Reinforcement Learning of Model Predictive Control Parameters for Autonomous Vehicle Guidance

Thesis · September 2020

DOI: 10.13140/RG.2.2.36051.60960

1 author:

Baha Zarrouki
Technische Universität München
3 PUBLICATIONS **7** CITATIONS

SEE PROFILE

Reinforcement Learning of Model Predictive Control Parameters for Autonomous Vehicle Guidance

Master's Thesis of

M. Baha E. Zarrouki

374291

Software and Embedded Systems Engineering
Department of Electrical Engineering and Computer Science
Technical University of Berlin

Reviewer:	Prof. Dr. Sabine Glesner
Second reviewer:	Prof. Dr. Clemens Gühmann
Advisor:	Dr.-Ing. Verena Klös
	Dr.-Ing. Rick Voßwinkel
	M.Sc. Simon Schwan
	M.Sc. Nikolas Heppner

Cooperation with:
IAV automotive engineering

September – 2020

Anmelde-/Freigabebogen für Diplom-/Studienarbeiten
(Anlage zum Einstellungsbogen)

automotive engineering iav

ANMELDUNG

Titel der Arbeit ~~Lernbasierte Optimierung einer prädiktiven Fahrdynamikregelung~~
Reinforcement Learning of Model Predictive Control Parameters for Autonomous Vehicle Guidance

Kandidat(in)	Baha Zarrouki		
Abteilung	TI-M24		
Betreuer IAV	Nikolas Heppner, Dr. Rick Vosswinkel (TI-F1), Maximilian Gerwien (TI-F1)	Telefon	+491721580840 +49 371 237-35209
Hochschule	Technische Universität Berlin		
Fachbereich	Elektrotechnik		

Professoren	Erstkorrektor: Prof.Dr. Sabine Glesner	Zweitkorrektor:

1. Die Arbeit beinhaltet
- ☐ geheimhaltungsbedürftige Informationen der IAV
- ☐ geheimhaltungsbedürftige Informationen von Kundenseite
- ☒ keine geheimhaltungsbedürftigen Informationen

2. Einverständniserklärung des Kunden
- ☐ liegt bei
- ☒ nicht erforderlich

Der/die Student/in erklärt, dass er/sie zur Diplomarbeit zugelassen ist
- ☒ Der schriftliche Nachweis über die Zulassung bzw. die Anmeldung zur Diplomarbeit (bei ausländischen Studenten) ist beizufügen

13.03.2020 — 17.03.20

1.1.1 Datum, Unterschrift Student(in)/Diplomand(in) — Datum, Unterschrift FL

FREIGABE (erst vor Abgabe der Arbeit ausfüllen)

1.1.2 Sperrvermerk zur Veröffentlichung
- ☒ keine Sperre
- ☐ Sperre bis

Ausfertigung eines Sperrvermerks, soweit erforderlich (siehe IAV-Vorlagen); Einfügen vor der jeweils 1. Seite der Arbeit

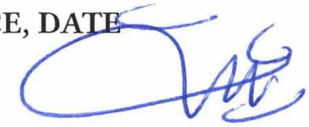
- ☐ Ablage im Wissensspeicher erfolgt

25.09.2020 — Dr. Rick Vosswinkel

1.1.3 Datum, Unterschrift Student(in)/Diplomand(in) — Datum, Unterschrift FL

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

PLACE, DATE



Berlin, 30.09.2020

(M. Baha E. Zarrouki)

Abstract

Within the scope of highly-automated driving, an important part is the automated control of vehicle dynamics for path-following. For that we examine two approaches. The first is Model Predictive Control (MPC). It takes hard constraints into consideration, but remains challenging regarding its parameters. The second is Reinforcement Learning (RL). This approach promises a high performance, lacks however safety-guarantees. To overcome the drawbacks of both approaches, we propose to combine both frameworks to achieve safety and performance at the same time and to save the hand tuning workload. To compensate for changes in the MPC environment, we define a novel Parameter-Varying MPC (PMPC) driven by a deep neural network. The latter is trained with state-of-the-art RL algorithms. Thus, we have a cascaded control system: PMPC determines the vehicle control actions, while RL introduces it with the best cost function parameter set. Apart from the objectives formulated in the MPC cost function, our RL driven PMPC is able to consider additional optimisation objectives formulated in a novel general multi-objective cascaded Gaussian (MOCG) reward function. We formulate optimisation objectives for a tracking- and for a comfort mode. Experimental results demonstrate that our approach trains autonomously and outperforms an MPC tuned by a human expert. The approach we propose is generic and, thus, can also be applied to other control areas, such as nanoelectronic semiconductor production, autonomous robotic medical surgeries or autonomous guidance of spacecrafts.

Zusammenfassung

Im Rahmen des hochautomatisierten Fahrens ist die automatisierte Regelung der Fahrdynamik zur Pfadverfolgung ein wichtiger Bestandteil. Dazu untersuchen wir zwei Ansätze. Der erste ist die modellbasierte prädiktive Regelung (Model Predictive Control, MPC). Sie berücksichtigt harte Systembegrenzungen, bleibt aber hinsichtlich ihrer Parameter anspruchsvoll. Der zweite ist das Bestärkende Lernen (RL). Dieser Ansatz verspricht eine hohe Leistungsfähigkeit, es fehlt ihm jedoch an Sicherheitsgarantien. Um die Nachteile beider Ansätze zu überwinden, schlagen wir vor, beide Ansätze zu kombinieren, um Sicherheit und Leistungsfähigkeit gleichzeitig zu erreichen und den hohen Aufwand beim manuellen Tuning einzusparen. Um Veränderungen in der MPC Umgebung zu kompensieren, definieren wir eine neuartige parameter-variable MPC (PMPC), die von einem mehrschichtigen neuronalen Netzwerk angetrieben wird. Letzteres wird mit modernsten RL-Algorithmen trainiert. Somit haben wir ein kaskadiertes Regelsystem: PMPC bestimmt die Fahrzeugregelaktionen, während RL die kostenoptimalen Funktionsparameter bestimmt. Zusätzlich zu den in der MPC-Kostenfunktion formulierten Zielsetzungen ist unsere RL-betriebene PMPC in der Lage, zusätzliche Optimierungsziele zu berücksichtigen, die in einer neuartigen allgemeinen Mehrziel-Kaskaden-Gaussian-Belohnungsfunktion (MOCG) formuliert sind. Wir formulieren Optimierungsziele für einen Tracking- und für einen Komfortmodus. Versuchsergebnisse zeigen, dass unser Ansatz autonom trainiert und eine von einem Experten abgestimmte MPC übertrifft. Der von uns vorgeschlagene Ansatz ist generisch und kann daher auch auf andere Regelungsbereiche angewendet werden, wie z.B. die nanoelektronische Halbleiterproduktion, autonome medizinische Roboterchirurgie oder die autonome Führung von Raumfahrtschiffen.

Contents

1. Introduction

Freeing up our time on the road for more social interaction, self-development or simply having some rest is one of the biggest drives and motivation for autonomous driving. Moreover, it provides mobility to incapable members of our society, e.g., elderly and people with disabilities. Further, we strive for increasing the road capacity, reducing the number of road fatalities, the energy consumption and the environmental pollution [48].
Towards achieving full autonomy, a lot of work is being put into highly-automated driving systems. Within this scope, the autonomous longitudinal and lateral vehicle guidance is a very challenging control task.

One possible approach to achieve the control goals is the use of Model Predictive Control (MPC), which gained popularity in industrial applications and academic research [28][49]. The reason for such acceptance is that the control objectives and operating constraints are explicitly considered in a single optimisation problem. In addition, it makes future predictions based on a system model for a certain prediction horizon. The optimisation problem is solved at each time-step [10].
However, a big challenge in MPC is determining a suitable parameter set that delivers the desired performance. Mostly, this is done by hand using the trial-and-error heuristic method. The quality of the results depends a lot on the experience and skill of the designer [47][49]. The process takes a lot of time, becomes more complex with an increasing number of parameters and can lead to a sub-optimal control. Thus, an automation of the process is needed.

Another possible approach is Reinforcement Learning (RL), which is gaining attention in solving optimal control problems [35]. RL is represented by an agent interacting with its environment. The agent won't be told how to behave, instead it explores the environment by taking an action that leads to a new state [66]. Then, a feedback is obtained in form of observations from the new state and a reward. If the taken action increases the expected reward, then the tendency to take such an action is strengthened, i.e, reinforced [70]. The main goal of RL is to maximise the long term cumulative reward over a succession of chosen actions and observed states [68][15].
Although RL agents seem to deliver a good performance, standard approaches are considered incompatible for safety-critical systems, e.g., autonomous vehicles [14]. While exploring the environment, the agent is often susceptible to taking actions that lead to unsafe states [49]. In addition, the closed loop behaviour of the resulting system is difficult for formal analysis. Therefore, it is still difficult to guarantee reproducible and thus verifiable behaviour for a system controlled by an RL agent [24].

1.1. Goal and objectives

In this work, our goal is to autonomously control vehicle dynamics to follow a given reference path, i.e., lateral and longitudinal control. The solution must meet the following objectives/criteria:

- Safety: the system constraints must not be violated, e.g., steering wheel angle- or acceleration constraints.
- Performance of the dynamic driving style: the system should deliver the best performance w.r.t. the chosen driving mode. For instance, the lateral- and velocity error should be minimised for a tracking-optimised mode. Additionally, we put more weight on minimising the longitudinal jerk and lateral acceleration for a comfort mode.
- Automation: the parameter should be determined and adapted automatically, i.e., not manually tuned, to achieve the best performance.

Given the above, both MPC and RL approaches do not solve our problem. The MPC fails in performance and automation. RL fails in safety.

1.2. Solution concept

Our intuition is the following: the two frameworks, RL and MPC, should complement each other. Thus, we take advantage of safety (MPC), performance as well as automation (RL). Specifically, we propose the following approach: the MPC computes each time-step the optimal actions for a given prediction horizon, ensures the stability and satisfies system constraints. On the other side, the RL agent focuses on delivering the best parameters for the MPC. By that, we realise a hybrid cascade control. The output of the RL agent (first controller) provides the parameters of the MPC (second controller).
Figure 1.1 depicts our general concept. We expand the MPC-plant closed loop. Based on observed plant- and controller states and collected rewards, the RL agent chooses the right action to execute, i.e., MPC cost function weights. We change the MPC cost function parameters while the system is running. Thus, we introduce in this work a novel Parameter-varying Model Predictive Control (PMPC). The PMPC is basically consists of an MPC with and a parameter scheduler. We extend the optimisation objectives specified in the MPC problem with additional objectives that we specify in a novel multi-objective cascaded Gaussian (MOCG) reward function.

This work is done in cooperation with IAV (Automotive Engineering). They provided simulation framework for a nonlinear MPC for path-following which is in Matlab/Simulink. It is based mainly on a vehicle model for longitudinal and lateral dynamics, a path generation unit and an MPC that controls the vehicle. The existing framework has three main goals: velocity-, path convergence to the given reference and constraint satisfaction [54]. In our application case, the plant is the vehicle model.

Given the above, our goal in this master thesis is to automatically compute suitable MPC parameters with an RL agent for an autonomous vehicle path-following control task

as shown in fig.1.1. Our strive is to ensure safety and to achieve a desired performance, e.g., regarding comfort. The combination of the two frameworks (MPC and RL) promises to improve controlling the system and to overcome the respective mentioned problems. Thus, it could advance automated driving by saving development effort and increasing controller performance.

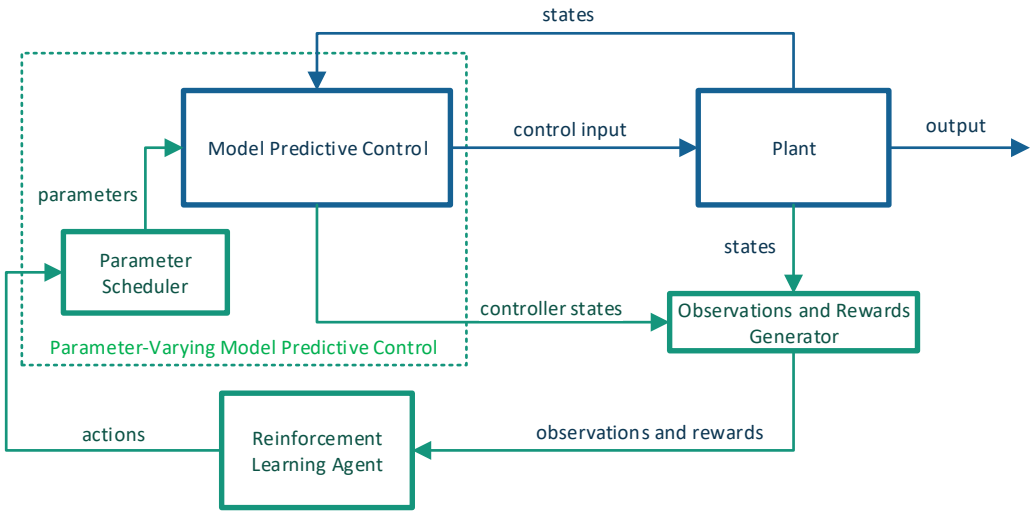


Figure 1.1.: Model of the suggested closed-loop structure. Blue refers to already existing system-blocks (classic MPC-plant closed loop). Green refers to the expansion in this master thesis.

1.3. Related Work

Our work expands the work done in [54], which introduces a nonlinear Model Predictive Path-Following Control (NMPFC) for highly automated driving. The expert determines the MPC parameters manually. The manual tuning process is demanding, time-consuming, requires MPC expert knowledge and may lead to suboptimal control quality. The process needs to be manually repeated once the optimisation goals are changed. In our work, we build on the NMPFC and expand it. Instead of hand tuning, we go towards automation through a learner. In our case the learner is a RL agent. Our concept outperforms the hand-tuned MPC regarding the path-tracking performance as well as regarding comfort. In case of change, our approach provides a new parameter model more easily.
A similar idea to learning the MPC cost function paramters with RL is presented by Mehndiratta [47]. They introduce an automated tuning of the MPC with RL for a simple quadcopter application. The work contains a lot of ambiguities. In fact, neither the action space nor the used observations are defined. The MPC parameter weights should be continuous positive real values. Inferring from their work, they use a discrete action space. Moreover, they claim that the learning process with the CPU takes only 40 minutes to determine 2 sets with 50 training episodes each, i.e., each episode takes only 0.4s. We summarise the differences between the two approaches as following:

- Algorithm: The work in [47] uses an outdated RL algorithm (Q-Learning). The latter is only applicable on a discrete action space. Moreover, its application is not clarified. In our work, we adapt state-of-the-art algorithms suitable for continuous action spaces, e.g., Twin Delayed Deep Deterministic Policy Gradient (TD3).
- Policy: Whereas the policy in [47] is a simple Q-table and its outcome is a single parameter-set, our policy is a deep neural network that adapts in real-time the MPC parameters to occurring changes to improve its performance. We extend the MPC theory with a new Framework: Parameter-Varying Model Predictive Control (PMPC).
- Reward function: Whereas the work in [47] uses a hybrid reward function, we define a continuous rewards attraction region and introduce a multi-objective-cascaded Gaussian.
- Use case: The use-case in [47] is not clear. The application of the concept is on a quadcopter. Having more degrees of freedom and being less affected to safety constraints, the quadcopter is not as challenging as the automated vehicle guidance.
- Learning process: The interaction between the RL agent and the MPC in [47] is not clear. We define a general RL-MPC algorithm, that defines the interaction between the MPC and the RL agent.
- Training conditions: While the work [47] learns parameters for simple quadratic path references, we train our agents on complex realistic urban scenarios that describe a useful use case, i.e., a daily driving scenario.
- Tracking performance: Overall, our concept delivers a better tracking quality, as their error median is approximately 8 cm with a small quadcopter. In contrast, our approach has a deviation median of only 2.7 cm, i.e., 296% better, on a significantly bigger vehicle (Volkswagen Passat Variant) and more challenging path reference.

Hence, the both idea of learning the MPC parameters with RL is related, but the approaches are totally different.
In the following, we discuss works that change the controlling structure online. For instance, the work in [37] defines a multiple MPC (MMPC). The latter is designed to cope with model nonlinearitites and the continuous variation in the operating point. The system operating region is divided into subregions. Each subregion is represented by a linear model. For each model a linear MPC is designed. Then, a model that switches between the different MPCs is designed based on switching criteria such as the change in the operating region. Thus, the main components of an MMPC are a prediction model bank and a model switching criterion [63]. Altogether, the MMPC switches its prediction models from pre-calculated models. A similar concept that handles a similar nonlinearities problem is also called Switching Model Predictive Control [3].
The work of Masár in [44] introduces a gain-scheduled controller. The latter is similar to the MMPC but uses a linear–quadratic regulator (LQR) instead. It copes with model nonlinearities by switching between LQR controllers that are designed for different operating

points. The nonlinear model is linearised at all equilibrium points and for each of them a corresponding LQR controller (gain) is computed. While running, the system controller is switched based on gain-scheduling variables.
The patent work in [31] introduces a gain scheduled MPC. They multiply the optimal control input computed by the MPC with pre-computed gain matrices to obtain a new control input.
The work [52] of Rajan G. Patel and Japan J. Trivedi introduces an adaptive and gain-scheduled MPC. The latter is similar to MMPC. It is used to handle nonliniearities. They switch between a defined set of MPC controllers designed for different prediction models as a single model can not provide the desired accuracy. Gain-Scheduled MPC, decomposes the subcool control problem in a parallel manner and uses a bank of multiple controllers rather than only one controller.
The Work of Falcone in [18] introduces the linear time-varying (LTV) MPC. They design the MPC problem based on successive online linearisation of the nonlinear plant model.

The approaches cited above do not solve our problem of determining the cost function parameters. None of the works cited above handles online adapting the cost function parameters of the MPC. In our work, we do not have to cope with nonlinearities as we are basing our work on a nonlinear MPC, that already considers nonlinear models. However, we do changes to the MPC online as well. In fact, we neither switch between multiple controller models nor between prediction models. We keep the same MPC structure and the same prediction model, but adapt the cost function weights online with a multilayer perceptron (MLP) neural network. We do so to optimise the MPC performance with respect to objectives that we formulated in a novel multi-objective cascaded Gaussian (MOCG) reward function. We train our MLP neural network (policy) with Reinforcement Learning (RL). We call the concept of varying MPC cost function parameters Parameter-Varying MPC (PMPC). Our approach is applicable to linear and to nonlinear system models.

1.4. Structure of the Thesis

This work is divided into four main parts, which together form the solution concept. Chapter 2 presents the theoretical background that the work is based on. Chapter 3 describes how to connect RL to MPC. We introduce the general concept for learning the MPC parameters with RL. We propose a framework called *Parameter-Varying Model Predictive Control* (PMPC), which increases the performance of the system and accelerates the learning process. Then, we apply the theory to our autonomous vehicle guidance use case. In Chapter 4, we define which actions the agent has to learn how to take and which observations the agent needs to take into consideration while making a decision. Additionally, we discuss the importance of scaling the MPC cost function terms. To handle multiple optimisation goals, we introduce a multi-objective cascaded Gaussian reward function (MOCG). We evaluate our approach in Chapter 5. The agents learn autonomously a tracking-optimised- and a comfort mode. Furthermore, we evaluate our approach w.r.t. its capability on handling new driving environments, model mismatch and the use of a different vehicle.

