as α_f and α_r, the side slip angle as β and the yaw rate as $\dot{\psi}$. The dashed gray vector represents the position (X, Y) in the global coordinate system. The longitudinal velocity v_x, lateral velocity v_y, and yaw rate $\dot{\psi}$ are referenced to the body-fixed center of gravity.

The complete model is expressed as:

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}), \tag{3.2}$$

where the state is denoted as $\mathbf{x} = \left[X, Y, \psi, \dot{\psi}, v_\mathrm{x}, v_\mathrm{y}\right]^T$ and the input is denoted as $\mathbf{u} = [\delta, D]^T$. The differential equations of the dynamic bicycle model [69] are adapted for the defined state vector $\mathbf{x}$ and are as follows:

$$\dot{X} = v_\mathrm{x}\cos\psi - v_\mathrm{y}\sin\psi, \tag{3.3a}$$
$$\dot{Y} = v_\mathrm{x}\sin\psi + v_\mathrm{y}\cos\psi, \tag{3.3b}$$
$$\dot{\psi} = \dot{\psi}, \tag{3.3c}$$
$$\ddot{\psi} = \frac{1}{J_\mathrm{z}}(F_\mathrm{f,y} l_\mathrm{f}\cos\delta - F_\mathrm{r,y} l_\mathrm{r}), \tag{3.3d}$$
$$\dot{v}_\mathrm{x} = \frac{1}{m}(F_\mathrm{x} - F_\mathrm{f,y}\sin\delta + m v_\mathrm{y}\dot{\psi}), \tag{3.3e}$$
$$\dot{v}_\mathrm{y} = \frac{1}{m}(F_\mathrm{r,y} + F_\mathrm{f,y}\cos\delta - m v_\mathrm{x}\dot{\psi}). \tag{3.3f}$$

where the lateral forces for both axles are denoted as $F_\mathrm{f,y}$ and $F_\mathrm{r,y}$, and the longitudinal force is denoted as F_x. The underlying tire model for the lateral forces is introduced in Section 3.2.3.

3.2.2 Longitudinal model

The longitudinal motion of the vehicle is modeled as a single force:

$$F_\mathrm{x} = f_\mathrm{D} D - f_\mathrm{R} - f_\mathrm{A} v_\mathrm{x}^2, \tag{3.4}$$

applied to the CoG. Here, f_D represents the drive-train coefficient, f_R the roll resistance coefficient, and f_A the air resistance coefficient. The desired driver command, denoted as D, is part of the input vector $\mathbf{u}$ as mentioned earlier. This

command, within the interval $[-1, 1]$, signifies a desired acceleration or deceleration, where 1 corresponds to maximum acceleration and -1 corresponds to maximum deceleration.

3.2.3 Nonlinear tire model

The tire model is the most critical component of the dynamic bicycle model as it characterizes the interaction between the vehicle and the road. The differential equations employ the tire model to represent a single lateral force for each axle. Figure 3.3 demonstrates that the lateral tire force exhibits linearity for small tire slip angles but becomes nonlinear as the slip angles increase. Consequently, to provide a high-fidelity model for high lateral dynamics, a nonlinear tire model must be utilized.

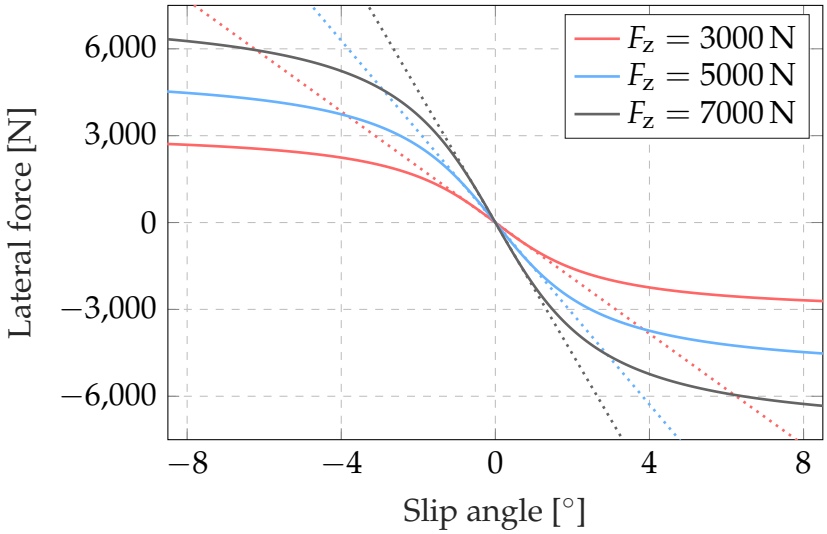


FIGURE 3.3: Plots of lateral tire force F_y over slip angle α for different nominal forces F_z, based on [70]. The dashed lines indicate the linear behavior for small slip angles before becoming nonlinear.

The vehicle model in this thesis employs the Pacejka tire model [71] to simulate the lateral forces of the vehicle motion. These forces are described as follows:

$$F_{y,f} = F_{z,f} \sin(C_f \arctan(B_f \alpha_f)), \tag{3.5}$$

$$F_{y,r} = F_{z,r} \sin(C_r \arctan(B_r \alpha_r)), \tag{3.6}$$

with

$$\alpha_f = \arctan(\frac{v_y + l_f \dot{\psi}}{v_x}) - \delta, \tag{3.7}$$

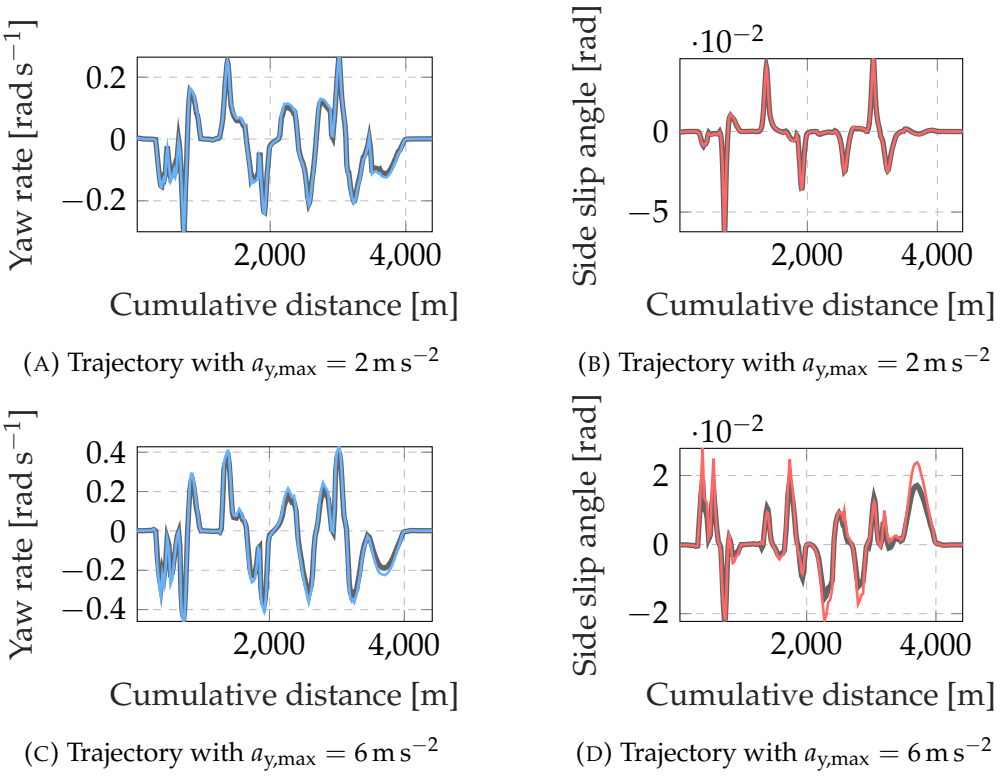
$$\alpha_r = \arctan(\frac{v_y - l_r \dot{\psi}}{v_x}), \tag{3.8}$$

where $F_{z,f/r}$ denotes tire forces in the vertical direction, and $B_{f/r}$ and $C_{f/r}$ represent the tire model coefficients. $\alpha_{f/r}$ denotes the current slip angle of the tire. All quantities are specified for the front and rear axle, respectively.

3.2.4 Model validation

To assess the accuracy of the derived dynamic bicycle model, a comparative analysis is conducted against the vehicle dynamics of the simulation environment. The validation process involves two distinct runs, each following different trajectories on the same closed-loop circuit. One trajectory involves a maximum lateral acceleration of $a_{y,max} = 2\,m\,s^{-2}$ to evaluate the linear tire behavior, while the other trajectory enforces $6\,m\,s^{-2}$ to evaluate the nonlinear behavior.

Figure 3.4 displays the results of both validation runs for the yaw rate (blue) and the side slip angle (red). The colored lines represent the data of the derived model, while the bold gray lines represent the data from the vehicle in the simulation environment.



(A) Trajectory with $a_{y,max} = 2\,m\,s^{-2}$

(B) Trajectory with $a_{y,max} = 2\,m\,s^{-2}$

(C) Trajectory with $a_{y,max} = 6\,m\,s^{-2}$

(D) Trajectory with $a_{y,max} = 6\,m\,s^{-2}$

FIGURE 3.4: Results of model validation for trajectories on the same closed-loop circuit, each with a different maximum lateral acceleration $a_{y,max}$.

The validation results depicted in subplots 3.4a and 3.4b indicate a close match between the derived model and the vehicle behavior observed in the simulation environment. However, at high lateral accelerations, there is a slight

deviation between the model and the simulation environment, as evidenced by the yaw rate in subplot 3.4c and the side slip angle in subplot 3.4d.

3.3 Adapted MPC-based tracking controller

The baseline controller used for the learning-based approaches in this thesis is derived from model predictive control (MPC), which is based on an iterative solution to an optimal control problem. To accommodate the introduced spatially discrete trajectories, the MPC needs to be adapted accordingly. This adaptation results in additional implementation changes within the GRAMPC framework.

3.3.1 MPC formulation

The fundamental concept of MPC, as outlined in [72], involves using a dynamic model to predict system behavior. By optimizing the outcome of a cost function over a finite time horizon, a sequence of control inputs is generated, from which the first value is applied to the controlled system. Consistent with preceding works [23] and [24], the MPC formulation for the tracking controller in this thesis is presented as follows:

$$\min_{\mathbf{u}(\tau)} \quad J(\mathbf{u};\mathbf{x}_\mathrm{k}) = V(\mathbf{x}(T)) + \int_0^T l(\mathbf{x}(\tau),\mathbf{u}(\tau))d\tau \tag{3.9a}$$

$$\text{s.t.} \quad \dot{\mathbf{x}}(\tau) = f(\mathbf{x}(\tau),\mathbf{u}(\tau)), \;\; \mathbf{x}(0) = \mathbf{x}_0, \tag{3.9b}$$

$$g(\mathbf{x}(\tau),\mathbf{u}(\tau)) = 0, \;\; g_\mathrm{T}(\mathbf{x}(T),\mathbf{u}(T)) = 0, \tag{3.9c}$$

$$h(\mathbf{x}(\tau),\mathbf{u}(\tau)) \leq 0, \;\; h_\mathrm{T}(\mathbf{x}(T),\mathbf{u}(T)) \leq 0, \tag{3.9d}$$

$$\mathbf{x}(\tau) \in [\mathbf{x}_\mathrm{min},\mathbf{x}_\mathrm{max}], \tag{3.9e}$$

$$\mathbf{u}(\tau) \in [\mathbf{u}_\mathrm{min},\mathbf{u}_\mathrm{max}], \tag{3.9f}$$

where $\tau \in [0,T]$ represents the internal time range over the prediction horizon T. Here, x_0 denotes the initial system state, g and h represent the equality and inequality constraints, and g_T and h_T denote the terminal equality and inequality constraints. The cost function to be minimized comprises the Mayer term V and the Lagrangian term l, both of which have a quadratic form:

$$V(\mathbf{x}) = \|\mathbf{x}-\mathbf{x}_\mathrm{des}\|^2_\mathbf{P}, \tag{3.10a}$$

$$l(\mathbf{x},\mathbf{u}) = \|\mathbf{x}-\mathbf{x}_\mathrm{des}\|^2_\mathbf{Q} + \|\mathbf{u}-\mathbf{u}_\mathrm{des}\|^2_\mathbf{R}, \tag{3.10b}$$

where $\mathbf{x}_\mathrm{des}$ and $\mathbf{u}_\mathrm{des}$ represent the desired set points, and the weight matrices $\mathbf{P}$, $\mathbf{Q}$, and $\mathbf{R}$ are positive (semi-)definite.

3.3.2 Adaptation to spatial discrete trajectories

Within the cost function, both the Mayer term V (3.10a) and the Lagrangian term l (3.10b) rely on set points $\mathbf{x}_{\text{des}}$ and $\mathbf{u}_{\text{des}}$. These set points must be known a priori for the discretized time horizon. Since the issued controller trajectory is spatially discretized, the desired set point has to be determined at each optimization step when the cost function is evaluated. Therefore, the Mayer and Lagrangian terms are adapted to

$$V(\mathbf{x}) = \|\xi(\mathbf{x}, \Theta)\|^2_{\mathbf{P}}, \tag{3.11a}$$

$$l(\mathbf{x}, \mathbf{u}) = \|\xi(\mathbf{x}, \Theta)\|^2_{\mathbf{Q}} + \|\mathbf{u} - \mathbf{u}_{\text{des}}\|^2_{\mathbf{R}}, \tag{3.11b}$$

where ξ is a function that calculates the lateral tracking errors e_{lat} and e_ψ, as well as the longitudinal error e_{vx}. The second function argument Θ denotes the information for all trajectory nodes.

Additionally, all equality and inequality state constraints are removed from the formulation because the provided trajectory is expected to be obstacle-free and within the physical limits of the vehicle. Therefore, the final MPC formulation has the reduced form:

$$\min_{\mathbf{u}(\tau)} \quad J(\mathbf{u}; \mathbf{x}_{\text{k}}) = V(\mathbf{x}(T)) + \int_0^T l(\mathbf{x}(\tau), \mathbf{u}(\tau)) d\tau \tag{3.12a}$$

$$\text{s.t.} \quad \dot{\mathbf{x}}(\tau) = f(\mathbf{x}(\tau), \mathbf{u}(\tau)), \quad \mathbf{x}(0) = \mathbf{x}_0, \tag{3.12b}$$

$$\mathbf{u}(\tau) \in [\mathbf{u}_{\min}, \mathbf{u}_{\max}]. \tag{3.12c}$$

3.3.3 Implementation

In the GRAMPC framework, the problem formulation follows a generic structure based on a single `probfct.c` file. The implementation of the introduced MPC formulation deviates in some points from the examples given in [60]. Therefore, this subsection provides detailed implementation information.

System function

Listing 3.1 illustrates the implementation of the dynamic bicycle model as the system function within the `probfct.c`. The `out` array represents $\dot{\mathbf{x}}$, and the variables in all capitals represent macros for constant values. These constants serve as indices to retrieve information from the `userparam` array, where all model parameters, constraints, weights, and trajectory information are stored.

The models for the nonlinear tire and the powertrain are implemented as separate functions in lines 11-12, and they write to their corresponding force variables. In addition to the differential equations of the OCP, the GRAMPC framework requires derivatives with respect to the state $\mathbf{x}$ and the control $\mathbf{u}$

to evaluate the optimality conditions [60]. Therefore, both models are also directly implemented into the differential equations to define the Jacobian matrices. This implementation is analogous and not shown here.

```
1  /** System function f(t,x,u,p,userparam)
2      ------------------------------------ **/
3  void ffct(typeRNum *out, ctypeRNum t, ctypeRNum *x, ctypeRNum *
       u, ctypeRNum *p, typeUSERPARAM *userparam)
4  {
5      const double lf = userparam[VEH_PARAM_OFFSET+0];
6      const double lr = userparam[VEH_PARAM_OFFSET+1];
7      const double  m = userparam[VEH_PARAM_OFFSET+2];
8      const double Jz = userparam[VEH_PARAM_OFFSET+3];
9
10     double Fy_f, Fy_r, Fx;
11     nonlinear_tire_model(x, userparam, &Fy_f, &Fy_r);
12     powertrain_model(x, u, userparam, &Fx);
13
14     out[0] = x[4]*cos(x[2]) - x[5]*sin(x[2]);
15     out[1] = x[4]*sin(x[2]) + x[5]*cos(x[2]);
16     out[2] = x[3];
17     out[3] = (Fy_f*lf*cos(u[0]) - Fy_r*lr) / Jz;
18     out[4] = (Fx - Fy_f*sin(u[0]) + m*x[5]*x[3]) / m;
19     out[5] = (Fy_r + Fy_f*cos(u[0]) - m*x[5]*x[3]) / m;
20 }
```

LISTING 3.1: Implementation of the bicycle model in `probfct.c` featuring separate functions for the powertrain and nonlinear tire models.

Cost functions

Listing 3.2 illustrates the implementation of the integral cost function l with dynamic set points. The terminal cost function V is handled similarly. By design, both cost functions are provided with the static set points `xdes` and `udes` for the state $\mathbf{x}$ and the input $\mathbf{u}$, respectively. Since the set points change for each time step along the time horizon, these set points must be constantly adapted.

The code in lines 5-11 parses the weight matrices from the `userparam` array. Line 14 determines the reference point on the trajectory as a cumulative distance s_{ref}. This cumulative distance is then used in lines 15-18 to determine the updated set points `xdes` for the global position X and Y, the heading angle ψ and the longitudinal velocity v_x. The set points for the input vector `udes` remain unchanged (0) to penalize high steering angles and driver commands.

```
1  /** Integral cost l(t,x(t),u(t),p,xdes,udes,userparam)
2      ------------------------------------------------ **/
3  void lfct(typeRNum *out, ctypeRNum t, ctypeRNum *x, ctypeRNum *
       u, ctypeRNum *p, ctypeRNum *xdes, ctypeRNum *udes,
       typeUSERPARAM *userparam)
4  {
5      double Q[6];
6      for (size_t i=Q_WGHT_OFFSET; i<Q_WGHT_OFFSET+6; i++) {
7          Q[i] = userparam[i];
8      }
9      double R[2];
10     R[0] = userparam[R_WGHT_OFFSET];
11     R[1] = userparam[R_WGHT_OFFSET+1];
12
13     // Determine reference values for error calculation
14     const double s_ref = reference_on_traj(userparam, x);
15     xdes[0] = interp_traj(s_ref, userparam, X_TRAJ_OFFSET);
16     xdes[1] = interp_traj(s_ref, userparam, Y_TRAJ_OFFSET);
17     xdes[2] = interp_traj(s_ref, userparam, PSI_TRAJ_OFFSET);
18     xdes[4] = interp_traj(s_ref, userparam, VX_TRAJ_OFFSET);
19
20     out[0] = (Q[0] * POW2(x[0] - xdes[0])
21             + Q[1] * POW2(x[1] - xdes[1])
22             + Q[2] * POW2(x[2] - xdes[2])
23             + Q[3] * POW2(x[3] - xdes[3])
24             + Q[4] * POW2(x[4] - xdes[4])
25             + Q[5] * POW2(x[5] - xdes[5])
26             + R[0] * POW2(u[0] - udes[0])
27             + R[1] * POW2(u[1] - udes[1]));
28 }
```

LISTING 3.2: Implementation of the integral costs in `probfct.c` with dynamic set points.

Chapter 4

Online adaptation of the vehicle model's yaw intensification

This chapter introduces an adaptive model predictive controller (AMPC) designed to adjust the vehicle model's yaw intensification to compensate for lost tracking accuracy resulting from model errors and changing conditions. The approach combines the introduced baseline MPC with an online learning algorithm based on a trajectory-dynamic lookup table. This lookup table is periodically updated according to the trajectory dynamics and the tracking performance of the controller. Beginning with a discussion of current research in Section 4.1, emphasis is placed on the contributions and novelties of the proposed approach. This is followed by the introduction of yaw intensification in Section 4.2 as an important parameter of the vehicle model regarding lateral dynamics. Section 4.3 provides a detailed description of the approach and its learning algorithm. The proposed approach is evaluated with a thorough simulation study in Section 4.4. The chapter concludes with a summary in Section 4.5.

4.1 Literature and contribution

The problem of trajectory tracking control for automated vehicles has witnessed an increased research activity [73], [74]. The objective is for the vehicle to follow a desired trajectory as accurately as possible. Both industry and academia are studying trajectory tracking control, applying various approaches such as pure pursuit, PID control, linear quadratic regulator (LQR), and model predictive control (MPC). Yao et al. [1] provide an overview and discussion of these different control approaches. MPC for trajectory tracking in autonomous vehicles has already been frequently applied, as demonstrated by Du et al. [2], Shen et al. [3], and Lima et al. [4].

Such control algorithms require robust and high-fidelity models to ensure optimal performance. However, during controller synthesis, not all static parameters are known or can be determined. Furthermore, model parameters such as weight, weight distribution, and tire behavior change over time. Another aspect is the changing environmental conditions, such as road friction or crosswinds. To address these dynamic parameters, researchers are exploring

and applying adaptive, optimization-based, and learning-based control algorithms.

Kebbati et al. [41] utilize an improved particle swarm optimization (PSO) algorithm to optimize and tune MPC parameters (weights, horizon, etc.) to cope with varying conditions during a double-lane change. Lin et al. [40] present an MPC that employs a recursive least squares algorithm to estimate tire stiffness and friction coefficients online, also to improve tracking during a double-lane change. Vaskov et al. [42] propose a stochastic nonlinear MPC (SNMPC) that uses a linear tire-force model, with the mean and covariance of the cornering stiffness estimated online. Additionally, Vaskov et al. [43] introduce an SNMPC that learns tire-road friction using a Bayesian approach, where the relationship is modeled as a Gaussian process. Both approaches aim to manage abrupt road condition changes, such as transitions from dry to snowy surfaces. However, all these methods are restricted to specific maneuvers, such as a standard double-lane change.

The goal is to develop an AMPC that minimizes lateral deviation during generic maneuvers. Unlike existing works, the proposed AMPC is capable of handling unknown trajectories. The approach relies on a trajectory-dynamic lookup table, which is adjusted online using trajectory data and the tracking errors of the controller. Based on this acquired data, the parameters of the vehicle model utilized within the MPC are adapted.

A thorough simulation study conducted in a high-fidelity simulation environment confirms the following novelties of the approach: (**i**) the generic nature of the AMPC, which enhances trajectory tracking even for unknown trajectories, and (**ii**) the capability to continuously adapt to new instances of condition changes.

4.2 Introduction of the yaw intensification

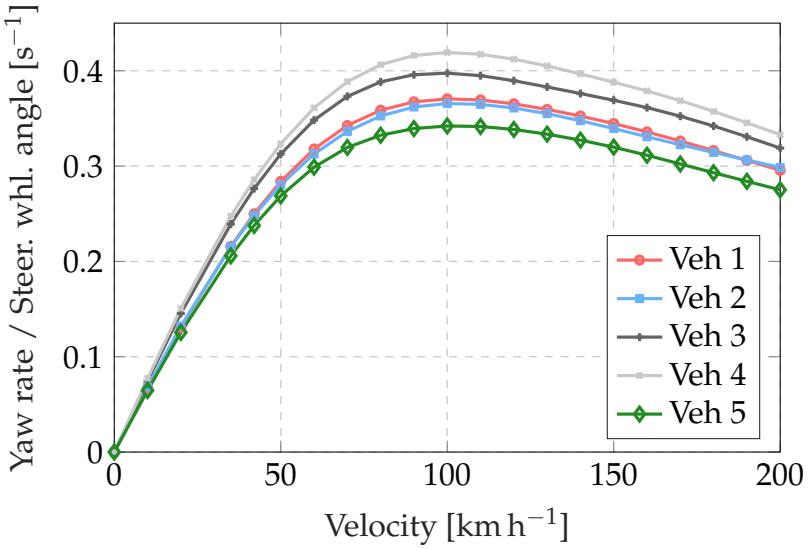
An essential design criterion for a vehicle's steering behavior is its ability to respond adequately to steering inputs at all times [75]. For example, at high velocities, a steering input should not result in high lateral acceleration. To quantify this behavior, the stationary yaw intensification is introduced as:

$$\frac{\dot{\psi}}{\delta} = \frac{1}{l} \frac{v}{1 + \frac{v^2}{v_{\mathrm{ch}}^2}} \tag{4.1}$$

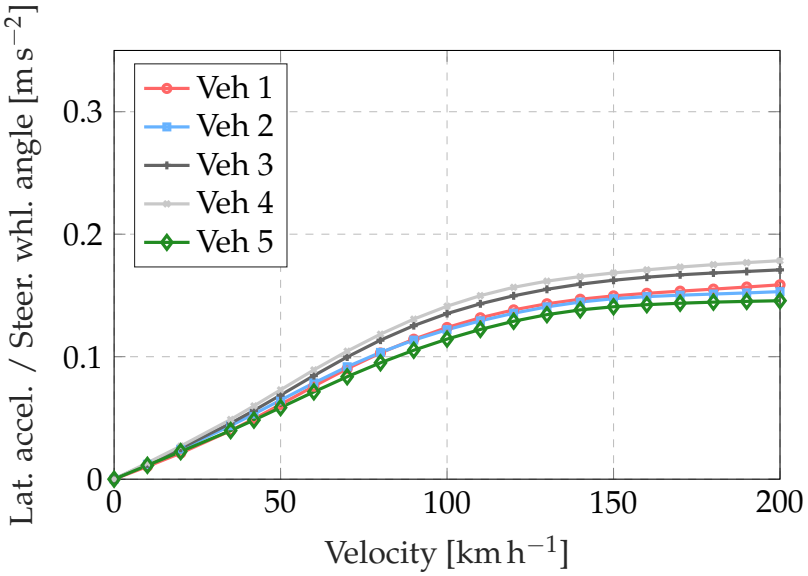
where $\dot{\psi}$ denotes the yaw rate, δ represents the steering angle, v is the velocity, and l is the wheelbase of the vehicle. The characteristic velocity v_{ch}^2 is defined by the vehicle parameters as:

$$v_{\mathrm{ch}}^2 = \frac{c_{\alpha,\mathrm{f}} c_{\alpha,\mathrm{r}} l^2}{m(c_{\alpha,\mathrm{r}} l_{\mathrm{r}} - c_{\alpha,\mathrm{f}} l_{\mathrm{f}})}, \tag{4.2}$$

where $c_{\alpha,\mathrm{f}}$ and $c_{\alpha,\mathrm{r}}$ denote the stiffness of the front and rear axle, respectively, and m is the mass of the vehicle.



(A) Stationary yaw intensification



(B) Stationary steering sensitivity

FIGURE 4.1: Illustration of stationary yaw intensification and stationary steering sensitivity for different vehicles based on steering models. Adapted from [75].

As an example, based on a steering model, Figure 4.1 illustrates the stationary yaw intensification and steering sensitivity for different vehicles. All vehicles exhibit the highest yaw intensification around 90 – 100 km h^{-1} and then it starts to decrease. According to [76], the optimum yaw intensification for a vehicle is at 80 km h^{-1} with approximately 0.3 s^{-1}. Despite the decrease in yaw intensification, lateral acceleration continues to increase. Therefore, yaw intensification essentially describes how much steering angle, with respect to lateral acceleration, is needed to cause a desired yaw motion of the vehicle.

4.3 Learning approach

Typically, a mismatch between an applied vehicle model and a real-world vehicle arises from the tire model, particularly at high dynamics where the behavior becomes nonlinear. To address this, the learning approach of the AMPC includes an algorithm designed to continuously improve the accuracy of the applied vehicle model. The approach relies on driving data that is constantly collected while the AMPC controls the vehicle along the trajectory. This driving data comprises vehicle, trajectory, and tracking information.

The algorithm adjusts the tire stiffness, modifying the parameters of the nonlinear tire model for the front axle (Eq. 3.5) and for the rear axle (Eq. 3.6) to change the yaw intensification.[1] Modifying the tire model changes the characteristic velocity, thereby impacting the yaw intensification. These alterations in yaw intensification influence the vehicle's response to steering inputs. Figure 4.2 provides an overview of the learning algorithm and its interface with the MPC, the planning layer, and the simulation environment. The black connections indicate the most critical data for the learning algorithm.

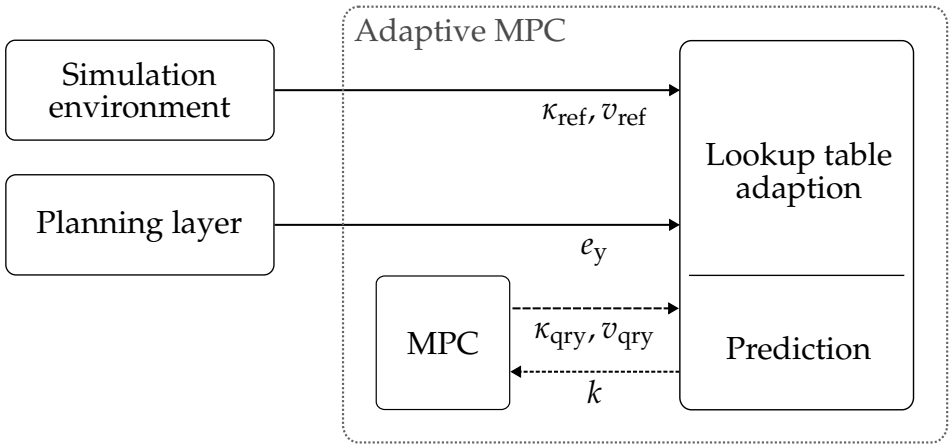


FIGURE 4.2: Overview of the AMPC approach and its interface with the simulation environment and the planning layer, showing only relevant information and connections.

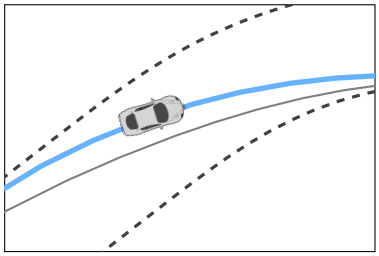
The approach is implemented as a lookup table that is updated at 5 Hz. This table spans the curvature κ_{ref} and the velocity v_{ref} at the reference point as the breakpoints. The curvature and the velocity are predetermined because they implicitly reflect the desired yaw rate and lateral acceleration for the vehicle on the trajectory. The table data is an overall stiffness factor k that influences the ratio between the stiffness of the front and rear axles. Unlike the update rate of the table, the prediction of k is invoked at each optimization of the MPC.

The stiffness factor k is adjusted at each update step based on an educated guess function

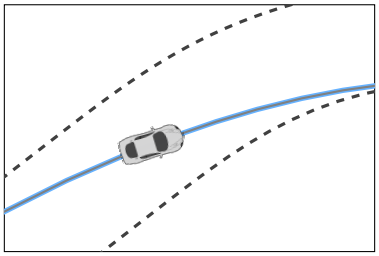
[1] Typically, the stiffness of an axle is defined by the tires, suspension, steering, etc. In this thesis, a single value for the tire stiffness represents an entire axle.

$$k_g = k + \Gamma \frac{|e_y|}{\bar{e}_y} \frac{1}{v_{ref}^2}, \tag{4.3}$$

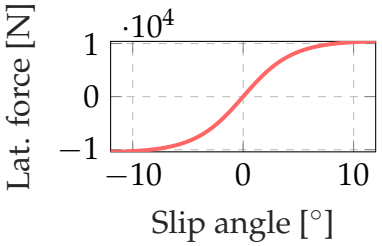
where k and k_g denote the current and updated stiffness factors, respectively. Γ denotes the steering behavior, indicating whether the vehicle travels on a larger (-1) or smaller (1) radius, and thus determining whether k will be increased or decreased. e_y denotes the lateral deviation and $\bar{e}_y$ denotes a tunable base lateral deviation, determining the extent to which the stiffness factor will be adjusted. In this thesis, the tunable base lateral deviation is always set to $\bar{e}_y = 0.2\,\text{m}$.



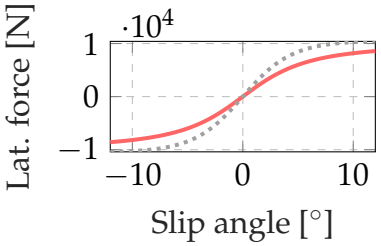
(A) Vehicle travels on a larger radius due to model mismatch.



(B) Vehicle travels on the desired radius due to an adjusted tire model.



(C) Original tire model that does not match the actual vehicle.



(D) Tire model adjusted by k to decrease the yaw intensification.

FIGURE 4.3: Illustration of the basic concept of adjusting the tire stiffness to change the yaw intensification and improve tracking behavior.

Figure 4.3 illustrates the basic concept of the learning algorithm. The top left image (4.3a) depicts a vehicle traveling on a larger radius (blue line) compared to the desired path (gray line) of the trajectory. The bottom left plot (4.3c) shows the currently applied tire model for the front axle. Assuming that the parameters of the applied vehicle model are correct or at least sufficient, the tire model and thus the yaw intensification cause a model mismatch. Due to this model mismatch, the MPC expects a stronger yaw response from the vehicle when optimizing for the steering angle as a control output. In other words, the MPC applies a tire model that is stiffer than the actual vehicle.

The bottom right plot (4.3d) displays a tire model adjusted by k, which reduces the tire stiffness of the front axle. This adjustment shifts the stiffness ratio to the rear and decreases the yaw intensification of the entire vehicle model. Consequently, the MPC outputs an increased steering angle as a control output. The top right image (4.3b) illustrates how the increased steering angle improves the tracking behavior of the vehicle.

To adjust the tire model of the front axle, only the parameter C_f (Eq. 3.5) is influenced by k. This decision is based on experiments that revealed a negligible impact of adapting both axles simultaneously.

4.4 Simulation study

The simulation study for this approach comprises four experiments conducted using a *Tesla Model S* as the test vehicle. Each experiment consists of three scenarios, each taking place on one of the three distinct closed-loop circuits introduced in Section 2.2.2. The trajectories corresponding to these circuits are all generated with a maximum lateral acceleration of $a_{\mathrm{y,max}} = 6\,\mathrm{m\,s^{-2}}$. The parameters for the bicycle model and the nonlinear tire model are provided in Table 4.1.

The first experiment in Section 4.4.1 conducts reference measurements of a condition change using only the baseline MPC as the trajectory tracking controller. The second experiment in Section 4.4.2 repeats the first experiment, but this time with the AMPC enabled. The third experiment in Section 4.4.3 demonstrates how the AMPC handles an additional condition change. In the fourth experiment in Section 4.4.4, an already adapted AMPC is applied to unknown trajectories and evaluated.

For all experiments, the changing conditions are emulated by adding additional weight, which is always placed at the vehicle's center of gravity (CoG). To evaluate and compare the tracking performance of the AMPC, all experiments are conducted in laps. Although the AMPC is not designed for repetitive maneuvers, this is done for the sake of comparability.

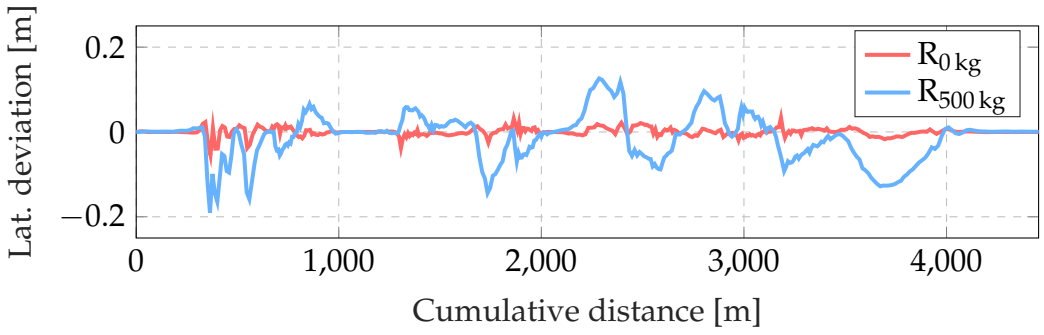
Parameter	Value	Description
l_f	1.47 m	Distance from front axle to CoG
l_r	1.50 m	Distance from rear axle to CoG
m	2108 kg	Total vehicle mass
J_z	4648 kg m^2	Moment of inertia around the z-axis
B_f	9.82	Tire parameter front
C_f	1.33	Tire parameter front
B_r	23.16	Tire parameter rear
C_r	1.07	Tire parameter rear

TABLE 4.1: Applied vehicle parameters of the Tesla Model S for the dynamic bicycle model with nonlinear tire model.

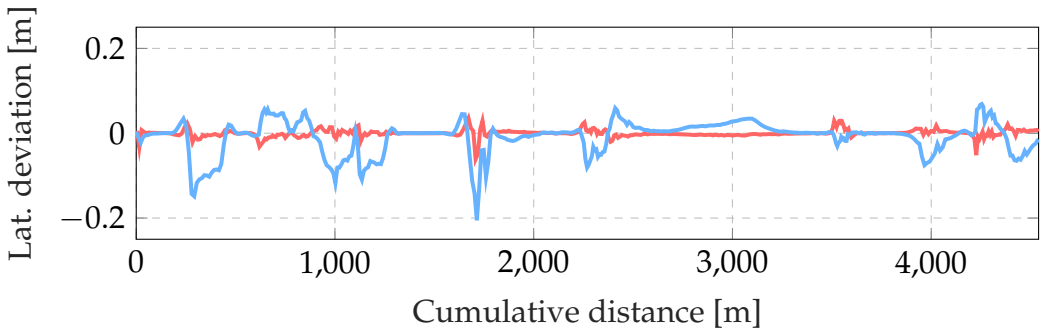
4.4.1 Reference measurements

Prior to evaluating the AMPCs learning approach, the underlying baseline MPC is assessed in three scenarios. Each scenario follows the same procedure but on a different closed-loop circuit. The procedure consists of two laps, during

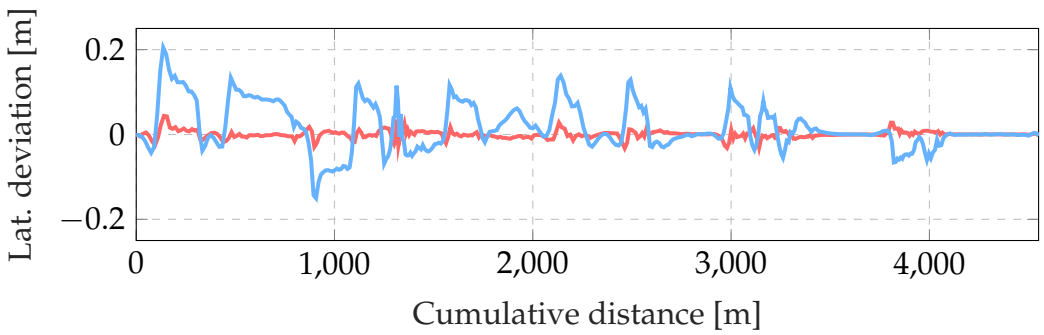
which the vehicle is controlled solely by an untrained AMPC. At the start of the second lap, an additional weight of 500 kg is placed at the vehicle's CoG.



(A) First scenario: Algarve International Circuit



(B) Second scenario: Hockenheimring



(C) Third scenario: Thunderhill Raceway

FIGURE 4.4: Results from all three scenarios of the first experiment: The first lap (red) was conducted under normal conditions, while the second lap (blue) involved the vehicle being affected by an additional weight of 500 kg.

Figure 4.4 illustrates the lateral deviation over the cumulative distance for all three scenarios. The red data represents the results for the first lap under regular conditions, while the blue data represents the results for the second lap with the vehicle affected by additional weight. In all three scenarios, the plots indicate a significant increase in lateral deviation.

Applying the introduced tracking performance metric $P_{e,lat}$ (Eq. 3.1) reveals more detailed results, which are listed in Table 4.2. Under regular conditions, the tracking performance exceeds 95 % in all scenarios. However, when affected

by a condition change, the performance drops by more than 50 % in two out of the three scenarios. Additionally, the maximum lateral deviation $e_{\text{lat,max}}$ increases by at least a factor of 2 in all scenarios.

These results demonstrate that a mismatch between the vehicle model used within the MPC and the actual vehicle leads to less accurate tracking behavior, resulting in deteriorated tracking performance.

Scenario	Lap	$P_{\text{e,lat}}$	$e_{\text{lat,max}}$
1st Algarve	1	96 %	0.08 m
	2	42 %	0.19 m
2nd Hockenheim	1	95 %	0.07 m
	2	59 %	0.21 m
3rd Thunderhill	1	95 %	0.05 m
	2	40 %	0.21 m

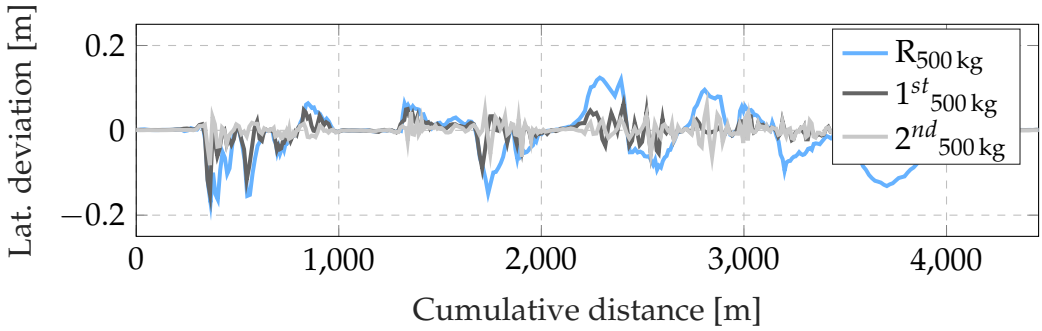
TABLE 4.2: Detailed results for Figure 4.4: $P_{\text{e,lat}}$ denotes the tracking performance and $e_{\text{lat,max}}$ represents the maximum lateral deviation.

Scenario	Lap	$P_{\text{e,lat}}$	$e_{\text{lat,max}}$
1st Algarve	1	42 %	0.19 m
	2	77 %	0.15 m
	3	86 %	0.08 m
2nd Hockenheim	1	59 %	0.21 m
	2	82 %	0.15 m
	3	90 %	0.08 m
3rd Thunderhill	1	40 %	0.21 m
	2	71 %	0.16 m
	3	83 %	0.08 m

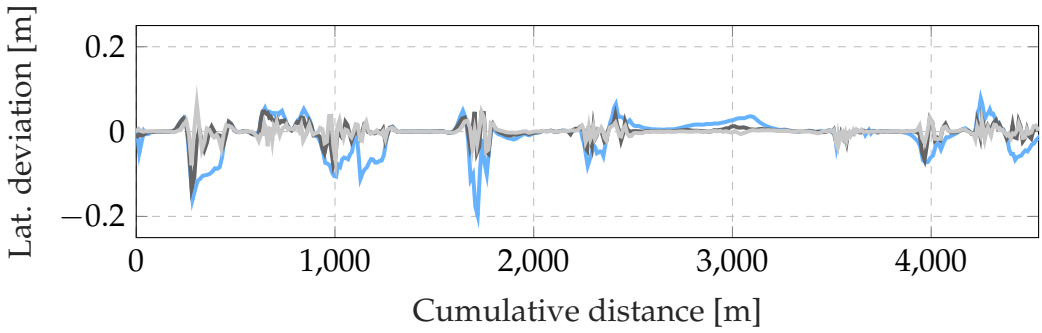
TABLE 4.3: Detailed results for Figure 4.5: $P_{\text{e,lat}}$ denotes the tracking performance and $e_{\text{lat,max}}$ represents the maximum lateral deviation.

4.4.2 Adapting to condition changes

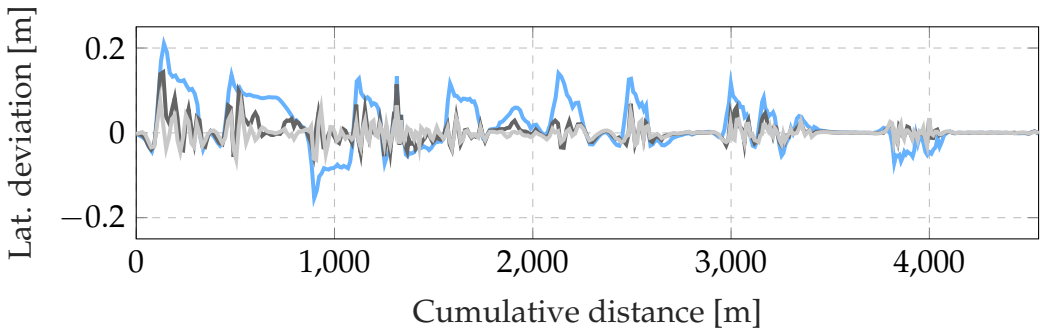
The second experiment aims to demonstrate the adaptability of the AMPC to changing conditions. The procedure consists of three laps, all with the vehicle affected by an additional weight of 500 kg. At the beginning of the second lap,



(A) First scenario: Algarve International Circuit



(B) Second scenario: Hockenheimring



(C) Third scenario: Thunderhill Raceway

FIGURE 4.5: Results of all three scenarios from the second experiment. All laps are conducted with an additional weight of 500 kg. The AMPC was disabled for the first lap (blue) and then enabled for the second and third lap (dark and light gray).

the AMPC is enabled for the remaining two laps to adjust the MPC model and compensate for the model mismatch.

Figure 4.5 presents the results of the second experiment for all three scenarios. The legend entries are ordered according to the laps. Comparing the data from the first lap (blue) and the third lap (light gray) indicates that the losses in tracking performance are largely compensated after two laps.

The detailed results are listed in Table 4.3. In all three scenarios, the losses in tracking performance are recovered by at least 30 %, with a regained tracking performance between 80–90 %. Additionally, the maximum lateral deviation is reduced by over 50 % in all scenarios.

A closer examination of the start of the second lap (dark gray), representing the first lap with the enabled AMPC, reveals the immediate impact of the proposed learning approach on tracking accuracy. In all three scenarios, this is evidenced by the reduced lateral deviation after the initial small peak or corner (since straights are followed quite accurately, peaks of lateral deviation essentially represent corners). The relatively consistent trend in lateral deviation towards the end of the laps suggests significant adaptation to the model mismatch after just one lap. Overall, the second learning lap is primarily necessary to ensure a complete lap after the adaptation for comparison purposes.

4.4.3 Repeatedly adapting to condition changes

The third experiment extends the second experiment to evaluate the AMPCs ability to repeatedly adapt to condition changes. This experiment begins with the third lap of the second experiment, which is the second lap with the AMPC enabled. After completing this lap, the additional weight is reduced back to 0 kg to simulate another condition change. The AMPC then conducts two more laps to adjust the vehicle model to the new situation.

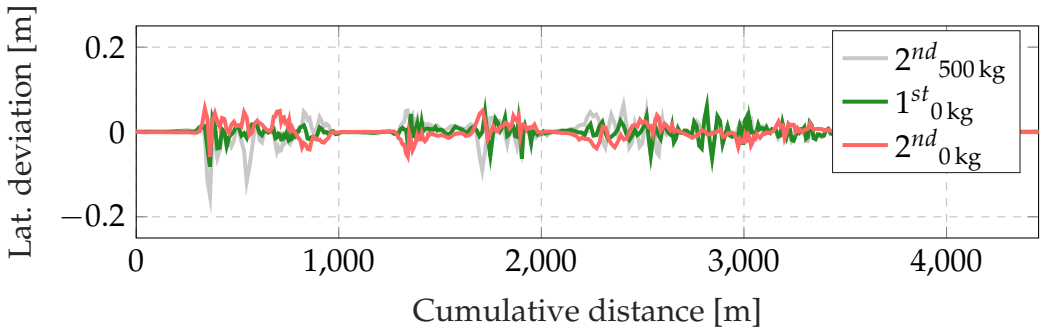
Figure 4.6 displays the lateral deviation for all three scenarios. As per the procedure, the light gray data is identical to the data from the third lap in Figure 4.5. The green and red data represent the adaptation to the second condition change. Detailed results are listed in Table 4.4.

Scenario	Lap	$P_{e,lat}$	$e_{lat,max}$
1st Algarve	1	86 %	0.08 m
	2	80 %	0.08 m
	3	95 %	0.07 m
2nd Hockenheim	1	90 %	0.08 m
	2	84 %	0.09 m
	3	93 %	0.07 m
3rd Thunderhill	1	83 %	0.08 m
	2	79 %	0.07 m
	3	93 %	0.05 m

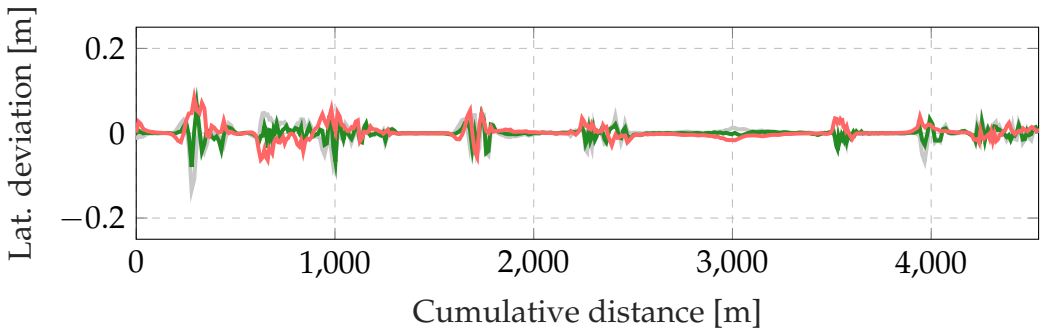
TABLE 4.4: Detailed results for Figure 4.6: $P_{e,lat}$ represents the tracking performance and $e_{lat,max}$ indicates the maximum lateral deviation.

The results show a slight decrease in tracking performance during the second lap when the vehicle is affected by another condition change. However, after the third lap, which includes two laps of adaptation, the tracking performance has increased to above 90 % in all three scenarios. Comparing this to the first reference laps (red) in Figure 4.4, the tracking performances are only off

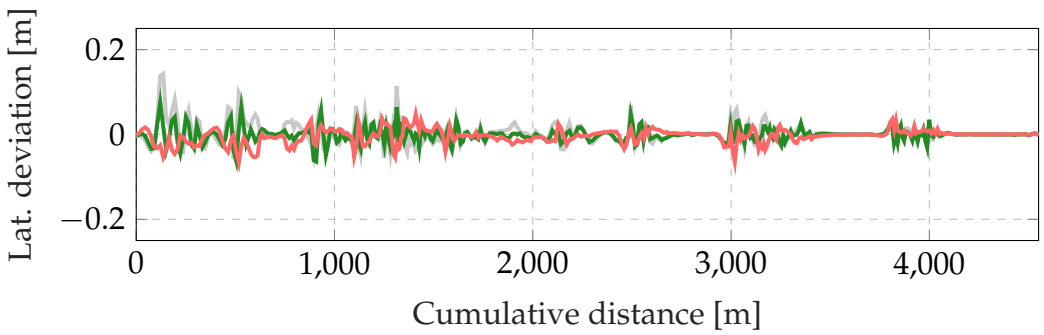
by 1 – 2 %. This demonstrates that the AMPC is capable of compensating for a model mismatch by adjusting the vehicle model accordingly.



(A) First scenario: Algarve International Circuit



(B) Second scenario: Hockenheimring



(C) Third scenario: Thunderhill Raceway

FIGURE 4.6: Results of all three scenarios from the third experiment. The AMPC re-adapts to a second condition change at the start of the second lap (green). After the second lap of adaptation (red), the lateral deviation is close to the desired lateral deviation of the reference lap with 0 kg in Figure 4.4.

4.4.4 Applying adapted AMPC to unknown trajectories

To evaluate the applicability of learned behavior to unknown trajectories, the AMPC is trained on one circuit and then applied for one lap to all three circuits. This procedure is repeated for the three differently trained AMPCs.

The results are listed in Table 4.5. For all evaluation laps, the AMPC does not adapt further but only utilizes the learned behavior.

Applied / *Trained*	*Algarve*	*Hockenheim*	*Thunderhill*
Algarve	**86 %**	75 %	82 %
Hockenheim	84 %	**90 %**	86 %
Thunderhill	71 %	81 %	**83 %**

TABLE 4.5: Resulting tracking performances $P_{e,lat}$ for AMPCs trained on individual circuits and applied to all circuits for one lap with an additional weight of 500 kg. The bold numbers indicate the results of the AMPC trained on each respective circuit.

As expected, the AMPC achieves the highest tracking performances on the circuits where it was trained, as indicated by the bold results in the table's diagonal. The results for the remaining circuit permutations, compared to the highest performances, vary from 2 % (Hockenheim applied to Thunderhill) up to 14 % (Algarve applied to Thunderhill). Comparing these results to the reference measurements in Table 4.2, the AMPC is able to recover at least 30 % of tracking performance for all unknown trajectory permutations.

4.5 Summary and conclusion

As detailed in the simulation study, experiments 1 through 3 were conducted sequentially. To highlight their correlation and offer a comprehensive overview, Figure 4.7 illustrates a summary of all conducted laps on a timeline for one scenario. The condition changes are indicated at the top by the amount of additional weight added to the vehicle's CoG. Subsequently, the state of the AMPC is depicted, indicating whether the learning algorithm was enabled or not. The histogram shows the resulting tracking performances for each lap.

Overall, the proposed AMPC effectively manages condition changes and significantly improves tracking performance (2^{nd} Experiment). The simulation results also demonstrate that the AMPC can repeatedly adapt to condition changes (3^{rd} Experiment). Furthermore, the generic nature of the approach enables the AMPC to handle unknown trajectories and still achieve a significant improvement in tracking accuracy (4^{th} Experiment).

The efficacy of using trajectory dynamics and tracking errors in conjunction with an MPC for trajectory tracking has been demonstrated, but there remains significant potential for enhancement. One area of improvement is the update mechanism of the lookup table based on collected data. High lateral deviation events often indicate a fundamental model mismatch due to condition changes. Instead of merely adapting specific data points, the algorithm could extrapolate from multiple adjacent entries in the lookup table. This proactive approach would enable the prediction and prevention of potential mismatches, leading to more rapid improvements in tracking performance and ensuring the system remains robust and accurate.

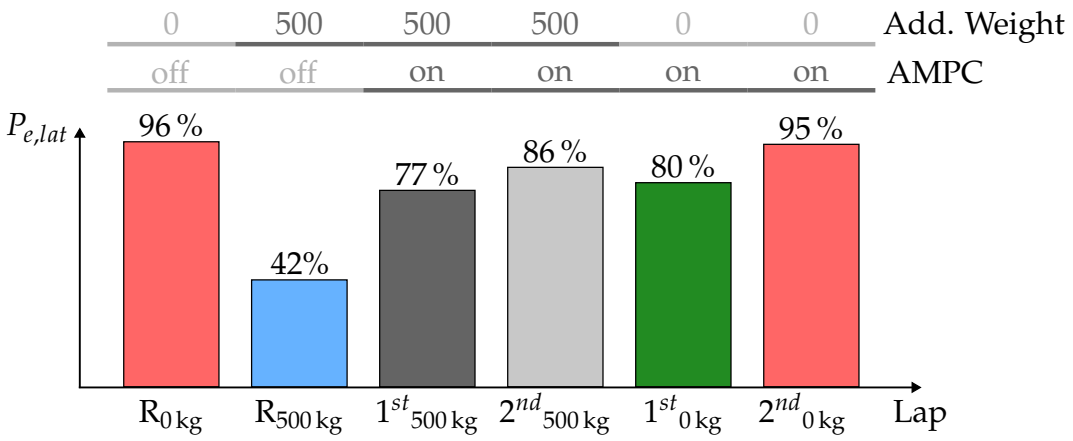


FIGURE 4.7: Combined summary of Experiments 1 to 3 on the Algarve International Circuit (1^{st} scenario). The lap names and colors correspond to the labels and colors used in Figures 4.4, 4.5, and 4.6.

Moreover, advancing beyond the reliance on a lookup table for approximation could involve integrating Gaussian process regression (GPR) or similar techniques. GPR would facilitate the incorporation of additional dimensions, enabling the learning algorithm to recall previously encountered conditions. This would significantly reduce the need for re-adaptation with each new condition change, offering a more robust and efficient system. The advanced method described in Chapter 5 addresses these enhancements, aiming to retain adapted behaviors while further refining model adaptation.