2. Background

In this chapter we introduce the necessary background, which our work is based on. We first introduce the problem scope which is autonomous vehicle guidance in Section 2.1. Then, we introduce Model Predictive Control in Section 2.2, the framework dedicated to solve it. In Section 2.3, we present the general concept of Neural Networks before we present Reinforcement Learning as way to train Neural Networks in Section 2.4.

2.1. Autonomous Vehicle Guidance

An autonomous vehicle has to be able to navigate all alone, i.e., it is capable of operating and reacting to its environment without assistance [72]. The architecture of an automated navigation vehicle system can be delineated in four layers [7]:

- Environment perception and modelling
- Localisation
- Motion planning and decision making
- Actuator level execution of the desired motion

Path following is a fundamental part of the vehicle control module (third layer). The objective of path following is to control the steering and acceleration of the vehicle (fourth layer) in order to minimise the distance between the vehicle (second layer) and path, i.e., guide the vehicle along the precomputed path from the motion planning layer [64]. The path can be computed in various ways, e.g., path planner-generation system [61] or defined using the perception system (first layer).
A lot of control concepts have been used to tackle the path following problem, e.g., classical algorithms [29][22], optimal algorithms[73], robust algorithms [2][41]. Model predictive control (MPC, to be introduced in the next Section 2.2) is gaining popularity in the control of autonomous vehicles, due to its capability of systematically considering system input, state constraints and future predictions in an optimal control problem (OCP). Faulwasser calls MPC for path-following in [19] as model predictive path-following control (MPFC). To deal with the path-following challenges, a model of the vehicle is required. There are mainly 3 different approaches: using only a kinematic model [36], using only a dynamic model [55], or considering both a kinematic and a dynamic problem [34] [54].
The geometric reference path is denoted by $p(\theta) = \begin{bmatrix} x_{ref}(\theta) & y_{ref}(\theta) & \psi_{ref}(\theta) \end{bmatrix}$, where x and y denote the coordinates in the space and ψ the yaw angle. The path-following problem has 3 main requirements [54]:

1. The vehicle position should converge to the reference path.
2. The vehicle velocity should converge to a given reference velocity ϑ_{ref}.
3. The constraints on states (x) and control inputs (u) denoted by the polyhedra $\mathbb{X}$ and $\mathbb{U}$ should always be satisfied.

In this work, a MPC is used as a control framework to solve the path-following problem. Doing so, it meets the given constraints. The general concept of the MPC is described in the next section.

2.2. Model Predictive Control

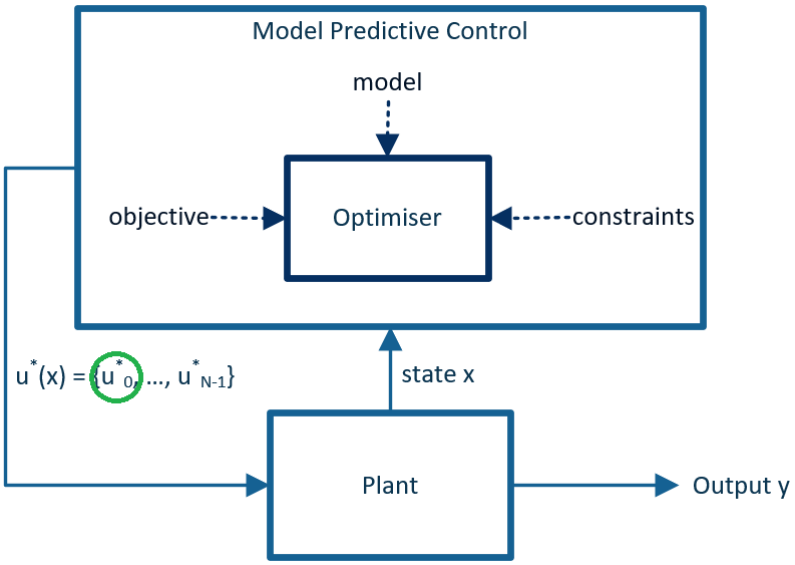


Figure 2.1.: plant controlled with an MPC

Model Predictive Control is a feedback control framework. At each sampling instant, the current control input is calculated by iteratively solving a finite horizon open-loop Optimal Control Problem (OCP). For that, a model of the plant is required to make predictions over the horizon in the future. Also, the current state of the plant is considered as the initial state. Solving the optimisation problem with the optimiser returns a control sequence over the given prediction horizon. Only the first control of the sequence is applied to the plant (fig.2.1). At the next sampling instant, the optimisation problem is solved again and the first control is applied. This is the so called "receding horizon" principle [38]. In fact, the horizon recedes as time proceeds, see fig.2.2. One of the important characteristics of MPC is its ability to consider hard constraints on control variables and states already during the design step [46].

Figure 2.2 represents the discrete time evolution of the predicted outputs y (blue curve) and the control inputs u (red curve) computed by the solver. It demonstrates the sliding of the horizon with time. The white area represents the finite horizon interval. The sliding

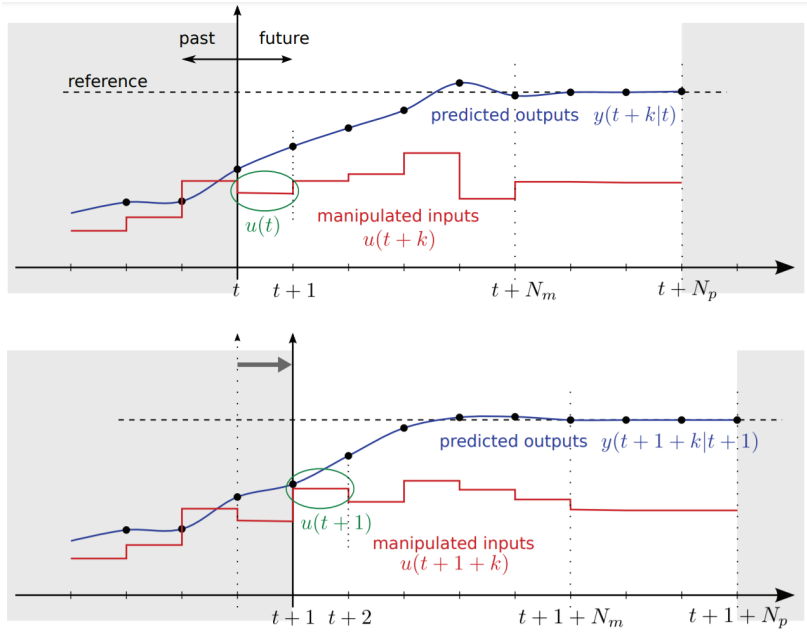


Figure 2.2.: receding horizon principle [17]

vertical axis denotes the current sampling instant. The subplot on top is at time t, whereas the bottom is at the next time step $t+1$. It is clear to see, that the curves of both predicted outputs and manipulated inputs differ from the first time step to the next. This is because the OCP is solved once again at each step, which allows the MPC to update its predictions based on the current plant states. Thus, it allows the MPC to take better actions and anticipate, to optimally meet its objectives. Only the first computed input (green circle) is applied to the system at each time step.

The following is a standard formulation of a linear MPC with quadratic cost/performance measure [16]:

$$
\begin{aligned}
u^*(x) := \arg\min_{u} \quad & x_N^T Q_f x_N + \sum_{i=0}^{N-1} x_i^T Q x_i + u_i^T R u_i \\
\text{subject to} \quad & x_i \in \mathbb{X} \qquad i \in \{1, \dots, N-1\} \\
& u_i \in \mathbb{U} \qquad i \in \{1, \dots, N-1\} \\
& x_N \in \mathcal{X}_f \\
& x_{i+1} = Ax_i + Bu_i
\end{aligned}
\tag{2.1}
$$

Where u^* is the optimal control input, x is the system state vector, u contains the control variables (inputs) and N is the prediction horizon. The linear system dynamics are denoted by $x_{i+1} = Ax_i + Bu_i$.
$\mathbb{X}, \mathcal{X}_f$ and $\mathbb{U}$ are polyhedra, and represent respectively the set of states (state space), the terminal set (terminal constraint) and the set of admissible input values.
The term $x_i^T Q x_i + u_i^T R u_i$ is called **stage cost**. It describes the "cost" of being in state x_i and applying input u_i. The term $x_N^T Q_f x_N$ represents the **terminal cost**, which provides recursive stability [50]. On the other side, the terminal constraint $x_N \in \mathcal{X}_f$ provides recursive feasibility. This holds under the following assumptions: $Q = Q^T \succcurlyeq 0$, $Q_f =$

$Q_f^T \succ 0$ and $R = R^T \succ 0$.
The weighting matrices Q, Q_f and R tell how strong a signal is penalised compared to another one in the cost function. A heavier weight means a bigger penalty. For instance, if we want to save control input efforts, e.g., behave energy efficiently, we specify heavier scalar weights in the matrix R than in Q. Consequently, the cost function is heavily impacted with the control inputs term and the optimiser tries to reduce their value to converge to zero. Similarly, if we want the plant states to converge to a zero steady-state, we define heavier weights in Q than in R. The weighting matrices are the most challenging parameters to tune. Each weight of the cost function terms denotes the significance of minimising it. A big relative weight means a big penalty in the cost function, which means the MPC tries to minimise it more. The task is to find a set of relative weights so that the system achieves a desired behaviour.
If non linear system modelas are used in prediction, the Nonlinear MPC (NMPC) is then used [25], where T_p is the prediction horizon.

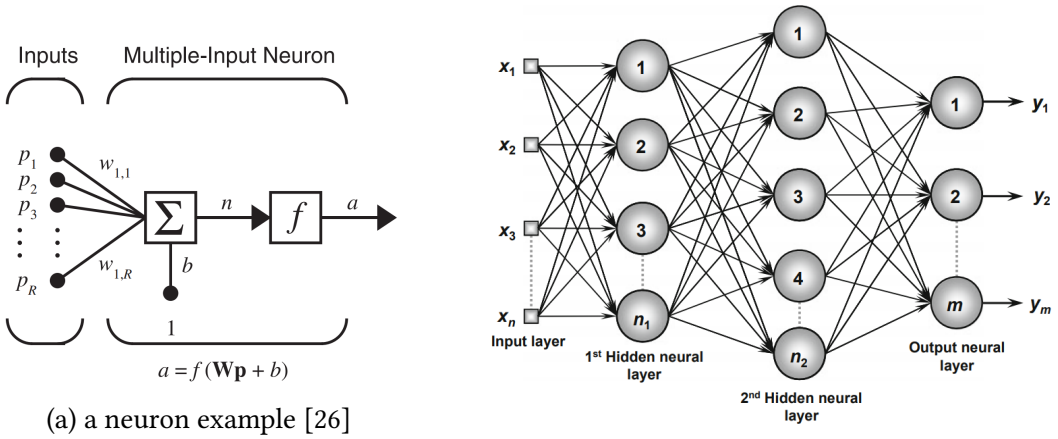
$$\begin{aligned} u^*(x) := \arg\min_u \quad & x(T_p)^T\, Q_f\, x(T_p) + \int_{t=0}^{T_p} x(t)^T\, Q\, x(t) + u(t)^T\, R\, u(t)\; dt \\ \text{subject to} \quad & x(t) \in \mathbb{X} \\ & u(t) \in \mathbb{U} \\ & x(T_p) \in \mathcal{X}_f \\ & \dot{x}(t) = f(x(t), u(t)) \end{aligned} \tag{2.2}$$

The MPC problem can be formulated in many forms. For instance, it can be addressed as a state-tracking problem. In this case, the state vector in the objective function is replaced, e.g., with $(x - x_r)$, where x_r contains the reference states. Thus, the MPC tries to minimise the difference term. Hence, the states converge to their prescribed references.

To determine a suitable parameter-set for the MPC in this work, we make use of Reinforcement Learning with Neural Networks as function approximator. Both are introduced in the following sections.

2.3. Neural Networks

Artificial Neural Networks (ANN) simulate how the human brain analyses and processes information. They involve a set of processing units called nodes (also called neurons or perceptrons) that act like neurons in the human brain. They are interconnected via a set of weights. Signals pass the network in parallel as well as serially. The weighted sum of all incoming signals is computed and compared to a threshold. If the threshold is surpassed, the neuron fires. Otherwise it remains inactive [12]. Figure 2.3a represents a simple neuron with multiple inputs p_i. The output a is the weighted sum of the inputs added to a bias b then passed through an **activation function** f, that represents the firing condition. The most common activation functions are rectified linear unit (ReLu), logistic (sigmoid) and tangent hyperbolic (tanh) and are presented in Fig. 2.4.



(a) a neuron example [26]

(b) MLP architecture [13]

Figure 2.3.: a neuron and a MLP

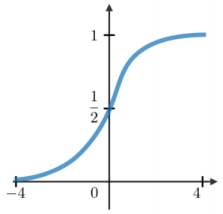
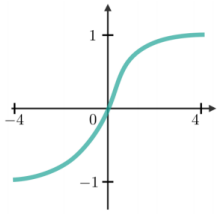
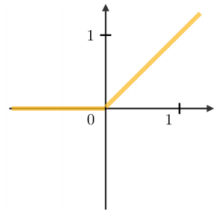
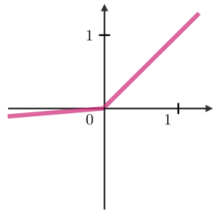
Sigmoid	**Tanh**	**ReLU**	**Leaky ReLU**
$g(z) = \frac{1}{1+e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0,z)$	$g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$
			

Figure 2.4.: activation functions [4]

One of the most frequently used neural network architectures is the **multilayer perceptron (MLP)**. It is a network of neurons aligned in layers. There are mainly 3 types of layers: input layer, hidden layers and output layer. In the input layer, there are no computational operations. The hidden layers consist of one or more layers. The MLP architecture with 2 hidden layers is displayed in Fig. 2.3b. The signals available in the input layer are propagated to the hidden layers and the network neurons perform their calculations layer-by-layer to and including the output layer. The outputs of the MLP are available in the output nodes.
A common technique to train the MLP is back propagation (BP). For further reading on BP, consider the book of Bishop [6] and the paper of Johansson [33].

2.4. Reinforcement Learning

Reinforcement learning is a machine learning category, mainly alongside supervised- and unsupervised learning. It is inspired from psychology [71][67] and neuroscientific animal behaviour [60].

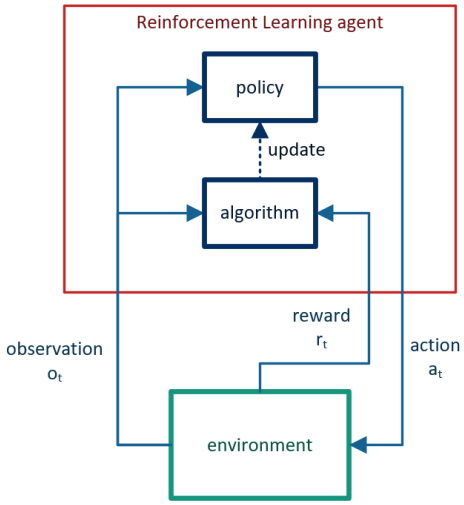


Figure 2.5.: main RL interaction loop

Figure 2.5 shows the main interaction between an agent and its environment. The **agent** is the learner and concretely a piece of software. It can influence its environment by choosing an action [69]. In contrast to most forms of machine learning, the agent is not assisted in his decision making, instead it explores the environment and looks for actions that lead to the highest reward [65]. Also, the agent can sense aspects of its environment in form of observations. Its explicit goal is maximising the cumulative rewards defined in the reward function. Mainly, an agent consists of two parts:

- A **policy** that maps the perceived states of the environment (observations) into actions to be executed in that state of the environment. It can have a simple structure, e.g., a look-up table or a more complex structure, e.g., a neural network [69].
- A **learning algorithm** that continuously updates the policy to find its optimal structure.

The agent can sample arbitrary actions from the environment's action space. The **action space** describes the set of the valid actions that can be taken in an environment. It can be continuous (real-valued vectors) or discrete (finite). The type of the action space have consequences on the choice of the learning algorithm. Not all RL algorithms apply on both action spaces.
A **state** contains all information about the state of the environment, whereas an **observation** is a partial description of the state.
A **reward** is a direct response from the environment on a taken action in form of a numerical value. It tells how good or how bad it is. As the goal is to maximise the long term

rewards, they help the agent learn the best actions to take. For instance, positive rewards emphasise a good action and negative rewards imply a dissatisfaction. Thus, the reward function objectively defines the RL goal and motivates the agent to act corresponding to a desired behaviour.
The cumulative reward over a trajectory is called **return**. A **trajectory** or **episode** is a sequence of states s_{t_i} and actions a_{t_i} in the environment $\tau = (s_0, a_0, s_1, a_1, s_2, ...)$, where t is the time. The return can be formulated in two ways:

- finite-horizon undiscounted return: $R(\tau) = \sum_{t=0}^{T} r_t$.
- infinite-horizon discounted return: $R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t$.
  where γ is called discount factor and has a value between 0 and 1. It tells how important future rewards are to the current state.

Considering a policy with parameters θ, e.g., parameters (weights,biases) of a neural network that represents the policy model, it can be deterministic or stochastic:

- deterministic:
  $$a_t = \mu_\theta(s_t)$$
  is the action at time t.
- stochastic:
  $$a_t = \pi_\theta(s_t)$$
  There is mainly two kinds: categorical policies for discrete action spaces and Gaussian policies for continuous action spaces. The agent uses the stochastic policy to sample actions and to compute log likelihoods of particular actions: $\log \pi_\theta(\mathrm{a}|\mathrm{s})$.

2.4.0.1. Long term reward prediction

The reward is fundamental for the policy's updating process. For instance, if an action leads to a low reward, updating may lower the probability of taking that action in the future. Conversely, the algorithm may higher the probability of taking an action that was followed by a high reward.
While a reward signal designates an immediate feedback regarding the taken action, the **value function** denotes a long term evaluation, i.e., a value of a state specifies the expected sum of rewards that the current policy can get starting from the current state towards the future. Therefore, the value function makes a long term reward prediction. To achieve that, it takes into account the probable following states and the corresponding gained rewards. This is beneficial because some states may produce a low immediate reward but the following states give high rewards. In this case, the value of the state is high, although the immediate reward is not. As the aim is to maximise the long term sum of rewards, it makes more sense to choose actions based on values and not on immediate rewards [69].

2.4.0.2. Value functions

There are two kinds of values when following a policy π:

- The value of a state denoted by the Value Function $V^\pi(s)$. It describes the expected return starting in a state s and sampling actions corresponding to a policy π:

$$V^\pi(s) = \mathop{\mathbb{E}}_{\tau\sim\pi}[R(\tau)|s_0 = s]$$

- The value of a state-action pair denoted by the Action-Value Function $Q^\pi(s,a)$. It describes the expected return starting in a state s, taking a random action a from the action space (not necessarily from the policy) and then sampling actions corresponding to the policy π:

$$Q^\pi(s,a) = \mathop{\mathbb{E}}_{\tau\sim\pi}[R(\tau)|s_0 = s, a_0 = a]$$

The value functions can also be formulated in a **Bellman Equation** form:

$$V^\pi(s) = \mathop{\mathbb{E}}_{a\sim\pi, s'\sim P}[r(s,a) + \gamma V^\pi(s')]$$

$$Q^\pi(s,a) = \mathop{\mathbb{E}}_{s'\sim P}[r(s,a) + \gamma Q^\pi(s',a')]$$

Where s' stands for the next state starting from state s and sampled following the transition rule and a' stands for the next action sampled from the policy π in the state s. Here, the Bellman Equation indicates that the value of the current state is the immediate reward received in that state, plus the value of the next state.

The **Advantage function** $A^\pi(s,a)$ describes the advantage of taking a specific action a according to the policy π over first taking a random action from the action space and then acting according to the policy π. It is then defined as following:

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$$

2.4.0.3. State-transitions

Following the taken action a_t at time t, the environment makes a state transition from s_t to s_{t+1}. The state transitions of the environment can be deterministic ($s_{t+1} = f(s_t, a_t)$) or stochastic ($s_{t+1} \sim P(\cdot|s_t, a_t)$). Considering a stochastic policy and stochastic environment transitions, the probability of a certain trajectory including T steps can be computed as following:

$$P(\tau|\pi) = \rho_0(s_0)\prod_{t=0}^{T-1} P(s_{t+1}|s_t, a_t)\pi(a_t|s_t)$$

where ρ_0 stands for the random sampling of the environment's initial state.
As the main RL goal consists in finding the optimal policy π^* that maximises the expected return, it can be formulated as following:

$$\pi^* = \arg\max_\pi \mathop{\mathbb{E}}_{\tau\sim\pi} = \arg\max_\pi \int_\tau P(\tau|\pi)\Phi_t = \arg\max_\pi J(\pi) \tag{2.3}$$

Where $J(\pi)$ is the expected return over a trajectory and the objective function of the optimisation problem and Φ_t might be one of the following [57]:

- $\Phi_t = R(\tau)$the return over a trajectory
- $\Phi_t = \sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1})$ is called **reward-to-go** and designates the sum of rewards obtained after a point of a trajectory at time t.
- $\Phi_t = \sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1}) - b(s_t)$ this is a baselined version of previous formula.
- $\Phi_t = A^{\pi}(s_t, a_t)$
- $\Phi_t = Q^{\pi}(s_t, a_t)$
- $\Phi_t = r_t + V^{\pi}(s_{t+1}) - V^{\pi}(s_t)$ represents a TD residual.

The choice of Φ_t has an impact on the variance, e.g., taking the advantage function yields to the lowest possible variance.

2.4.0.4. Policy optimisation

Policy Optimisation is an approach to train model-free RL. It addresses the policy $\pi_\theta(a|s)$ and optimises its parameters θ. In Addition, it usually approximates the value function $V^{\pi}(s)$ with an approximator $V_\phi(s)$ which gets trained along with the policy.

The goal is optimising the policy in Equation (2.3) by gradient ascent:

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\pi_\theta)|_{\theta_k}$$

Where $\nabla_\theta J(\pi_\theta)$ is called **policy gradient** and α is the step size. Computing the policy gradient $\nabla_\theta J(\pi_\theta)$ gives the following:

$$\nabla_\theta J(\pi_\theta) = \mathop{\mathbb{E}}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \Phi_t \right]$$

Proof can be founded in the work of Josh Achiam [1].
Collecting a set of trajectories $D = \{\tau_i\}_{i=1,\dots,N}$ by acting with the policy π_θ, allows the estimation of the expectation in the equation above with a sample mean:

$$\hat{g} = \frac{1}{|D|} \sum_{\tau \in D} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \Phi_t$$

$|D|$ stands for the the number of trajectories in D.

2.4.0.5. Vanilla Policy Gradient

The Vanilla Policy Gradient (VPG) is a policy gradient method and uses for Φ_t the Advantage function, i.e., uses the value function as a baseline b. Based on the discussed above, the VPG algorithm can be formulated as in Alg.1.

Algorithm 1 Vanilla Policy Gradient algorithm [9]

initialise policy parameters θ_0 and value function parameters ϕ_0
for k = 0,1,2 ... **do**

1. Collect a set of trajectories $D_k = \{\tau_i\}$ by executing the current policy $\pi_k = \pi(\theta_k)$
2. For each trajectory compute:
   a) The *rewards-to-go* $\hat{R}_t$ for each trajectory
   b) The *advantage estimates* $\hat{A}_t$ based on the current value function V_{ϕ_k}, e.g. using the generalised advantage estimator GAE [56]:

$$\hat{A}_t^{GAE(\gamma,\lambda)} = \sum_{l=0}^{\infty} (\gamma\lambda)^l \, \delta_{t+l}^V$$

   where the TD residual [69] is:

$$\delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t)$$

3. Compute the policy gradient estimate:

$$\hat{g}_k = \frac{1}{|D_k|} \sum_{\tau \in D_k} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)|_{\theta_k} \hat{A}_t$$

4. Update the policy, e.g. using gradient ascent: $\theta_{k+1} = \theta_k + \alpha \, \hat{g}_k$
5. Fit the value function to the rewards-to-go via gradient descent:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|D_k|T} \sum_{\tau \in D_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2$$

end for

2.4.0.6. Proximal Policy Optimisation

Compared to VPG, Proximal Policy Optimisation (PPO) [58] has a different policy update method. Policy gradient methods have generally the following objective function:

$$J(\theta) = \hat{\mathbb{E}}_t \left[\log \pi_\theta(a_t|s_t) \, \hat{\Phi}_t \right]$$

Where Φ_t can be for example the advantage function as in VPG.
PPO is the successor of the Trust Region Policy Optimisation (TRPO) method [59], which tries to make a big improvement to the policy during the update phase, while staying near the old one to avoid performance collapse. This is achieved by considering the ratio of the old policy $\pi_{\theta_{old}}$ to the new policy π_θ. Hence, TRPO tries to maximise a "surrogate objective":

$$J(\theta) = \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \hat{A}_t \right]$$

TRPO still has the inconvenience of large policy updates. PPO solves the problem by penalising the changes that push the ratio away from 1. It introduces a pessimistic bound to the objective function that keeps the ratio inside the interval $[1-\epsilon, 1+\epsilon]$, where ϵ is a hyperparameter that has a small value (usually between 0.1 and 0.3). This is achieved by taking the minimum from the original surrogate objective and by clipping the ratio to the given interval. The final objective looks as following:

$$J(\theta) = \hat{\mathbb{E}}_t \left[\min \left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \hat{A}_t, clip \left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}, 1-\epsilon, 1+\epsilon \right) \hat{A}_t \right) \right]$$

The PPO algorithm (Alg.4) looks the same as the VPG apart from the objective function and can be found in the appendix A.1.1.

2.4.0.7. Deep Deterministic Policy Gradient (DDPG)

Deep Deterministic Policy Gradient [62][42] is an off-policy RL algorithm, i.e., it can learn even from experience collected with an outdated policy. Basically, it learns a Q-function and a policy. The optimal action is computed differently as in Equation (2.3) and it is as following:

$$a^*(s) = \arg\max_a Q^*(s,a) \tag{2.4}$$

DDPG uses mainly two tricks. First, they introduce a replay buffer that stores a set of past experiences. Second, they use target networks. Basically, a target network is an updated scheduled copy of the main network. For more readings, consider the work of Achiam [1].

2.4.0.8. Twin Delayed Deep Deterministic Policy Gradient (TD3)

DDPG has the disadvantage of overestimating Q-values. Twin Delayed DDPG solves this problems with 3 tricks. First, *Twin* stands for double-Q Learning, as the agent learns with 2 Q-functions. Second, *Delayed* stands for updating the policy every two Q-function updates [23]. Third, to reduce the effect of the policy using vulnerabilities of the Q-function, noise is added to the target action. Thanks to the 3 tricks, the performance of DDPG is significantly improved.
The full algorithm of TD3 that we use in this work can be found in the work of Achiam [1].

2.4.1. Tools

In this work, we use the Tensorflow library to learn with Neural Networks. It is an open source software for numerical computation using data flow graphs developed by Google [43]. A multi-dimensional data array is called tensor. Tensors are the graph edges. Graph nodes consist of mathematical operations, called tensorflow operations. Tensors (inputs) flow through the graph. Thus the name: Tensorflow. For further reading, consider the documentation of Tensorflow [43].
The library comes with a visualisation software for inspecting and a better understanding of the Tensorflow runs and graphs. It has 5 types of visualisations: graphs, scalars, histograms,

images and audio. This enables tracking of experiment metrics like loss. The visualisations are not integrated and have to be programmed separately. As a library for RL, we consider the work of Spinning Up in Deep RL [1]. It is an educational resource developed by OpenAI. It contains an implementation of 6 well documented RL algorithms: VPG, TRPO, PPO, Deep Deterministic Policy Gradient (DDPG), Twin Delayed DDPG (TD3) and Soft Actor-Critic (SAC). All algorithms are implemented in both Tensorflow and PyTorch of Facebook.
The code is designed to be simple and highly consistent when the algorithm changes. We chose this framework because its code is well commented, flexible to change and to adapt to our needs. Note that the big part of background on RL introduced here is based on the work of Achiam [1] and other cited authors.

3. Connecting RL and MPC

The MPC optimises only the objectives formulated in its cost function. The more objectives we formulate, the more degrees of freedom we have, i.e., the more weights we have to tune. RL learns based on objectives expressed as physically interpretable rewards and needs less tuning parameters. We propose combining MPC and RL frameworks. On the one hand, we make use of the predictive planning and system constraints consideration capabilities of the MPC. On the other hand, we profit from RL agents to explore and exploit MPC parameter sets and to optimise objectives that are not formulated in the MPC cost function. Thus, we propose a hybrid cascaded control system: the RL agent controls the MPC parameters and the MPC controls the plant. Hence, we aim that this combination pushes the closed loop system to its performance limits.
In this work, we have 2 sets of objectives w.r.t. our closed loop system goals:

- We consider objectives regarding the path-following problem and meeting constraints formulated in the MPC problem.
- We consider objectives regarding optimising the path-tracking and comfort. These objectives are formulated in the reward function.

In this chapter, we derive our solution concept, then we define a general framework that learns MPC parameters with RL. We apply the general concept to our specific highly automated driving use case and introduce the architecture to realise it.

3.1. General concept for PMPC with RL

In this work, we are defining a hybrid MPC-RL multi-objective optimisation problem. Towards optimising the system behaviour w.r.t. our objectives, we expand the MPC theory with the idea of adjusting the MPC cost function parameters during its execution. We introduce a novel **Parameter-Varying Model Predictive Control (PMPC)**. Significantly, we propose altering the PMPC cost function parameters autonomously with a RL agent. Thus, we reduce enormously the developing effort of manually tuning. We aim with the adaptive parameters of the PMPC for an increased performance of the closed loop system w.r.t. changes in its environment. To exemplify, we consider our autonomous vehicle guidance use case. The RL agent adapts the PMPC weights to the road- and weather condition changes, e.g., rain or strong wind. Moreover, as the vehicle- and the prediction model change their behaviour depending on the vehicle speed, the RL policy adapts the PMPC weights to compensate correspondingly, e.g., the RL agent generates different weights for a highway drive than for a parking situation.
The PMPC is an MPC that updates its cost function parameters after a certain constant

switching time period T_{sw}. Figure 3.1 illustrates the schedule of the PMPC. Within a switching time period, the PMPC uses every sample time step T_s the same parameter-set for its computations of the optimal control input to be applied on the plant (PMPC step). A PMPC step denotes solving the MPC problem with the actual cost function weighting matrices and applying the first control input on the plant. Then, new parameters are introduced during the parameter update phase. Concretely, an PMPC is an MPC with a cost function parameter scheduler.

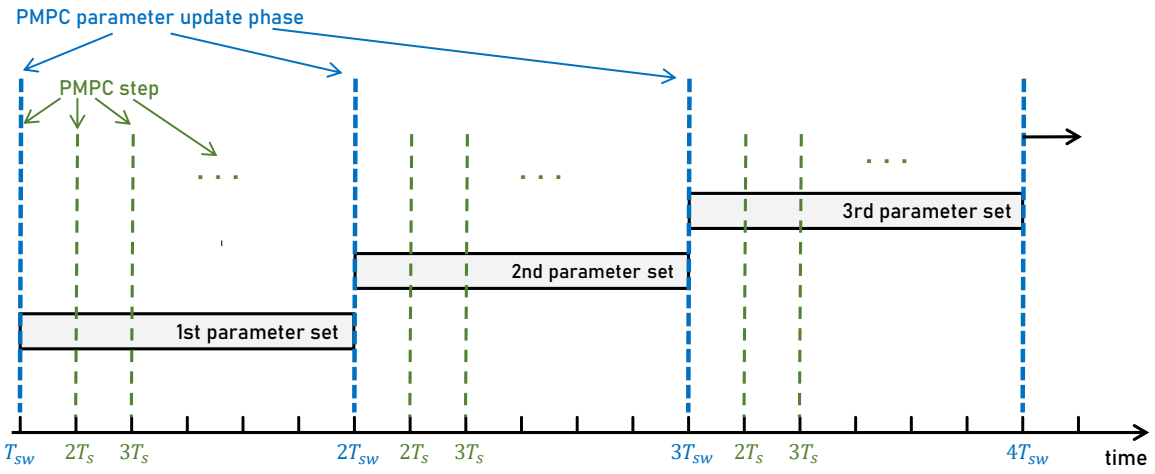
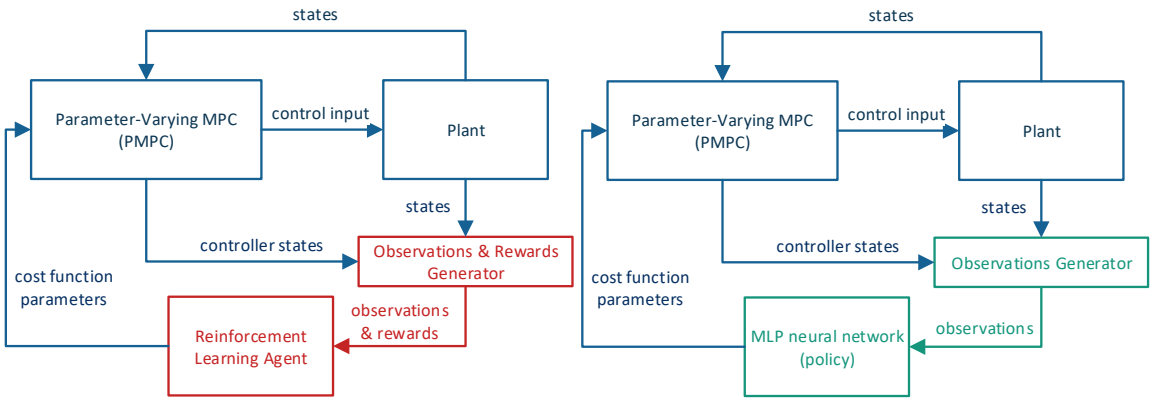


Figure 3.1.: PMPC schedule

We always follow the PMPC schedule. It is valid both during RL training (offline) as well as after training (online). Note that online and offline are defined w.r.t. the PMPC execution. Figure 3.2 depicts the closed loop architecture difference between RL training- and deployment-phase.



(a) offline: training (b) online: after training

Figure 3.2.: deep neural network policy driven PMPC closed loop architecture: during and after training

During training, a RL agent composed of a policy (multilayer perceptron (MLP) neural network) and an algorithm interacts the PMPC-plant closed loop. The agent trains the MLP neural network policy based on the objectives we define in a reward function, the observations it collects and the actions it tries (Fig.3.2a). Once the policy is trained, i.e., no learning is required anymore, the MLP neural network policy generates cost function parameters based on its observations and following the PMPC schedule (Fig.3.2b). Namely, neither a RL algorithm nor rewards are needed anymore.
Note that the PMPC concept is general and can be driven with other concepts than RL in future works.

In the following discussion, we focus on the learning phase and we define a general RL algorithm of PMPC cost function parameters. In order to connect a RL agent to an MPC framework, we analyse first the interaction between an agent and a general environment. We formulate a default interaction loop as shown in Algorithm 2, where the terminal signal d indicates whether s‘ is a terminal state.
Throughout this work, we define the following equivalences:

- RL action $\Longleftrightarrow$ PMPC cost function parameters
- RL observations $\Longleftrightarrow$ chosen states from the MPC framework environment

Algorithm 2 general RL interaction with environment approach

Environment instantiation and initialisation
for $k = 0, 1, 2, \ldots$ **do**

- **for** $i = 1, \ldots$ steps per episode **do**
  1. Select action a (randomly or based on a policy)
  2. Execute action a in the environment
  3. Observe next state **s‘**, terminal signal **d** and get the reward **r**
  4. If d is a terminal state, reset the environment
- **End for**
- Update the parameters of the agent

End for

The agent's action represents the MPC input. As a feedback, the agent receives an observation and a reward signal. The feedback is generated based on MPC- or plant states. Figure 1.1 and Figure 3.2a denote our suggested concept. To learn during an episode, the agent collects knowledge about his environment by executing each time step a new action. Thus, the MPC needs to try different parameter sets. A manual tuning expert would usually try a parameter set, then wait for a simulation to be executed and finally evaluate his choice. In our case, waiting for a full simulation-run to have a single observation after an action is very time-consuming w.r.t. the learning process. With our proposed PMPC concept, the agent only waits until the next switching time T_{sw}.

We determine the new set of weights by the RL agent. Thus, the PMPC parameter update phase is equivalent to a simple step of the RL agent. It consists on executing a new action and getting an observation and a reward. The action - and consequently the new set of weights - is generated by the policy based on the observations collected during a phase $T_{sw} = k\ T_s$, where $k \in \mathbb{N}_1$ is the number of PMPC steps executed before the next parameter update phase. The RL policy itself is updated after a certain amount of time called episode $T_{ep} = l\ T_{sw}$, where l represents the training episode length. Figure 3.3 illustrates the schedule of the PMPC parameter- and the RL policy updates.