Chapter 5

Online adaptation using Gaussian process regression and clustering

This chapter introduces an advancement of the proposed AMPC from Chapter 4. This advanced AMPC also adapts the vehicle model's yaw intensification to compensate for lost tracking accuracy caused by model errors and condition changes. However, the approach combines the baseline MPC with an online learning algorithm based on Gaussian process regression (GPR) and clustering. The learning algorithm is provided with periodically acquired observation data on the trajectory dynamics and the tracking performance of the controller. Section 5.1 emphasizes the contribution and novelties of the proposed approach. It is followed by the introduction of Gaussian process regression in Section 5.2 and clustering in Section 5.3. Section 5.4 provides a detailed description of the approach and its learning algorithm. The proposed approach is evaluated through a thorough simulation study in Section 5.5 and is simultaneously compared to the first AMPC approach. This chapter concludes with a summary and conclusion in Section 5.6.

5.1 Literature and contribution

As discussed in Section 4.1, control algorithms like MPC require robust and high-fidelity models to ensure their performance. Model mismatch may occur during controller synthesis due to unknown or indeterminable parameters. Over time, parameters such as weight distribution and tire conditions may change. Additionally, environmental factors like wind and road conditions can also lead to model mismatch.

To address these problems, optimization-based and learning-based control algorithms are being researched and applied. The approaches of Kebbati et al. [41], Lin et al. [40], and Vaskov et al. [42], [43] have been previously discussed. However, all these approaches are restricted to specific maneuvers. The proposed approach in Chapter 4 has already addressed this limitation. It is capable of adapting repeatedly to changing conditions, although it does not retain previously adapted behavior.

The goal of this chapter is to revamp the previous approach by replacing the learning algorithm based on a lookup table with a GPR combined with a clustering algorithm. The new approach retains all the features of the previous

learning algorithm while also offering improvements such as memorization, faster adaptation of the applied vehicle model, and overall improved tracking performance.

MPC in combination with GPR has been discussed in the literature. In 2004 Kocijan et al. [25] applied Gaussian processes to model-based predictive control to highlight areas of the input space where prediction quality is poor due to a lack of data or its complexity. The approach was demonstrated on an exemplary nonlinear system. Hewing et al. [37] presented an MPC approach that integrates a nominal system with an additive nonlinear part of the dynamics, modeled as a Gaussian process (GP). Jain et al. [44] utilized Gaussian processes to correct model mismatch, aiming to converge their extended kinematic model to match the actual vehicle dynamics. Both approaches employed remote-controlled cars at a 1:10 scale in their simulations. Kabzan et al. [33] formulated an online learning data-driven MPC using GPR to account for residual model uncertainty and achieve safe driving behavior. They tested their approach on a full-size AMZ driverless race car (Formula Student). All these approaches utilize GPR to improve model quality with observation data. However, the MPCs used in these studies are primarily implemented to improve lap times rather than accurately follow a desired trajectory. In contrast to this thesis, all approaches are applied to a single condition or state of vehicle model error.

With a thorough simulation study in the high-fidelity simulation environment, the following novelty of the approach is verified: (**i**) the capability to memorize experienced condition changes and react accordingly when encountered again. Furthermore, the study demonstrates that the approach exhibits enhanced performance in adapting to new instances of condition changes compared to the initial approach.

5.2 Gaussian process regression

Gaussian process regression, also known as *kriging*, is a non-parametric Bayesian approach used for regression tasks in machine learning and statistics. Due to its Bayesian framework, which allows expressing uncertainty in terms of probability distributions, GPR is valuable when dealing with complex, nonlinear relationships between variables or when the underlying data distribution is not well defined.

GPR utilizes Gaussian processes, which are defined as follows:

Definition 1 (Gaussian process)**.** *A Gaussian process is a collection of random variables, any finite number of which have a consistent joint Gaussian distribution [77].*

A Gaussian process is fully described by a mean function $m(x)$ and a covariance function $k(x, x')$. This is analogous to a Gaussian distribution with a mean vector and a covariance matrix. Therefore, a GP is generally expressed as:

$$g(x) \sim \mathcal{GP}(m(x), k(x, x')), \tag{5.1}$$

which implies: „g is distributed as a GP with the mean function $m(x)$ and the covariance function $k(x, x')$“.

5.2.1 Prior

A GP with only the mean function m and the covariance function k defines a prior distribution. This prior is subsequently refined into a posterior distribution through the inclusion of observation data. To exemplify a prior, consider a GP given by $g \sim \mathcal{GP}(m, k)$ with

$$m(x) = 0, \tag{5.2a}$$

$$k(x, x') = \exp(-\frac{1}{2}(x - x')^2), \tag{5.2b}$$

where $k(x, x')$, also known as the kernel function, represents the Squared Exponential (SE) covariance function. This function is consistently used throughout this thesis and in all implementations. Additional covariance functions are discussed in [77], Chapter 4.

For a finite number n of distinct locations, the mean and covariance functions from Equation 5.2 can be evaluated, thereby defining a Gaussian distribution as follows:

$$g \sim \mathcal{N}(\mu, \Sigma), \tag{5.3}$$

with

$$\mu_i = m(x_i) = 0, \tag{5.4a}$$

$$\Sigma_{i,j} = k(x_i, x_j) = \exp(-\frac{1}{2}(x_i - x_j)^2), \qquad i, j = 1, \ldots, n, \tag{5.4b}$$

where μ and Σ also represent the mean and covariance functions, respectively, to distinguish between the process and the distribution. A random vector can now be generated from this distribution with $g(x_*)$. Figure 5.1 illustrates samples from a GP prior for three random vectors. Due to the absence of observation data, the prior uncertainty regarding the underlying function remains constant.

5.2.2 Posterior

Although the GP prior is independent of observations, it still possesses certain properties. For instance, the samples depicted in Figure 5.1 exhibit a smoothness resulting from the defined kernel. The goal of the GP posterior is to refine the prior using observation data (training data), thus facilitating predictions of the underlying function for unseen query points (test points).

Let $\mathbf{g}$ be the vector of the training data outputs and $\mathbf{g}_*$ be the vector of test data outputs, with X representing the training data inputs and X_* representing the test data inputs. According to the prior, the joint distribution follows as

$$\begin{bmatrix} \mathbf{g} \\ \mathbf{g}_* \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu} \\ \boldsymbol{\mu}_* \end{bmatrix}, \begin{bmatrix} \Sigma & \Sigma_* \\ \Sigma_*^T & \Sigma_{**} \end{bmatrix} \right), \tag{5.5}$$

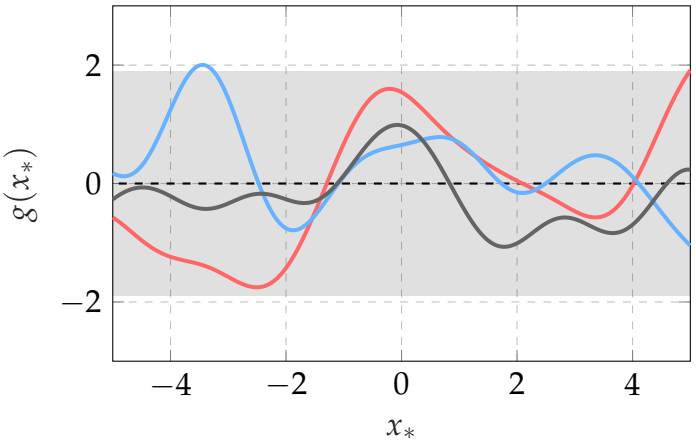


FIGURE 5.1: Samples from a GP prior are depicted by the colored solid lines. The dashed line represents the mean function, while the gray area indicates the 95 % confidence interval of the Gaussian distribution.

where $\boldsymbol{\mu}$ and $\boldsymbol{\mu}_*$ denote the means for the training and test data, Σ denotes the covariance matrix for the training data, Σ_* denotes the covariance matrix for the training-test data, and Σ_{**} denotes the covariance matrix for the test data.[1] Since the training data is known, the conditional distribution of $\mathbf{g}_*$ given $\mathbf{g}$ can be expressed as

$$\mathbf{g}_*|\mathbf{g} \sim \mathcal{N}(\boldsymbol{\mu}_* + \Sigma_*^T\Sigma^{-1}(\mathbf{g}-\boldsymbol{\mu}), \Sigma_{**} - \Sigma_*^T\Sigma^{-1}\Sigma_*). \tag{5.6}$$

The function values $\mathbf{g}_*$ corresponding to the test data inputs X_* can be sampled from the joint posterior distribution.[2] Figure 5.2 illustrates three samples from a GP posterior with five observations for an underlying function $y = x\sin x$ (dotted line).

5.2.3 Noise

In realistic modeling situations, the observation data is not gathered as function values $\mathbf{g}(X)$ but rather as measurements with noise, as in $\mathbf{y} = \mathbf{g}(X) + \epsilon$. Assuming additive, independent, and identically distributed Gaussian noise ϵ with variance σ_n^2, the joint distribution of the observation data and the test data inputs changes to

$$\begin{bmatrix}\mathbf{y}\\ \mathbf{g}_*\end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix}\boldsymbol{\mu}\\ \boldsymbol{\mu}_*\end{bmatrix}, \begin{bmatrix}\Sigma + \sigma_n^2 I & \Sigma_*\\ \Sigma_*^T & \Sigma_{**}\end{bmatrix}\right). \tag{5.7}$$

5.2.4 Hyperparameters

Covariance functions are accompanied by free parameters known as hyperparameters. For the SE covariance function (Eq. 5.4), it has the following form

[1] $\Sigma = k(X,X)$, $\Sigma_* = k(X,X_*)$, $\Sigma_{**} = k(X_*,X_*)$.

[2] A method for sampling from the joint posterior distribution is described in [77], App. 2.

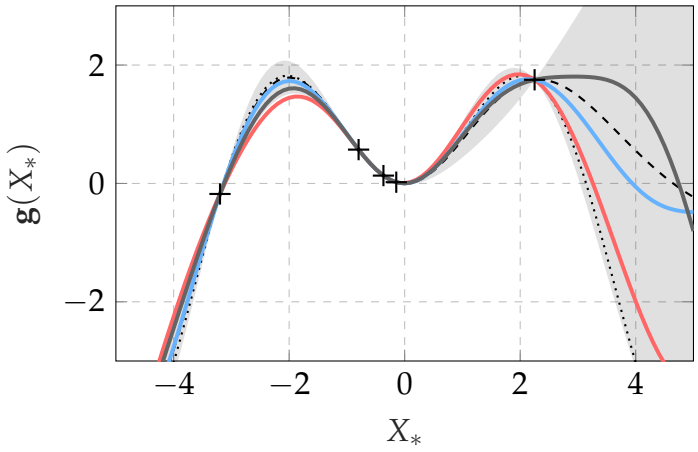
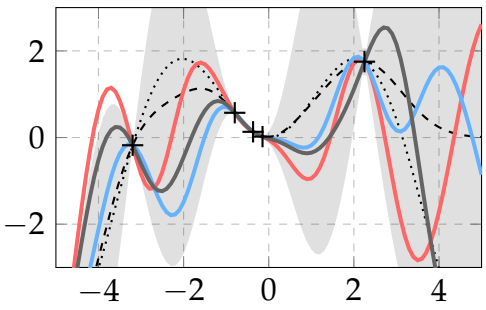


FIGURE 5.2: Samples from a GP posterior with five observations marked by the black crosses. The colored solid lines represent the samples. The dashed line represents the mean function. The gray area represents the 95 % confidence interval of the Gaussian distribution. The dotted line represents the underlying function.

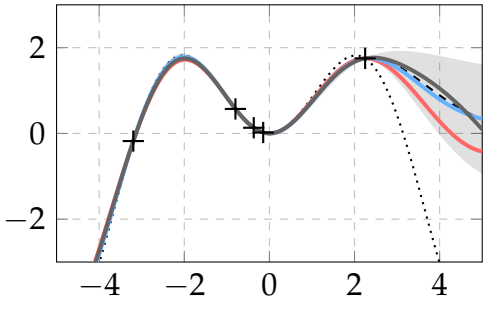
in one dimension:

$$\Sigma_{i,j} = k(x_i, x_j) = \sigma_f^2 \exp(-\frac{1}{2l^2}(x_i - x_j)^2) + \sigma_n^2 I, \tag{5.8}$$

where l^2 denotes the length scale, σ_f^2 denotes the signal variance, and σ_n^2 denotes the noise variance introduced earlier. Figure 5.3 shows, as an example, how the length scale and signal variance influence the GP posterior and, consequently, the sampling from it. For a more detailed discussion on the effects of the hyperparameters, refer to [77], Chapter 2.



(A) Length scale $\frac{1}{2}l^2$



(B) Signal variance $\frac{1}{4}\sigma_f^2$

FIGURE 5.3: Illustrations to emphasize the impact of the hyperparameters on sampling from a GP posterior. The length scale is changed by $\frac{1}{2}$ and the signal variance by $\frac{1}{4}$ compared to the GP posterior in Figure 5.2.

Because of the influence of hyperparameters, this thesis employs the *log marginal likelihood* method to optimize these hyperparameters when the training data is updated. The log marginal likelihood is defined as:

$$\log p(\mathbf{y}|X) = -\frac{1}{2}\mathbf{y}^T(K+\sigma_n^2 I)^{-1}\mathbf{y} - \frac{1}{2}\log|K+\sigma_n^2 I| - \frac{n}{2}\log 2\pi. \tag{5.9}$$

In [77], Chapter 5 introduces various other methods for optimizing hyperparameters. Additionally, [78] covers the use of Gaussian processes for reinforcement learning (RL) and provides a well-written introduction to GPR.

5.3 Clustering

Data clustering techniques are used in data analysis and machine learning to group data points based on specific characteristics or features. The primary objective is to partition data into subsets (or clusters), where the data points within each cluster are more similar to each other than to those in other clusters. These clustering methods aid in revealing underlying structures or patterns within unlabeled data. Due to its wide-ranging applications in learning, summarization, and segmentation, data clustering has been extensively addressed in the literature on data mining and machine learning (e.g., [79], [80]).

An overview of common techniques used in cluster analysis is provided in [81], which includes references to books and surveys that discuss these methods in more detail. These techniques encompass feature selection methods, probabilistic and generative models, as well as distance-based algorithms. The method utilized in this thesis employs a distance-based algorithm due to its simplicity and ease of implementation for various problems. Furthermore, these algorithms can be used with almost any type of data when an appropriate distance function is derived. Consequently, the design of the distance function has become a separate research area, as discussed in [82].

Generally, these algorithms are divided into flat and hierarchical types. Flat types divide data points into several clusters in a single iteration using partitioning representatives. Such partitioning representatives and distance functions are crucial to the underlying algorithm because they regulate its behavior. Hierarchical types represent the clusters through a dendrogram[3], illustrating the arrangement of clusters formed at different levels of granularity. The hierarchical representation is either created top-down or bottom-up.

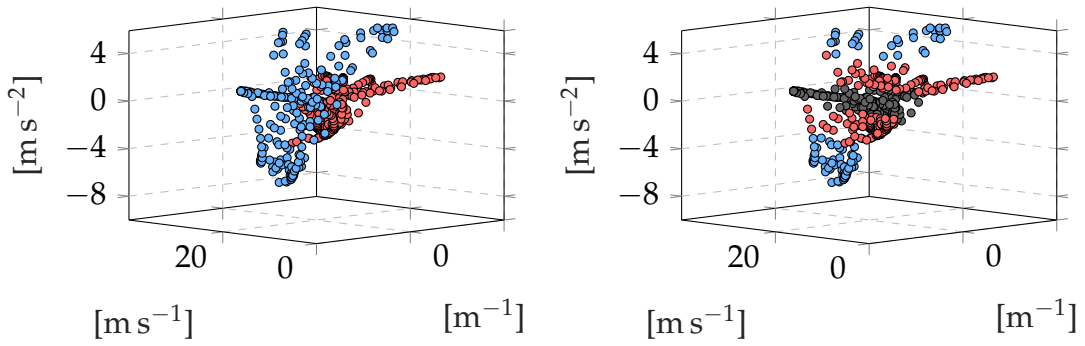
As previously mentioned, the approach used in this thesis is a distance-based algorithm known as the k-means clustering algorithm. In this method, the partitioning representatives are the means of each cluster. These are not derived from the original data points but are instead computed based on the underlying cluster structure. The Euclidean distance is used as the distance function. This method is widely recognized as one of the simplest and most classical approaches in clustering [83].

Given a set of n data points and predefined number of m clusters, the algorithm determines the centroids $\mu_1, \ldots, \mu_{\mathrm{m}}$ by minimizing the mean squared distance for each data point to its nearest centroid:

[3] In the context of clustering, a dendrogram is a tree-like structure that illustrates the arrangement of the clusters formed at each stage of the clustering process.

$$\min \sum_{i=1}^{m} \sum_{x_n \in C_i} ||x_n - \mu_i||^2, \tag{5.10}$$

where C_i represents the i-th cluster. Figure 5.4 illustrates how the k-means clustering algorithm is applied to multidimensional data. The data points originate from two recorded laps on one circuit with trajectories generated from different maximum lateral accelerations. The resulting clusters represent different levels of lateral acceleration: low, medium, and high.



(A) Two sets of data points from simulations with low and high lateral dynamics.

(B) Clustered data points based on the magnitude of lateral acceleration.

FIGURE 5.4: As an example, the results from a k-means clustering algorithm applied to data points consisting of velocity, curvature, and lateral acceleration are illustrated to categorize data based on lateral acceleration.

5.4 Learning approach

The structure of this AMPC is quite similar to the approach introduced in Chapter 4. Data consisting of vehicle, trajectory, and tracking information is collected while the AMPC controls the vehicle along the trajectory. The tire model is periodically updated to improve the accuracy of the applied vehicle model and, consequently, improve the tracking performance. When updating the tire model, the tire stiffness is adjusted, i.e., the parameters of the nonlinear tire model (Eq. 3.5 and Eq. 3.6) are modified to alter the yaw intensification. This adjustment influences the vehicle's response to steering input, as illustrated previously in Figure 4.3.

Figure 5.5 presents an overview of the AMPC and its interface with the simulation environment and the planning layer. The AMPC consists of three modules: clustering, GPR, and the baseline MPC. The clustering module receives data from the simulation environment and the planning layer, along with the currently predicted stiffness factor k from the GPR module. The GPR module is supplied with a data dictionary $\mathcal{D}_i$ from the clustering module to train the posterior. The MPC can query the GPR by providing the data κ_{qry} and v_{qry} to receive the stiffness factor k for adjusting the tire model.

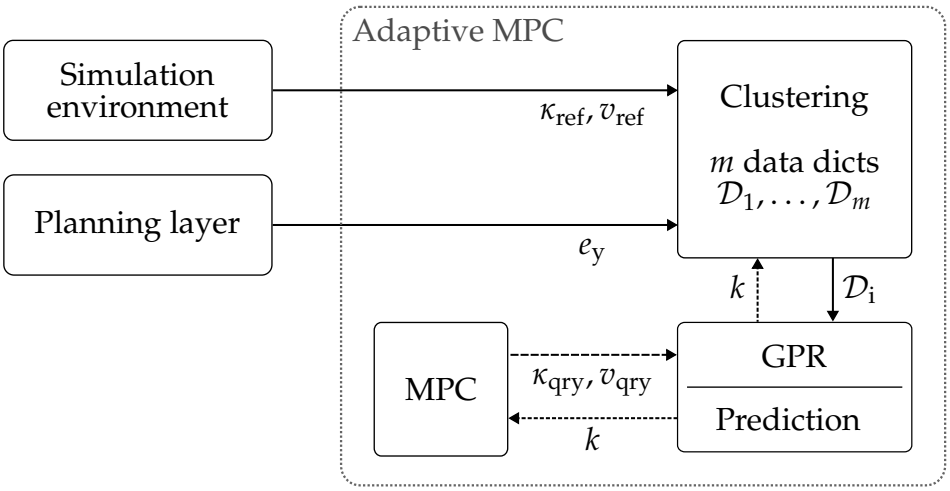


FIGURE 5.5: Overview of the AMPC approach with GPR and clustering, and its interface with the simulation environment and the planning layer, showing only relevant information and connections.

5.4.1 Clustering

The clustering module contains a master data dictionary that holds up to $n_{\mathcal{D}} = 60$ feature vectors. A feature vector is given as:

$$\mathbf{f}_{\mathcal{D}} = [\kappa_{\text{ref}}, v_{\text{ref}}, e_y, k_{\text{edu}}]^T, \tag{5.11}$$

where κ_{ref} and v_{ref} denote the reference curvature and reference velocity of the trajectory, respectively. e_{y} denotes the lateral deviation and k_{edu} denotes an educated guess for the stiffness factor to compensate for the model mismatch.

The master data dictionary is input into the k-means clustering method from Equation 5.10 to determine m_{C} clustered dictionaries based on k_{edu} of the feature vectors. The clustered dictionary with its centroid closest to the current k is then passed to the GPR module. Each clustered dictionary must adhere to a maximum number of entries given by the $n_{\mathcal{D}}/m_{\text{C}}$. If this number is exceeded, the first in - first out (FIFO)[4] organization method is applied to discard data from the clustered dictionary. All clustered dictionaries and the newly collected data are then combined into a new master data dictionary to start the cycle again.

5.4.2 Gaussian process regression

The GPR module holds a Gaussian process based on a SE kernel, as shown in Equation 5.8. This module utilizes the provided data dictionary $\mathcal{D}_{\text{i}}$ from the

[4]Method of organizing and manipulating data or items, where the first element added is the first one to be removed.

clustering module as observation or training data for the posterior. The input X and the output $\mathbf{y}$ are extracted as:

$$X = [\kappa_{\text{ref}}, v_{\text{ref}}, e_y]^T, \tag{5.12a}$$
$$\mathbf{y} = k_{\text{edu}}, \tag{5.12b}$$

from the feature vectors contained in the data dictionary. The calculation of the posterior is always accompanied by an optimization of the hyperparameters. Here, the log marginal likelihood from Equation 5.9 is utilized. The clustering module and the GPR are executed at a rate of 2 Hz to update the posterior, a frequency that proved to be sufficient during the development process. Additionally, optimizing the hyperparameters is a computationally expensive operation.

Similar to the approach outlined in Chapter 4, the MPC also queries the GPR for a stiffness factor k (function value $\mathbf{g}_*$) at each optimization step. The query (test) input X_* is defined as:

$$X_* = [\kappa_{\text{qry}}, v_{\text{qry}}, 0]^T, \tag{5.13}$$

where κ_{qry} and v_{qry} represent the curvature and velocity on the trajectory at the current optimization step, respectively. The query input is set with a lateral deviation $e_{\text{y}} = 0$ to predict the stiffness factor for the most accurate tracking behavior under the current circumstances.

5.5 Simulation study

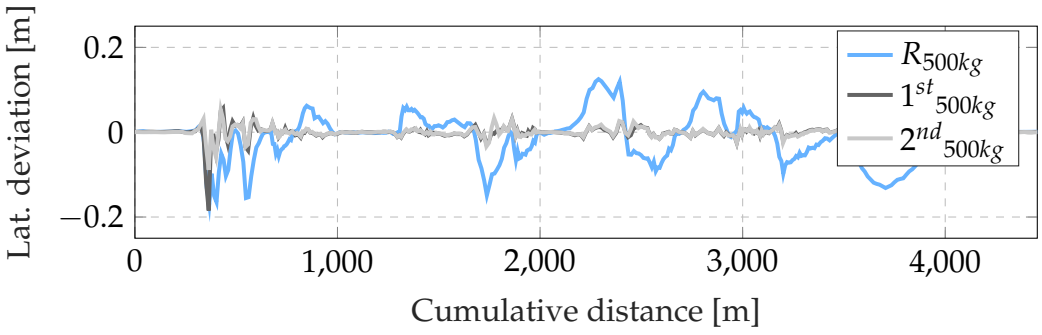
The simulation study for this approach closely resembles the one conducted for the AMPC introduced in Chapter 4. Thus, a direct comparison of both approaches is feasible. The test vehicle used is a *Tesla Model S* with parameters detailed in Table 4.1. The scenarios take place on one of three distinct closed-loop circuits shown in Figure 2.3. The trajectories for all circuits are generated with a maximum lateral acceleration of $a_{\text{y,max}} = 6\,\text{m}\,\text{s}^{-2}$.

The first experiment in Section 5.5.1 conducts reference measurements to evaluate the impact of a condition change using only the baseline MPC as the trajectory tracking controller. This procedure is then replicated in the second experiment in Section 5.5.2, but this time with an enabled AMPC. The third experiment in Section 5.5.3 assesses the AMPCs performance in handling multiple condition changes. Finally, the fourth experiment in Section 5.5.4 applies the already trained AMPC to unknown trajectories.

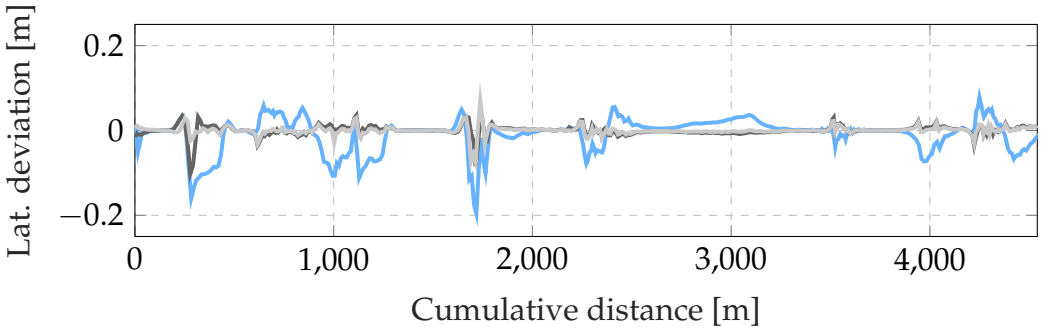
The condition changes for the experiments are emulated using an additional weight placed in the vehicle's CoG. To ensure comparability, all experiments are conducted in laps and utilize the introduced tracking performance metric. Although the experiments are conducted in laps, the AMPC does not rely on laps or maneuvers of the same distance.

5.5.1 Reference measurements

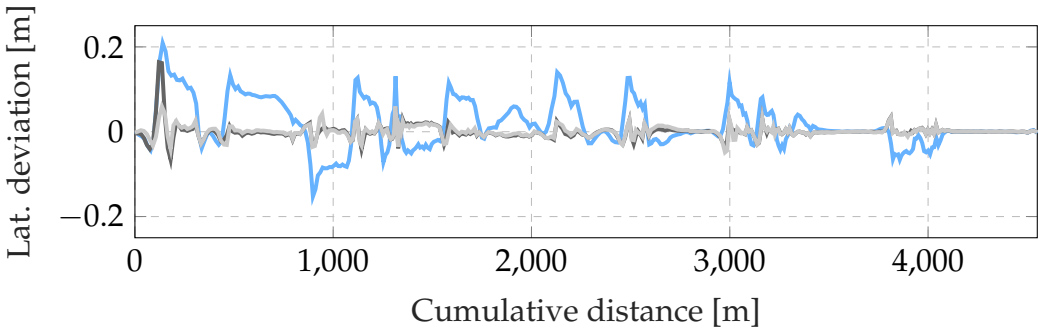
The results of the reference measurements for this experiment are identical to those of the experiment in Section 4.4.1, as the same test vehicle and trajectories for the closed-loop circuits are used. The results indicate that the tracking performance decreases by more than 50 % on two of the three circuits when the vehicle experiences a condition change. The detailed results are listed in Table 4.2.



(A) First scenario: Algarve International Circuit



(B) Second scenario: Hockenheimring



(C) Third scenario: Thunderhill Raceway

FIGURE 5.6: Results of all three scenarios from the second experiment. All laps are conducted with an additional weight of 500 kg. The AMPC is disabled for the first lap (blue) and then enabled for the second and third lap (dark and light gray).

5.5.2 Adapting to condition changes

The second experiment evaluates how well the AMPC adapts to a condition change. The experiment consists of three scenarios, each taking place on one of the three closed-loop circuits. Each scenario follows the same procedure, consisting of three laps where the vehicle is affected by an additional weight of 500 kg. At the beginning of the second lap, the AMPC is enabled for the remaining two laps to compensate for model mismatch caused by the condition change. Figure 5.6 displays the lateral deviation over all three laps for all three scenarios. Comparing the first (blue) and the third (light gray) lap reveals that the increased lateral deviation is largely reduced by adapting the vehicle model. Another interesting detail is that the second (dark gray) and the third laps only differ at the beginning for a couple of hundred meters. This indicates that the AMPC immediately and accurately adjusts the vehicle model as soon as the condition change occurs.