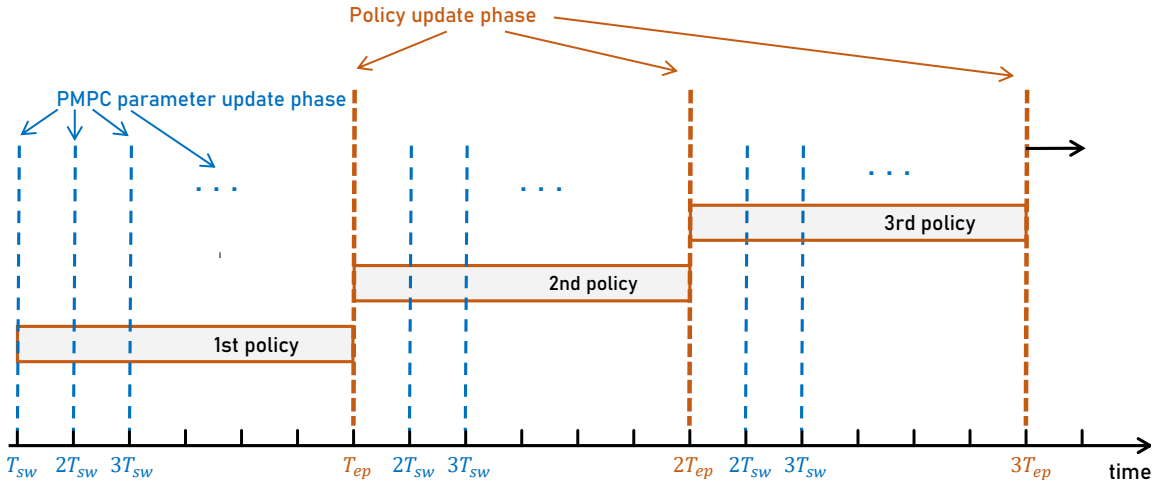


Figure 3.3.: RL-PMPC training schedule

Note that the RL-PMPC schedule is a level above the PMPC schedule and is only applied during training. After training, there is no need for policy updates anymore. However, the PMPC schedule is valid to change the cost function parameters with the trained MLP policy.

Ultimately, we connect RL to PMPC to learn the policy by combining both schedules. Namely, we introduce the PMPC schedule (Fig. 3.1) in the action execution step of Algorithm 2. Based on the discussions above, we define a general algorithm that learns the PMPC parameters with RL (Algorithm 3). Since our cost function weights are real-valued, We choose a RL algorithm suitable for continuous action spaces. Our MLP policy is represented by f. The counter k denotes the training episodes, is a factor of T_{ep} and represents the policy update phase in Fig.3.3. The variable i counts the number of cost function parameter alterations during an episode and denotes the PMPC parameter update phase in Fig.3.3 and Fig.3.1. We scale the signals appearing in the MPC cost function to faster the learning process and reduce the search space of the agent. The variable j counts the number of solving the MPC problem and applying the optimal control input on the plant between two parameter switching periods. It denotes the PMPC step in the PMPC schedule (Fig.3.1). Once the maximum number of PMPC steps $\frac{T_{sw}}{T_s}$ between two switching episodes is reached, the *Observations & Rewards Generator* (3.2a) generates an observation and a reward as a feedback to the last chosen MPC cost function parameter set. Then, the policy computes a new parameter set based on the last observation and the loop continues.

Algorithm 3 general RL-PMPC algorithm

Choose a RL algorithm suitable for continuous action spaces.
Set up the agent's policy f to generate only positive actions.
Initialise the RL agent's parameters.
Initialise the MPC environment and get the initial observation o_0.
For $k = 0, 1, 2, \ldots$ **do**

- **For** $i = 1, \ldots \frac{T_{ep}}{T_{sw}}$ steps per episode **do**
  1. Execute the current policy $a_i = f(o_{i-1})$.
  2. Scale the signals appearing in the MPC cost function.
  3. Introduce a_i to the MPC weighting matrices Q, R and P.
  4. **For** $j = 1, \ldots \frac{T_{sw}}{T_s}$ samples per parameter-set **do**
     a) Solve the MPC problem and get the optimal control input u^*_{ij}.
     b) Apply the optimal control input u^*_{ij} on the plant and store the outputs in a buffer.

     end for
  5. Process the output buffer and generate an observation o_i and a reward r_i.
- Execute needed RL agent updates, e.g. updating the policy network and the value functions.
- **End for**

End for

Note that our concept for the RL-PMPC algorithm is not specific for the autonomous vehicle guidance application nor for a particular RL algorithm, but it is general and applicable to plants controlled with an MPC and a large variety of RL algorithms suitable for continuous action spaces.
Figure 3.4 depicts the transition from the the learning step (offline) to the deployment step (online). First We train the policy with RL to learn PMPC cost function parameters. Then, we apply only the PMPC schedule: we generate a parameter set with the trained policy based on the last- or initial observations we execute PMPC steps with the actual parameter set as long as the PMPC parameter update phase T_{sw} is reached. When the latter holds, we generate an observation and retrieve a new parameter set from the MLP policy. We do not compute rewards anymore.

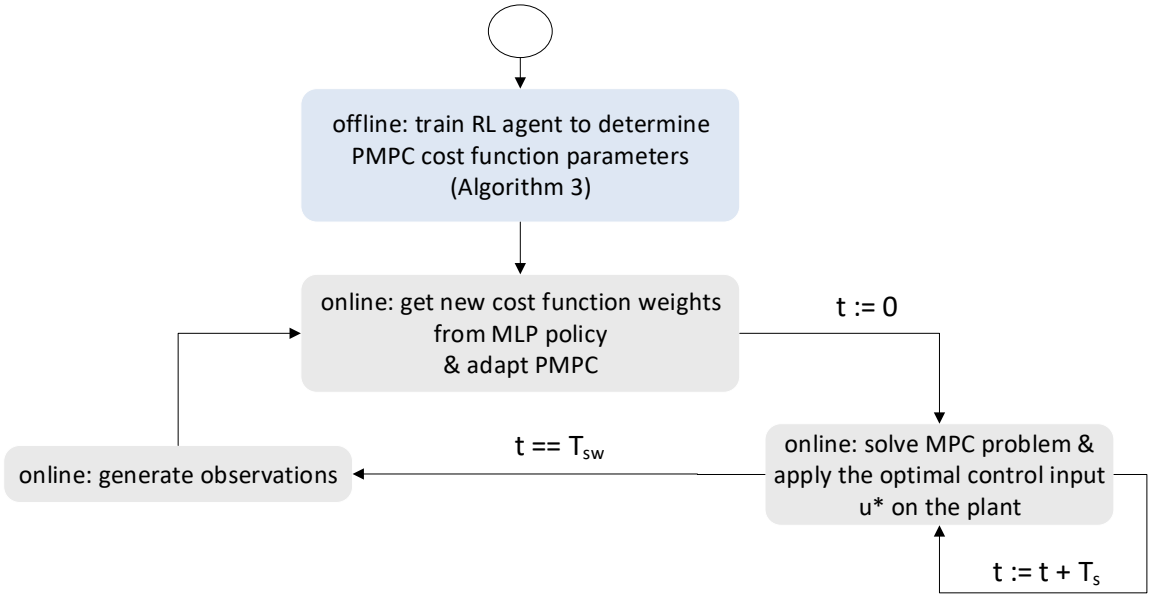


Figure 3.4.: PMPC: online and offline activity

3.2. Concept application on Autonomous Vehicle Guidance

Next, we apply the concept to our use case. The plant is the vehicle model. In IAV, we have a simulation framework in Matlab/Simulink based on the work presented in [54]. In this work, we expand it to achieve the thesis goals. We propose the concept architecture illustrated in Fig. 3.5. To build a PMPC, the standard used MPC is introduced every T_{sw} by the RL agent with new weighting matrices through the *weights introduction interface*. The former acquires its reference from the reference path generator. The MPC computes the optimal control input every T_s and acts on the vehicle model, which is simulated on the road. The observations are collected from the simulation environment by the *observations generator unit*.
Since most of RL libraries are built in Python and have more training time efficiency, we choose to write the RL part in Python. Accordingly, we create a RL environment for the PMPC in Python. The latter is suitable for interaction with common RL libraries. Also, we face the challenge of connecting the simulation framework (Simulink) with the RL agent (Python), as they are built with different software frameworks. We come with the idea of establishing a communication interface on both sides to exchange data via a TCP/IP Server/Client communication. We implement the Simulink-TCP client in C++ so it is compatible with code generation.
We present the architecture blocks in the following sections. The grey blocks of the Simulink framework are presented in Section 3.3. Then, we introduce our expansion of the Simulink simulation framework in Section 3.4. Finally, we present the advantages of our architecture w.r.t. accelerating the run-time of the training process in Section 3.5.

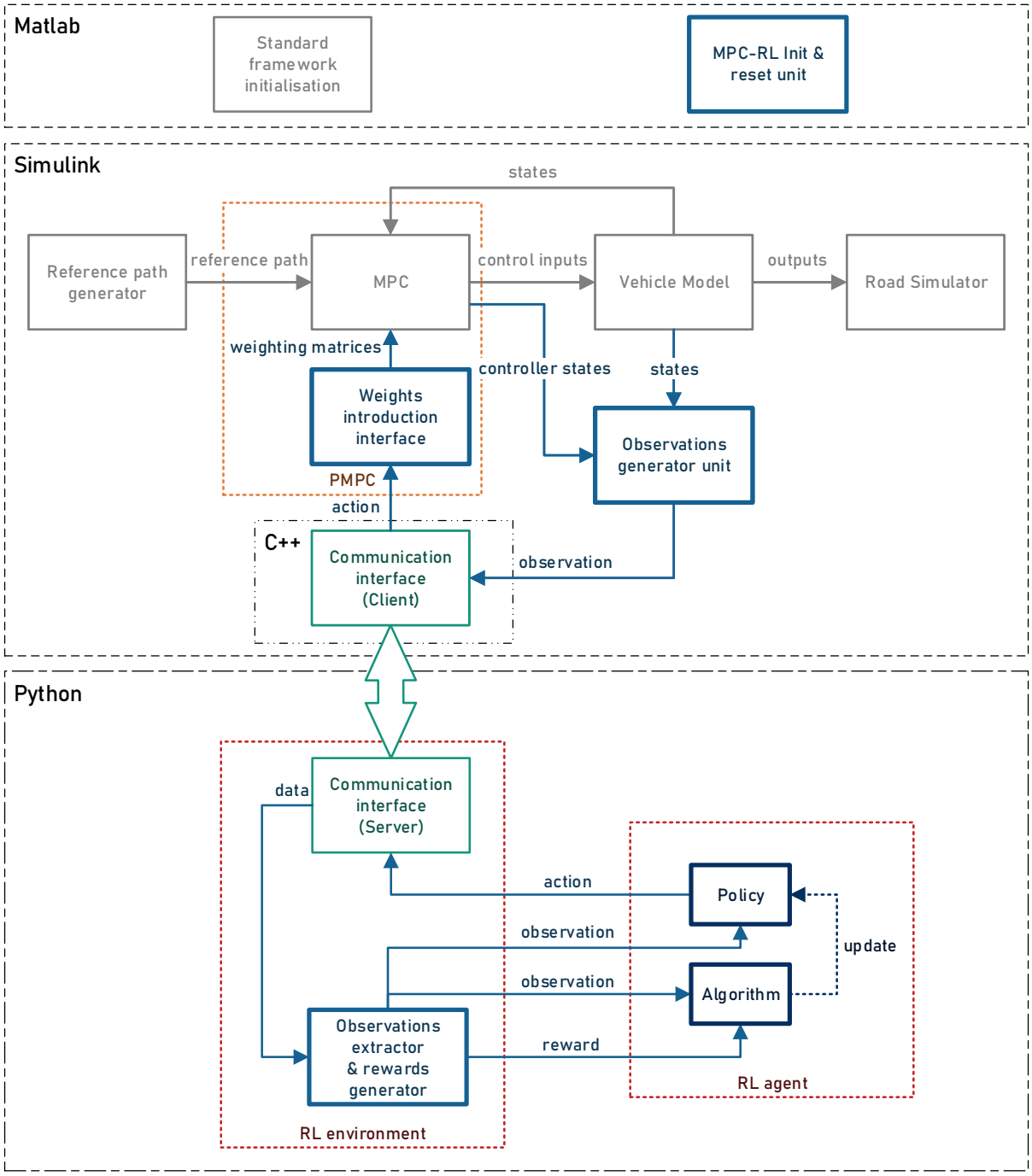


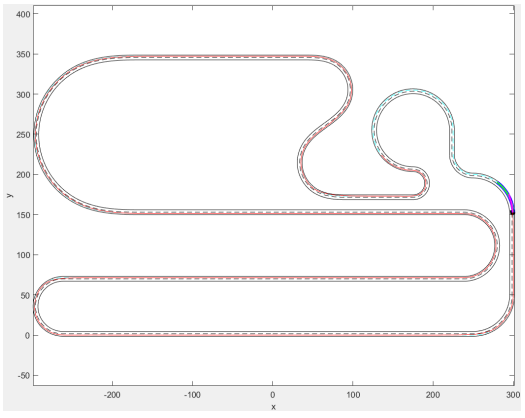
Figure 3.5.: Concept architecture: grey blocks existed before and the rest represents our expansion in this work. Grey dashed lines denote the software framework.

3.3. Simulation framework for highly automated driving

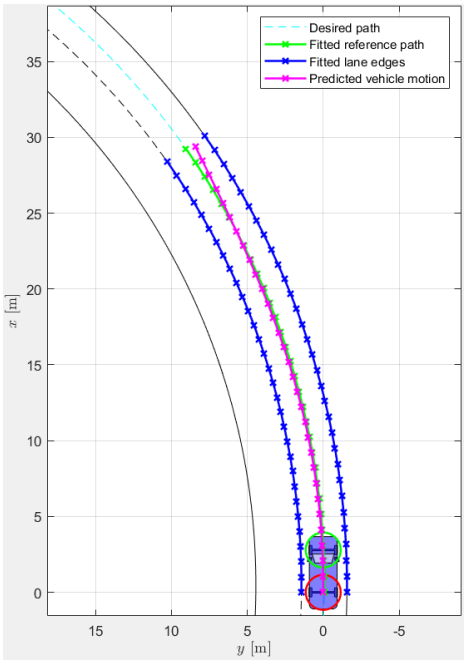
The simulation framework in IAV contains a vehicle model, an MPC controller, a path processing unit and other logging features. The MPC and the environment are initialised in Matlab. The simulation provides reality resembling valid data. The Matlab/Simulink framework facilitates its deployment on prototype vehicles and thus real-world testing. In this section, we present the blocks contained in the simulation framework, i.e., the blocks in the Simulation framework.

3.3.1. Reference path generator and Road Simulator

In real case, the reference path is generated online through the environment perception layer. In this simulation framework, the reference path is pre-computed. It represents a sequence of x- and y- coordinates of the middle of the lane. Each (x_{ref},y_{ref}) pair is joined with a velocity reference information v_{ref}.
The road simulator is a road like simulation track. There are mainly 2 scenarios available: an urban and a highway scenario. An urban simulation track is depicted in Fig. 3.6a and a bird's eye view on the vehicle in Fig. 3.6b. The track has one lane per direction. The reference is in the middle of the lane. The predicted vehicle motion calculated by the MPC is displayed over the prediction horizon and updated every MPC step. The urban scenario simulates challenging curvatures that resemble to reality. The reference velocity ranges in $[11\,\mathrm{m\,s^{-1}}, 20\,\mathrm{m\,s^{-1}}]$.



(a) default urban simulation track



(b) Bird's eye view of the simulated vehicle in a curve

Figure 3.6.: simulation track

3.3.2. Vehicle Model

The vehicle model represents our plant. It is a switchable single track model (for more background on single track models, see the book *Vehicle dynamics and control* of Rajamani [53]). A continuous-time nonlinear system is used for the longitudinal and lateral vehicle dynamics:

$$\dot{x}(t) = f(x(t), u(t)), \qquad x(t_o) = x_0$$

The control input is $u = \begin{bmatrix} a_{ref} & \omega_{s,ref} \end{bmatrix}$, where a_{ref} is the reference longitudinal acceleration and $\omega_{s,ref}$ is the steering wheel target angular velocity. The plant state $\mathbf{x}$ is

$$\mathbf{x} = \begin{bmatrix} x, & y, & \psi, & \dot{\psi}_{dyn}, & \beta_{dyn}, & v, & v_{ref}, & \delta_s, & \delta_{s,ref} \end{bmatrix}$$

where x and y denote the vehicle position, ψ the yaw angle, $\dot{\psi}_{dyn}$ the yaw rate, β_{dyn} the side slip angle, v, the vehicle velocity, δ_s the steering wheel angle and the subscript ref the reference. The full vehicle model and its numerical parameters can be found in the work of Ritschel [54]. The nonlinear function f that represents the dynamics can be derived from the latter as well. The vehicle model is appropriate for both high and low speeds. The vehicle dynamics are identified based on real Volkswagen vehicles. Model static parameters, e.g., vehicle mass, are calculated based on the vehicle's specific geometry and mass.
Through this work, we use the Volkswagen Passat Variant dynamics as depicted in Fig. 3.7.

Figure 3.7.: IAV Volkswagen Passat Variant test vehicle

3.3.3. MPC

In the following, the model predictive path-following control (MPFC) for highly automated driving implemented in the simulation framework of IAV is considered [54]. As nonlinear dynamics are used, a nonlinear MPFC was formulated (NMPFC). For readability reasons, we represent the NMPFC problem through our work as in Equation (3.1).

$$
\begin{aligned}
\arg\min_{u,\vartheta} \quad & \int_{\tau=t_k}^{t_k+T_p} \begin{pmatrix} x - x_{ref} \\ y - y_{ref} \\ \psi - \psi_{ref} \\ a_{lat} \end{pmatrix}^T Q \begin{pmatrix} x - x_{ref} \\ y - y_{ref} \\ \psi - \psi_{ref} \\ a_{lat} \end{pmatrix} + \begin{pmatrix} a_{ref} \\ \omega_{s,ref} \\ \vartheta - \vartheta ref \end{pmatrix}^T R \begin{pmatrix} a_{ref} \\ \omega_{s,ref} \\ \vartheta - \vartheta ref \end{pmatrix} d\tau \\
& + \begin{pmatrix} x(t_k+T_p) - x_{ref}(t_k+T_p) \\ y(t_k+T_p) - y_{ref}(t_k+T_p) \\ \psi(t_k+T_p) - \psi_{ref}(t_k+T_p) \\ a_{lat}(t_k+T_p) \end{pmatrix}^T P \begin{pmatrix} x(t_k+T_p) - x_{ref}(t_k+T_p) \\ y(t_k+T_p) - y_{ref}(t_k+T_p) \\ \psi(t_k+T_p) - \psi_{ref}(t_k+T_p) \\ a_{lat}(t_k+T_p) \end{pmatrix} \qquad (3.1) \\
\text{subject to} \quad & x(\tau) \in \mathbb{X} \\
& u(\tau) \in \mathbb{U} \\
& \vartheta(\tau) \in \mathcal{V} \\
& l(\omega_{s,ref}(\tau), \delta_{s,ref}(\tau), \vartheta(\tau)) \leq 0 \\
& \dot{x}(\tau) = f(x(\tau), u(\tau))
\end{aligned}
$$

Where $\mathcal{V}$ represents the velocity constraints, $\mathbb{X}$ the set of states, $\mathbb{U}$ the set of admissible input values, a_{lat} the lateral acceleration and $l(\omega_{s,ref}(\tau), \delta_{s,ref}(\tau), \vartheta(\tau))$ considers predefined limits of the steering actuator depending on the velocity [54]. Looking at the optimisation problem, it is straightforward to infer that it tries to minimise, considering the constraints, the error between (x, y) coordinates, yaw angle ψ, velocity ϑ and their respective references. Also, it tries to reduce the lateral acceleration a_{lat}, the reference longitudinal acceleration a_{ref} and the steering wheel target angular velocity $\omega_{s,ref}$. The weighting matrices are Q, R and P, where the latter represents the weight of the terminal cost in this formulation. Our idea in this work is to learn the weighting matrices with RL.
The MPC is implemented using the ACADO framework for automatic control and dynamic optimisation[30]. The optimal control problem in Equation (3.1) is solved by sequential quadratic programming namely qpOASES [20].
The MPFC has a prediction horizon $T_p = 3\,\text{s}$ and a sampling period $T_s = 0.04\,\text{s}$. The concrete optimisation constraints are as following:

- $\mathbb{X} = \{x \in \mathbb{R}^9 \mid 0 \leq v_{ref} \leq 60\,\text{m}\,\text{s}^{-1}, -460^\circ \leq \delta_{s,ref} \leq 460^\circ\}$
- $\mathbb{U} = [-3\,\text{m}\,\text{s}^{-2}, 2\,\text{m}\,\text{s}^{-2}] \times [-250^\circ\,\text{s}^{-1}, 250^\circ\,\text{s}^{-1}]$
- $\mathcal{V} = [0, 60\,\text{m}\,\text{s}^{-1}]$

3.4. Expanding the simulation framework for learning

3.4.1. Generating observations from the environment/Simulation

The observations generator unit is a fundamental part in realising the PMPC-RL interaction. We extract states from the simulation environment and observe them.
The observations are not necessarily transmitted at every MPC time step T_s to the RL agent, but at PMPC parameter switching time steps T_{sw}. Thus, we need to generate a single observation that summarises the significant information about the development of the environment state during the non-RL-interaction period.
Accordingly, we collect the observed states between two parameter switching time instants T_{sw} and every MPC step time T_s in a buffer. At each PMPC parameter update step T_{sw}, we generate useful observations for the RL agent. Generally we're interested in the following values:

- The average of a certain signal over the switching period T_{sw}:

$$\mu(x) = \frac{1}{\frac{T_{sw}}{T_s}} \sum_{k=1}^{\frac{T_{sw}}{T_s}} x(k\,T_s) \tag{3.2}$$

- The extreme absolute value of a signal in buffer:

$$\alpha = \max \left\{ |x(k\,T_s)| : k = 1..\frac{T_{sw}}{T_s} \right\} \tag{3.3}$$

3.4.2. Introducing MPC with weights

The weights introduction interface is the main module behind transforming an MPC into a PMPC. It is responsible for 4 tasks:

1. It scales the different signals appearing in the MPC cost function, so that all terms have the same order of magnitude.

2. It initialises the MPC, i.e., it delivers the first parameter set used at the start of a simulation or after a reset of the environment. We choose for the initial weights a default setup, i.e., all weights are equal to 1. In addition, initially, we scale the cost function terms. Thus, we have equally weighted terms with an equivalent order of magnitude.

3. It associates the policy action vector elements with the weighting matrices Q, R and P elements.

4. It schedules the parameters of the MPC by introducing a new set of parameters Q, R and P every switching time T_{sw}.

3.4.3. Communicating between RL and MPC frameworks

Our idea to exchange data in real time between the RL framework (Python) and the MPC framework (Matlab/Simulink) is to create a TCP Server/Client communication. Our main communication goal is to exchange data and simulation controlling signals, e.g., order from Python to stop the simulation in Simulink. There is mainly two communication directions:

- Python Server to C++ Client in Simulink: The message transmitted in this direction is usually the agent's action and the reset signal.
  When the agent resets the simulation, it sends a stop simulation signal. The communication interface in Simulink recognises the signal and internally stops itself. Then, it restarts again.
  The C++/Simulink Client is able to block the simulation framework in order to wait for the RL agent to complete processing the last observation and generating a new action. As soon as the client receives the new action, the simulation is resumed.

- Simulink/C++ Client to Python Server: The message transmitted in this direction is usually the observation and the terminal state signal.

3.4.4. RL agent

We make use of the provided RL algorithms by Spinning Up [1] (introduced in 2.4.1). To adapt the algorithms to our PMPC requirements and to optimise the running time, we have rewritten the algorithms that we use in our work: Vanilla Policy Gradient (VPG), Proximal Policy Optimisation (PPO) and TD3.

3.4.5. RL environment

The RL environment represents the interaction interface with the RL agent in Python. In our case, the RL environment summarises the highly automated driving simulation framework and provides the RL agent with all needed information about the environment (observations, rewards, terminal states). Moreover, we define in the RL environment our action- and the observation spaces. Also, our RL environment controls the Simulink simulation framework. It transfers actions and resets the simulation. We achieve that through TCP communication.
Our idea is to transform our custom use case environment to a general RL environment. The latter is suitable with the majority of the state-of-the-art available RL libraries, e.g., Tensorforce, OpenAI Baselines, Stable Baselines, TF Agents or Open AI Spinning Up. Thus, it is applicable to a large range of RL algorithms, e.g., Twin Delayed Deep Deterministic Policy Gradient (TD3).
We create an OpenAI Gym [9] based environment. It has an episodic setting of RL. The agent learns through a series of episodic interactions with its environment. At the start of every episode, the initial state of the agent is sampled randomly from a distribution. The agent continues interacting, until a terminal state of the environment is reached or the episode ends.

- **Initial state**: Our initial state is the following
  - Vehicle position: After each reset, we start the vehicle on a random position on the track. Thus, we increase randomness and avoid overfitting.
  - Vehicle speed: We choose to start the vehicle with a high velocity, usually the reference velocity. The reason behind starting with a high velocity, is to faster the learning process. Doing so, the agent already experiences the feeling of driving and not standing still. Thus, the agent experiences a high reward. This encourages the agent in the further leaning process to emulate the reference velocity by accelerating.
- **Terminal state**: In our Simulink simulation framework, the vehicle can usually drive on the track for an infinite period of time. Concretely, we define a terminal state, when we consider that the simulation has to come to an end. For instance, if the vehicle stands still or drives very slowly, i.e., below a certain velocity threshold, for a certain period of time, e.g., 2 s. Another case is when the vehicle leaves the lane and the lateral deviation is too large, e.g., 10 m. We can associate terminal states in the rewards generator unit with penalties. In our use case, experiments have shown that using terminal states slows down the learning process and leads to a degraded performance. We presume that the agent jams in repeating bad actions leading to the terminal state without exploring new options. The agent thinks it is already collecting the maximum rewards. However, when the vehicle leaves the lane without terminal states, the agent tries actions to go back on the track. Thus, the agent experiences the good actions that lead to bigger rewards and tries to reinforce them. Hence, we proceed without terminal states, i.e., the simulation ends only if the episode comes to an end.

3.5. Challenge: accelerating the experiments run time

The biggest challenge that we faced while creating the environment, is the time constraint. In our case, the training process is in range of hundreds of thousands to millions of interactions. Hence, we need to optimise both the MPC simulation framework run-time as well as the interaction with the RL agent. Notably, our MPC environment is very slow compared to standard RL environments, e.g., OpenAI Gym environments.
The advantage of writing the communication Client in C++, is that we can generate code of the simulation framework in Simulink including the communication interface. Thus, we build the code as a standalone application. Results in table 3.1 have shown massive improvement of the experiments run-time. In fact, we score approximately 240% run-time improvement. Moreover, our architecture concept supports parallelisation. In fact, we can create many instances of the simulation framework. Therefore, we can run parallel experiments either on the same computer or on multiple computers, i.e., run several experiments at the same time.
Thanks to the code generation, we need neither to install Matlab/Simulink nor the simulation framework when training on parallel computers, since both are replaced by a generated executable application. Altogether, taking advantage of the parallisation- and

Table 3.1.: Comparison: VPG agent experiment with 2 million interactions and a switching time $T_{sw} = 0.1$ s

	Matlab/Simulink online	standalone application
run-time	8h 35min	3h 35min

code-generation capabilities of our architecture presented in 3.5, we saved several weeks of additional training time by running parallel experiments on 4 computers possessing Intel Core CPUs: i7-10th-, 2 computers with i7-6th- and i5-4th generation.

In this chapter, we presented our solution concept to learn the MPC parameter weights with RL. We extended MPC to support online parameter change (PMPC). Then, we presented a general RL-PMPC framework. The concept discussed is also a general concept applicable to multiple MPC applications. In our case, we applied the concept on the vehicle guidance use case. We designed an architecture applicable to standard RL libraries and to a wide range of RL algorithms.
In the next chapter, we present design challenges and how we solved them.

4. Design Challenges & Solutions

In this chapter, we present the design challenges that we faced following our approach. First, we discuss the impact of the switching time on the MPC performance in 4.1. Second, we design the objective of the agent by formulating reward functions for two driving modes in 4.2. Third, we consider requirements for the RL agent's actions and define them in 4.3. Moreover, we scale the MPC cost function terms to make the different physical quantities have an equivalent order of magnitude in 4.4. Lastly, we discuss concepts for the choice of the observations needed to feed the RL agent.

4.1. Switching time

In this section, we choose a suitable PMPC cost function parameter switching time T_{sw}. We base our choice on 2 objectives: first, the PMPC feasibility and second, the performance of the learning process, i.e., the cumulative rewards collected.
First of all, we try small values: $T_{sw} = 0.1\,\text{s}$. The latter yields to infeasible solutions of the MPC problem. Then, we try larger values: 1 s, 2 s, 3 s, 5 s, 10 s and 20 s. All of the last values meet the feasibility criteria. Thus, we choose the switching time that yields the highest cumulative rewards during training: $T_{sw} = 10\,\text{s}$. We discuss the choice of T_{sw} with more details in the following argumentation.
Undesired switching of MPC parameter degrades the performance of the control system. Taking a small switching time, e.g., 0.1 s, leads to unfeasible problems. Figure 4.1 illustrates that the solver frequently reaches the maximum allowed number of Quadratic Programming (QP) iterations in attempt to solve the MPC problem. This yields in the most cases to infeasibility.

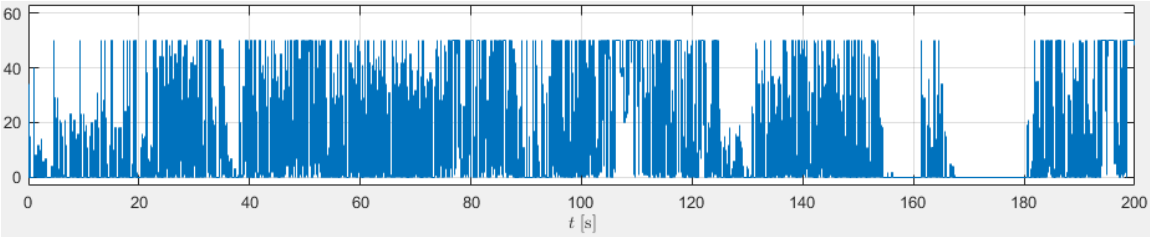


Figure 4.1.: Quadratic Programming (QP) iterations needed with a switching time $T_{sw} = 0.1\,\text{s}$. Horizontal axis stands for simulation time [s], vertical axis for number of QP iterations. 50 is the maximum number allowed.

We can analyse this as following: $T_{sw} = 0.1\,\text{s} = 2.5\,T_s$ is a short time period for the controller and the closed loop system to compensate for the weight changes, where T_s is

the MPC sampling time, i.e., calculating new control inputs.
To avoid performance degradation of the PMPC, we choose a large enough switching time, that does not lead to infeasible problems. We try the following values: 1 s, 2 s, 3 s, 5 s, 10 s and 20 s. We notice that all the mentioned values lead always to feasible MPC problems. Figure 4.2 shows the effect of taking a larger switching time. The QP iterations reach a maximum of 28 iterations per MPC sample time T_s. The number of QP iterations needed in the usual case is under 10 and never reaches the limit. Thus, the problem is always feasible. A large T_{sw} led to remarkable performance improvement of the MPC. With $T_{sw} = 10\,\text{s} = 250\,T_s$, the closed loop system have enough reaction time.

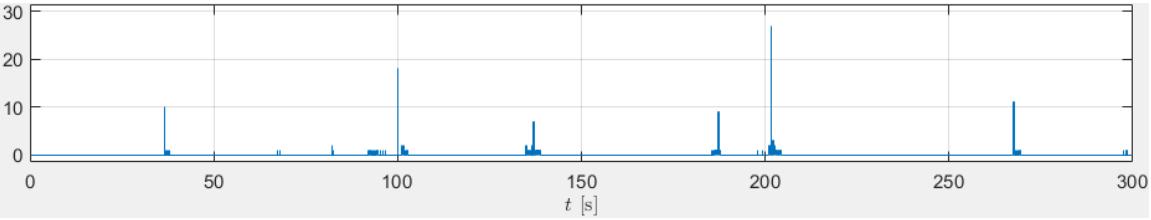


Figure 4.2.: QP iterations needed with a switching time $T_{sw} = 10\,\text{s}$. Horizontal axis stands for simulation time [s], vertical axis for number of QP iterations.

To choose between the above mentioned values, that we tried, we consider the learning performance of the agent. With $T_{sw} = 10\,\text{s}$, the agent reaches the highest cumulative rewards. Thus, we choose the latter. An extensive study of the stability and the feasibility of the PMPC is an idea for a future work.

4.2. Designing a reward function

Designing the reward function is the most challenging task in our approach. It should be well formulated to express objectively our desired goals, in a way that the agent can learn based on that. In this section, we present our optimisation goals, discuss the challenges we faced while designing a reward function and then formulate a function that overcomes them.

4.2.1. Optimisation objectives

In this work, we want to train agents that optimise two driving modes:

- **Tracking-optimised mode** is a mode where we concentrate only on minimising the deviation to the reference path. Concretely, we seek for a compromise between minimising the lateral error e_{lat} and the velocity error e_{vel} with more weight on the lateral behaviour, as it is more safety relevant.
- **Comfort mode** is a mode where we want to minimise the longitudinal jerk j_{long} and the lateral acceleration a_{lat} as much as possible. The jerk is the rate of change of acceleration with time:

$$j_{long} = \frac{da_{long}(t)}{dt} \tag{4.1}$$

where a_{long} is the longitudinal acceleration. The approximation of the lateral acceleration is:

$$a_{lat}(t) = v(t)\dot{\psi}(t) \tag{4.2}$$

where v is the velocity and $\dot{\psi}$ is the yaw rate. We set less weight on optimising the lateral deviation and the velocity error. Still, the vehicle should follow the path reference, not exceed the velocity reference and stay in the lane. Hence, the lateral- and velocity error are still an optimisation objective but errors are more tolerable than in tracking-optimised mode. Thus, we have a 4 objective optimisation objective: j_{long}, a_{lat}, e_{lat} and e_{vel}.

Throughout our work, we consider the optimised path-tracking mode in a first place and then extend to the comfort mode.

4.2.2. Challenges: the Cobra Effect

We designed different kinds of reward functions: discrete, hybrid and continuous (see Appendix A.5). We did not get good results with those formulations.
We experienced with RL that the agent exploits vulnerabilities of our formulation, this is the so called "Cobra Effect" [5]. This problem is also well known in RL as reward hacking [32]. The agent tricks the system and shows an unintended behaviour that is still rewarded by the reward function although it may make the situation even worse. To illustrate, when learning for a comfort mode, we observed that the agent does not accelerate at all and stands still. In fact, stand still is jerk and lateral acceleration optimal. Therefore, the agent collects the maximum rewards acting so.
When using a certain formulation of the reward function, the tracking mode agent tends to drive extremely slow or to stand still in order to keep the lateral error marginal. With another configuration, the vehicle collects the highest rewards when drifting and driving in circles with high velocities.
In the following sections, we demonstrate how we have overcome these difficulties. As we have a continuous action- and observation space, we choose to define a continuous reward function as well. Unlike discrete rewards, continuous rewards provide faster convergence while training and need simpler network architectures [45].

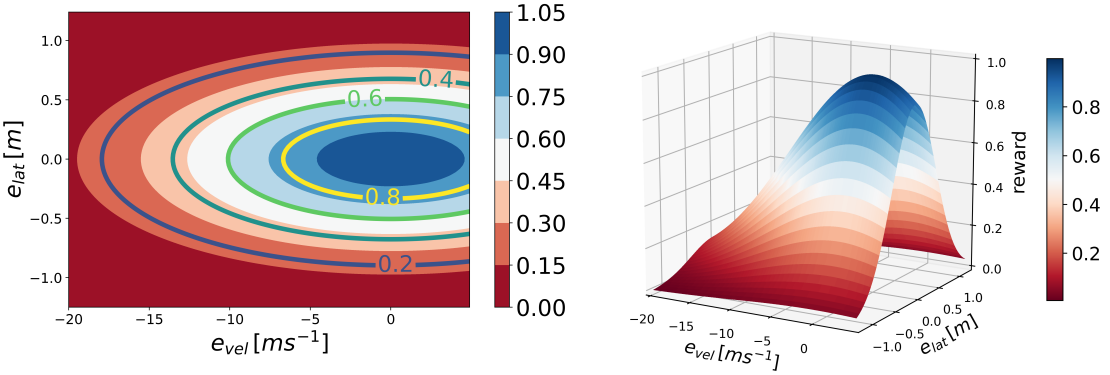
4.2.3. Bi-objective reward function

In our work, we get inspired from the social psychology and namely the reward theory of attraction [27]: people are attracted to other people that make them feel good in some way.
For the tracking-optimised mode, we want our agent to be attracted to the pair (lateral error free, velocity error free), which we call the **attraction point**. To get our agent attracted to this, we make it feel good by giving it high rewards. Our idea is to create an **attraction region** in the continuous space. This region has the shape of an ellipsoid, is symmetric and centred around the attraction point, where the rewards reach their peak. The rewards decrease the further the agent moves away from the centre in any direction, i.e., the reward is always dependent on the two variables. For instance, if one of the lateral-

or the velocity error is small but the other one is large, a low reward is returned. A high reward is delivered only when both errors are marginal, i.e., near the attraction point. In this way, the attraction point serves as a rewards magnet. In order to collect the highest possible cumulative reward, the agent needs to converge to the centre of the attraction region.
A physical object that shapes our idea of the attraction region can be a bell. There is a variety of functions possessing a bell shape, e.g., Gaussian function, Fuzzy Logic, Hyperbolic secant, Witch of Agnesi, Bump function, raised cosine type and other algebraic functions. Considering its easy extensibility to a second dimension and thus having ellipsoid level sets, we choose the Gaussian function.
Thus, we define our reward function that considers the two objectives as following:

$$R(e_{vel}, e_{lat}) = A \exp\left(-\left(\frac{e_{vel}^2}{2\sigma_v^2} + \frac{e_{lat}^2}{2\sigma_l^2}\right)\right) \tag{4.3}$$

where A is the height of the peak. σ_v and σ_l are the e_{vel} and e_{lat} spreads of the blob, i.e., control the width of the bell. Figures 4.3a and 4.3b show a contour- and a 3D representation of the reward function with an amplitude $A = 1$ and the deviations in e_{vel} and in e_{lat} directions are respectively $10\,\mathrm{m\,s^{-1}}$ and $0.5\,\mathrm{m}$. We represent the velocity only in range $[-20\,\mathrm{m\,s^{-1}}, 5\,\mathrm{m\,s^{-1}}]$ in our plot, as that is the common range that the agent experiences during training.