Scenario	Lap	$P_{\mathrm{e,lat}}$	$e_{\mathrm{lat,max}}$
1st Algarve	1	42 % | 42 %	0.19 m | 0.19 m
	2	94 % | 77 %	0.18 m | 0.15 m
	3	94 % | 86 %	0.09 m | 0.08 m
2nd Hockenheim	1	59 % | 59 %	0.21 m | 0.21 m
	2	94 % | 82 %	0.10 m | 0.15 m
	3	94 % | 90 %	0.08 m | 0.08 m
3rd Thunderhill	1	40 % | 40 %	0.21 m | 0.21 m
	2	89 % | 71 %	0.18 m | 0.16 m
	3	90 % | 83 %	0.07 m | 0.08 m

TABLE 5.1: Detailed results for Figure 5.6. $P_{\mathrm{e,lat}}$ denotes the tracking performance and $e_{\mathrm{lat,max}}$ denotes the maximum lateral deviation. For comparison, the values in light gray show the detailed results from Table 4.3 for the same experiment conducted with the AMPC introduced in Chapter 4.

The detailed results of the tracking performance and the maximum lateral deviation are listed in Table 5.1. In all three scenarios, tracking performance losses are recovered by at least 45 %, achieving a subsequent tracking performance of over 90 %. The maximum lateral deviation is also reduced by over 50 % in all scenarios. The data in light gray represent the corresponding results from Table 4.3 for the experiment conducted with the previously introduced AMPC.

This advanced AMPC approach, incorporating GPR and clustering, achieves significantly better results in all three scenarios compared to the AMPC utilizing a lookup table. This is evident in the final tracking performances after the third lap, and particularly notable in the performances of the second lap. In two

of the three scenarios, the tracking performance is already on par with that of the third lap. Additionally, the tracking performances for these scenarios are only 1-2 % lower compared to the performances of the reference measurement in Table 4.2, where the vehicle is not affected by any condition changes. The AMPC almost completely compensates for the model mismatch caused by the condition change. Furthermore, it is important to consider that an additional weight of 500 kg during cornering maneuvers at lateral accelerations of $6\,\mathrm{m\,s^{-2}}$ significantly impairs the physical capabilities of the vehicle. Therefore, achieving nearly the same tracking performance as an unaffected vehicle underscores the capability of the approach in compensating for model mismatch.

5.5.3 Repeatedly adapting to condition changes

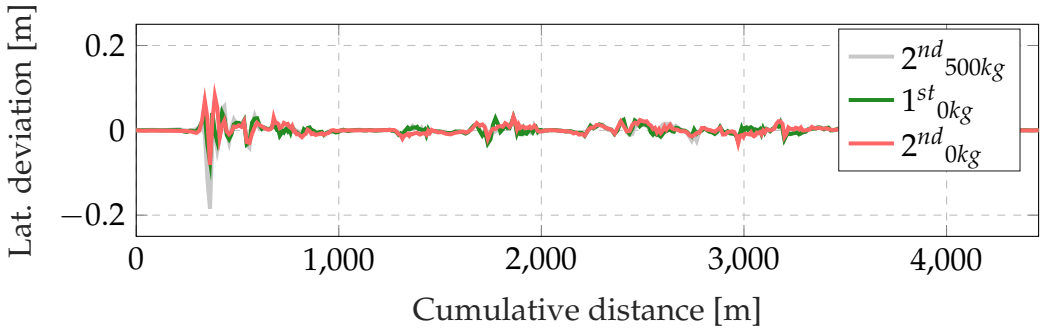
The third experiment aims to evaluate the AMPCs ability to repeatedly adapt to condition changes. It is essentially an extension of the second experiment, beginning where the second experiment left off. The procedure starts with the second lap of an enabled AMPC and a vehicle with an additional weight of 500 kg added to its CoG. Following this, another condition change is emulated by reducing the additional weight to 0 kg.

Figure 5.7 displays the lateral deviation for all three scenarios. As per the procedure, the light gray data is identical to the data from the third lap in the second experiment shown in Figure 5.6. The green and red lines depict the lateral deviation during the re-adaptation process over two laps. Detailed results for this experiment are listed in Table 5.2.

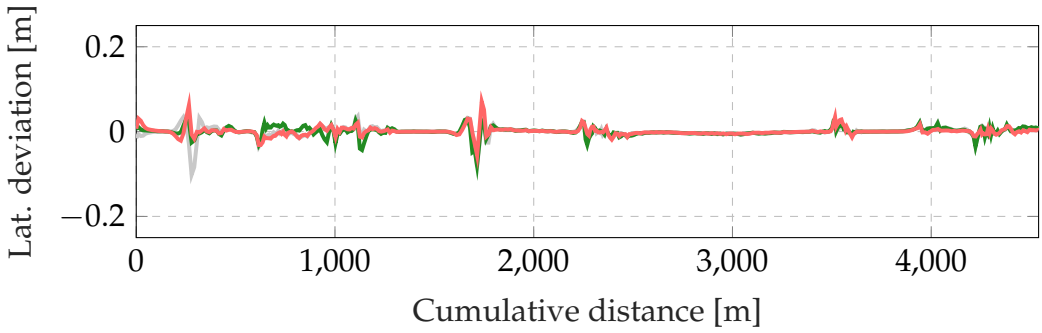
Scenario	Lap	$P_{\mathrm{e,lat}}$	$e_{\mathrm{lat,max}}$
1st Algarve	1	94 % | 86 %	0.08 m | 0.08 m
	2	95 % | 80 %	0.09 m | 0.08 m
	3	96 % | 95 %	0.08 m | 0.07 m
2nd Hockenheim	1	94 % | 90 %	0.08 m | 0.08 m
	2	94 % | 84 %	0.09 m | 0.09 m
	3	95 % | 93 %	0.08 m | 0.07 m
3rd Thunderhill	1	90 % | 83 %	0.08 m | 0.08 m
	2	93 % | 79 %	0.08 m | 0.07 m
	3	95 % | 93 %	0.05 m | 0.05 m

TABLE 5.2: Detailed results for Figure 5.7. $P_{\mathrm{e,lat}}$ denotes the tracking performance and $e_{\mathrm{lat,max}}$ denotes the maximum lateral deviation. The values in light gray show the detailed results from Table 4.4 for the same experiment conducted with the AMPC introduced in Chapter 4.

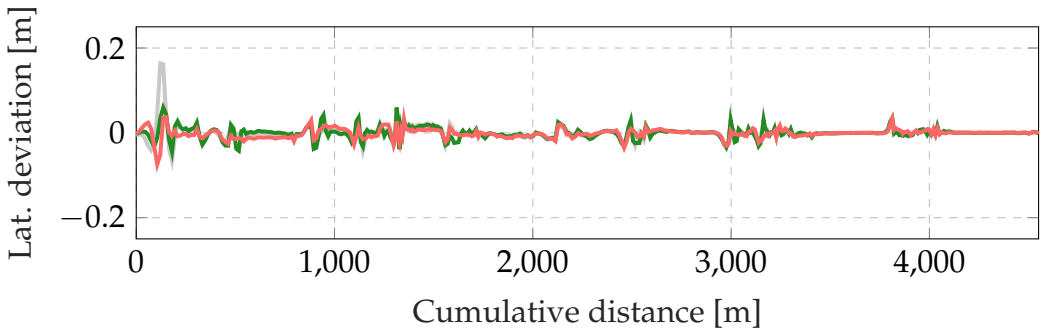
These results demonstrate that the AMPC is capable of adapting once again to a condition change in a manner similar to the previous experiment, where the



(A) First scenario: Algarve Inter. Circuit



(B) Second scenario: Hockenheimring



(C) Third scenario: Thunderhill Raceway

FIGURE 5.7: Results of all three scenarios from the third experiment. The AMPC re-adapts to a second condition change at the beginning of the second lap (green). After the second lap of adaptation (red), the lateral deviation almost matches the lateral deviation of the reference lap with 0 kg in Figure 4.4, indicating nearly the same tracking performance.

tracking performance after one lap is already nearly comparable to the performance after two laps of adaptation. This trend is consistent across all three scenarios. After one lap, the tracking performance reaches at least 93% in all three scenarios, and after the second lap, it is at least 95%. These performances align closely with the tracking performances of the reference measurements in Table 4.4. Achieving similar performances to the reference measurements indicates that the AMPC accurately adjusts the model based on the observed mismatch.

The values in light gray represent the results of the same experiment but

with the AMPC from Chapter 4. Compared to that approach, the AMPC adapts faster and more accurately to the second condition change as well. In combination with the results from the previous experiment, this more sophisticated approach, utilizing a GPR as an approximation function rather than a lookup table, proves to be the superior method.

5.5.4 Applying adapted AMPC to unknown trajectories

The fourth experiment aims to evaluate the AMPCs ability to transfer adapted behavior to unknown trajectories. The AMPC is trained for two laps on one of the three closed-loop circuits and then applied to the remaining two circuits for one lap each. This procedure is repeated for all three circuits. During the lap on the unknown circuit, the AMPC does not adjust any further, it only utilizes the adjusted behavior for predictions. The results for all three scenarios are listed in Table 5.3.

The bold results in the table's diagonal illustrate that, in general, the AMPC achieves the highest tracking performance on the circuits where it was trained. Some adaptations of the AMPC on other circuits are partially able to match even the best results. Overall, the maximum discrepancy is just 1 %.

In comparison to the same experiment in Section 4.4.4 for the AMPC with the lookup table, this AMPC approach yields significantly better results regarding its applicability to unknown trajectories. For instance, with the AMPC using the lookup table, the maximum discrepancy from the best performance was 14 %, whereas this approach achieves just 1 %.

Applied / *Trained*	*Algarve*	*Hockenheim*	*Thunderhill*
Algarve	**94 %**	94 %	94 %
Hockenheim	93 %	**94 %**	93 %
Thunderhill	89 %	90 %	**90 %**

TABLE 5.3: Resulting tracking performances $P_{e,lat}$ for AMPCs trained on individual circuits and applied to all circuits for one lap with an additional weight of 500 kg. The bold numbers indicate the results of the AMPC trained on each respective circuit.

5.5.5 Memorization of multiple condition changes

Since the previous experiments have shown that this AMPC is capable of achieving tracking performances close to the reference measurements, a new additional experiment aims to demonstrate how the approach can memorize adapted behavior for multiple condition changes. This experiment spans over nine laps in total. Initially, the AMPC encounters three different condition changes, each lasting for two laps. Subsequently, the active learning of the AMPC is disabled, with only the current data utilized for predictions. The three

condition changes are then repeated for one lap each. As always, this procedure is applied to three scenarios on different closed-loop circuits. The detailed results are presented in Table 5.4.

The table's top section shows the results for the active AMPC encountering three distinct condition changes, while the bottom section shows the results when the AMPC does not adapt further but must deal with the condition changes again, relying solely on predictions based on previously adapted behavior. The results demonstrate that the tracking performances mostly differ by just 1 %, with only two cases showing a difference of up to 3 % (bold values). These results suggest that the clustering implemented within the AMPC allows the approach to adapt to multiple condition changes while organizing data according to encountered conditions, and subsequently utilize them again under similar circumstances.

Lap	Weight	Algarve	Hockenheim	Thunderhill
1.-2.	500 kg	94 %	95 %	89 %
3.-4.	0 kg	96 %	95 %	95 %
5.-6.	250 kg	96 %	95 %	94 %
7.	500 kg	94 %	94 %	88 %
8.	0 kg	95 %	**92 %**	**93 %**
9.	250 kg	95 %	94 %	93 %

TABLE 5.4: Resulting tracking performances for three different scenarios where the AMPC adapts to three distinct condition changes during laps 1 to 6. These condition changes reoccur from laps 7 to 9, during which the AMPC does not adapt further but relies solely on the previously adapted data for predictions.

5.6 Summary and conclusion

As detailed in the simulation study, Experiments 1 through 3 were conducted sequentially. To highlight their connection and provide a clearer overview, Figure 5.8 illustrates all conducted laps on a timeline for one scenario. The condition change is indicated at the top by the amount of additional weight added to the vehicle's CoG. The state of the AMPC is depicted below the additional weight. The bars represent the tracking performance for the conducted laps. For direct comparison, the dashed lines on the four bars furthest to the right indicate the results of the AMPC from Chapter 4 for Experiments 2 and 3, while the reference measurements in Experiment 1 remain unchanged.

In summary, the advanced AMPC proposed in this chapter effectively manages condition changes and significantly improves tracking performance (2$^{\text{nd}}$ experiment). The simulation results also demonstrate that the AMPC can repeatedly adapt to condition changes (3$^{\text{rd}}$ experiment). Moreover, the approach's generic nature allows the AMPC to manage unknown trajectories and achieve

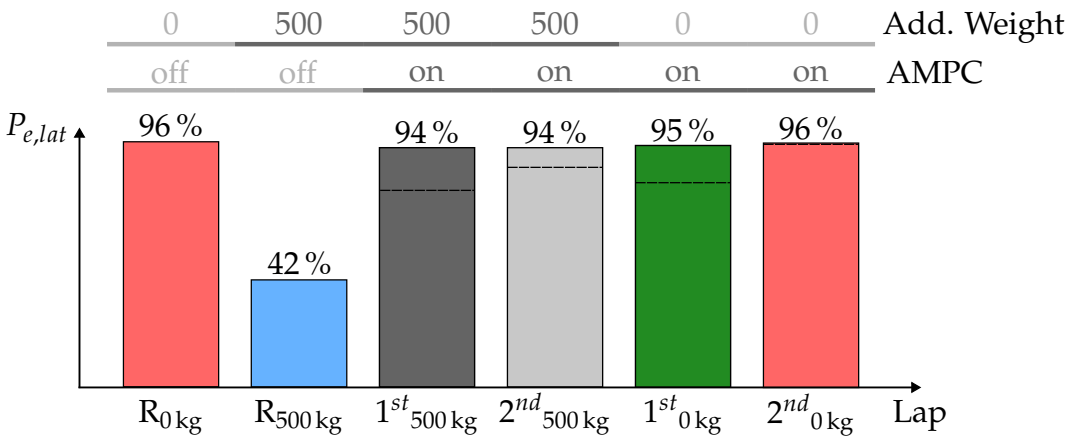


FIGURE 5.8: Combined summary of Experiments 1 to 3 on the Algarve International Circuit (1st scenario). The lap names and colors correspond to the labels and colors used in Figures 4.4, 5.6, and 5.7. The dashed lines on the four bars furthest to the right indicate the results of the AMPC from Chapter 4.

nearly the same improvements in tracking accuracy as for known trajectories (4th experiment).

In all experiments, this AMPC achieves better results than its predecessor from Chapter 4, almost reaching the tracking accuracy of the reference measurement for the unaffected vehicle. Unlike the previous AMPC, this approach also demonstrates its ability to memorize adapted behavior through the application of clustering. Consequently, the AMPC can react immediately to changing conditions, leading to significant improvements in tracking performance.

Although the AMPC has effectively demonstrated its ability to enhance tracking accuracy, transfer learned behaviors to new trajectories, and retain previously adjusted behaviors, there remains significant potential for further development. One avenue for improvement involves incorporating additional meaningful feature data, which would enable the GPR to more precisely differentiate tracking behaviors and optimize yaw intensification more effectively. However, a notable drawback of GPR is its computational inefficiency with large datasets, given its time complexity of $\mathcal{O}(n^3)$. Investigating the use of multiple smaller GPRs could provide a valuable solution, potentially optimizing multiple parameters more efficiently.

Given the computational limitations of GPR, neural networks offer a promising alternative for future research. Neural networks are capable of robust function approximation for system behavior prediction and can handle high-dimensional data, allowing for a more comprehensive selection of features and the potential to approximate entire sections of the model. Additionally, neural networks could potentially obviate the need for clustering algorithms due to their superior generalization capabilities.

However, implementing neural networks presents its own set of challenges, particularly in the training process. An a priori supervised learning approach

that relies on real-world and simulation data may be difficult to execute, as the data must encompass the condition changes the learning system is designed to manage. Therefore, generating a diverse and extensive set of training data is crucial. Robust online reinforcement learning or transfer learning, where the neural network adapts in real-time to changing conditions, could be effective alternatives. Nonetheless, ensuring robustness is essential, as the tracking behavior must remain stable and safe. Achieving adaptation speeds comparable to those of GPR, which can quickly adjust with substantial improvements from limited data points, represents a significant challenge for neural network-based approaches.

Chapter 6

Fault detection for trajectory tracking controller

In contrast to the two preceding chapters, this chapter explores the use of neural networks as a fault detection system for a trajectory tracking controller. The tracking controller in this context comprises only the baseline MPC without any additional learning approaches introduced in Chapters 4 and 5. Four different types of neural networks, namely, an Artificial Neural Network (ANN), a Recurrent Neural Network (RNN), a Long Short-Term Memory (LSTM) network, and a Gated Recurrent Unit (GRU) network, are applied and evaluated for their suitability as fault detection systems. The selection of features for the training data of the neural networks is based on a common automated driving function, its sensors, and the tracking controller.

This chapter is structured as follows: Section 6.1 discusses fault detection approaches for trajectory tracking controllers, emphasizing the significance of their evaluation. Section 6.2 introduces the four types of neural networks employed in this study. Section 6.3 provides insight into the composition and preparation of the training data. Subsequently, all network types are evaluated and compared in a thorough simulation study in Section 6.4, where they are applied to detect both known and unknown fault conditions. Finally, the chapter concludes with a summary in Section 6.5.

6.1 Literature and contribution

As control systems and applications progress toward SAE Level 5 of driving automation, the complexity of these systems is constantly increasing. To minimize risks and prevent accidents, the monitoring systems for fault detection, isolation, and identification must also advance. However, current fault diagnosis and health monitoring algorithms are not sufficiently effective for such complex systems [5], where incorrect behavior may result from multiple sensors or components simultaneously.

A trajectory tracking control system for automated driving typically relies on a localization service, a planning layer, and a tracking controller. Implementing a fault detection system requires consideration of a variety of sensors as well as software components.

Biddle et al. [5] utilize Support Vector Machine techniques to detect and identify faults in sensors of control systems for automated vehicles. Alsuwian et al. [51] applied a fuzzy neural network to identify wheel speed problems and thereby avoid instability issues. While these examples solely monitor certain sensors of the automated driving system, others focus on monitoring the control output itself. Dai et al. [52] use a Linear Quadratic Gaussian (LQG) controller as a benchmark to detect abnormal behavior in general control software. Salsbury et al. [53] propose a feed-forward control scheme with static simulation models to detect discrepancies between the actual and expected control output.

Moreover, there are approaches that consider the entire trajectory tracking control loop. Geng et al. [54], [55], propose a fault-tolerant model predictive control algorithm for robust lane change path tracking. They utilize a Kalman filter and a Chi-square based algorithm for detection. Asadi et al. [56] propose a fault-tolerant trajectory tracking control strategy for quadcopters, using a discrete extended Kalman filter based on the control outputs and the angular rate. Loquasto et al. [57] employed a neural network to classify unusual disturbances or significant changes in plant dynamics for a model predictive control system. They demonstrated their approach in a case study involving the Wood-Berry distillation column model.

Since neural networks have rarely been applied as fault detection systems for trajectory tracking controllers, this study aims to investigate their use in detecting incorrect behavior based solely on tracking performance, controller outputs, and vehicle ego motion. The approach focuses exclusively on detecting incorrect behavior, without addressing the isolation and identification of the fault source.

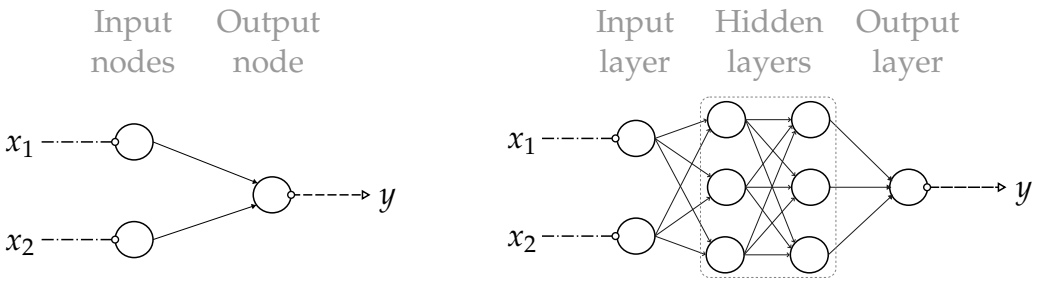
Using a high-fidelity vehicle dynamics simulation, a thorough simulation study is conducted to examine and compare the effectiveness of various types of neural networks in monitoring a trajectory tracking controller within an automated driving system. The experiments aim to: (**i**) evaluate the networks' ability to detect known incorrect behavior using a training dataset that includes data from such fault conditions, and (**ii**) assess their capability to detect unknown fault conditions.

6.2 Neural networks

Inspired by the structure and functioning of the human brain, neural networks are a subset of machine learning algorithms. A neural network can be trained to solve specific problems or tasks, such as classification, pattern recognition, and data clustering. Neural networks find applications in several domains, including object and face identification, understanding spoken languages, diagnosing diseases from medical images, assessing risk for credit, predicting stock market trends, and generating realistic graphics [84]. They are known for their efficient and adaptive learning capabilities [85] and possess a certain robustness to noise in the training data [86].

The task of letter recognition, for instance, is frequently used as an example to introduce neural networks. The *MNIST* database of handwritten digits, containing up to 70,000 examples [87], serves this purpose by enabling pattern recognition methods on real-world data using preprocessed and formatted training data. Established in 1994 [88], the database is a subset of the original collection from the National Institute of Standards and Technology (NIST).

Neural networks consist of interconnected artificial neurons, also known as nodes, that are organized into layers. These can be categorized into single-layer and multi-layer neural networks. In a single-layer neural network, the input layer is directly connected to the output layer. Conversely, in a multi-layer neural network, the input layer is connected to the output layer through a series of hidden layers. This layered architecture is also referred to as a *feed-forward network* [89]. Figure 6.1 illustrates, as an example, the structure and interconnections of both single-layer and multi-layer neural networks.



(A) Example of a single-layer neural network with two input neurons directly connected to one output neuron.

(B) Example of a multi-layer neural network with two input neurons, two hidden layers, and one output neuron.

FIGURE 6.1: Illustrative structure and interconnections of a single-layer and a multi-layer neural network.

The simplest neural network is the single-layer *perceptron*, which is one of the first approaches to apply artificial neurons in a model [89]. It was developed in the 1950s [90] and inspired by the earlier work of McCulloch et al. [91].

Each node in the network is connected to other nodes and contains an associated weight, a bias, and a threshold or activation function. The weight influences the information input to the node, while the bias is added to the sum of all inputs, effectively offsetting it. When the threshold is exceeded, the node is activated and sends its information to connected nodes. If an activation function is applied instead of a threshold, the output of the corresponding function is transmitted.

Figure 6.2 illustrates the computations at a node along with all its components. In this case, the computation is expressed as:

$$y = \Phi(x_1 w_1 + x_2 w_2 + b), \tag{6.1}$$

where y denotes the output value of the node. The activation function Φ takes as input the sum of the inputs x_1 and x_2, each multiplied by their corresponding

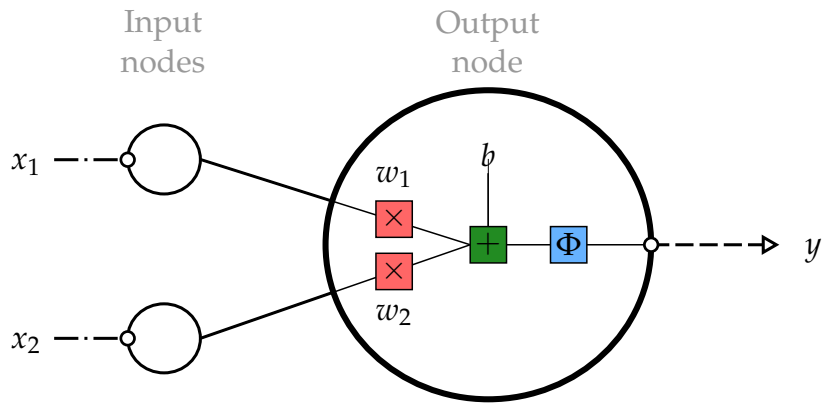


FIGURE 6.2: Illustration of the computation at a single node. All inputs x are multiplied by their corresponding weights w and summed together with the node's bias b. The result is then passed to the activation function Φ to calculate the node's output y.

weights w_1 and w_2, along with the node's bias b. From this example, a generic formulation can be derived, which is written as:

$$y = \Phi(\sum_{i=1}^{n} w_i x_i + b). \tag{6.2}$$

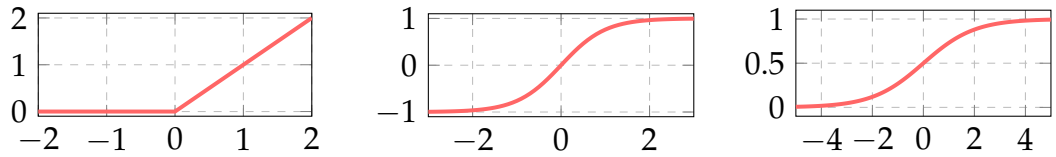
All input values and their corresponding weights can be collected into vectors $\mathbf{x} = [x_1, \ldots, x_n]$ and $\mathbf{w} = [w_1, \ldots, w_n]$, where n denotes the number of inputs or feature variables. Given that both vectors are of the same length, the dot product operation can be applied. Consequently, the generic formula is revised to:

$$y = \Phi(\mathbf{x} \cdot \mathbf{w} + b). \tag{6.3}$$

It is noteworthy that no computations are performed at the input layer; instead, only the feature values are transmitted to the next layer. Hence, the *perceptron* is referred to as a single-layer neural network because the output layer is the only layer where computations occur.

Regarding the previously mentioned activation function Φ, Figure 6.3 illustrates three common activation functions: the *Rectified Linear Unit* (ReLU), the *Hyperbolic Tangent*, and the *Sigmoid Function*. Since activation functions are not covered in more detail in this thesis, further information on various activation functions can be found in Chapter 6 of [92]. Moreover, the choice of activation functions and number of hidden layers plays a crucial role in the design process of a neural network, just as the selection and number of output nodes do [89].

The first type of neural network evaluated as a fault detection system is an ANN, which has already been covered in this section through the multi-layer feedforward network. The following subsections provide a brief introduction to the remaining neural network types.



(A) Rectified Linear Unit
$f(z) = max(0, z)$
range = $(0, \infty)$

(B) Hyperbolic Tangent
$f(z) = (e^z - e^{-z})(e^z + e^{-z})^{-1}$
range = $(-1, 1)$

(C) Sigmoid Function
$f(z) = (1 + e^{-z})^{-1}$
range = $(0, 1)$

FIGURE 6.3: Illustration of three common activation functions: Rectified Linear Unit, Hyperbolic Tangent, and Sigmoid Function.

6.2.1 Recurrent neural network

In contrast to the ANN discussed in the preceding section, which is a unidirectional feedforward neural network, the RNN is a bi-directional artificial neural network. This configuration allows nodes to use their outputs to influence subsequent inputs at the same node. Consequently, an RNN is capable of dealing with sequential or time-series data [89]. These recurrent neurons utilize an internal state (memory or hidden state) to represent information from previous input data at a given time step, which is updated at every subsequent time step to refine the accumulated knowledge about past inputs. This ability to utilize an internal state renders an RNN particularly effective for tasks such as speech [93] and handwriting recognition [94]. Figure 6.4 provides an illustration of an RNN, showcasing its recurrent neurons in the hidden layer, as indicated by the dashed arrowheads.

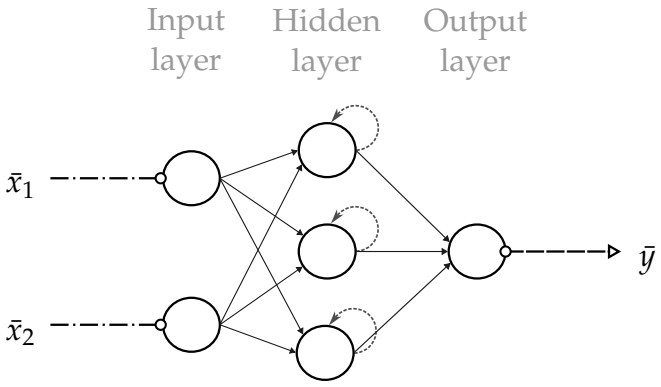


FIGURE 6.4: Illustration of an RNN. The dashed arrowheads represent recurrent neurons. The bars above the variables signify sequence vectors, whereas indices denote the feature input index, not the series index.

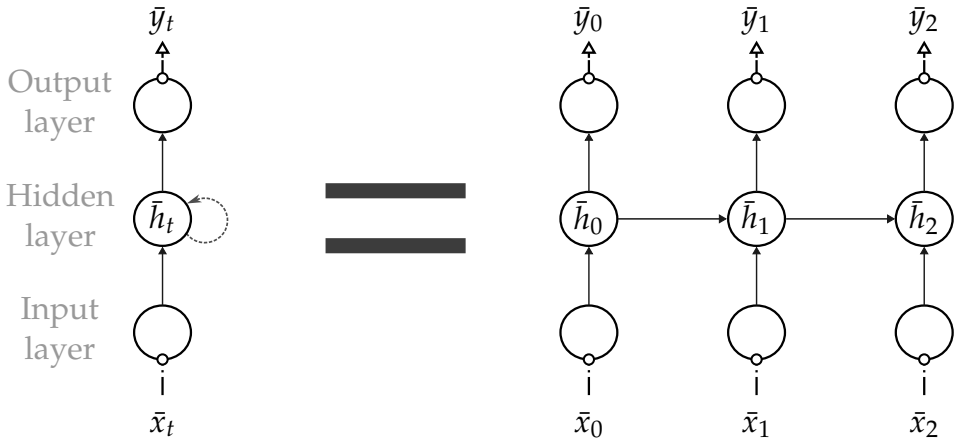
The basic architecture of a simple RNN is illustrated in Figure 6.5. For each of the three time steps in the sequence, an input $\bar{x}_t$ is fed to the recurrent neuron, while the internal state $\bar{h}_t$ changes according to

$$\bar{h}_t = \Phi_{hid}(\bar{h}_{t-1}, \bar{x}_t), \tag{6.4}$$

where Φ_{hid} denotes the activation function. The outputs are determined by another activation function, which takes the internal state as its argument, as given in

$$\bar{y} = \Phi_{out}(\bar{h}_t). \tag{6.5}$$

It is worth mentioning that the time-layered representation may be misleading, as the topology of a single recurrent neuron (see Figure 6.5b) does not accurately reflect the layered topology of a full recurrent neural network. What appears to be layered is, in fact, a sequence of time steps executed by the recurrent neuron.



(A) RNN (B) Time-layered representation of the RNN.

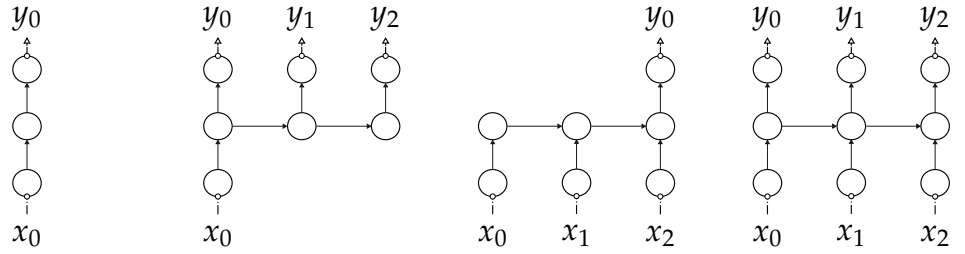
FIGURE 6.5: Illustration of an RNN (Many to Many) with one node in each of its three layers and its time-layered representation. This figure is based on [89].

The RNN illustrated in Figure 6.5 is of the *Many to Many* type. There are multiple types of RNNs regarding the length of input features and output values. Figure 6.6 illustrates all four RNN types. For example, a *One to Many* type RNN is used for music generation, a *Many to One* type is used for sentiment classification, and a *Many to Many* type is used for machine translation.

Even though the RNN is capable of addressing more problems than an ANN, the standard RNN architecture has a limited context range that can be practically accessed [95]. The influence of the input on the hidden layer, and consequently on the output layer, either decays or increases exponentially as it cycles through the recurrent connections of the RNN. In the literature, this effect is known as the vanishing gradient problem [96], [97].

6.2.2 Long Short-Term Memory network

To tackle the vanishing gradient problem, the LSTM architecture was proposed by Hochreiter et al. in 1997 [98]. This architecture comprises memory blocks, implemented through recurrently connected sub-nets. Each block hosts



(A) One to One (B) One to Many (C) Many to One (D) Many to Many

FIGURE 6.6: Different types of RNNs categorized by the length of their inputs and outputs.

one or more self-connected memory cells and three multiplicative units: the input, output, and forget gates. These gates provide a write, read, and reset functionality for a cell, allowing LSTM cells to maintain information over extended periods and thereby mitigate the vanishing gradient problem. For instance, as long as the input gate remains closed (i.e., its activation value is near zero), the cell's information will not be overwritten by new inputs, preserving the information for later use in the input sequence. The LSTM architecture has been proven successful in tasks that require long-range memory and has addressed challenges that remain insurmountable for other RNN architectures [95].

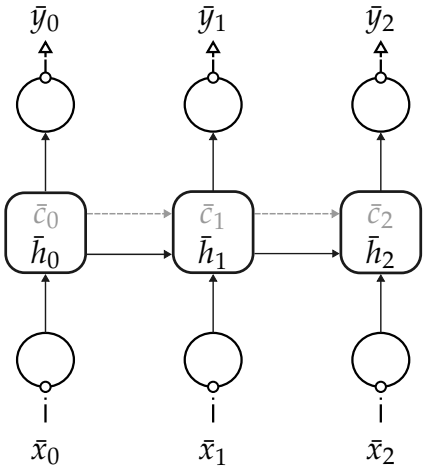


FIGURE 6.7: Time-layered representation of a neuron from an LSTM network, similar to the illustration of a standard RNN in Figure 6.5b, but with an additional cell state carried through the recurrent process.

Figure 6.7 illustrates the time-layered representation of an LSTM network (Many to Many), similar to the standard RNN shown in Figure 6.5b. In addition to the internal state $\bar{h}_t$, the cell state $\bar{c}_t$ is also carried through the recurrent process. Because of the cell state and the three gates, the LSTM incorporates four activation functions instead of only one, as in the standard RNN (Eq. 6.4). Each of these four activation functions comes with an associated weight and bias.

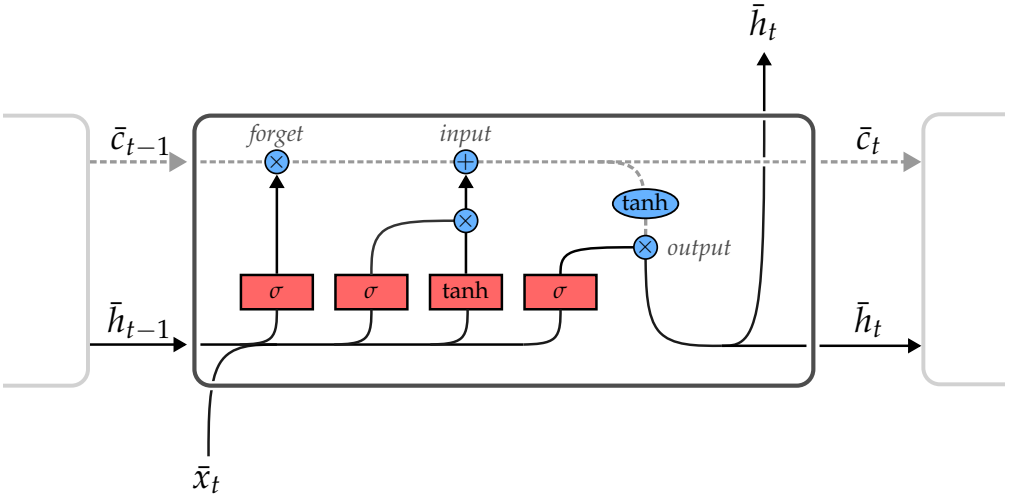


FIGURE 6.8: Illustration of a recurrent neuron in an LSTM network (based on [99]). The red-filled rectangles represent activation functions with weights and biases, while the blue circles represent point-wise operations. The three gates are labeled at their intersections with the cell state.

A detailed illustration of the architecture of an LSTM neuron is shown in Figure 6.8. The gray dashed line at the top denotes the cell state. In the first step, the cell state is influenced by the forget gate, which is represented by the leftmost multiplication symbol and a sigmoid activation function. The sigmoid function takes the concatenated $\bar{h}_{t-1}$ and $\bar{x}_t$ as input, and based on the output values, the corresponding cell states are either retained or discarded.

In the next step, the cell state is influenced by the input gate, which determines the information to be stored in the cell state. The input gate comprises a second sigmoid function, a hyperbolic tangent function, a multiplication operation, and an addition operation. The updated cell state is then carried forward to the output gate.

The output gate consists of the rightmost sigmoid function, a hyperbolic tangent function, and a multiplication operation. The sigmoid function takes the input and hidden state as arguments, while the hyperbolic tangent function takes the cell state as an argument. The results of both functions serve as arguments to the multiplication operation to determine the output.

The literature offers comprehensive resources for delving deeper into LSTM networks, such as [89], providing valuable insights into their architecture, training, and applications.

6.2.3 Gated Recurrent Unit network

Another LSTM variant utilized in this thesis is the Gated Recurrent Unit (GRU) network, introduced in 2014 by Cho et al. [100]. This architecture, similar to LSTM, employs only the forget and input gates, resulting in fewer parameters and enhanced computational efficiency. Moreover, the GRU network does

not utilize explicit cell states. Figure 6.9 illustrates a GRU variant, where the gates depend solely on the previous hidden state. This variant, along with two others, is evaluated in [101], with the authors concluding that their accuracy performance is comparable.

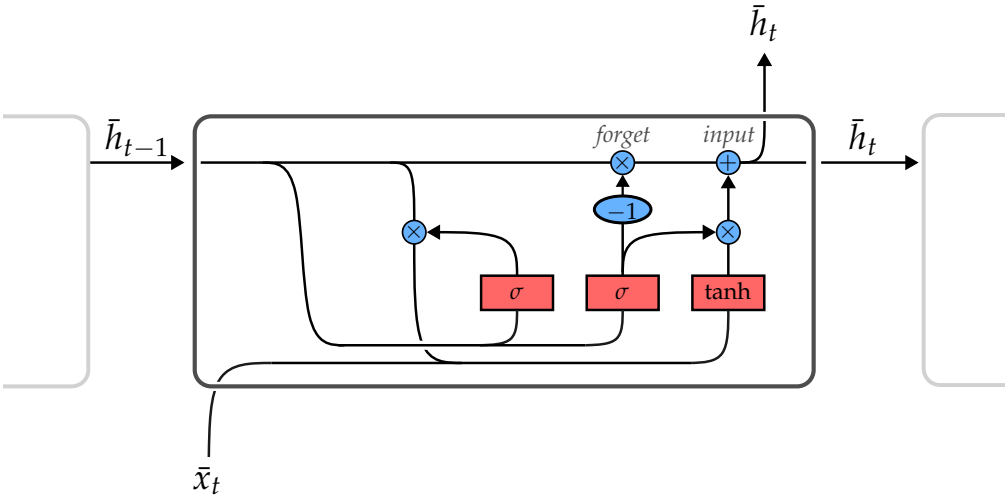


FIGURE 6.9: Illustration of a recurrent neuron in a GRU network (based on [99]). The red-filled rectangles represent activation functions with weights and biases, while the blue circles represent point-wise operations. The two gates are labeled at their intersections with the memory/hidden state.

A direct comparison between LSTM and GRU in [102] demonstrated that both architectures are comparable in the context of polyphonic music modeling and speech signal modeling. Additionally, there are further LSTM variations, such as LSTM with "peephole connections" [103], Depth-Gated LSTM [104], and Clockwork RNN [105]. In [106], a comparison of popular variants was conducted, without finding significant differences in their ability to solve tasks. Conversely, [107] identified RNN architectures that outperformed LSTM networks for certain tasks.

6.3 Preparation for simulation study

This section briefly discusses how the neural networks are integrated into the simulation environment and how they are trained for the fault detection task.

6.3.1 Integration into the simulation environment

The four different types of neural networks are implemented using TensorFlow.[1] Since the simulation environment is based on Matlab, built-in functionality enables the integration of TensorFlow-created neural networks into Matlab code, thus incorporating them into the simulation environment with Simulink and IPG CarMaker.

This means that the training process for any neural network type is outsourced to work solely with Python scripts. However, the simulation environment provides the training data and is responsible for preparing the data for the training process. Since Python scripts can be executed from within Matlab, the simulation environment serves as the primary interface for the user, assuming the scripts to train the neural networks function as intended.

6.3.2 Fault conditions

To evaluate the effectiveness of neural networks as a fault detection system, several fault conditions must be defined within the simulation environment. Table 6.1 presents six different fault conditions that are emulated during the following simulation study. The table is divided into two sections: the top section lists fault conditions used to generate training data for the neural networks, making these conditions known to the networks. The bottom section lists fault conditions not included in the training data, remaining unknown to the networks. Additionally, the table provides brief implementation details for each fault condition.

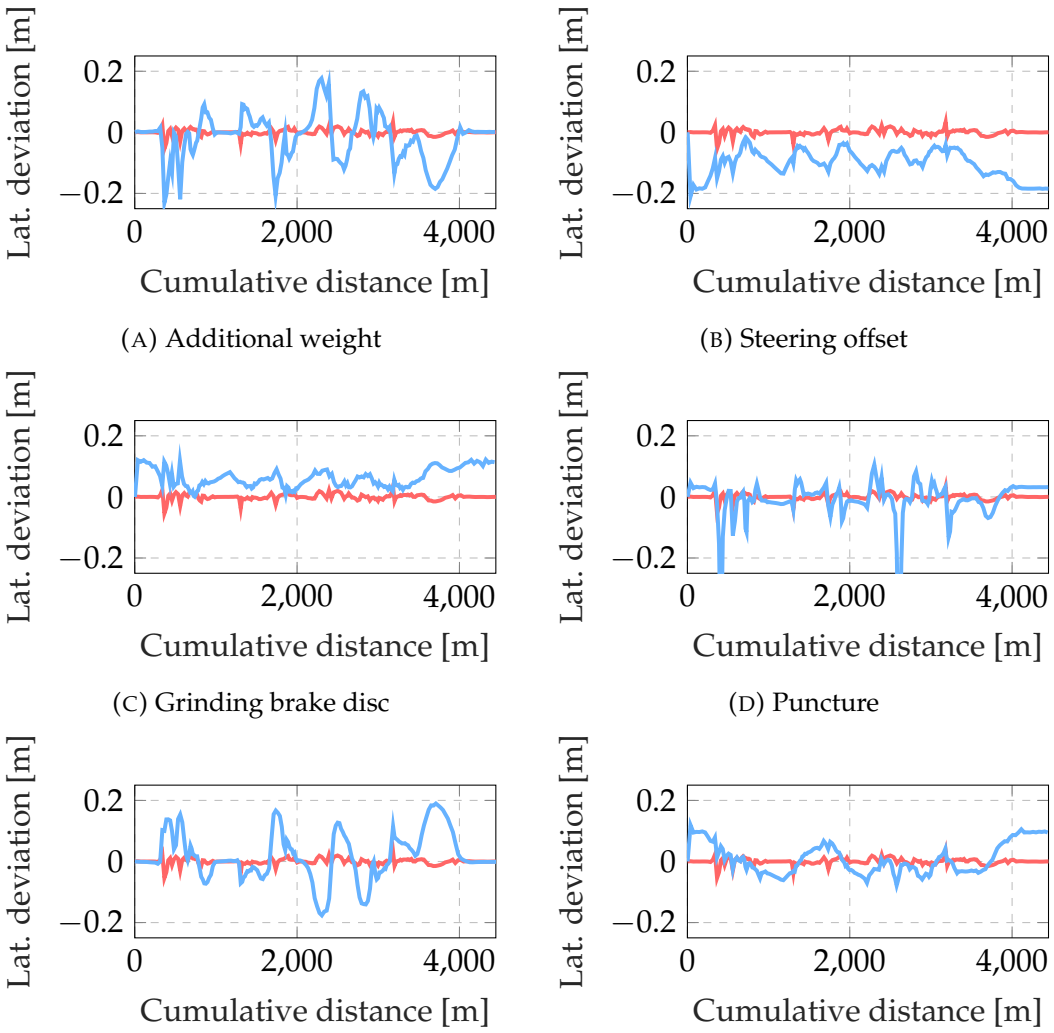
Fault Condition	Implementation Detail
Additional weight	750 kg placed in the CoG
Steering offset	10° steering wheel offset
Grinding brake disc	Constant brake torque applied
Puncture	Friction reduced and additional forces applied
Wind	Wind from a fixed direction at $20\,\mathrm{m\,s^{-1}}$
Parameter error	Yaw inertia off by 75 %

TABLE 6.1: Fault conditions defined for the simulation study, along with concise implementation details.

The impact of these fault conditions on the tracking behavior is presented in Figure 6.10. The lateral deviation induced by each fault condition is compared to that of an unaffected vehicle over a single lap on the closed-loop circuit shown in Figure 2.3a.

The trajectory used for comparison is generated with a maximum lateral acceleration of $a_{\mathrm{y,max}} = 6\,\mathrm{m\,s^{-2}}$. Among the fault conditions, only the puncture condition exceeds a lateral deviation of $\pm 0.25\,\mathrm{m}$. This could be attributed to

[1] TensorFlow is a free and open-source software library for machine learning and artificial intelligence, particularly focused on training and inference of deep neural networks.



(A) Additional weight

(B) Steering offset

(C) Grinding brake disc

(D) Puncture

(E) Parameter error

(F) Wind

FIGURE 6.10: Illustration of the impact of fault conditions on tracking behavior during a single lap on a closed-loop circuit. The red line represents the lateral deviation for an unaffected vehicle, while the blue line represents the lateral deviation for an affected vehicle.

its implementation as reduced friction, potentially causing an abrupt change in vehicle behavior when the lateral acceleration surpasses the friction limit.

6.3.3 Training data collection

All neural network types used in the following simulation study are based on the same training dataset. This dataset is generated separately in preparation for the evaluation. Training for all neural network types occurs only once, prior to the start of the simulation study. Each data point in the training dataset contains selected information regarding trajectory tracking behavior, the tracking controller, and the vehicle, known as features. The input layer size of the neural networks must be defined according to the number of features. Additionally,

for the RNN, LSTM, and GRU, the sequence length must be considered. Table 6.2 provides an overview of all selected features.

Tracking behavior	MPC outputs	Vehicle
Lateral deviation	Steering angle	Gas pedal position
Yaw error	Acceleration	Brake pedal position
Ref. curvature		Velocity in x-direction
Ref. velocity		Velocity in y-direction
		Yaw rate
		Yaw acceleration

TABLE 6.2: Selected features for a data point of the training dataset, comprising data related to tracking behavior, the tracking controller, and the vehicle.

The selection of lateral deviation and yaw error aims to highlight the performance of the tracking controller within the dataset. To contextualize this performance in relation to the dynamics of the desired trajectory, reference curvature and reference velocity are also included. Curvature and velocity implicitly contain information about the predetermined yaw rate and lateral acceleration of the trajectory. Controller outputs are incorporated into the training data to facilitate fault detection, even when the controller compensates for them in terms of tracking performance. Vehicle pedal positions are included to enable fault detection during non-cornering situations. For instance, an unusually strong brake pedal actuation dictated by the controller to achieve the desired negative acceleration may indicate a heavier vehicle or worn brakes. The remaining four vehicle information are selected to identify faults affecting cornering behavior.

The training data is generated on five different closed-loop circuits. Three of these circuits were previously introduced and shown in Figure 2.3. The additional two circuits, Motorsport Arena Oschersleben and Autodrom Most, are also real-world circuits. For each circuit, four different trajectories with maximum lateral accelerations of 2, 4, 6, and $8\,\mathrm{m\,s^{-2}}$ are generated. Each of these twenty trajectories is then subjected to the trajectory tracking controller for four laps. During these four laps, the vehicle experiences the known fault conditions: additional weight, steering offset, and grinding brake disc, as well as one lap where the vehicle is not affected by any fault condition. All 80 laps are sampled at 100 Hz to extract data points for the training dataset. Therefore, the training dataset contains approximately 650 000 data points.

6.3.4 Training of the neural networks

Training for all four types of neural networks follows the same procedure. The training data is randomly divided into training and test sets using an 80 – 20 ratio. The training set is then used to fit the data to the neural networks via supervised learning. Once training is complete, the trained neural network is evaluated on the test set to assess the model's accuracy.

To address potential variations in accuracy resulting from data splitting and the training procedure, an additional algorithm is employed. This algorithm repeats the procedure until a receding target accuracy is reached. The pseudo code for this algorithm is provided in Figure 6.11. This relatively simple algorithm is highly time-intensive because a full training sequence for a neural network is executed at each iteration. However, for the purpose of ensuring accurate training of the neural networks, this approach is sufficient within a reasonable amount of time.

Figure 6.11 Search for most accurate neural network

```
Require: fullTrainData ≠ NULL
 1: BestNN ← NULL
 2: BestAcc ← 0
 3: TargetAcc ← 1.0
 4: while BestAcc < TargetAcc do
 5:     TargetAcc ← TargetAcc − 0.01
 6:     for 1 to 3 do
 7:         randomize fullTrainData into TrainSet, TestSet
 8:         NN ← compile neural network
 9:         NN ← train neural network with TrainSet
10:         Acc ← evaluate neural network with TestSet
11:         if Acc > BestAcc then
12:             BestAcc ← Acc
13:             BestNN ← NN
14: return BestNN
```

The training for all neural network types is conducted over 10 epochs, meaning the process iterates through the training set ten times. All neural network types utilize the adaptive moment estimation (ADAM) optimizer and a binary cross-entropy loss function. ADAM is a popular optimization algorithm that outperforms many others, as it incorporates adaptive learning rates for all parameters, serving as an extension to the stochastic gradient descent (SGD) algorithm. The binary cross-entropy loss function, also known as logistic log loss, measures the disparity between predicted probabilities and the true binary outputs. It is commonly employed in binary classification tasks, especially in the context of training neural networks. Detailed information regarding various optimizers and loss functions can be found in the literature, such as [89].

The Tables 6.3, 6.4, 6.5, and 6.6 provide summaries of the architecture for the four different neural network types, utilizing TensorFlow's summary functionality.

Given the objective of this thesis to evaluate the feasibility of neural networks as fault detection systems for trajectory tracking, there is no exhaustive assessment of various configurations for each neural network type, nor of different optimizers, loss functions, or numbers of epochs. These configurations and settings were established during the development phase and achieved accuracies exceeding 98 % on the test data. Consequently, these neural network

Layer	Output size	Activation function
Flatten	12	None
Dense	256	ReLU
Dense	128	ReLU
Dense	1	Sigmoid

TABLE 6.3: Summary of the ANN architecture in TensorFlow.

Layer	Output size	Activation function
Input	(10, 12)	None
SimpleRNN	128	ReLU
Dense	1	Sigmoid

TABLE 6.4: Summary of the RNN architecture in TensorFlow.

Layer	Output size	Activation function
Input	(10, 12)	None
LSTM	128	ReLU
Dense	1	Sigmoid

TABLE 6.5: Summary of the LSTM architecture in TensorFlow.

Layer	Output size	Activation function
Input	(10, 12)	None
GRU	128	ReLU
Dense	1	Sigmoid

TABLE 6.6: Summary of the GRU architecture in TensorFlow.

configurations were retained for the simulation study. Detailed results for the trained neural networks are provided in Table 6.7.

Network	Accuracy	Loss	Data points
ANN	0.9880	0.0321	639430
RNN	0.9866	0.0344	639430
LSTM	0.9887	0.0269	639430
GRU	0.9899	0.0268	639430

TABLE 6.7: Evaluation results on the test set for trained neural network types.

6.4 Simulation study

The simulation study is divided into two sections. In Section 6.4.1, the capability of the neural network types to detect known fault conditions is evaluated,

while in Section 6.4.2, their ability to detect unknown fault conditions is assessed. At the beginning of the simulation study, all four neural network types are pre-trained, and no further training occurs during the study. Evaluations generally take place on the three closed-loop circuits shown in Figure 2.3, each featuring different trajectories based on their maximum lateral acceleration. The simulation study concludes in Section 6.5 with a brief summary.

6.4.1 Detecting known fault conditions

The evaluation of the neural networks' capability to detect known fault conditions is divided into three parts. The first part presents detailed results for a single trajectory on one closed-loop circuit to provide a closer illustration of the evaluation process. The second part displays the results for the two remaining closed-loop circuits. Finally, the last part presents the results for different trajectories on the same closed-loop circuits to assess the operation across a wide range of vehicle dynamics.

Detailed results for a single trajectory

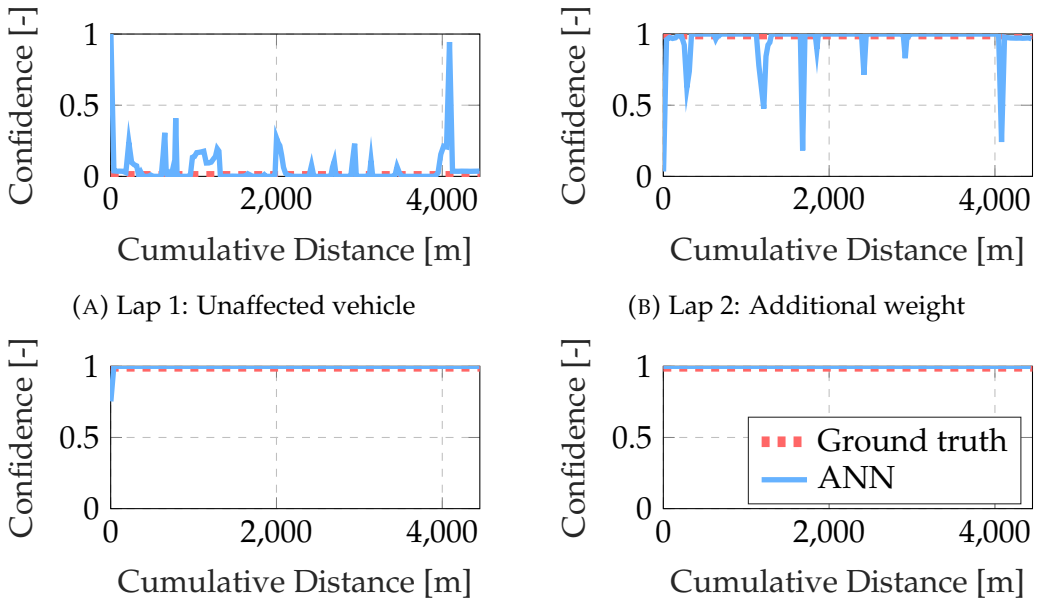
The following results are derived from evaluating all four neural network types on the first closed-loop circuit (Figure 2.3a), using a trajectory generated with a maximum lateral acceleration of $a_{\mathrm{y,max}} = 6\,\mathrm{m\,s^{-2}}$.

The simulation procedure to evaluate each neural network type consists of four laps. During the first lap, the vehicle is unaffected by any fault condition, while during the remaining three laps, the vehicle is affected by one of the known fault conditions for the entire lap. This procedure remains consistent for each neural network type. Figure 6.12 illustrates, as an example, the output of the ANN type during this procedure. Each subplot displays the confidence of the ANN and the ground truth for one lap. For the unaffected vehicle in the first lap, the ground truth is 0, while for the remaining three laps where the vehicle is affected by a fault condition, the ground truth is 1.

The top left subplot displays the results for an unaffected vehicle. The ANN often exhibits uncertainty, even erroneously showing more than 50 % confidence that the vehicle is affected by a fault condition. Similarly, in the top right plot, when the vehicle is actually affected by a fault condition, the ANN also frequently exhibits uncertainty. However, for the other two fault conditions, the ANNs predictions are nearly 100 % certain and accurate.

The ANNs difficulty in accurately distinguishing between an unaffected vehicle and one with additional weight may stem from training data points that are quite similar. The impact of additional weight on tracking behavior is not always significant enough to be clearly evident in the feature data. For instance, when traveling straight at a constant velocity, there may be no discernible difference in the feature data. However, pedal positions may reveal a fault condition, as they can indicate a need for more power or brake pressure to achieve desired accelerations.