(a) rewards contour plot

(b) 3D representation

Figure 4.3.: reward function

Shaping the reward function in a way that it covers a large area in the space has a huge impact on the speed of the learning process. Instead of having sparse rewards, where the agent does not get rewarded very often, the agent gets gradual feedback with shaped rewards and knows if it's getting better and closer to its objectives.
This reward function generates only positive rewards, which have many advantages. In fact, as the actions must be positive as well, we can design the same neural network architecture for both the value function and the policy. On the other side, positive rewards encourage the agent to accumulate as much as possible.

4.2.4. Multi-objective reward function

The RL world lacks of general reward function formulations that consider optimisation problems. This motivates us to propose a novel general **multi-objective Gaussian (MOG)** reward function for n goals as following:

$$R(x_1, x_2, \ldots, x_n) = A \exp\left(-\left(\sum_{k=1}^{n} \frac{(x_k - x_{0,k})^2}{2\sigma_k^2}\right)\right) \tag{4.4}$$

where all x_k denote our **objective signals**, $x_{0,k}$ and σ_k represent their respective **centre (objective value)** and **deviation** for $k = 1, \ldots, n$. Analysing this function, the rewards are maximised when all centred signals approach zero, i.e., when all signals x_k approach their respective objective values $x_{0,k}$. The further a signal or multiple signals move away from their objective values, the less the reward w.r.t. the distribution of their values σ_k.
We apply the Equation (4.4) to consider the comfort mode with $n = 4$ and all objective values are zero, as we want the velocity, lateral error, jerk and lateral acceleration to be minimal:

$$R(e_{vel}, e_{lat}, a_{lat}, j_{long}) = A \exp\left(-\left(\frac{e_{vel}^2}{2\sigma_v^2} + \frac{e_{lat}^2}{2\sigma_l^2} + \frac{a_{lat}^2}{2\sigma_a^2} + \frac{j_{long}^2}{2\sigma_j^2}\right)\right) \tag{4.5}$$

where σ_l and σ_j are the a_{lat} and j_{long} deviations.
Moreover, we propose a second reward function formulation for the multi-objective optimisation which we intend to compare its performance with the MOG w.r.t. the learning performance. This reward function is composed of cascaded 2-dimensional Gaussian functions. We call this novel general reward function as **multi-objective cascaded Gaussian (MOCG)**. This reward function can be applied to $n \geq 2$ objectives:

$$\begin{aligned}
n = 2 \quad & R_2(x_1, x_2) = A \exp\left(-\left(\frac{(x_1 - x_{0,1})^2}{2\sigma_1^2} + \frac{(x_2 - x_{0,2})^2}{2\sigma_2^2}\right)\right) \\
n = 3 \quad & R_3(x_1, x_2, x_3) = R_2(R_2(x_1, x_2) - A,\ x_3) \\
n = 4 \quad & R_4(x_1, \ldots, x_4) = R_2(R_2(x_1, x_2) - A,\ R_2(x_3, x_4) - A) \\
n = 5 \quad & R_5(x_1, \ldots, x_5) = R_4(R_2(x_1, x_2) - A,\ x_3,\ x_4,\ x_5) \\
n = 6 \quad & R_6(x_1, \ldots, x_6) = R_4\left(R_2(x_1, x_2) - A,\ R_2(x_3, x_4) - A,\ x_5,\ x_6\right) \\
\vdots \quad & \quad \vdots
\end{aligned} \tag{4.6}$$

The output of every R_n function reaches its maximum A when all n input signals are 0. Thus, when cascading (R_n as input for R_k), we should centre the resulting output of the R_n function by subtracting its amplitude A from it. Consequently, $R_n - A$ approaches 0 when all objectives are minimal, which is suitable as input of the R_k function. Hence, the latter outputs its maximum value when all k centred inputs converge to 0.
We apply the Equation (4.6) to consider the comfort mode with $n = 4$ and all objective values are zero, as we want the velocity, lateral error, jerk and lateral acceleration to be minimal:

$$R_4(e_{vel}, e_{lat}, a_{lat}, j_{long}) = R_2(R_2(e_{vel}, e_{lat}) - 1,\ R_2(a_{lat}, j_{long}) - 1) \tag{4.7}$$

4.3. Generating actions

In this section, we introduce the restrictions on the MPC weights and how the RL agent can meet them. Moreover, we define the action space. Our goal in this section is to reduce the action space as much as possible. Thus, we have smaller networks and faster learning process.

4.3.1. Requirements

To ensure the recursive stability and feasibility of the MPC, the following assumptions must hold: $Q = Q^T \succcurlyeq 0, P = P^T \succ 0, R = R^T \succ 0$ (Section 2.2). To reduce the workload in implementation, we take a more strict assumption that keeps the recursive stability and feasibility holding. Instead of assuming that Q is positive semidefinite, we assume it is positive definite. Thus, all three weighting matrices are positive definite, i.e., $Q = Q^T \succ 0, P = P^T \succ 0, R = R^T \succ 0$. Concretely, this means that cost function elements weighted with the matrix Q can not be weighted with zero, i.e., can not disappear from the cost function. Our main motivation is that all agent actions have the same domain. This does not have a big impact on the MPC performance: if a term should be insignificant, its weight is set to a small positive value and consequently the other terms are relatively penalised with larger positive weights.

4.3.2. Defining the action space

As introduced in Section 2.4, the action space is the set of valid actions. In our case, the action space must reflect the elements of the 3 matrices Q, R and P. The elements of the matrices are real-valued. Hence, our action space must be continuous. This induces restrictions on the choice of the RL algorithms, e.g., we can not use Q-learning or Deep Q Network (DQN) as they are suitable only for discrete action spaces, but we can use algorithms suitable for continuous action spaces such as Vannilla Policy Gradient (VPG), Proximal Policy Optimisation (PPO) or Twin Delayed Deep Deterministic Policy Gradient (TD3).
Usually the weighting matrices are diagonal. Thus, we can formulate them as following:

$$Q = \begin{pmatrix} q_x & 0 & 0 & 0 \\ 0 & q_y & 0 & 0 \\ 0 & 0 & q_\psi & 0 \\ 0 & 0 & 0 & q_a \end{pmatrix} \qquad P = \begin{pmatrix} p_x & 0 & 0 & 0 \\ 0 & p_y & 0 & 0 \\ 0 & 0 & p_\psi & 0 \\ 0 & 0 & 0 & p_a \end{pmatrix} \qquad R = \begin{pmatrix} r_a & 0 & 0 \\ 0 & r_\omega & 0 \\ 0 & 0 & r_\vartheta \end{pmatrix} \tag{4.8}$$

where q_x, q_y, q_ψ and q_a are respectively the weights for x-, y-, yaw errors and the lateral acceleration, p_x, p_y, p_ψ and p_a physically represent the same as the latter but for the terminal cost. r_a, r_ω and r_ϑ designate respectively the weights for the reference longitudinal acceleration, the steering wheel target angular velocity and the longitudinal velocity error. To accelerate the learning process, we take $Q = P$. Thus, the stage cost and the terminal cost are weighted equally. With this assumption, we give up on freedom degrees that can be exploited for convergence and stability. However, we reduce the action space, and thus,

the learning load. Then, we instantiate our MPC Problem formulated in Equation (3.1) by inserting the results of the discussions above. We get the following problem: [1]

$$\min_{u,\vartheta} \int_{\tau=t_k}^{t_k+T_p} q_x \left(\Delta x(\tau)^2 + \Delta x(T_p)^2\right) + q_y \left(\Delta y(\tau)^2 + \Delta y(T_p)^2\right) + q_\psi \left(\Delta \psi(\tau)^2 + \Delta \psi(T_p)^2\right) + q_a \left(a_{lat}(\tau)^2 + a_{lat}(T_p)^2\right) + r_a\, a_{ref}(\tau)^2 + r_\omega\, \omega_{s,ref}(\tau)^2 + r_\vartheta\, \Delta\vartheta(\tau)^2 \quad d\tau \tag{4.9}$$

where $\Delta x = x - x_{ref}$, applies similarly to y, ψ and ϑ.

4.3.2.1. 6 dimensional action space

To reduce the action space, we choose $q_x = q_y$. Optimising the lateral behaviour is achieved by minimising the x- and y- errors in the coordinate space. Thus, it makes more sense to reduce the same amount of error on both coordinates. We make use of Equation (4.9). Thus, we obtain a 6-dimensional action space:

$$Q = P = \begin{pmatrix} q_x & 0 & 0 & 0 \\ 0 & q_x & 0 & 0 \\ 0 & 0 & q_\psi & 0 \\ 0 & 0 & 0 & q_a \end{pmatrix} \qquad R = \begin{pmatrix} r_a & 0 & 0 \\ 0 & r_\omega & 0 \\ 0 & 0 & r_\vartheta \end{pmatrix} \tag{4.10}$$

The weighting matrices are square matrices. We assume that their eigenvalues are real. Thus, taking into account the positive definiteness requirement of the Q, R and P discussed in section 4.3.1, all eigenvalues should be positive [21]. It follows that all diagonal terms should be positive, as the matrices are diagonal based on our first assumption.
The outputs of the policy network correspond to the MPC cost function weights. Since we model our policy with a multilayer perceptron (MLP) neural network, the output layer should generate positive values. This is achieved by specifying the activation function of the output layer, e.g., Sigmoid or the tangent hyperbolic (Tanh) (introduced in Section 2.3). However, we face the following challenge: tanh can generate non positive values and the limit of the sigmoid is zero for negative infinity. To avoid the problem, we clip the output of the network to a chosen minimum value a_{min}. We take the minimum as a small value near zero, e.g., $a_{min} = 10^{-4}$.

$$a_t = \max\left(\pi_\theta(s_t), a_{min}\right) \tag{4.11}$$

Hence, the agent generates actions in range $[a_{min}, 1]$.

4.3.2.2. 5 dimensional action space

We can reduce the action space even more. To achieve this, we can remodel our MPC problem (Equation (4.9)) and reduce the freedom degrees (weights) by fixing a single weight and scaling the rest. This can be done by taking any weight as a factor of the sum

[1] For readability reasons, we do not write the constraints each time we are developing the MPC problem. The constraints remain unchanged.

and thus fix one weight on 1. For example, we consider the first appearing weight q_x. Then, the optimisation problem can be formulated as the following:

$$\min_{u,\vartheta} \quad q_x \int_{\tau=t_k}^{t_k+T_p} 1\left(\Delta x(\tau)^2 + \Delta x(T_p)^2\right) + \frac{q_y}{q_x}\left(\Delta y(\tau)^2 + \Delta y(T_p)^2\right) + \frac{q_\psi}{q_x}\left(\Delta\psi(\tau)^2 + \Delta\psi(T_p)^2\right) + \frac{q_a}{q_x}\left(a_{lat}(\tau)^2 + a_{lat}(T_p)^2\right) + \frac{r_a}{q_x}a_{ref}(\tau)^2 + \frac{r_\omega}{q_x}\omega_{s,ref}(\tau)^2 + \frac{r_\vartheta}{q_x}\Delta\vartheta(\tau)^2 \quad d\tau \tag{4.12}$$

Minimising the cost function with the factor q_x (Equation (4.12)) or without (Equation (4.13)) provides the same result.

$$\min_{u,\vartheta} \quad \int_{\tau=t_k}^{t_k+T_p} 1\left(\Delta x(\tau)^2 + \Delta x(T_p)^2\right) + \frac{q_y}{q_x}\left(\Delta y(\tau)^2 + \Delta y(T_p)^2\right) + \frac{q_\psi}{q_x}\left(\Delta\psi(\tau)^2 + \Delta\psi(T_p)^2\right) + \frac{q_a}{q_x}\left(a_{lat}(\tau)^2 + a_{lat}(T_p)^2\right) + \frac{r_a}{q_x}a_{ref}(\tau)^2 + \frac{r_\omega}{q_x}\omega_{s,ref}(\tau)^2 + \frac{r_\vartheta}{q_x}\Delta\vartheta(\tau)^2 \quad d\tau \tag{4.13}$$

Considering the above, our weighting matrices are formulated as the following:

$$Q = P = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{q_\psi}{q_x} & 0 \\ 0 & 0 & 0 & \frac{q_a}{q_x} \end{pmatrix} \qquad R = \begin{pmatrix} \frac{r_a}{q_x} & 0 & 0 \\ 0 & \frac{r_\omega}{q_x} & 0 \\ 0 & 0 & \frac{r_\vartheta}{q_x} \end{pmatrix} \tag{4.14}$$

Taking into consideration the outcomes of Equation (4.13) and Equation (4.14), the policy network should generate positive values relative to 1, i.e., greater or smaller than 1. The weights tell how big a term should be penalised compared to another term in the MPC cost function. Accordingly, the output activation function can be ReLu (introduced in section 2.3). ReLu generates non negative output. Thus, we clip the output of the network to the minimum value a_{min}.

4.3.2.3. Conclusion

We tried experiments with 5 actions relative to 1 and with 6 actions non relative to a fixed value. With less actions, we aimed for a faster learning process. However, the performance of the agent was worse. Figure 4.4 depicts the rewards gained over the training episodes. It is straightforward to see that the non relative to 1 weights deliver a better learning performance. Fixing weights makes the training process harder. Thus, the agent performs better when given the total freedom in specifying the relativity of the weights.
Altogether, we choose the 6 dimensional action space (Equation (4.10)) and formulate it as following:

$$\mathcal{A} = \{a |\, a \in \mathbb{R}^6_{>0}\} \tag{4.15}$$

Where a is the action generated by the RL agent and contains the 6 unknown diagonal elements of the weighting matrices. Thanks to the analysis in this section, we reduced our action space to 6 dimensions.
Apart from the assumption $q_x = q_y$, the work done above can hold for general MPC

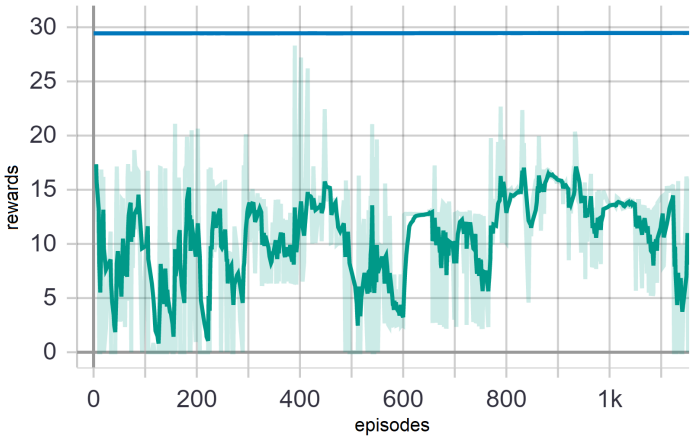


Figure 4.4.: comparison w.r.t. cumulative rewards of 2 agents learning with 2 different action spaces. Green: 5-, blue: 6-dimensional action space

frameworks, since we took no further use case specific assumptions. Thus, we can define an action space for the general RL of MPC parameters framework introduced in the concept Algorithm 3, where n is the number of diagonal elements to be determined:

$$\mathcal{A} = \{a | \, a \in \mathbb{R}^n_{>0}\} \tag{4.16}$$

4.4. Scaling MPC cost function terms

4.4.1. Challenges

To increase the efficiency and reliability of our solver and to accelerate the learning process of the agent, we propose scaling the signals appearing in the MPC cost function. Considering the MPC problem in Equation (4.9), it is straightforward to notice that we are adding up different physical quantities. Certain quantities have much smaller or larger magnitudes than others in the cost function. Compared to a default MPC problem, where all terms have equivalent weights, i.e., all weights are 1, MPC delivers often a substandard performance when the cost function terms have different orders of magnitude.

4.4.2. Solution

In this section, we solve the challenge introduced above by scaling the different MPC cost function terms. Once the signals are scaled, the default configuration is that all cost function terms are dimensionless and equivalent. This is a good kickoff for training our models. The RL agent focuses on the relative priority of each cost function term rather than trying to find the best combination of term priority and signal scale. This reduces significantly the exploration time and helps to prevent that the algorithm-updates are stuck in local minima. Also, scaling improves the numerical conditioning, as calculations are less affected with round-off errors, i.e., a better optimisation and state estimation calculations.

We consider the cost function in Equation (4.9) and introduce the scaling factors:

$$\begin{aligned}\min_{u,\vartheta} \quad & \int_{\tau=t_k}^{t_k+T_p} q_x \left(\left(\frac{\Delta x(\tau)}{s_x}\right)^2 + \left(\frac{\Delta x(T_p)}{s_x}\right)^2\right) + q_y \left(\left(\frac{\Delta y(\tau)}{s_y}\right)^2 + \left(\frac{\Delta y(T_p)}{s_y}\right)^2\right) \\ & + q_\psi \left(\left(\frac{\Delta \psi(\tau)}{s_\psi}\right)^2 + \left(\frac{\Delta \psi(T_p)}{s_\psi}\right)^2\right) + q_a \left(\left(\frac{a_{lat}(\tau)}{s_{lat}}\right)^2 + \left(\frac{a_{lat}(T_p)}{s_{lat}}\right)^2\right) \\ & + r_a \left(\frac{a_{ref}(\tau)}{s_{long}}\right)^2 + r_\omega \left(\frac{\omega_{s,ref}(\tau)}{s_\omega}\right)^2 + r_\vartheta \left(\frac{\Delta\vartheta(\tau)}{s_\vartheta}\right)^2 \quad d\tau\end{aligned} \tag{4.17}$$

$$\begin{aligned}\min_{u,\vartheta} \quad & \int_{\tau=t_k}^{t_k+T_p} q_x \frac{1}{s_x^2}\left(\Delta x(\tau)^2 + \Delta x(T_p)^2\right) + q_y \frac{1}{s_y^2}\left(\Delta y(\tau)^2 + \Delta y(T_p)^2\right) \\ & + q_\psi \frac{1}{s_\psi^2}\left(\Delta \psi(\tau)^2 + \Delta \psi(T_p)^2\right) + q_a \frac{1}{s_{lat}^2}\left(a_{lat}(\tau)^2 + a_{lat}(T_p)^2\right) \\ & + r_a \frac{1}{s_{long}^2} a_{ref}(\tau)^2 + r_\omega \frac{1}{s_\omega^2} \omega_{s,ref}(\tau)^2 + r_\vartheta \frac{1}{s_\vartheta^2} \Delta\vartheta(\tau)^2 \quad d\tau\end{aligned} \tag{4.18}$$

Table 4.1.: scaling factors

scaling factor	meaning
s_x	x error
s_y	y error
s_ψ	yaw error
s_{lat}	lateral acceleration
s_{long}	longitudinal reference acceleration
s_ω	steering wheel angular velocity
s_ϑ	velocity error

Table 4.1 denote the introduced scaling factors. The goal is not to define exact values for each scaling factor. They are meant only as help for the agent, i.e., a first kickoff to reduce the search space. The factors are chosen with intuition to make the cost function terms approximately in same order of magnitude. Trying to choose the scaling factors exactly would make us degrade to manual tuning. In this thesis, we want to avoid that. We choose each scaling factor to be approximately equal to half the span of the variable, where the span of a variable stands for the difference between its maximum and minimum values. Generally, the maximum or mean of a variable is not known and we choose a desired span. For example, if we aim for the lateral error to be in range $[-0.25\,\text{m}, 0.25\,\text{m}]$, we take $0.25\,\text{m}$ as scaling factor. We scale the MPC cost function terms in the Weights Introduction Interface (section 3.4.2). We represent the factors in a vector:

$$s = \left[\tfrac{1}{s_x^2}, \tfrac{1}{s_y^2}, \tfrac{1}{s_\psi^2}, \tfrac{1}{s_{lat}^2}, \tfrac{1}{s_{long}^2}, \tfrac{1}{s_\omega^2}, \tfrac{1}{s_\vartheta^2}\right] \tag{4.19}$$

As we assumed that $q_x = q_y$, we take $s_x = s_y$ too. In our work, we consider a pure path-tracking mode and a comfort driving style. As they have different objectives, we need two different models to be trained. Through experiments, we noticed that choosing two different kickoffs that express the desired variables span for the two different modes yields a better performance. Thus, we have defined two different scaling sets.