The detailed results for the remaining three networks are not illustrated, as they are quite similar in terms of their informative value. However, to compare



(A) Lap 1: Unaffected vehicle

(B) Lap 2: Additional weight

(C) Lap 3: Steering offset

(D) Lap 4: Grinding brake disc

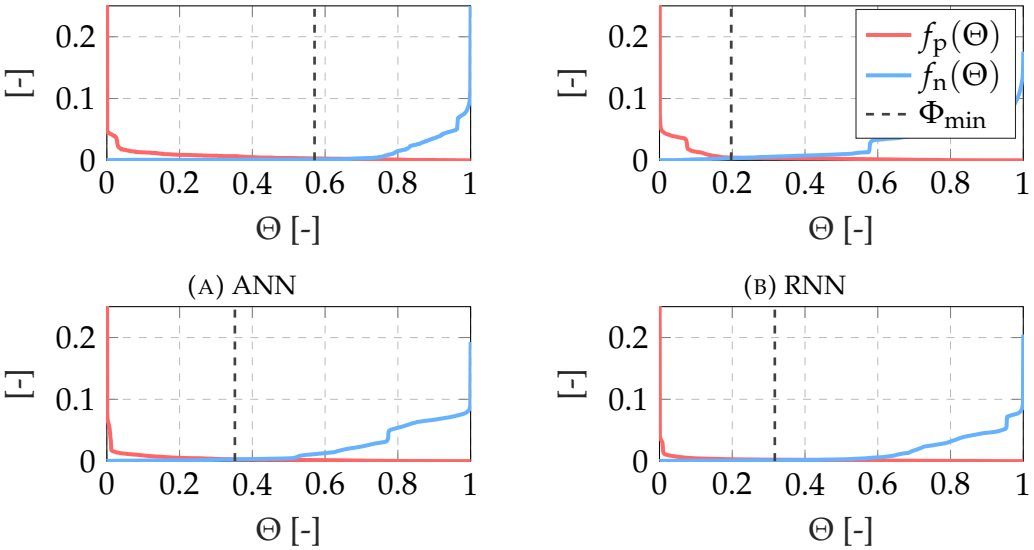
FIGURE 6.12: Fault detection confidence of the ANN for each lap of the evaluation procedure. The dashed red line indicates the ground truth regarding whether the vehicle is affected by a fault condition.

all neural network types, a decisive threshold Φ is introduced. By applying this threshold, the output of the neural networks provides a clear prediction of whether a fault condition is detected. The threshold ranges from 0 to 1, and adjusting it, results in changes to the false-positive and false-negative rates of the neural networks. Figure 6.13 illustrates these rates over the entire threshold range for each neural network type. The red lines represent the false-positive rates, while the blue lines represent the false-negative rates. The gray dashed line indicates the threshold value Φ_{min}, where the sum of both rates results in a minimum, indicating the neural network's highest correct prediction rate across all conditions.

The threshold Φ_{min} can then be used to compare the neural network types. Table 6.8 lists the percentages of correct fault detection for each neural network type across all four conditions individually.

All neural network types are capable of detecting all fault conditions with at least 98 % accuracy. These results suggest that all neural network types reliably detect fault conditions when their training data includes samples of the specific fault conditions. When comparing all four conditions, the GRU type achieves the best overall results, followed by the ANN, LSTM, and RNN.

In order to compare the neural network types without a specified threshold, a different method is applied. Figure 6.14 illustrates the false-positive rates plotted against the false-negative rates across the full threshold range of $[0, 1]$. This results in a curve for each neural network type that approaches the point $(0, 0)$, representing an ideal neural network that consistently issues correct detections.



(A) ANN (B) RNN

(C) LSTM (D) GRU

FIGURE 6.13: Illustration of the false-positive and false-negative rates for each neural network type across the threshold range of [0, 1]. The red line represents the false-positive rate, the blue line represents the false-negative rate, and the gray dashed line represents the threshold where the sum of both rates is minimized.

Network	Unaffected	Add. weight	Steering off.	Grind. brake disc
ANN	99.6	98.8	100.0	100.0
RNN	98.8	98.4	99.9	100.0
LSTM	99.3	98.7	100.0	100.0
GRU	99.6	99.1	100.0	100.0

TABLE 6.8: Percentages of correct fault detection with the applied threshold Φ_{min} for all neural network types, categorized by each condition separately.

The area under these curves can be interpreted as an indicator of overall correctness. Table 6.9 provides the area under the curve for each neural network type. A smaller area indicates a higher frequency of correct detections.

In comparing the neural network types, both methods yield similar results. The GRU achieves the best performance, followed by the ANN, LSTM, and RNN. Nevertheless, the differences in performance do not exceed 1 % for a single fault condition, even between the best and the worst neural network types. In summary, all neural network types proved to be applicable to a certain extent as a fault detection system for the simple task of detecting known fault conditions.

Different closed-loop circuits

Since the detailed evaluation before considered only one closed-loop circuit, the next part applies the same procedure to two additional closed-loop

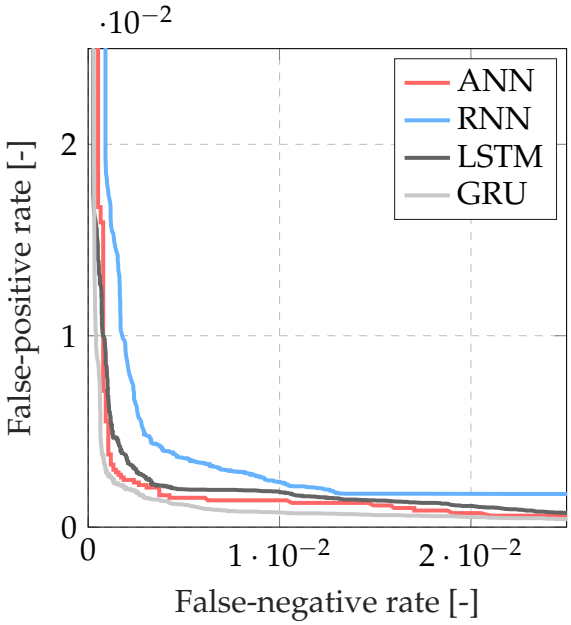


FIGURE 6.14: False-positive rates plotted against false-negative rates across the full threshold range of $[0, 1]$.

Network	Area under curve
ANN	1.4814e−4
RNN	2.3685e−4
LSTM	1.5058e−4
GRU	8.8029e−5

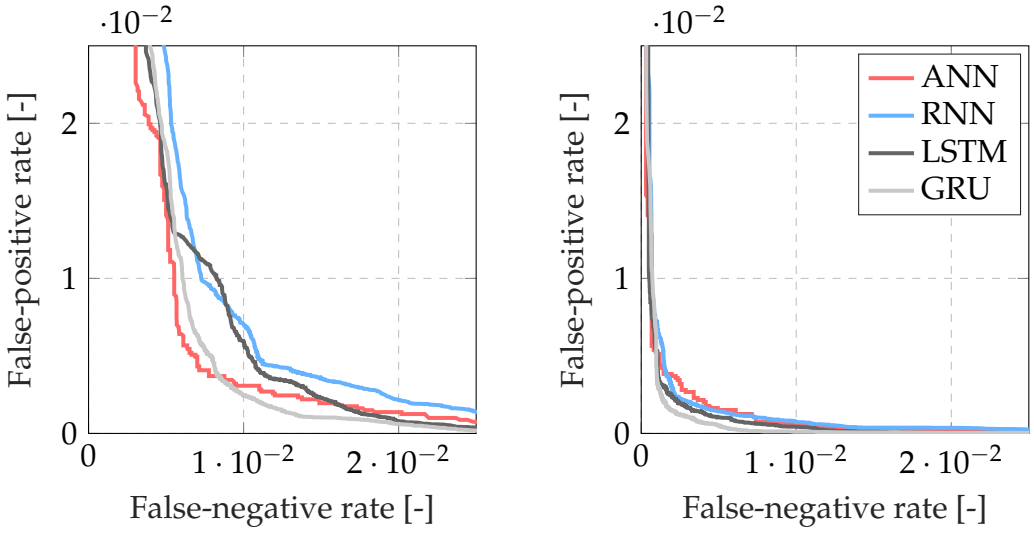
TABLE 6.9: The area under the false-positive over false-negative curves, illustrated in Figure 6.14, serves as an indicator of the correctness of the neural network type across the full threshold range. Lower values indicate higher correctness.

circuits that contributed simulation data to the training dataset. The trajectories used for the evaluation are generated with a maximum lateral acceleration of $a_{y,max} = 6\,m\,s^{-2}$. Table 6.10 lists the percentages of correct fault detection for both circuits. The first value denotes the results for the Hockenheimring and the second value corresponds to the Thunderhill Raceway.

Network	Unaffected	Add. weight	Steering off.	Grind. brake disc
ANN	97.4 | 99.5	98.2 | 98.6	100.0 | 99.9	100.0 | 100.0
RNN	95.7 | 99.3	98.1 | 99.0	99.9 | 100.0	100.0 | 100.0
LSTM	95.8 | 99.5	98.4 | 99.0	100.0 | 100.0	100.0 | 100.0
GRU	96.7 | 99.4	98.6 | 99.4	100.0 | 100.0	99.9 | 100.0

TABLE 6.10: Percentages of correct fault detection for two additional closed-loop circuits. The values to the left of the vertical bars represent the results from the Hockenheimring, while the values to the right represent the results from the Thunderhill Raceway.

Figure 6.15 shows the false-positive rates plotted against the false-negative rates for both evaluations. For each circuit individually, the types of neural networks do not differ significantly from each other. This confirms the previously observed behavior, where no single type of neural network appears to be superior to the others. However, when comparing the evaluations, the results at Thunderhill Raceway (6.15b) are similar to those previously discussed but are significantly better than the results at Hockenheimring (6.15a).



(A) Hockenheimring (B) Thunderhill Raceway

FIGURE 6.15: Evaluation results for two additional closed-loop circuits are presented. The plots show the false-positive rates plotted against the false-negative rates across the full threshold range of [0, 1].

For example, the RNN at the Hockenheimring correctly detected an unaffected vehicle only 95.7 % of the time, while all neural network types at the Thunderhill Raceway were correct more than 99 % of the time for all fault conditions, with only one exception. Since all neural network types appear to struggle at that particular circuit, it is possible that the characteristics and dynamics of the circuit impact the results. The detection data over the traveled distance reveal that the neural networks consistently produce a false positive at one particular corner when the vehicle is not affected by any fault condition. This corner is the sharp right-hand corner to the east of the circuit (see Figure 2.3b). Due to its curvature, the velocity at that corner is quite low compared to other corners. The issue, however, is not the type of corner per se but its representation within the training data. Since all circuits are proper racetracks, the velocities at most corners are relatively high for series production road cars, thereby potentially underrepresenting slower corners within the training data.

Different trajectories

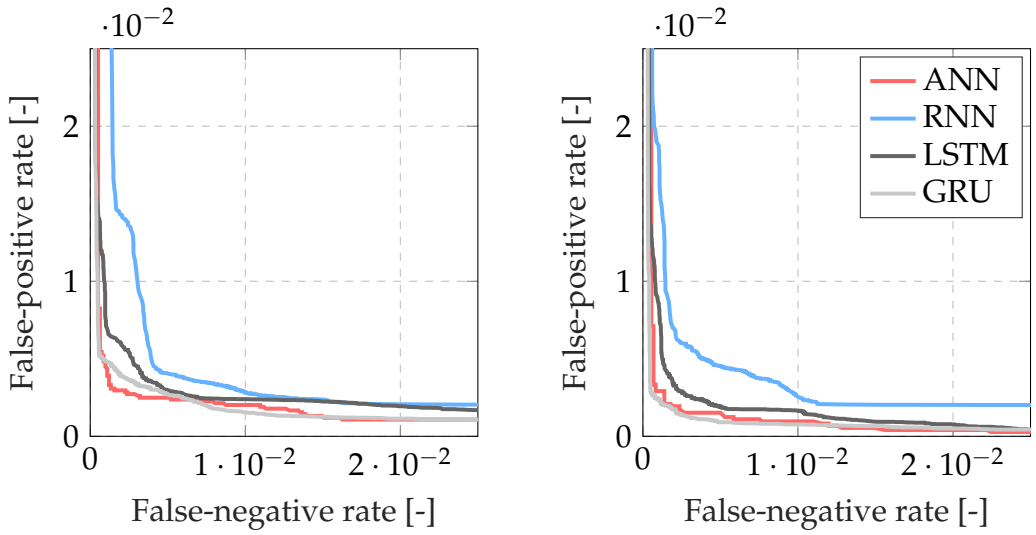
The final evaluation involves applying the test procedure to various trajectories on a single closed-loop circuit. In this case, the trajectories are generated

for the same circuit discussed in Section 6.4.1, but with a maximum lateral accelerations of $4\,\mathrm{m\,s^{-2}}$ and $7\,\mathrm{m\,s^{-2}}$ used in the generation process.

Network	Unaffected	Add. weight	Steering off.	Grind. brake disc
ANN	99.5 | 99.6	98.8 | 99.2	100.0 | 100.0	100.0 | 100.0
RNN	98.4 | 99.2	98.4 | 98.1	100.0 | 100.0	99.9 | 100.0
LSTM	98.8 | 99.2	98.7 | 98.8	100.0 | 100.0	99.8 | 100.0
GRU	99.8 | 99.6	98.7 | 99.2	100.0 | 100.0	99.3 | 100.0

TABLE 6.11: Percentages of correct fault detection for two additional trajectories on the Algarve closed-loop circuit. The values to the left of the vertical bars represent the results for the trajectory based on $a_{y,max} = 4\,\mathrm{m\,s^{-2}}$ and the values to the right represent the results for the trajectory based on $a_{y,max} = 7\,\mathrm{m\,s^{-2}}$.

Table 6.11 lists the percentages of correct fault detection for both trajectories. These results indicate that all types of neural networks achieve a fault detection correctness of at least 98 % for all fault conditions. Figure 6.16 illustrates the false-positive rates plotted against the false-negative rates across the full threshold range. Based on these results, it appears that the correctness of fault detection by the neural network types is independent of the trajectory dynamics for known fault conditions, as there are no significant differences among all neural network types.



(A) $a_{y,max} = 4\,\mathrm{m\,s^{-2}}$ (B) $a_{y,max} = 7\,\mathrm{m\,s^{-2}}$

FIGURE 6.16: Evaluation results for two additional trajectories based on different maximum lateral accelerations. The plots show the false-positive rates plotted against the false-negative rates across the full threshold range $[0, 1]$.

Summary

This section evaluates the capabilities of all four types of neural networks (ANN, RNN, LSTM, and GRU) in detecting known fault conditions through three test cases: a single trajectory on one circuit, different circuits, and different trajectories on the same circuit. Initially, the analysis concentrates solely on the Algarve circuit, utilizing one specific trajectory. This evaluation involves simulating four laps for each type of neural network, with the first lap under normal conditions and the subsequent laps each featuring a distinct known fault condition. This procedure is consistently applied across all test cases. The results are presented using a decisive threshold Φ and include the false-positive and false-negative rates across the full threshold range. This approach allows for a detailed comparison of the neural network types, both in specific and overall terms.

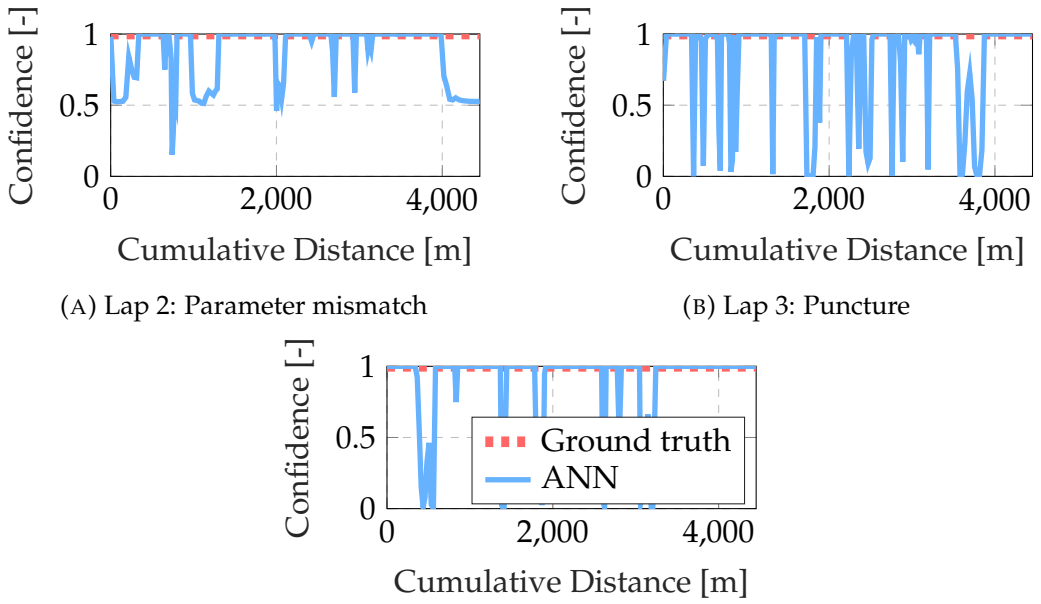
All test cases demonstrated that all types of neural networks are capable of detecting known fault conditions with a high level of correctness. Moreover, the results indicated no significant difference among the capabilities of the different neural network types. The dynamics of different trajectories had almost no influence on the effectiveness of the neural networks. For two of the three circuits utilized, the correctness was at least 98 % for all neural network types across all fault conditions and for the unaffected vehicle. On only one circuit, the correctness of detecting the unaffected vehicle was reduced to at least 95 % for all neural network types, likely due to the underrepresentation of specific trajectory characteristics in the training data, as indicated by measurements from the simulation.

6.4.2 Detecting unknown fault conditions

The evaluation of the neural networks' capability to detect unknown fault conditions follows a methodology similar to that described in Section 6.4.1, with three test cases employing the same test procedure. The first part presents detailed results for a single trajectory on one closed-loop circuit, providing a more in-depth illustration of the evaluation process. The second part showcases results from the two other closed-loop circuits. Lastly, the third test case presents the results for different trajectories on the same closed-loop circuits to assess the operation across a wide range of vehicle dynamics.

Detailed results for a single trajectory

The first test case evaluates the types of neural networks on the Algarve closed-loop circuit (see Figure 2.3a) using a trajectory generated with a maximum lateral acceleration of $a_{\mathrm{y,max}} = 6\,\mathrm{m\,s^{-2}}$. The test procedure encompasses four laps. During the first lap, the vehicle is not affected by any fault condition, while in the remaining three laps, the vehicle is affected by the unknown fault conditions listed in Table 6.1. Since the results for the first lap with an unaffected vehicle are identical to those in Section 6.4.1, they are not shown here again.



(A) Lap 2: Parameter mismatch

(B) Lap 3: Puncture

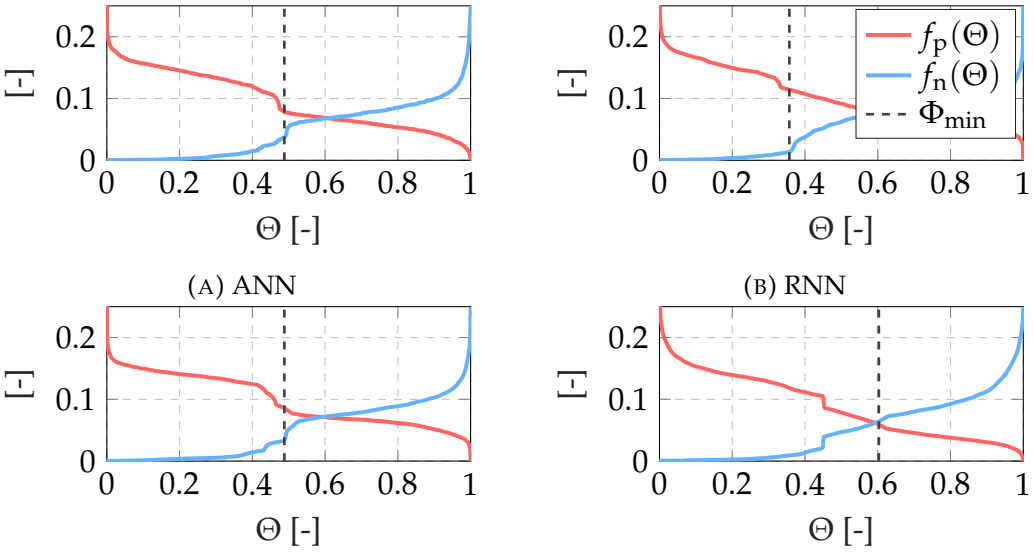
(C) Lap 4: Wind

FIGURE 6.17: Fault detection confidence of the ANN for each lap of the evaluation procedure. The dashed red line indicates the ground truth regarding whether the vehicle is affected by a fault condition.

Figure 6.17 serves as an example, illustrating the output of the ANN during the test procedure. Each subplot displays the ANNs confidence as a blue line and the ground truth as a red dashed line for one lap. As before, the ground truth is set to 1 when the vehicle is affected by a fault condition. For all unknown fault conditions, the ANN exhibits a lack of full confidence in its predictions. This is particularly evident for the fault conditions puncture and wind, where there are instances in which the ANNs prediction drops to 0, erroneously indicating full confidence that the vehicle is not affected by any fault condition. A possible explanation, beyond the absence of sufficient training data, could be the minimal impact of these fault conditions on the vehicle's tracking behavior. For instance, in the simulation, wind blows in a fixed global direction, which has a minimal effect on the vehicle when it travels in alignment with the wind. As for the puncture fault condition, the relatively straightforward implementation approach of applying additional forces to a tire and reducing the friction coefficient might be inadequate. Consequently, the vehicle's tracking behavior remains largely unaffected by these fault conditions in certain situations.

As before, the detailed results for the other neural network types are similar in terms of their informative value. To compare all neural network types, the decisive threshold Φ is determined. Figure 6.18 displays the false-positive and false-negative rates for each neural network type as blue and red lines, respectively. The determined threshold Φ_{min} is illustrated as a gray dashed line.

Applying these thresholds enables the comparison of all neural network types. Table 6.12 lists the percentages of correct fault detection for each neural network type across all unknown fault conditions.



(A) ANN (B) RNN

(C) LSTM (D) GRU

FIGURE 6.18: Illustration of the false-positive and false-negative rates for each neural network type across the threshold range $[0, 1]$. The red line represents the false-positive rate, the blue line represents the false-negative rate, and the gray dashed line indicates the threshold where the sum of both rates is at a minimum.

Network	Parameter	Puncture	Wind
ANN	97.9	84.3	86.9
RNN	88.9	80.5	85.1
LSTM	94.0	84.1	86.9
GRU	97.9	88.5	88.5

TABLE 6.12: Percentages of correct fault detection with the applied threshold Φ_{min} for all neural network types, for each condition separately.

In contrast to the evaluation of known fault conditions, all types of neural networks exhibit overall lower percentages of correct fault detection in this assessment. The highest percentage of correct fault detection in this evaluation is 97.9 %, whereas in the previous evaluation, the lowest percentage was 98.4 %. Therefore, all neural network types perform worse on fault conditions that were not represented in the training data.

In addition, significant differences exist between the individual fault conditions. For all types of neural networks, the fault condition of parameter mismatch is correctly detected approximately 10 % more often than the other fault conditions. However, as mentioned previously, the implementation of the puncture and wind fault conditions may influence these results, as they do not always significantly affect the tracking behavior.

Furthermore, this evaluation of unknown fault conditions reveals that the RNN overall does not perform at the same level as the other neural network

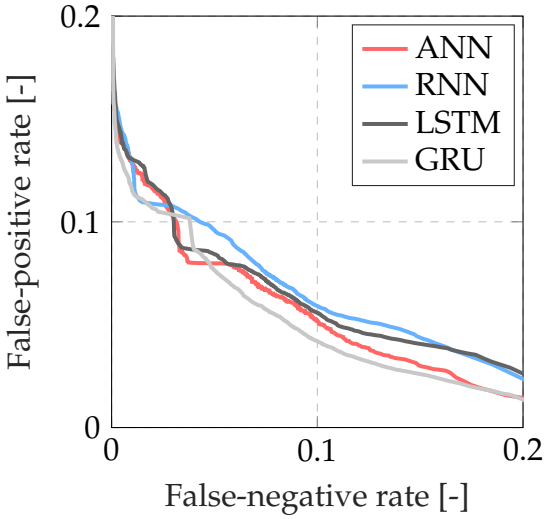


FIGURE 6.19: False-positive rates plotted against false-negative rates across the full threshold range $[0, 1]$.

types. The RNN exhibits the lowest correct detection rates across all fault conditions. Notably, in detecting the parameter mismatch, where the other neural network types perform close to their results in the previous evaluation, the RNNs performance is inferior by up to 8 %.

Network	Area under curve
ANN	1.11e−2
RNN	1.37e−2
LSTM	1.28e−2
GRU	1.06e−2

TABLE 6.13: Area under the curve for false-positive rates plotted against false-negative rates, as illustrated in Figure 6.19. This serves as an indicator of the accuracy of each neural network type across the full threshold range, where a lower value indicates higher accuracy.

In contrast to the detection of known fault conditions, these results indicate a difference in performance among the neural network types. Figure 6.19 displays the false-positive rates plotted against the false-negative rates across the full threshold range. Table 6.13 lists the corresponding areas under these curves. Similar to the comparison using a decisive threshold, the performance of the RNN is again worse than that of the other neural network types, while the ANN and GRU demonstrate the best performance.

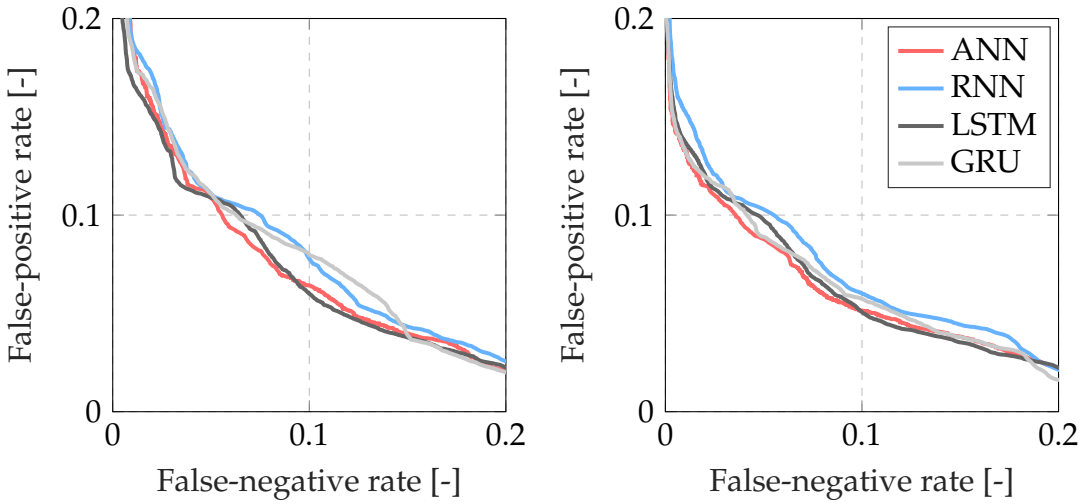
Different closed-loop circuits

The next part of the evaluation applies the same test procedure as before to two remaining closed-loop circuits that contributed to the training data. The trajectories used for the evaluation are also generated with a maximum lateral

acceleration of $a_{y,max} = 6\,m\,s^{-2}$. The percentages of correct fault detection for both circuits are listed in Table 6.14. The first value represents the results for the Hockenheimring and the second value represents the results for the Thunderhill Raceway.

Network	Puncture	Wind	Parameter
ANN	81.1 | 89.8	89.4 | 86.1	91.9 | 84.8
RNN	79.3 | 87.1	87.4 | 84.2	87.0 | 85.4
LSTM	79.5 | 87.8	86.9 | 83.1	86.9 | 83.6
GRU	85.1 | 91.7	86.2 | 85.1	86.5 | 90.5

TABLE 6.14: Percentages of correct fault detection for two additional closed-loop circuits. The values to the left of the vertical bars represent the results for the Hockenheimring and the values to the right represent the results for the Thunderhill Raceway.



(A) Hockenheimring (B) Thunderhill Raceway

FIGURE 6.20: Evaluation results for two additional closed-loop circuits. Plots illustrate the false-positive rates plotted against the false-negative rates across the full threshold range $[0, 1]$.

The percentages of correct fault detection indicate that the ANN and GRU neural network types generally perform better than the RNN and LSTM. However, in detecting wind and parameter mismatch at the Hockenheimring, the RNN and LSTM outperform the GRU. The ANN and GRU each excel in three out of the six evaluations. The false-positive and false-negative rates across the full threshold range are depicted in Figure 6.20. The areas under the curves for the ANN and GRU are smaller than those for the RNN and LSTM, indicating higher performance.

In summary, the ANN and GRU outperform the RNN and LSTM both at a threshold of Φ_{min} and across the full threshold range.

Different trajectories

The third part of the evaluation involves applying the test procedure to different trajectories on the same closed-loop circuit. The trajectories are generated with maximum lateral accelerations of $4\,\mathrm{m\,s^{-2}}$ and $7\,\mathrm{m\,s^{-2}}$ on the Algarve closed-loop circuit. Table 6.15 lists the percentages of correct fault detection for both trajectories based on the evaluations.

Network	Puncture	Wind	Parameter
ANN	93.3 | 78.3	89.3 | 87.5	92.6 | 98.3
RNN	86.7 | 77.5	86.4 | 83.9	78.7 | 88.3
LSTM	92.8 | 80.1	88.2 | 87.3	89.8 | 96.3
GRU	92.0 | 79.5	89.2 | 87.3	86.6 | 82.3

TABLE 6.15: Percentages of correct fault detection for two additional trajectories on the Algarve closed-loop circuit. The values to the left of the bar represent the results for the trajectory based on $a_{\mathrm{y,max}} = 4\,\mathrm{m\,s^{-2}}$ and the values to the right of the bar represent the results for the trajectory based on $a_{\mathrm{y,max}} = 7\,\mathrm{m\,s^{-2}}$.

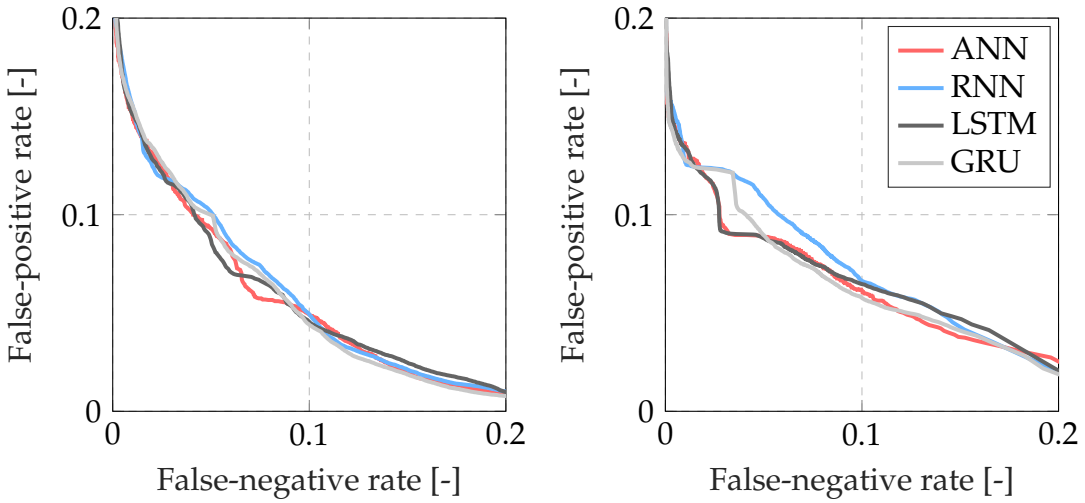
Overall, the ANN demonstrates the best performance, while the RNN again performs worse than all other neural network types. In contrast to the first two parts of the evaluation, the LSTM performs better than the GRU. The LSTM and GRU exhibit similar performances for the fault conditions of puncture and wind, while the parameter mismatch presents an outlier where the GRUs performance is almost 15 % worse than that of the LSTM.

Figure 6.21 illustrates the false-positive rates plotted against the false-negative rates for both evaluations. The curves inidicate that the LSTM outperforms the GRU across the full threshold range. There is a significant discrepancy between the ANN/LSTM and the GRU/RNN in the right plot at approximately $(0.05, 0.1)$. Such a discrepancy could explain the outlier between the GRU and LSTM for the parameter mismatch observed in Table 6.15.

Summary

This section evaluates the capabilities of four types of neural networks (ANN, RNN, LSTM, and GRU) in detecting unknown fault conditions through three test cases: a single trajectory on one circuit, various circuits, and different trajectories on the same circuit. The procedures are identical to those used in the evaluations for known fault conditions in the previous section.

All test cases demonstrated that all types of neural networks are capable of detecting unknown fault conditions with at least 78.7 % accuracy. None of the neural network types exceed 98 % accuracy in any combination of closed-loop circuits and trajectories. In two of the three test cases, the ANN and the GRU outperform the LSTM and the RNN, both at a threshold of Φ_{min} and across the full threshold range. Overall, the RNN performs worse than the other neural network types in all test cases, while the ANN exhibits the best performance.



(A) $a_{y,max} = 4ms^{-2}$ (B) $a_{y,max} = 7ms^{-2}$

FIGURE 6.21: Evaluation results for two additional trajectories based on different maximum lateral accelerations. Plots illustrate the false-positive rates plotted against the false-negative rates across the full threshold range $[0, 1]$.

6.5 Conclusion

The simulation study evaluated four different types of neural networks (ANN, RNN, LSTM, and GRU) for their applicability as fault detection systems. The simulation environment utilized CarMaker, Matlab/Simulink, and TensorFlow. The study was divided into evaluations of known and unknown fault conditions. A fault condition is classified as known when the training data includes simulation data representing it. The feature selection for the training data was based on an automated driving function and its sensors.

The evaluation of known fault conditions indicates that all types of neural networks are generally capable of detecting fault conditions and an unaffected vehicle state with an accuracy of at least 95 %. Furthermore, the results demonstrate that there is no significant difference between the neural network types in terms of their detection accuracy for known fault conditions. Additionally, the findings from the Hockenheimring suggest that specific circuit characteristics, which are underrepresented in the training data, are likely to result in incorrect detection outcomes from the neural networks.

In contrast to the detection of known fault conditions, the evaluation of unknown fault conditions reveals differences among the neural network types. Generally, the ANN and GRU exhibit better performances than the RNN and LSTM. Except for one outlier case involving the GRU, the performances of the ANN and GRU are not significantly different. Nonetheless, all neural network types are capable of detecting unknown fault conditions with an accuracy of at least 77 %.

Notably, the study utilized a single, fixed training dataset throughout, suggesting that further exploration into the impact of training data composition

could yield significant insights. Future work should investigate how neural networks perform when trained with datasets that vary in the ratio of unaffected vehicle data points to fault conditions. This could provide a deeper understanding of how data diversity influences network performance.

Moreover, the generation of synthetic training data presents a promising area for exploration. By creating synthetic data, researchers can cover a broader spectrum of fault conditions without the need for exhaustive simulation of every possible fault scenario. This approach could greatly enhance the neural networks' detection accuracy by ensuring they are well-trained across a more extensive range of trajectory dynamics and vehicle behaviors. These efforts could pave the way for more robust and reliable fault detection mechanisms, ultimately contributing to the safety and efficiency of automated driving technologies.

Chapter 7

Fault detection with learning-based control

This chapter explores the application of a neural network as a fault detection system in conjunction with a learning-based control approach for trajectory tracking. This involves applying the neural network not only alongside the baseline MPC, as discussed in Chapter 6, but also in tandem with the learning-based MPC approach from Chapter 5. Consequently, the impact of fault conditions on tracking behavior may be altered or even nullified by the adaptive capabilities of the learning-based MPC.

The goal is to briefly evaluate the influence of the learning-based control approach on the performance of neural networks as fault detection systems, using training data that does not include any simulation data from the learning-based MPC. Two experiments are conducted in Section 7.1, one involving different trajectories and another involving different closed-loop circuits, with a detailed illustration provided for one of the trajectories. The chapter concludes with a summary in Section 7.2.

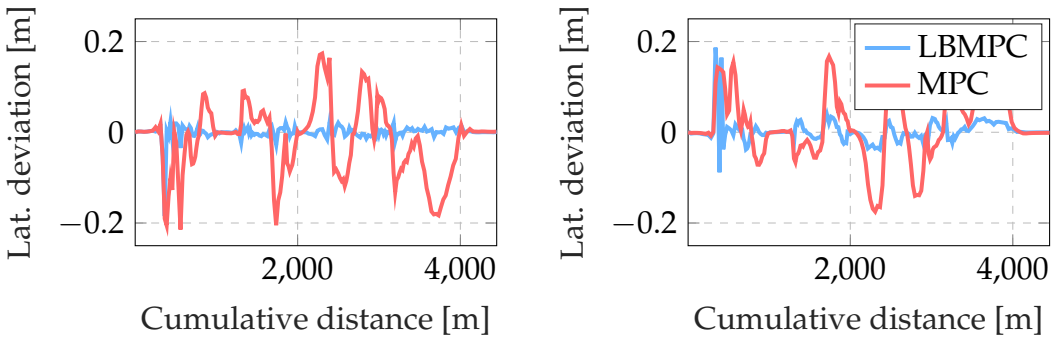
7.1 Experiments

This section presents the results of two experiments designed to evaluate the applicability of neural networks as a fault detection system in conjunction with a learning-based MPC approach for trajectory tracking control. Unlike the comprehensive simulation study conducted in the previous chapter, this analysis focuses on similar experiments for one known and one unknown fault condition. Before presenting the results, detailed outcomes for a single test case on the Algarve closed-loop circuit are discussed in greater depth.

All test cases in the experiments adhere to the same procedure, consisting of four laps: two laps where the vehicle is affected by a known fault condition and two laps where it is affected by an unknown fault condition. On the first lap of each pair, the learning algorithm is active, while on the second lap, only the baseline MPC is responsible for the trajectory tracking task. The known fault condition is simulated by adding additional weight, while the unknown fault condition is simulated through a parameter mismatch. The implementation of both fault conditions is consistent with those previously used, as listed in Table 6.1.

7.1.1 Detailed results for a single trajectory

The test case for a more detailed discussion is conducted on the Algarve circuit, utilizing the ANN as the neural network architecture and a trajectory generated with a maximum lateral acceleration of $a_{y,max} = 6\,m\,s^{-2}$. Figure 7.1 illustrates the lateral deviation over the traveled or cumulative distance. The left-hand plot shows the two laps where the vehicle is affected by a known fault condition, with the blue line representing measurements from the learning-based MPC approach and the red line representing those from the baseline MPC. The right-hand plot shows the lateral deviation for the two laps where the vehicle is affected by an unknown fault condition. As anticipated, in both scenarios, the learning-based MPC compensates for the model mismatch, reducing the lateral deviation and thereby improving the overall tracking behavior.

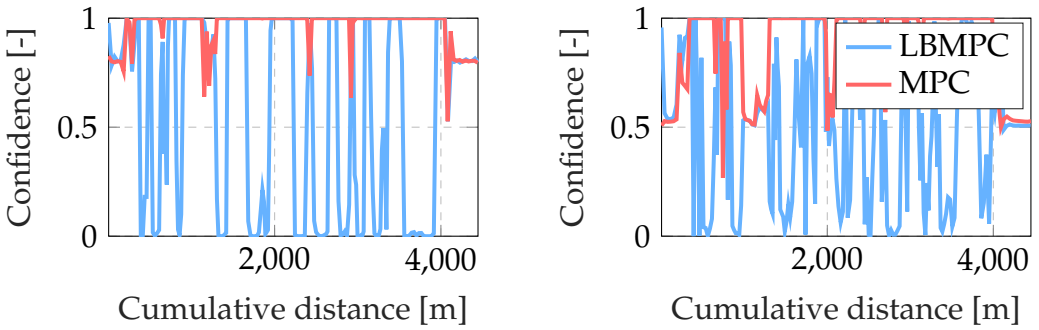


(A) Additional weight (known fault) (B) Parameter mismatch (unknown fault)

FIGURE 7.1: Lateral deviation over cumulative distance, featuring the learning-based MPC (blue) and the baseline MPC (red) as trajectory tracking controllers. In both plots, the controlled vehicle is affected by a fault condition.

The output of the ANN over the traveled distance is depicted in Figure 7.2, corresponding to Figure 7.1, where the blue lines represent the learning-based MPC, and the red lines represent the baseline MPC. The plot on the left-hand side displays the results for the simulation under a known fault condition (laps 1 and 2), and the plot on the right-hand side displays the results under an unknown fault condition (laps 3 and 4). A confidence value of 1 indicates that the ANN is fully confident the vehicle is affected by a fault condition. Since the vehicle is indeed affected by a fault condition during each lap, the optimal scenario would have the ANN output 1 over the entire traveled distance.

The plot on the left-hand side, showing the vehicle affected by a known fault condition, illustrates that the ANN, in combination with the baseline MPC (red), exhibits maximum confidence for most of the time over a full lap distance, while never dropping below 50 %. In contrast, the combination of the ANN with the learning-based MPC (blue) fluctuates heavily between maximum and minimum confidence levels. Further simulation data of the ANNs output over the trajectories' curvature indicate that the ANN remains correctly confident that the vehicle is affected by a fault condition when traveling in a straight line. However, during cornering, the ANNs confidence can drop to a value of 0,



(A) Additional weight (known fault) (B) Parameter mismatch (unknown fault)

FIGURE 7.2: Confidence output of the ANN over the cumulative distance, featuring the learning-based MPC (blue) and the baseline MPC (red) as trajectory tracking controllers. In both plots, the controlled vehicle is affected by a fault condition.

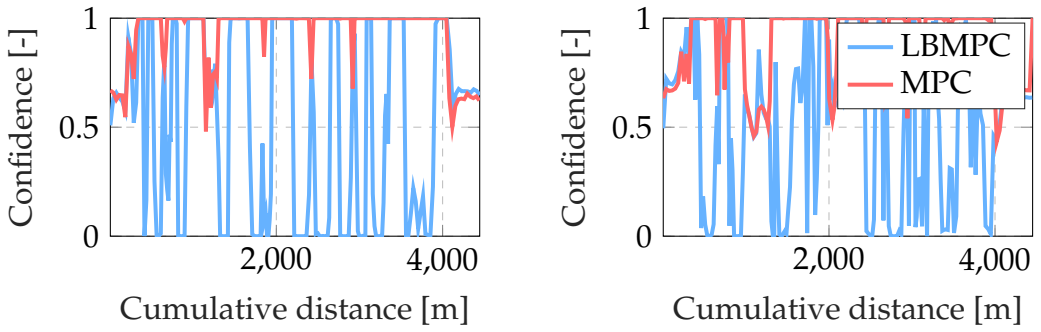
which essentially means that the ANN is incorrectly confident that the vehicle is not affected by a fault condition. The significant improvements in tracking performance (Fig. 7.1a) caused by the learning-based MPC seemingly disguise the fault condition to the ANN, especially during cornering. The tracking performance becomes quite similar to that of an unaffected vehicle. Consequently, the remaining differences, such as in yaw acceleration and control outputs, are insufficient for the ANN to distinguish between the tracking behavior of an unaffected vehicle and that which is improved via a learning-based MPC.

Analogous to the simulation results for the known fault condition, Figure 7.2b on the right-hand side illustrates the confidence output of the ANN for a vehicle affected by an unknown fault condition. Compared to the known fault condition, the ANN in combination with the baseline MPC more often lacks full confidence that a fault condition is present, as already discussed in Chapter 6. Similarly, the ANN combined with the learning-based MPC exhibits fluctuations in fault detection accuracy comparable to its performance for the known fault condition.

Results for the remaining neural network types

Given that the detailed results of the ANN suggest its efficacy as a fault detection system is affected by the learning-based MPC, the performance of the remaining neural network types has also been evaluated using the same test procedure. These results are depicted in Figure 7.3.

All other types of neural networks, namely the RNN, LSTM, and GRU, exhibit similar results for both known and unknown fault conditions. All networks display the same detection behavior as the ANN, with confidence levels fluctuating significantly. There are instances where the neural network is wrongly fully confident that the vehicle is not affected by a fault condition. In comparison to their application with the baseline MPC, there are significant differences regarding detection accuracy.



(A) RNN: Additional weight

(B) RNN: Parameter mismatch

(C) LSTM: Additional weight

(D) LSTM: Parameter mismatch

(E) GRU: Additional weight

(F) GRU: Parameter mismatch

FIGURE 7.3: Confidence output of the RNN (top), LSTM (middle), and GRU (bottom) over the cumulative distance, featuring the learning-based MPC (blue) and the baseline MPC (red) as trajectory tracking controllers. In the plots on the left-hand side, the vehicle is affected by a known fault condition, while in the plots on the right-hand side, the vehicle is affected by an unknown fault condition.

Overview of the results

To compare the different types of neural networks, the decisive threshold is set to $\Phi_{\text{min}} = 0.5$ for all networks. It is noteworthy that the results for the neural networks in conjunction with the baseline MPC differ slightly from the results in Chapter 6, as the applied threshold was determined by the minimum of false positives and false negatives.

The percentages of correct fault detection over one lap are listed in Table 7.1.

The columns correspond to the lap numbers of the test procedure introduced in this chapter. The results show that the difference between the types of neural networks does not exceed approximately 7 %. This finding also suggests that the various neural network types do not significantly differ as fault detection systems when applied alongside a learning-based MPC.

Interestingly, the performance order is reversed for the LBMPC when comparing the neural network types across different fault conditions. For instance, the GRU performs best with the unknown fault and worst with the known fault. Overall, all neural network types exhibit slightly better performance for an unknown fault condition when applied in conjunction with the LBMPC.

Network	Known fault		Unknown fault	
	MPC	LBMPC	MPC	LBMPC
ANN	99.7	54.4	98.4	60.6
RNN	99.5	53.9	96.3	62.0
LSTM	99.6	50.9	96.1	65.2
GRU	99.5	56.2	93.4	58.8

TABLE 7.1: Percentages of correct fault detection with an applied threshold $\Phi_{\text{min}} = 0.5$ for all neural network types in conjunction with the baseline MPC and a learning-based MPC, for both known and unknown fault conditions.

7.1.2 Results for different closed-loop circuits

The test procedure from the detailed results section has been applied to the two remaining circuits. The percentages of correct fault detection are listed in Table 7.2, based on the decisive threshold $\Phi_{\text{min}} = 0.5$. For completeness and comparability, the results for the Algarve circuit are included as well. The utilized trajectories for the different circuits are all generated with a maximum lateral acceleration of $6\,\text{m}\,\text{s}^{-2}$.

Overall, and as expected, all types of neural networks perform significantly better with the baseline MPC than with the learning-based MPC. Contrary to the results presented for the Algarve circuit, all types of neural networks achieve better performance in detecting known fault conditions than unknown fault conditions on the Hockenheimring and the Thunderhill Raceway. Furthermore, the discrepancy in performance among the neural networks is larger for the two additional circuits. For instance, the maximum difference in performance on the Algarve circuit was 4.6 % (for unknown faults with the LBMPC between ANN and LSTM). For the Hockenheimring and the Thunderhill Raceway, there are instances of up to 15 % difference in performance between types. Apart from that, the results for neural networks in conjunction with the LBMPC appear quite random, except for the observation that the LSTM consistently achieves the best performance in detecting unknown faults across all circuits.

Circuit	Network	Known fault		Unknown fault	
		MPC	LBMPC	MPC	LBMPC
Algarve	ANN	99.7	54.4	98.4	60.6
	RNN	99.5	53.9	96.3	62.0
	LSTM	99.6	50.9	96.1	65.2
	GRU	99.5	53.9	96.3	62.0
Hockenheim	ANN	99.7	67.7	87.8	44.5
	RNN	98.8	55.3	80.3	38.2
	LSTM	99.0	59.0	89.5	53.9
	GRU	98.9	65.9	78.4	39.7
Thunderhill	ANN	99.7	52.9	86.7	32.9
	RNN	99.4	45.9	81.7	30.5
	LSTM	99.4	51.0	90.5	46.4
	GRU	99.3	49.3	80.9	30.4

TABLE 7.2: Percentages of correct fault detection with an applied threshold $\Phi_{\text{min}} = 0.5$ for all neural network types across three different closed-loop circuits. Neural networks are utilized in conjunction with the baseline MPC and the learning-based MPC for both known and unknown fault conditions.

7.1.3 Results for different trajectories

Considering that all trajectories in the previous test case were generated with the same maximum lateral acceleration, this test case extends the procedure to include trajectories generated with $4\,\mathrm{m\,s^{-2}}$, $6\,\mathrm{m\,s^{-2}}$, and $7\,\mathrm{m\,s^{-2}}$ on the Algarve circuit. All results are listed in Table 7.3. The results for the trajectory generated with $6\,\mathrm{m\,s^{-2}}$ are consistent with those from the two previous test cases and are included again for completeness and comparability.

As in the other test cases, the performance of all neural network types is again better in conjunction with the baseline MPC than with the learning-based MPC. Furthermore, there are no significant differences among the neural network types for each combination of trajectory, fault condition, and MPC. The largest difference is 8 % between the GRU and the ANN for an unknown fault condition in conjunction with the LBMPC.

The performance difference among the trajectories is more pronounced. Both known and unknown fault conditions are detected more accurately for the trajectory with the lowest maximum lateral acceleration compared to the two other trajectories. When comparing the trajectory with $6\,\mathrm{m\,s^{-2}}$ to $7\,\mathrm{m\,s^{-2}}$, all types of neural networks detect known faults more accurately for $6\,\mathrm{m\,s^{-2}}$. However, for $7\,\mathrm{m\,s^{-2}}$, the detection of the unknown fault by all neural network types is less accurate.

Trajectory ($a_{y,max}$)	Network	Known fault		Unknown fault	
		MPC	LBMPC	MPC	LBMPC
$4\,m\,s^{-2}$	ANN	99.9	83.6	94.7	72.7
	RNN	99.7	79.9	91.9	67.5
	LSTM	99.7	82.7	93.1	69.3
	GRU	99.6	85.4	87.6	64.4
$6\,m\,s^{-2}$	ANN	99.7	54.4	98.4	60.6
	RNN	99.5	53.9	96.3	62.0
	LSTM	99.6	50.9	96.1	65.2
	GRU	99.5	53.9	96.3	62.0
$7\,m\,s^{-2}$	ANN	99.7	61.7	99.0	52.3
	RNN	98.8	62.0	96.5	53.8
	LSTM	99.6	63.0	97.6	55.2
	GRU	99.4	64.6	93.1	50.1

TABLE 7.3: Percentages of correct fault detection with an applied threshold $\Phi_{min} = 0.5$ for all neural network types across three different trajectories. Neural networks are utilized in conjunction with the baseline MPC and the learning-based MPC for both known and unknown fault conditions.

7.2 Conclusion

The test cases evaluate four types of neural networks as fault detection systems in conjunction with a learning-based MPC across various closed-loop circuits and trajectories. The outcomes highlight a notable decline in the accuracy of all neural network types for detecting both known and unknown fault conditions, with a more pronounced decrease for unknown fault conditions. For instance, the GRUs accuracy dropped by 50 % for detecting an unknown fault condition when used alongside the LBMPC on the Thunderhill Raceway.

As with the previous evaluation, there is no significant, consistent difference among neural network types. Larger discrepancies arise instead from different circuits and trajectories with their unique dynamics. For instance, the maximum discrepancy for any neural network type between the MPC and LBMPC is 20 % for the trajectory at $4\,m\,s^{-2}$, while for the trajectory at $6\,m\,s^{-2}$, it is up to 50 %. In both instances, the maximum difference between the neural network types amounts to only a few percent.

The learning-based MPC effectively mitigates model mismatch caused by both known and unknown fault conditions, thereby improving tracking performance and minimizing lateral deviation. Since lateral deviation is a crucial feature in the training data, neural networks may lack an essential factor for fault detection. This is particularly evident in corners, where the LBMPCs adjustment significantly changes tracking behavior, leading to a notable drop in detection accuracy despite correct fault detection on straights.

Chapter 8

Conclusion and outlook

The concluding chapter of this dissertation provides a concise summary of the research conducted and its contributions. This overview emphasizes the key insights and findings derived from all the simulation studies. The chapter concludes with a forward-looking perspective, proposing directions for future work that can build on the foundations laid in this dissertation.

8.1 Conclusion

This thesis is structured into two main parts: the development of adaptive model predictive controllers (AMPC) for trajectory tracking and the evaluation of various neural network types as fault detection systems for trajectory tracking controllers. Both parts focus on enhancing the safety and reliability of trajectory tracking control systems through the application of machine learning techniques. By integrating AMPC and fault detection mechanisms that utilize neural networks, this thesis aims to contribute to the development of robust and fault-tolerant control systems. These systems are designed to ensure higher safety standards and maintain operational integrity in dynamic environments for trajectory tracking.

In the first part of this thesis, two AMPC approaches are sequentially developed to mitigate the loss of tracking accuracy due to model errors and changing conditions. The initial approach utilizes a trajectory-dynamic lookup table to adjust the vehicle model's yaw intensification, while the second, more advanced approach employs Gaussian process regression (GPR) and clustering. Both approaches are evaluated in a thorough simulation study. The results demonstrate that the refined AMPC is capable of (**i**) effectively managing condition changes and significantly improving tracking performance, (**ii**) repeatedly adapting to condition changes, (**iii**) handling unknown trajectories with nearly the same improvements in tracking accuracy as known trajectories, and (**iv**) memorizing adapted behavior through the application of clustering.

The results in this dissertation demonstrate that the proposed approaches yield significantly better tracking performance than those discussed in the literature. According to the provided figures, these approaches achieved maximum lateral deviations ranging from 0.3 m to 1.0 m during double lane change maneuvers at speeds between $72\,\mathrm{km\,h^{-1}}$ and $90\,\mathrm{km\,h^{-1}}$. Both approaches presented in this dissertation resulted in a maximum lateral deviation of less than

0.1 m across three different closed-loop circuits, with a maximum lateral acceleration of $6\,\mathrm{m\,s^{-2}}$. It is noteworthy that some double lane change maneuvers discussed in the literature might require slightly higher lateral acceleration, making them more demanding for the vehicle and the learning-based control approach. Furthermore, the proposed approaches in this dissertation are partially capable of completely compensating for lost tracking performance, a capability not observed in other approaches.

None of the approaches in the literature evaluated their adapted behavior with respect to unknown trajectories. In contrast, the proposed approaches in this dissertation demonstrated that their adapted behavior is transferable to unknown trajectories with similar dynamics while achieving significant performance improvements. Furthermore, the approach using GPR and clustering introduced the concept of memorizing adapted behavior, which has not yet been explored in the literature. This approach proved capable of immediately reacting to known condition changes in accordance with previously adapted behavior.

In the second part, a simulation study assessed four types of neural networks: Artificial Neural Networks (ANN), Recurrent Neural Networks (RNN), Long Short-Term Memory (LSTM) networks, and Gated Recurrent Unit (GRU) networks for their effectiveness as fault detection systems for a trajectory tracking controller without learning capabilities. The study revealed that for known fault conditions, i.e., conditions included in the training data, the neural networks could detect both faults and unaffected vehicle states with at least 95 % accuracy, with no significant differences among the neural network types. For unknown fault conditions, all neural network types managed to detect faults with at least 77 % accuracy, with ANN and GRU slightly outperforming RNN and LSTM.

In addition, the developed AMPC from the first part was applied as a tracking controller. Simulation results demonstrated a significant decrease in the detection accuracy of all neural network types for both known and unknown fault conditions, with a more notable decrease for unknown fault conditions. Despite this, no significant or consistent differences were observed among the neural network types when paired with a learning-based MPC. Given that the training data did not include any data from an AMPC, these results were expected.

Existing literature has already demonstrated fault detection systems for single sensors or controllers using various approaches. Furthermore, neural networks have been applied to detect system changes or disturbances. This dissertation diverges from monitoring a single system or component and instead examines trajectory tracking control in its entirety. This approach suggests that future applications may not require a separate fault detection system for every component; rather, a single fault detection system could be utilized to detect anomalies in overall tracking behavior. The dissertation demonstrates that neural networks are capable of functioning as a fault detection system using data that is likely to be available in most automated driving functions in series production cars. It also shows that there are only slight differences among various

neural network architectures as fault detection systems, an aspect that has not yet been explored in the literature.

8.2 Outlook

The proposed AMPC with GPR and clustering has demonstrated significant performance during evaluation. This approach could be improved by incorporating additional meaningful feature data, which would help the GPR distinguish tracking behavior more precisely and optimize the yaw intensification more accurately. However, a major limitation is that GPR is computationally expensive for large datasets with its time complexity of $\mathcal{O}(n^3)$. Exploring the application of multiple smaller GPRs could also be a worthwhile approach, as it might allow for the optimization of multiple parameters more efficiently.