- **tracking-optimised mode**: Determining the scaling factors is simple. We set the factors to half of the variables span, if prior knowledge is available, e.g., $s_\psi = 7\,^\circ\,\mathrm{s}^{-1}$, $s_{lat} = 4\,\mathrm{m\,s}^{-2}$, $s_{long} = 3\,\mathrm{m\,s}^{-2}$ and $s_\omega = 250\,^\circ\,\mathrm{s}^{-1}$. Wehn prior knowledge is not available, we choose desired values. For the x and y coordinate we aim for an error less than 0.25 m. Our reference velocity has an approximate maximal value of $20\,\mathrm{m\,s}^{-1}$. The velocity error has commonly a range $[-5\,\mathrm{m\,s}^{-1}, 20\,\mathrm{m\,s}^{-1}]$. As our path-optimised mode requires a small velocity error, we choose the lower value as a scale factor. Accordingly, we designate $s_x = s_y = 0.25\,\mathrm{m}$ and $s_\vartheta = 5\,\mathrm{m\,s}^{-1}$

$$s_{tracking} = \left[\tfrac{1}{0.25^2},\ \tfrac{1}{0.25^2},\ \tfrac{1}{7^2},\ \tfrac{1}{4^2},\ \tfrac{1}{3^2},\ \tfrac{1}{250^2},\ \tfrac{1}{5^2}\right] \tag{4.20}$$

- **Comfort mode**: We have other requirements. Our priority is to minimise lateral acceleration and longitudinal jerk. For this reason, we allow larger lateral- and velocity errors. Hence, we adapt our variables span in the scaling factors. On the one side, we enlarge the span of the x-,y- coordinate. On the other side, we reduce the span of the lateral acceleration. As the latter is proportional to the yaw angle, we reduce its span too.

$$s_{comfort} = \left[\tfrac{1}{0.5^2},\ \tfrac{1}{0.5^2},\ \tfrac{1}{5^2},\ \tfrac{1}{1.5^2},\ \tfrac{1}{3^2},\ \tfrac{1}{250^2},\ \tfrac{1}{5^2}\right] \tag{4.21}$$

It is important to notice, that once we introduce the scaling factors, we hold them unchanged during the learning process. Scaling made the learning process way much faster, as we save the agent unnecessary exploration time in the positive real numbers space. Taking this step led also to a major performance improvement of the closed-loop system w.r.t. our optimisation goals.

4.5. Choosing observations

A big challenge that we face in refining our approach is the choice of the observations: which physical quantities/signals does our agent need to observe, in order to make the learning process easier and to achieve the optimal results? Considering our proposed general approach model in Fig. 1.1 and the use case specific architecture in Fig. 3.5, we sample observations from the plant states or from the controller states. We think of two possible concepts that are driven by two different arguments. In the first one, we only consider the states or outputs of the plant. We call it Plant in the Loop (PiL). The second one is considering controller relevant signals. If optimisation goals are needed, which are not included in the MPC formulation, these signals can be observed additionally. Hence, we call it hybrid MPC in the Loop.

4.5.1. Concept I: Plant in the Loop

In our case the plant is the vehicle model. This concept is driven by the idea that our main goal of implementing a controller is to achieve a desired output behaviour of our plant. Generally, the plant outputs are derived from the states. Hence, it makes sense that our agent directly observes plant states and learns based on them to find the optimal policy that meets our objectives. Accordingly, we choose the following signals as observations:

- Lateral error (mean & maximum absolute value) [m]
- Yaw angle (mean & maximum absolute value) [$^\circ\,\mathrm{s}^{-1}$]
- Steering angle (mean & maximum absolute value) [$^\circ$]
- Lateral acceleration (mean & maximum absolute value) [$\mathrm{m\,s}^{-2}$]
- Longitudinal acceleration (mean & maximum absolute value) [$\mathrm{m\,s}^{-2}$]
- Velocity mean squared error [$\mathrm{m\,s}^{-1}$]
- Longitudinal jerk (mean & maximum absolute value) [$\mathrm{m\,s}^{-3}$]

The signals are chosen w.r.t. the vehicle's output as discussed in section 3.4.1.

4.5.2. Concept II: hybrid MPC in the Loop

This concept is driven by the idea that the weights are actually for the the MPC cost function terms. So it makes sense that the agent observes exactly the terms that we intend to penalise. Thus, the agent observes all the terms appearing in the cost function of the MPC problem (Equation (3.1)). Hence, we observe the input signals of the MPC:

- x-coordinate error (mean & maximum absolute value) [m]
- y-coordinate error (mean & maximum absolute value) [m]
- Yaw angle error (mean & maximum absolute value) [$^\circ$]
- Lateral acceleration (mean & maximum absolute value) [$\mathrm{m\,s}^{-2}$]
- Longitudinal acceleration (mean & maximum absolute value) [$\mathrm{m\,s}^{-2}$]
- Velocity mean squared error [$\mathrm{m\,s}^{-1}$]
- Steering wheel angular velocity (mean & maximum absolute value) [$^\circ\,\mathrm{s}^{-1}$]

Yet, a set of observations delivered to the RL agent is not the same as delivered to the solver. Whereas, the solver receives new input signals every MPC sample time T_s, the agent receives input signals every parameter switching time $T_{sw} = kT_s$, where $k \in \mathbb{N}$. The observations are maximum absolute (Equation (3.3)) or mean values (Equation (3.2)) over k values in the past. Thus, the RL algorithm learns with pairs of executed actions in form of new MPC weights and their effect on the MPC cost function over time.
Since we optimise nonexistent quantities in the cost function, e.g., lateral error or jerk, we include them in the observation vector too:

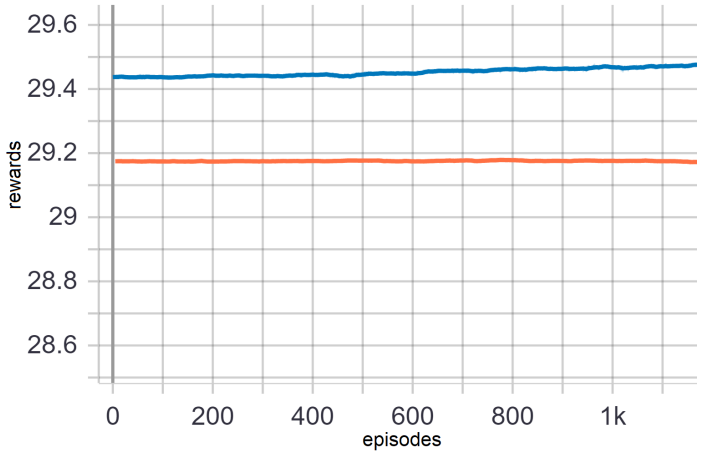


Figure 4.5.: PiL vs. hybrid MPC in the Loop: orange is PiL and blue is MPC in the Loop.

- Lateral error (mean & maximum absolute value) in [m]
- Longitudinal jerk (mean & maximum absolute value) [m s^{-3}]

We have learned 2 frameworks applying the first and the second concept. We have used the same algorithm (VPG) and the same bi-objective reward function specified in Section 4.2. The rewards gained with each concept are displayed in Fig. 4.5. Both concepts deliver comparable results. However, we notice a slight advantage of the MPC in the Loop compared to PiL. Namely, the agent is directly fed with the cost function terms, observes them and generates weights to reduce their weighted sum correspondingly.

In this chapter, we presented the challenges we faced while designing our experiments. First, we analysed the effect of the PMPC switching time T_{sw}. Second, we presented the challenges of expressing our optimisation objectives in a reward function that helps the RL agent to learn. We defined a reward attraction region expressed by a bell shaped function and applied a bi-objective Gaussian for the tracking-optimised mode. We extended our formulation to consider n optimisation objectives by introducing the multi-objective Gaussian (MOG)- and the multi-objective cascaded Gaussian (MOCG) reward functions. Then we applied them for the comfort mode.
Moreover, we discussed the restrictions on the action space imposed by the MPC formulation. We reduced our action space to 6 positive actions and presented the impact of the defined action space on designing the MLP neural networks and choosing a suitable activation function.
Additionally, we discussed the significance of scaling the MPC cost function terms on the controller performance and the learning process.
Finally, we deliberated the choice of the observations provided to the agent by presenting the Plant in the Loop and the hybrid MPC in the Loop concepts. The latter delivered a slightly better performance.
Based on the results of the discussions in this chapter, we design experiments for the tracking-optimised- and comfort modes. We let our agents train parallel on multiple

computers. In the next chapter, we evaluate the results delivered by the best trained neural networks, i.e., models with the most rewards collected during a training episode.

5. Evaluation

First, we define key performance indicators (KPIs) to compare different experiments and models. Second, we benchmark different RL algorithms used to learn the tracking-optimised mode. Furthermore, we analyse the time evolution of the policy output of each algorithm and check the MPC constraints satisfaction. Third, we evaluate the robustness of the learned model to environment change, model mismatch and different vehicles. Finally, we evaluate the comfort mode experiments. Throughout this chapter, we always compare the performance of the learned model of each experiment to the hand tuned MPC.

5.1. Defining KPIs

A big challenge that we face, is to measure the performance of the agent or to compare different agents/models. For instance, the agent knows what is good or bad from the objective that we specified in the reward function. But what if we change the design of the reward function itself? How can we measure objectively the quality of the trained models and benchmark them?
We need to define key performance indicators (KPIs), that can express how good the agent is doing w.r.t. our use case specific goals, i.e., concretely regarding longitudinal-, lateral- and comfort behaviour. Defining a single metric that evaluates the whole vehicle guidance system is difficult and uncommon. Therefore, we introduce individual metrics related to our goals. First, we define 2 metrics regarding longitudinal and lateral control as following.
Time Average of the absolute longitudinal error $TA(e_{long})$:

$$\begin{aligned} TA(e_{long}) &= \tfrac{1}{T_{ep}} \int_0^{T_{ep}} f\left(g(t)\right)\ dt \\ g &= \left|\mu_{e_{vel}}(kT_{sw})\right|, \quad k = 0, \ldots, \tfrac{T_{ep}}{T_{sw}} \end{aligned} \tag{5.1}$$

where T_{ep} is the episode length and f is an spline interpolation function that delivers a continuous differentiable output. During an episode, we get discrete mean values of the velocity signal $\mu_{e_{vel}}$ at each MPC parameters switching time T_{sw} as in Equation (3.2). This represents exactly our velocity error observation from the environment. The idea is that the surface area constructed by the x-axis and the interpolated signal represents a good performance measure of the longitudinal guidance. A smaller area means less cumulative velocity error. Also, it approximates the total longitudinal distance lost over a period of time. To compute the surface area, we integrate the absolute values over the time period (in our case, the episode length T_{ep}). To approximate the definite integral, we use the Composite Simpson's rule [11], as the function we are integrating can be not smooth over

the interval. We apply the approximation to our case in Equation (5.2).

$$\int_0^{T_{ep}} f(t)\, dt \approx \frac{T_{ep}}{3T_{sw}} \left[f(0) + 2 \sum_{j=1}^{\frac{T_{sw}}{2}-1} f(t_{2j}) + 4 \sum_{j=1}^{\frac{T_{sw}}{2}} f(t_{2j-1}) + f(T_{ep}) \right] \tag{5.2}$$

However, we want to compare agents/models running different time periods. To make this metric independent from the time accumulation, we divide the area by its corresponding time period. Thus, we get an approximate time average measure of the longitudinal error. We can interpret this metric as the following: the smaller the value, the higher the quality.
Percentage of exceeding the limit value (PEL): Sometimes we are not only interested in the fact that a signal exceeds a certain limit value, but also how often does that occur in a certain period of time. In our use case, we want to measure how often the maximum absolute lateral error exceeds the limit value L_{lat}. We choose $L_{lat} = 0.2\,\text{m}$, as a manually driver can clearly sense a lateral deviation of 20 cm.

$$PEL = \frac{100}{\frac{T_{ep}}{T_{sw}}} \sum_{k=0}^{\frac{T_{ep}}{T_{sw}}} exc\left(e_{lat}\left(kT_{sw}\right)\right) \tag{5.3}$$

where

$$exc(e_{lat}) = \begin{cases} 1 & \text{if } e_{lat} \geq L_{lat} \\ 0 & \text{else} \end{cases}$$

Similarly, we compute the PEL for the lateral acceleration. The limit for the lateral acceleration is $2\,\text{m}\,\text{s}^{-2}$.
We consider 2 objectives: path-tracking and comfort. For path-tracking, we are interested in the lateral- and longitudinal behaviour. For the comfort, we consider the longitudinal jerk and the lateral acceleration. Apart from PEL and TA we make use of other metrics, e.g., maximum/minimum or absolute maximum values, mean squared error (MSE) and root-mean-square error (RMSE). We present an overview of the corresponding metrics to each objective in Table 5.1.

Table 5.1.: goals and corresponding metrics

goal		KPIs
tracking	lateral error	max/min
		PEL
		MSE
	longitudinal error	TA
		RMSE
comfort	lateral acceleration	max/min
		PEL
	longitudinal jerk	max absolute

5.2. Tracking-optimised mode

We recall that the tracking-optimised mode minimises the lateral- and velocity errors. The tracking optimised mode is motivated by a variety of use-cases, e.g., in a parking situation, the vehicle should stick to the computed parking trajectory and minimise the deviation to avoid crashes.
For that, we benchmark the performance of 3 RL agents and compare them to the human expert work. Specifically we trained a Vanilla Policy Gradient (VPG), a Proximal Policy Optimisation (PPO) and a Twin Delayed Deep Deterministic Policy Gradient (TD3) agent. The TD3 multilayer perceptron (MLP) networks have 3 hidden layers with respectively 256, 512 and 256 nodes. For the VPG and PPO, we associate a (64,128,64) architecture. For all algorithms, the activation functions used in the hidden layers is Tangent hyperbolic (Tanh) and the output activation function for the policy is a Sigmoid. The value function networks have no output activation function.
The agents were trained based on the reward function that we formulated in Equation (4.3). Each agent's reward function has an amplitude $A = 1$. The deviations σ_v and σ_l of the velocity- e_{vel} and the lateral error e_{lat} were configured as presented in Table 5.2:

Table 5.2.: reward function configuration for the different agents

	VPG	PPO	TD3
$\sigma_v\,[\mathrm{m\,s^{-1}}]$	10	5	10
$\sigma_l\,[\mathrm{m}]$	0.5	0.5	0.5

We choose the deviations as approximately the half of the span of each variable. The PPO agent delivered a poor longitudinal performance with the latter. Thus, we associate a different configuration. During training, we usually give rewards along a large interval, e.g., $[-20\,\mathrm{m\,s^{-1}}, 20\,\mathrm{m\,s^{-1}}]$ for the velocity error, so that the agent explores more its environment and learns better actions. We take a velocity error deviation $\sigma_v = 10\,\mathrm{m\,s^{-1}}$ and a lateral error deviation of $\sigma_l = 0.5\,\mathrm{m}$. In evaluation phase, we consider a stricter configuration of the reward function compared to the learning process: $\sigma_v = 2\,\mathrm{m\,s^{-1}}$ and $\sigma_l = 0.1\,\mathrm{m}$ respectively.
To evaluate the agents, we only consider models that collected the highest cumulative rewards in a training episode. Then, we run the best model of each experiment 300 s simulation time (T_{ep}) in the MPC environment. Since the switching time $T_{sw} = 10\,\mathrm{s}$, the agent performs 30 PMPC parameter updates, i.e., an evaluation episode has the length $l = 30$. The PMPC performs $\frac{T_{ep}}{T_s} = \frac{300\,\mathrm{s}}{0.04\,\mathrm{s}} = 7500$ steps, i.e., 7500 times solving the MPC problem in Equation (3.1) and applying the first control input on the vehicle, where T_S is the MPC sampling time.
Through the evaluation chapter, we refer to the human expert's model as a vector of unchanged weights. The latter were determined through hand tuning by an MPC expert [54]. The goal of the hand tuned MPC was to find a trade-off between tracking and comfort. We can reproduce the training or evaluation rewards collected by the hand-tuned MPC. We remove the scaling of the cost function terms, ignore the output of our policy and always

introduce the constant weights: $q_x = q_y = 25000$, $q_\psi = 2700$, $q_a = 700$, $r_a = 2500$, $r_\omega = 0.2$ and $r_\vartheta = 1000$. In the following sections, we first present the results of our experiments w.r.t. the collected reward. Then, we evaluate the longitudinal- and lateral behaviour.

5.2.1. Rewards

During training with our bell-shaped reward function defined in Equation (4.3), the TD3 agent reaches a sum of rewards equal to 29.6 (Fig. 5.1) and so the highest rewards. VPG collects a close amount of rewards and it is slightly above the PPO performance. The same holds in evaluation.
Since the reward function generates a maximum instantaneous reward of 1, the maximum possible reward over a trajectory is 30. Thus, the TD3-agent collects 98.67% of the maximum possible rewards, i.e., it makes an average loss of 0.013 each time step. The maximum possible rewards over a trajectory means the vehicle follows perfectly the reference with absolutely neither lateral nor velocity errors. Note that our velocity reference adopted from the work in [54] is not feasible, as it has a lot of reference jumps (Fig. 5.4a) and requires high velocities in curves.
According to the strict reward function for evaluation, the TD3 agent collects 22.26 from 30 possible rewards, i.e., with 74.2% accuracy and 25.8% loss. Compared to training, the same policy collects 75.19% of its training rewards. In contrast, the manually-tuned MPC collects only 10.56 rewards. Thus, our concept performs **210.83% better** than the human expert w.r.t. the path-tracking objectives formulated in the reward function.