Given that GPR quickly becomes a computational bottleneck, exploring neural networks appears to be a viable alternative for future work. Neural networks offer robust function approximation capabilities for predicting system behavior and can handle high-dimensional data, allowing for a much broader selection of features, especially for approximating entire parts of the utilized model. Additionally, neural networks could potentially make the application of clustering algorithms for memorization obsolete due to their superior generalization capabilities.

However, a major challenge in applying neural networks lies in the training procedure. An a priori supervised learning approach that relies on real driving and simulation data may be difficult to realize since the data must represent the condition changes the learning approach is designed to manage. Generating a vast variety of training data is therefore necessary. Robust online reinforcement learning or transfer learning, where the neural network adapts online to condition changes, could be considered as alternatives. Yet, robustness is crucial, as the tracking behavior must not deteriorate or lead to unsafe states. Achieving adaptation speeds comparable to those of GPR, which can rapidly adjust with significant improvements from just a few data points, represents a substantial challenge for neural network-based approaches.

The evaluation of different neural network types as fault detection systems has demonstrated that monitoring full system behavior is feasible for detecting anomalies, although detecting an unaffected vehicle was the least accurate. Since the evaluation is based on a single defined training dataset, investigating the composition of the training data could be beneficial. It is necessary to explore how the ratio of training data for an unaffected vehicle to the data for fault conditions influences the performance of the neural networks and which additional features in the training data could further improve the distinction between a fault condition and an unaffected state.

The evaluation also revealed that all neural network types were significantly less accurate at detecting unknown fault conditions. Creating synthetic training data to cover a broader range of fault scenarios could significantly enhance detection accuracy. This approach would reduce the likelihood of unknown fault

conditions remaining undetected. When the overall detection accuracy is further increased, a logical next step would be to isolate the cause of the fault condition. However, this would likely require a substantial increase in the required feature data to enable the neural networks to distinguish between different subsystems effectively.

In conclusion, integrating AMPC with neural network-based fault detection systems offers a promising direction for creating a unified framework in autonomous vehicle control. This combined approach leverages the strengths of both systems: the AMPCs precision in trajectory tracking and the neural networks' proficiency in detecting anomalies. By merging these technologies, future research could focus on developing a cohesive system that not only enhances the accuracy and efficiency of trajectory tracking control but also improves the reliability and safety of autonomous operations.

Bibliography

[1] Q. Yao, Y. Tian, Q. Wang, and S. Wang, "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.

[2] X. Du, K. K. K. Htet, and K. K. Tan, "Development of a genetic-algorithm-based nonlinear model predictive control scheme on velocity and steering of autonomous vehicles," *IEEE Transactions on Industrial Electronics*, vol. 63, no. 11, pp. 6970–6977, 2016. DOI: 10.1109/TIE.2016.2585079.

[3] C. Shen, H. Guo, F. Liu, and H. Chen, "Mpc-based path tracking controller design for autonomous ground vehicles," in *2017 36th Chinese Control Conference (CCC)*, 2017, pp. 9584–9589. DOI: 10.23919/ChiCC.2017.8028887.

[4] P. F. Lima, M. Nilsson, M. Trincavelli, J. Artensson, and B. Wahlberg, "Spatial model predictive control for smooth and accurate steering of an autonomous truck," *IEEE Transactions on Intelligent Vehicles*, vol. 2, no. 4, pp. 238–250, 2017. DOI: 10.1109/TIV.2017.2767279.

[5] L. Biddle and S. Fallah, "A novel fault detection, identification and prediction approach for autonomous vehicle controllers using svm," *Automotive Innovation*, vol. 4, pp. 301–314, 2021. DOI: 10.1007/s42154-021-00138-0.

[6] L. Hewing, K. P. Wabersich, M. Menner, and M. N. Zeilinger, "Learning-based model predictive control: Toward safe learning in control," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 3, no. Volume 3, 2020, pp. 269–296, 2020. DOI: https://doi.org/10.1146/annurev-control-090419-075625.

[7] M. Tanaskovic, L. Fagiano, R. Smith, and M. Morari, "Adaptive receding horizon control for constrained mimo systems," en, *Automatica*, vol. 50, no. 12, pp. 3019–3029, 2014-12. DOI: 10.3929/ethz-b-000078391.

[8] R. Soloperto, M. A. Müller, S. Trimpe, and F. Allgöwer, "Learning-based robust model predictive control with state-dependent uncertainty," *IFAC-PapersOnLine*, vol. 51, no. 20, pp. 442–447, 2018, 6th IFAC Conference on Nonlinear Model Predictive Control NMPC 2018. DOI: https://doi.org/10.1016/j.ifacol.2018.11.052.

[9] F. Berkenkamp, A. P. Schoellig, and A. Krause, "Safe controller optimization for quadrotors with gaussian processes," *CoRR*, vol. abs/1509.01066, 2015.

[10] M. C. Priess, R. Conway, J. Choi, J. M. Popovich, and C. Radcliffe, "Solutions to the inverse lqr problem with application to biological systems analysis," *IEEE Transactions on Control Systems Technology*, vol. 23, no. 2, pp. 770–777, 2015. DOI: 10.1109/TCST.2014.2343935.

[11] J. F. Fisac, A. K. Akametalu, M. N. Zeilinger, S. Kaynama, J. H. Gillula, and C. J. Tomlin, "A general safety framework for learning-based control in uncertain robotic systems," *CoRR*, vol. abs/1705.01292, 2017.

[12] T. Gurriet, M. Mote, A. D. Ames, and E. Feron, "An online approach to active set invariance," in *2018 IEEE Conference on Decision and Control (CDC)*, 2018, pp. 3592–3599. DOI: 10.1109/CDC.2018.8619139.

[13] R. Zhao, R. Yan, Z. Chen, K. Mao, P. Wang, and R. X. Gao, "Deep learning and its applications to machine health monitoring: A survey," *CoRR*, vol. abs/1612.07640, 2016.

[14] J. Jiao, M. Zhao, J. Lin, and K. Liang, "A comprehensive review on convolutional neural network in machine fault diagnosis," *Neurocomputing*, vol. 417, pp. 36–63, Dec. 2020. DOI: 10.1016/j.neucom.2020.07.088.

[15] A. Borghesi, A. Bartolini, M. Lombardi, M. Milano, and L. Benini, "Anomaly detection using autoencoders in high performance computing systems," *CoRR*, vol. abs/1811.05269, 2018.

[16] A. S. Raihan and I. Ahmed, *A bi-lstm autoencoder framework for anomaly detection – a case study of a wind power dataset*, 2023.

[17] Y. Xu, Z. Li, S. Wang, W. Li, T. Sarkodie-Gyan, and S. Feng, "A hybrid deep-learning model for fault diagnosis of rolling bearings," *Measurement*, vol. 169, p. 108 502, 2021. DOI: https://doi.org/10.1016/j.measurement.2020.108502.

[18] Y. Gao, C. H. Kim, and J.-M. Kim, "A novel hybrid deep learning method for fault diagnosis of rotating machinery based on extended wdcnn and long short-term memory," *Sensors*, vol. 21, no. 19, 2021. DOI: 10.3390/s21196614.

[19] H. Chen, H. Luo, B. Huang, B. Jiang, and O. Kaynak, "Transfer learning-motivated intelligent fault diagnosis designs: A survey, insights, and perspectives," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 35, no. 3, pp. 2969–2983, 2024. DOI: 10.1109/TNNLS.2023.3290974.

[20] Z. Zhao, Q. Zhang, X. Yu, *et al.*, "Applications of unsupervised deep transfer learning to intelligent fault diagnosis: A survey and comparative study," *IEEE Transactions on Instrumentation and Measurement*, vol. 70, pp. 1–28, 2021. DOI: 10.1109/tim.2021.3116309.

[21] K. Jang, K. E. S. Pilario, N. Lee, I. Moon, and J. Na, "Explainable artificial intelligence for fault diagnosis of industrial processes," *IEEE Transactions on Industrial Informatics*, pp. 1–8, 2023. DOI: 10.1109/TII.2023.3240601.

[22] A. B. Arrieta, D. Molina, R. Benjamins, R. Chatila, and F. Herrera, "Explainable artificial intelligence (XAI): concepts, taxonomies, opportunities and challenges toward responsible AI," *CoRR*, vol. abs/1910.10045, 2019.

[23] T. Lubiniecki and G. Schildbach, "Adaptive mpc for trajectory tracking with online adaption of the vehicle model's yaw intensification," in *2023 European Control Conference (ECC)*, 2023, pp. 1–6. DOI: 10.23919/ECC57647.2023.10178261.

[24] T. Lubiniecki and G. Schildbach, "Trajectory tracking mpc with online model adaptation using gaussian process regression and clustering," in *2023 IEEE 26th International Conference on Intelligent Transportation Systems (ITSC)*, 2023, pp. 154–159. DOI: 10.1109/ITSC57777.2023.10422382.

[25] J. Kocijan, R. Murray-Smith, C. Rasmussen, and A. Girard, "Gaussian process model based predictive control," in *Proceedings of the 2004 American Control Conference*, vol. 3, 2004, 2214–2219 vol.3. DOI: 10.23919/ACC.2004.1383790.

[26] M. Maiworm, *Gaussian process in control : model predictive control with guarantees and control of scanning quantum dot microscopy*. 2021, p. 146. DOI: 10.25673/38878.

[27] L. Ortmann, D. Shi, E. Dassau, F. J. Doyle, S. Leonhardt, and B. J. Misgeld, "Gaussian process-based model predictive control of blood glucose for patients with type 1 diabetes mellitus," in *2017 11th Asian Control Conference (ASCC)*, 2017, pp. 1092–1097. DOI: 10.1109/ASCC.2017.8287323.

[28] T. Salzmann, E. Kaufmann, J. Arrizabalaga, M. Pavone, D. Scaramuzza, and M. Ryll, "Real-time neural mpc: Deep learning model predictive control for quadrotors and agile robotic platforms," *IEEE Robotics and Automation Letters*, vol. 8, no. 4, pp. 2397–2404, 2023. DOI: 10.1109/LRA.2023.3246839.

[29] J. M. Assael, N. Wahlström, T. B. Schön, and M. P. Deisenroth, "Data-efficient learning of feedback policies from image pixels using deep dynamical models," *CoRR*, vol. abs/1510.02173, 2015. DOI: 10.48550/arXiv.1510.02173.

[30] J. Quinonero-Candela and C. E. Rasmussen, "A unifying view of sparse approximate gaussian process regression," *The Journal of Machine Learning Research*, vol. 6, pp. 1939–1959, 2005.

[31] H. Bijl, J.-W. van Wingerden, T. B. Schön, and M. Verhaegen, "Online sparse gaussian process regression using fitc and pitc approximations**this research is supported by the dutch technology foundation stw, which is part of the netherlands organisation for scientific research (nwo), and which is partly funded by the ministry of economic affairs. the work was also supported by the swedish research council (vr) via the project probabilistic modeling of dynamical systems (contract number: 621-2013-5524).," *IFAC-PapersOnLine*, vol. 48, no. 28, pp. 703–708,

2015, 17th IFAC Symposium on System Identification SYSID 2015. DOI: 10.1016/j.ifacol.2015.12.212.

[32] Y. Lu, Y. Yue, G. Li, and Z. Wang, "Adaptive fault tolerant control for safe autonomous driving using learning-based model predictive control," in *2023 IEEE International Conference on Mechatronics and Automation (ICMA)*, 2023, pp. 2218–2223. DOI: 10.1109/ICMA57826.2023.10215568.

[33] J. Kabzan, L. Hewing, A. Liniger, and M. N. Zeilinger, "Learning-based model predictive control for autonomous racing," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3363–3370, 2019. DOI: 10.1109/LRA.2019.2926677.

[34] R. Rezvani Arany, "Gaussian process model predictive control for autonomous driving in safety-critical scenarios," 2019.

[35] W. Liu, Y. Zhai, G. Chen, and A. Knoll, "Gaussian process based model predictive control for overtaking scenarios at highway curves," in *2022 IEEE Intelligent Vehicles Symposium (IV)*, 2022, pp. 1161–1167. DOI: 10.1109/IV51971.2022.9827233.

[36] W. Liu, C. Liu, G. Chen, and A. Knoll, "Gaussian process based model predictive control for overtaking in autonomous driving," *Frontiers in Neurorobotics*, vol. 15, p. 723 049, 2021.

[37] L. Hewing and M. N. Zeilinger, "Cautious model predictive control using gaussian process regression," *CoRR*, vol. abs/1705.10702, 2017. DOI: 10.48550/arXiv.1705.10702.

[38] C. J. Ostafew, A. P. Schoellig, and T. D. Barfoot, "Learning-based nonlinear model predictive control to improve vision-based mobile robot path-tracking in challenging outdoor environments," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 4029–4036. DOI: 10.1109/ICRA.2014.6907444.

[39] G. Cao, E. M. Lai, and F. Alam, "Gaussian process model predictive control of an unmanned quadrotor," *CoRR*, vol. abs/1707.04515, 2017.

[40] F. Lin, Y. Chen, Y. Zhao, and S. Wang, "Path tracking of autonomous vehicle based on adaptive model predictive control," *International Journal of Advanced Robotic Systems*, vol. 16, Sep. 2019. DOI: 10.1177/1729881419880089.

[41] Y. Kebbati, V. Puig, N. Ait-Oufroukh, V. Vigneron, and D. Ichalal, "Optimized adaptive mpc for lateral control of autonomous vehicles," in *2021 9th International Conference on Control, Mechatronics and Automation (ICCMA)*, 2021, pp. 95–103. DOI: 10.1109/ICCMA54375.2021.9646218.

[42] S. Vaskov, K. Berntorp, and R. Quirynen, "Cornering stiffness adaptive, stochastic nonlinear model predictive control for vehicles," in *2021 American Control Conference (ACC)*, 2021, pp. 154–159. DOI: 10.23919/ACC50511.2021.9482699.

[43] S. Vaskov, R. Quirynen, M. Menner, and K. Berntorp, "Friction-adaptive stochastic predictive control for trajectory tracking of autonomous vehicles," 2022. DOI: 10.23919/ACC53348.2022.9867523.

[44] A. Jain, P. Chaudhari, and M. Morari, "Bayesrace: Learning to race autonomously using prior experience," *CoRR*, vol. abs/2005.04755, 2020. DOI: 10.48550/arXiv.2005.04755.

[45] D. J. Yeong, G. Velasco-Hernandez, J. Barry, and J. Walsh, "Sensor and sensor fusion technology in autonomous vehicles: A review," *Sensors*, vol. 21, no. 6, 2021. DOI: 10.3390/s21062140.

[46] H. Pan, W. Sun, Q. Sun, and H. Gao, "Deep learning based data fusion for sensor fault diagnosis and tolerance in autonomous vehicles," *Chinese Journal of Mechanical Engineering*, vol. 34, no. 1, 2021. DOI: 10.1186/s10033-021-00568-1.

[47] J. Ren, R. Ren, M. Green, and X. Huang, "A deep learning method for fault detection of autonomous vehicles," in *2019 14th International Conference on Computer Science & Education (ICCSE)*, 2019, pp. 749–754. DOI: 10.1109/ICCSE.2019.8845483.

[48] M. Al-Zeyadi, J. Andreu-Perez, H. Hagras, *et al.*, "Deep learning towards intelligent vehicle fault diagnosis," in *2020 International Joint Conference on Neural Networks (IJCNN)*, 2020, pp. 1–7. DOI: 10.1109/IJCNN48605.2020.9206972.

[49] S. Das, R. Hall, S. Herzog, G. Harrison, M. Bodkin, and L. Martin, "Essential steps in prognostic health management," in *2011 IEEE Conference on Prognostics and Health Management*, 2011, pp. 1–9. DOI: 10.1109/ICPHM.2011.6024332.

[50] H. E. Hadraoui, O. Laayati, A. E. Maghraoui, E. Sabani, M. Zegrari, and A. Chebak, "Diagnostic and prognostic health management of electric vehicle powertrains: An empirical methodology for induction motor analysis," in *2023 5th Global Power, Energy and Communication Conference (GPECOM)*, 2023, pp. 153–158. DOI: 10.1109/GPECOM58364.2023.10175674.

[51] T. Alsuwian, M. H. Usman, and A. A. Amin, "An autonomous vehicle stability control using active fault-tolerant control based on a fuzzy neural network," *Electronics*, vol. 11, no. 19, 2022. DOI: 10.3390/electronics11193165.

[52] C. Dai, S. Yang, and L. Tan, "An approach for controller fault detection," in *Fifth World Congress on Intelligent Control and Automation (IEEE Cat. No.04EX788)*, vol. 2, 2004, pp. 1637–1641. DOI: 10.1109/WCICA.2004.1340931.

[53] T. Salsbury and R. Diamond, "Fault detection in hvac systems using model-based feedforward control," *Energy and Buildings*, vol. 33, no. 4, pp. 403–415, 2001, Special Issue: BUILDING SIMULATION'99. DOI: 10.1016/S0378-7788(00)00122-5.

[54] K. Geng, N. A. Chulin, and Z. Wang, "Fault-tolerant model predictive control algorithm for path tracking of autonomous vehicle," *Sensors*, vol. 20, no. 15, p. 4245, 2020. DOI: 10.3390/s20154245.

[55] K. Geng and S. Liu, "Robust path tracking control for autonomous vehicle based on a novel fault tolerant adaptive model predictive control algorithm," *Applied Sciences*, vol. 10, no. 18, 2020. DOI: 10.3390/app10186249.

[56] D. Asadi, K. Ahmadi, and S. Y. Nabavi, "Fault-tolerant trajectory tracking control of a quadcopter in presence of a motor fault," *International Journal of Aeronautical and Space Sciences*, vol. 23, no. 1, pp. 129–142, 2022. DOI: 10.1007/s42405-021-00412-9.

[57] F. Loquasto and D. Seborg, "Monitoring model predictive control systems using pattern classification and neural networks †," *Industrial & Engineering Chemistry Research - IND ENG CHEM RES*, vol. 42, Apr. 2003. DOI: 10.1021/ie020870k.

[58] N. Panagiotopoulos, T. Lubiniecki, A. Turnwald, and N. Baldauf, "Robust machine learning systems for dependable space applications," in *2023 European Data Handling & Data Processing Conference (EDHPC)*, 2023, pp. 1–5. DOI: 10.23919/EDHPC59100.2023.10396116.

[59] N. Baldauf, A. Turnwald, T. Lubiniecki, K. Lakatos, and N. Panagiotopoulos, "Learning-based motion control of a rover on unknown ground," *Papers of ESA GNC-ICATT 2023*, 2023. DOI: 10.5270/esa-gnc-icatt-2023-061.

[60] T. Englert, A. Völz, F. Mesmer, S. Rhein, and K. Graichen, *A software framework for embedded nonlinear model predictive control using a gradient-based augmented lagrangian approach (grampc)*, 2018. DOI: 10.48550/arXiv.1805.01633.

[61] B. Käpernick and K. Graichen, "The gradient based nonlinear model predictive control software grampc," in *2014 European Control Conference (ECC)*, 2014, pp. 1170–1175. DOI: 10.1109/ECC.2014.6862353.

[62] Y. Nesterov, *Introductory Lectures on Convex Optimization: A Basic Course*, 1st ed. Springer Publishing Company, Incorporated, 2014. DOI: 10.1007/978-1-4419-8853-9.

[63] M. Diehl, R. Findeisen, F. Allgöwer, H. Bock, and J. Schloder, "Nominal stability of real-time iteration scheme for nonlinear model predictive control," *Control Theory and Applications, IEE Proceedings -*, vol. 152, pp. 296–308, Jun. 2005. DOI: 10.1049/ip-cta:20040008.

[64] K. Graichen and A. Kugi, "Stability and incremental improvement of suboptimal mpc without terminal constraints," *IEEE Transactions on Automatic Control*, vol. 55, no. 11, pp. 2576–2580, 2010. DOI: 10.1109/TAC.2010.2057912.

[65] B. Paden, M. Cáp, S. Z. Yong, D. S. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *CoRR*, vol. abs/1604.07446, 2016. DOI: 10.48550/arXiv.1604.07446.

[66] J. Kabzan, L. Hewing, A. Liniger, and M. N. Zeilinger, "Learning-based model predictive control for autonomous racing," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3363–3370, 2019. DOI: 10.1109/LRA.2019.2926677.

[67] W. Liu, Y. Zhai, G. Chen, and A. Knoll, "Gaussian process based model predictive control for overtaking scenarios at highway curves," in *2022 IEEE Intelligent Vehicles Symposium (IV)*, 2022, pp. 1161–1167. DOI: 10.1109/IV51971.2022.9827233.

[68] J. Kong, M. Pfeiffer, G. Schildbach, and F. Borrelli, "Kinematic and dynamic vehicle models for autonomous driving control design," in *2015 IEEE Intelligent Vehicles Symposium (IV)*, 2015, pp. 1094–1099. DOI: 10.1109/IVS.2015.7225830.

[69] M. Mitschke and H. Wallentowitz, *Dynamik der Kraftfahrzeuge* (VDI-Buch). Springer Fachmedien Wiesbaden, 2014. DOI: 10.1007/978-3-658-05068-9.

[70] B. Heißing, M. Ersoy, and S. Gies, *Fahrwerkhandbuch: Grundlagen, Fahrdynamik, Komponenten, Systeme, Mechatronik, Perspektiven* (ATZ/MTZ-Fachbuch). Vieweg+Teubner Verlag, 2011. DOI: 10.1007/978-3-658-01992-1.

[71] H. B. Pacejka and E. Bakker, "The magic formula tyre model," *VehicleSyst. Dyn.*, vol. 21, no. S1, pp. 1–18, 1992. DOI: 10.1080/00423119208969994.

[72] J. Rawlings, D. Mayne, and M. Diehl, *Model Predictive Control: Theory, Computation, and Design*. Nob Hill Publishing, 2017.

[73] A. Eskandarian, C. Wu, and C. Sun, "Research advances and challenges of autonomous and connected ground vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 2, pp. 683–711, 2021. DOI: 10.1109/TITS.2019.2958352.

[74] J. Guanetti, Y. Kim, and F. Borrelli, "Control of connected and automated vehicles: State of the art and future challenges," *Annual Reviews in Control*, vol. 45, pp. 18–40, 2018. DOI: 10.1016/j.arcontrol.2018.04.011.

[75] P. Pfeffer and M. Harrer, "Fahrdynamische grundlagen der querdynamik," in *Lenkungshandbuch: Lenksysteme, Lenkgefühl, Fahrdynamik von Kraftfahrzeugen*. Wiesbaden: Vieweg+Teubner, 2011. DOI: 10.1007/978-3-8348-8167-0_5.

[76] D. H. Weir and R. J. DiMarco, "Correlation and evaluation of driver/vehicle directional handling data," SAE Technical Paper, Tech. Rep., 1978. DOI: 10.4271/780010.

[77] C. E. Rasmussen, C. K. Williams, *et al.*, *Gaussian processes for machine learning*. Springer, 2006, vol. 1. DOI: 10.1007/978-3-540-28650-9_4.

[78] M. P. Deisenroth, *Efficient reinforcement learning using Gaussian processes*. KIT Scientific Publishing, 2010, vol. 9. DOI: 10.5445/KSP/1000019799.

[79] A. Patrikainen and M. Meila, "Comparing subspace clusterings," *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 7, pp. 902–916, 2006. DOI: 10.1109/TKDE.2006.106.

[80] W. M. Rand, "Objective criteria for the evaluation of clustering methods," *Journal of the American Statistical Association*, vol. 66, no. 336, pp. 846–850, 1971. DOI: 10.1080/01621459.1971.10482356.

[81] C. C. Agarwal and C. Reddy, "Data clustering, algorithms and applications," *Data Mining and Knowledge Discovery Series. Chapman & Hall/CRC Book, CRC Press, Taylor and Francis Group, Boca Raton*, 2014. DOI: 10.1201/9781315373515.

[82] J. Bezdek and N. Pal, "Some new indexes of cluster validity," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 28, no. 3, pp. 301–315, 1998. DOI: 10.1109/3477.678624.

[83] A. Patrikainen and M. Meila, "Comparing subspace clusterings," *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 7, pp. 902–916, 2006. DOI: 10.1109/TKDE.2006.106.

[84] R. Shanmugamani, *Deep Learning for Computer Vision: Expert techniques to train advanced neural networks using TensorFlow and Keras*. Packt Publishing, 2018.

[85] S. Haykin, *Neural networks and learning machines, 3/E*. Pearson Education India, 2009.

[86] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer Science+ Business Media, LLC, 2006.

[87] L. Deng, "The mnist database of handwritten digit images for machine learning research [best of the web]," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012. DOI: 10.1109/MSP.2012.2211477.

[88] L. Bottou, C. Cortes, J. Denker, *et al.*, "Comparison of classifier methods: A case study in handwritten digit recognition," in *Proceedings of the 12th IAPR International Conference on Pattern Recognition, Vol. 3 - Conference C: Signal Processing (Cat. No.94CH3440-5)*, vol. 2, 1994, 77–82 vol.2. DOI: 10.1109/ICPR.1994.576879.

[89] C. C. Aggarwal *et al.*, "Neural networks and deep learning," *Springer*, vol. 10, no. 978, p. 3, 2018. DOI: 10.1007/978-3-319-94463-0.

[90] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain.," *Psychological review*, vol. 65, no. 6, p. 386, 1958. DOI: 10.1037/h0042519.

[91] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, pp. 115–133, 1943. DOI: 10.1007/BF02478259.

[92] I. Goodfellow, Y. Bengio, and A. Courville, *Ian Goodfellow, Yoshua Bengio, and Aaron Courville: Deep learning*. MIT Press, 2016. DOI: 10.1007/s10710-017-9314-z.

[93] X. Li and X. Wu, "Constructing long short-term memory based deep recurrent neural networks for large vocabulary speech recognition," *CoRR*, vol. abs/1410.4281, 2014. DOI: 10.48550/arXiv.1410.4281.

[94] A. Graves, M. Liwicki, S. Fernández, R. Bertolami, H. Bunke, and J. Schmidhuber, "A novel connectionist system for unconstrained handwriting recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 5, pp. 855–868, 2009. DOI: 10.1109/TPAMI.2008.137.

[95] A. Graves, "Long short-term memory," in *Supervised Sequence Labelling with Recurrent Neural Networks*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 37–45. DOI: 10.1007/978-3-642-24797-2_4.

[96] S. Hochreiter, "Untersuchungen zu dynamischen neuronalen netzen," *Diploma, Technische Universität München*, vol. 91, no. 1, p. 31, 1991.

[97] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber, "Gradient flow in recurrent nets: The difficulty of learning long-term dependencies," 2001.

[98] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997. DOI: 10.1162/neco.1997.9.8.1735.

[99] C. Olah, "Understanding LSTM networks," 2015.

[100] K. Cho, B. Van Merriënboer, C. Gulcehre, *et al.*, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," 2014. DOI: 10.48550/arXiv.1406.1078.

[101] R. Dey and F. M. Salem, "Gate-variants of gated recurrent unit (gru) neural networks," in *2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS)*, IEEE, 2017, pp. 1597–1600. DOI: 10.48550/arXiv.1701.05923.

[102] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *CoRR*, 2014. DOI: 10.48550/arXiv.1412.3555.

[103] F. Gers and J. Schmidhuber, "Recurrent nets that time and count," in *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, vol. 3, 2000, 189–194 vol.3. DOI: 10.1109/IJCNN.2000.861302.

[104] K. Yao, T. Cohn, K. Vylomova, K. Duh, and C. Dyer, *Depth-gated LSTM*, 2015. DOI: 10.48550/arXiv.1508.03790.

[105] J. Koutník, K. Greff, F. J. Gomez, and J. Schmidhuber, "A clockwork RNN," *CoRR*, vol. abs/1402.3511, 2014. DOI: 10.48550/arXiv.1402.3511.

[106] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A search space odyssey," *CoRR*, 2015. DOI: 10.48550/arXiv.1503.04069.

[107] R. Jozefowicz, W. Zaremba, and I. Sutskever, "An empirical exploration of recurrent network architectures," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML'15, Lille, France: JMLR.org, 2015, pp. 2342–2350. DOI: 10.5555/3045118.3045367.