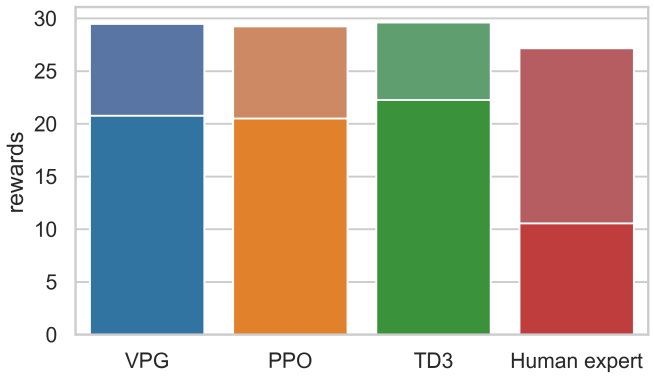
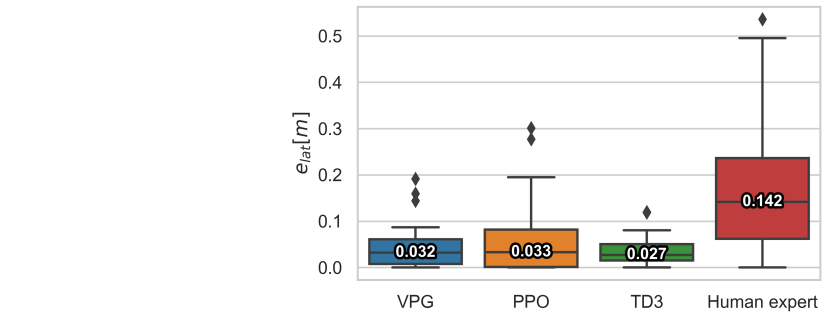


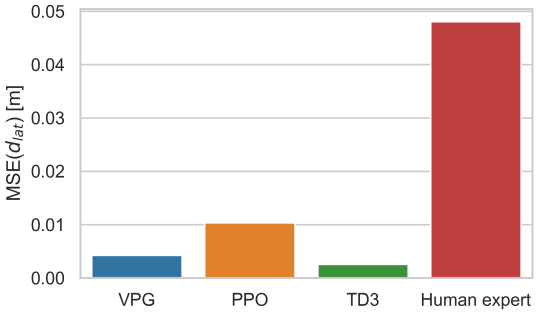
Figure 5.1.: cumulative rewards collected by 3 RL algorithm-models and the human expert parameter set: bright denotes training rewards and dark denotes evaluation rewards

5.2.2. Lateral behaviour

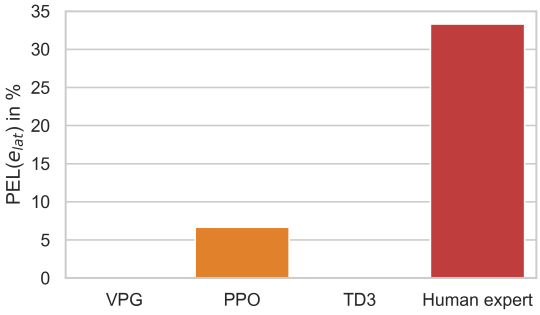
Figure 5.2a delineates a box plot of the lateral error variation during the evaluation run. The TD3 agent has the lowest median: 2.7 cm. In contrast, the hand-tuned MPC has a median of 14.2 cm, which is larger than the furthest TD3 outlier and 525.93% worse than its median. Whereas the work of the human expert delivers a maximum lateral



(a) lateral error boxplot



(b) lateral MSE



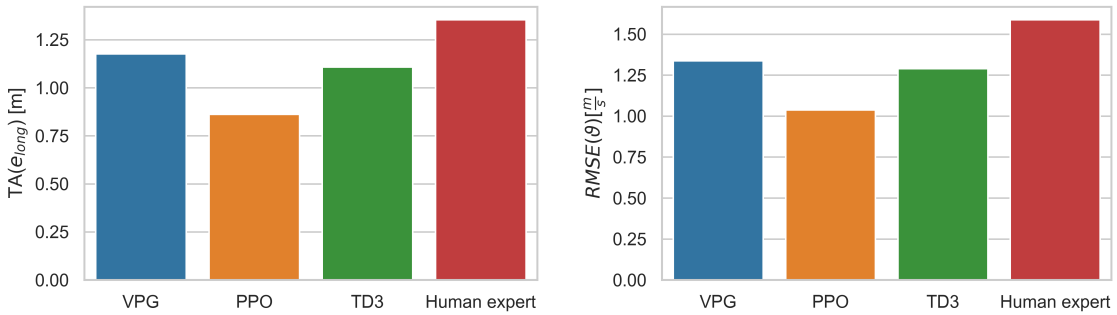
(c) PEL lateral error

Figure 5.2.: comparison of different used RL algorithm models and the human expert parameter set

error $e_{lat} = 50\,\text{cm}$, our concept with TD3 delivers a **625%** better performance with only 8 cm maximum lateral error and an outlier: 12 cm.We deduce similar observations when considering the MSE (Fig. 5.2b) or the PEL (Fig. 5.2c). TD3 has the lowest MSE. Regarding the PEL, both VPG and TD3 never exceed the limit value- 20 cm. But, the human expert and PPO are respectively 33.33% and 6.66% of the time above it. The VPG agent has a comparable performance to TD3 and both outperform the PPO agent. Altogether, the 3 RL agents surpass the manually-tuned MPC.

5.2.3. Longitudinal behaviour

Figure 5.3a delineates the time average (TA) of the absolute longitudinal error of different agents and the human expert. PPO provides the less error (0.86 m), which is 157% better than the human expert (1.35 m). Likewise, the TD3 agent delivers with 1.1 m a 122% better performance. VPG is at third place with 1.19 m.
PPO achieves less velocity errors, because the reward function configuration enforces it. With a deviation $\sigma_v = 5\,\text{m}\,\text{s}^{-1}$, the rewards for the velocity error e_{vel} are more concentrated around 0 than with $\sigma_v = 10\,\text{m}\,\text{s}^{-1}$. The metric RMSE, presented in Fig. 5.3b, infers to the same comparison. Overall, the RL agents outperform the human expert. The TD3 is the best w.r.t. the lateral- as well as longitudinal behaviour.

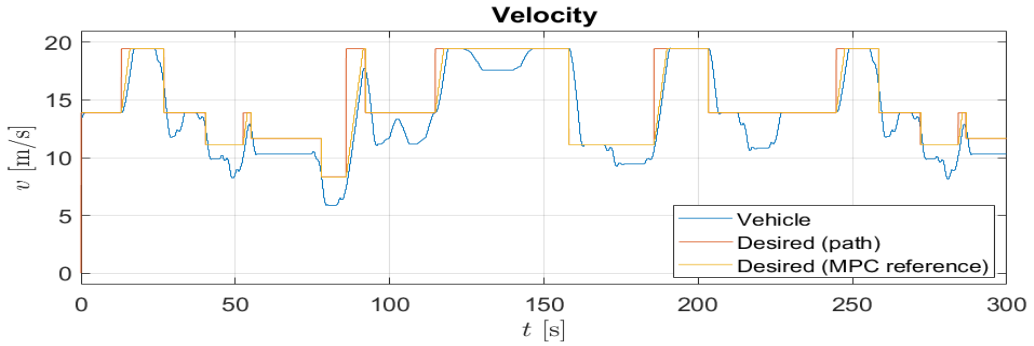


(a) TA longitudinal error

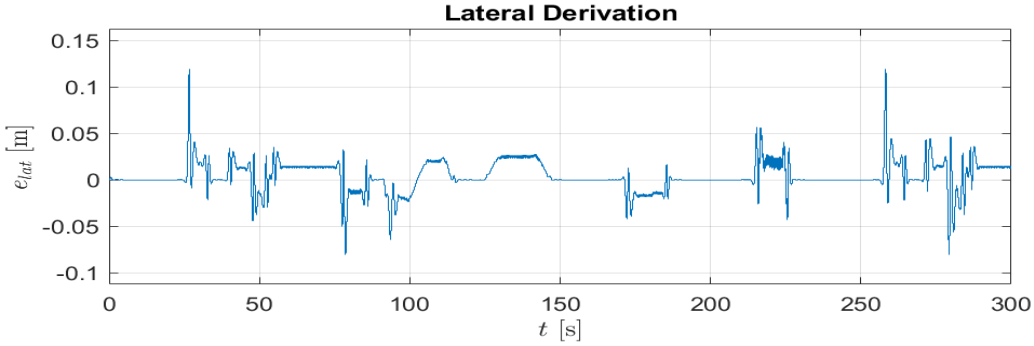
(b) velocity RMSE

Figure 5.3.: comparison of different used RL algorithm models and the human expert parameter set

We present the time evolution of the evaluated velocity (Fig. 5.4a) and the lateral error (Fig. 5.4b) of the TD3-agent. The velocity reference is mostly in range [28 km h^{-1},70 km h^{-1}]. Driving below the reference velocity is not as safety critical as above it. Our TD3 agent does not exceed the reference velocity. It yields a maximum lateral error of 8 cm with high velocities 70 km h^{-1} through urban curves. Thus, the TD3 agent delivers a good performance regarding path-tracking.



(a) velocity evolution



(b) lateral deviation evolution

Figure 5.4.: time evolutions

5.2.4. Policy output

Figures 5.5, 5.6 and 5.7 show the behaviour of respectively VPG, PPO and TD3 policies during the evaluation simulation. The output of the policy network has 6 dimensions and represent the MPC cost function weights. According to our action space definition in Section 4.3, the agent generates actions in range of $[a_{min}, 1]$, where $a_{min} = 10^{-4}$. As discussed in section 4.1, the agent introduces new weights to the PMPC every switching time $T_{sw} = 10\,\text{s}$. We assumed that $q_x = q_y$ for the weights of x-, and y-errors. q_ψ and q_a denote respectively the weights of the yaw error and the lateral acceleration. r_a, r_ω and r_ϑ designate respectively the longitudinal reference acceleration, the steering wheel target angular velocity and the longitudinal velocity error weights.
We notice that the policy of the different algorithms show different behaviour. Yet, they deliver comparable performances, as discussed above. Whereas the TD3 policy tend to generate constant values for the weights apart from r_ω, VPG and PPO constantly adapt their actions. Notably, the VPG agent updates the weights with small changes and not large leaps. The values of each weight seem to be distributed around a mean value. Conversely, the PPO agent is more aggressive and makes big steps.
The TD3 policy is more stable and smooth. For the tracking optimised mode, it collects the maximum episodic cumulative rewards while putting less relative weight on the lateral acceleration than on the x-, y-, yaw angle-, velocity errors and longitudinal acceleration. However, the steering wheel angular velocity weight is varied over time.
We notice that all 3 algorithms reach a good performance by keeping higher relative weights on the x-, y- and velocity errors than on the lateral acceleration weight. Moreover, they tend to generate comparable weights for the x-, y- and yaw angle errors.

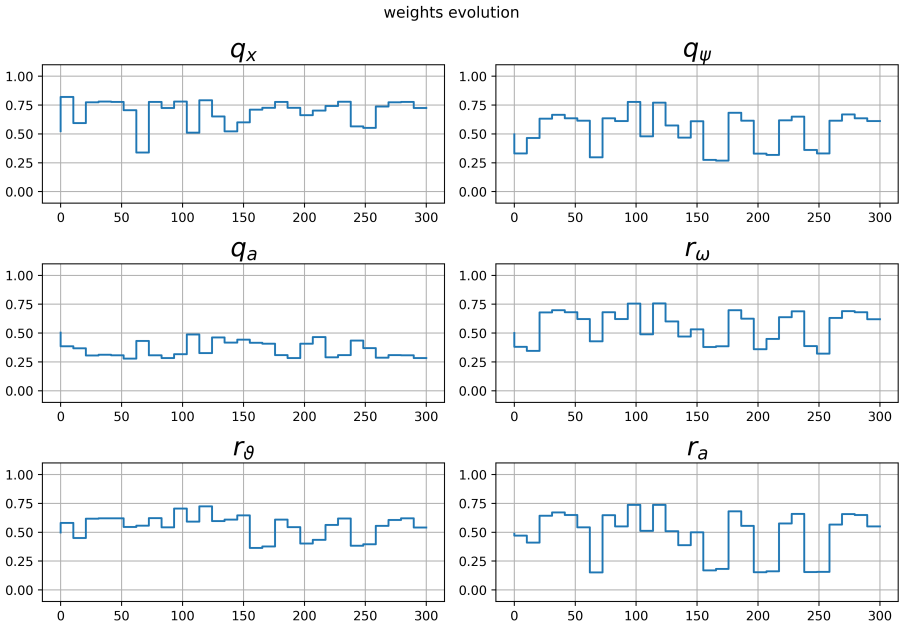


Figure 5.5.: VPG weights evolution: the x-axis denotes the simulation time [s] and the y-axis the weight

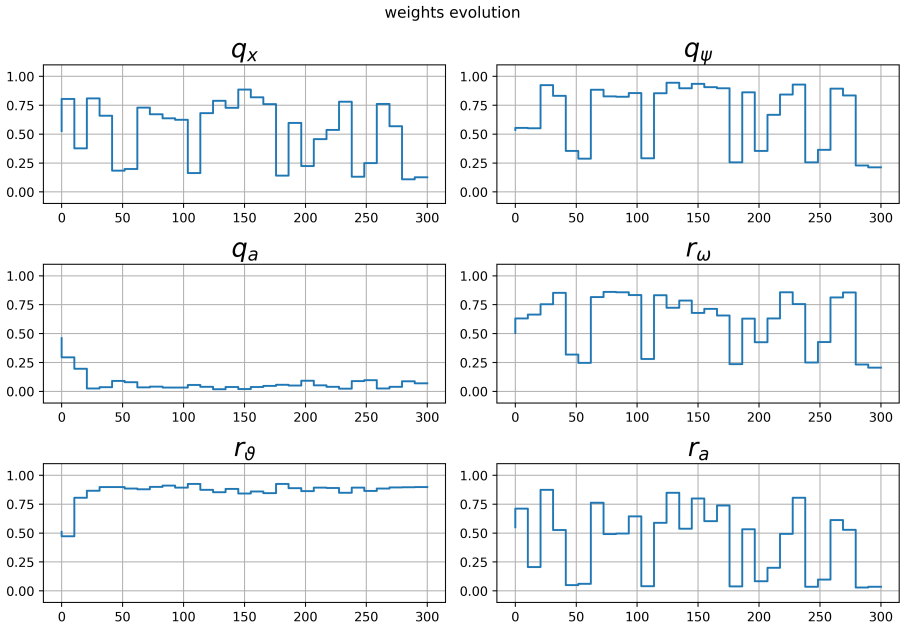


Figure 5.6.: PPO weights evolution: the x-axis denotes the simulation time [s] and the y-axis the weight

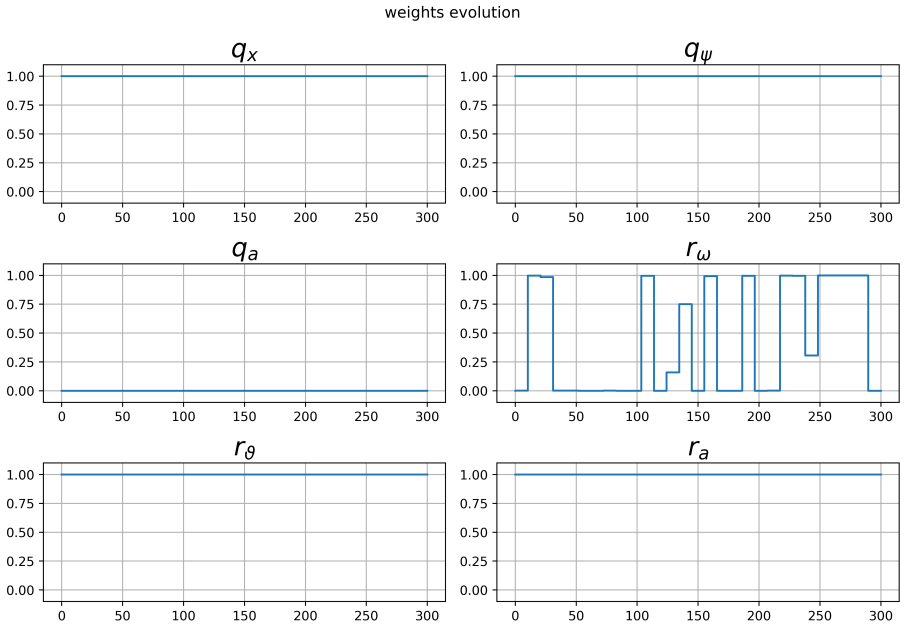


Figure 5.7.: TD3 weights evolution: the x-axis denotes the simulation time [s] and the y-axis the weight. The lowest weight value reached is 1e-4

5.2.5. constraints satisfaction

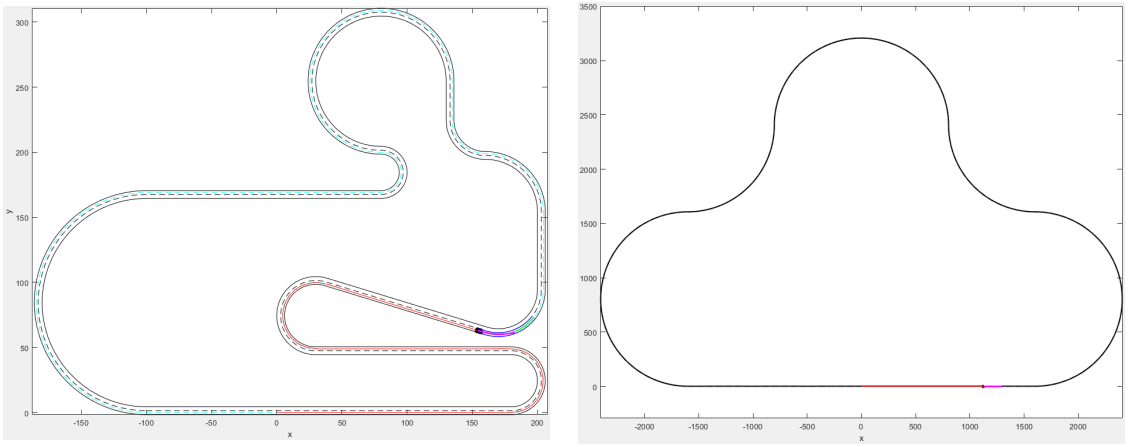
Our resulting concept still holds the requirement of constraints satisfaction introduced in Equation (3.1). The MPC longitudinal reference acceleration a_{ref}, depicted in Appendix B.1, is always in range $[-3\,\mathrm{m\,s^{-2}}, 2\,\mathrm{m\,s^{-2}}]$. The steering wheel reference angular velocity $\omega_{s,ref}$, displayed in Fig. B.2, is always in range $[-250\,^\circ\,\mathrm{s^{-1}}, 250\,^\circ\,\mathrm{s^{-1}}]$. Furthermore the steering wheel reference angle $\delta_{s,ref}$ constraints are met (Fig. B.3). It is straightforward to see that the velocity constraints are met as well. In Fig. 5.4a, the vehicle velocity is in range $[0\,\mathrm{m\,s^{-1}}, 20\,\mathrm{m\,s^{-1}}]$, which is below our constraint $60\,\mathrm{m\,s^{-1}}$.

5.3. Generalising and testing robustness

In this section, we test our learned agent in new unexperienced situations and evaluate the robustness of our concept.

5.3.1. Handling different driving environments

Our agent learned to drive only in one environment, i.e., the urban track presented in Fig. 3.6a in Chapter 3. To validate that our agent is not over-fitting to the training track, we evaluate the trained model on other tracks, e.g., another urban- and a highway scenario (Fig.5.8). We do not let the agent continue learning the new environment. We only run a test simulation in the new situation.



(a) second urban simulation track

(b) highway simulation track

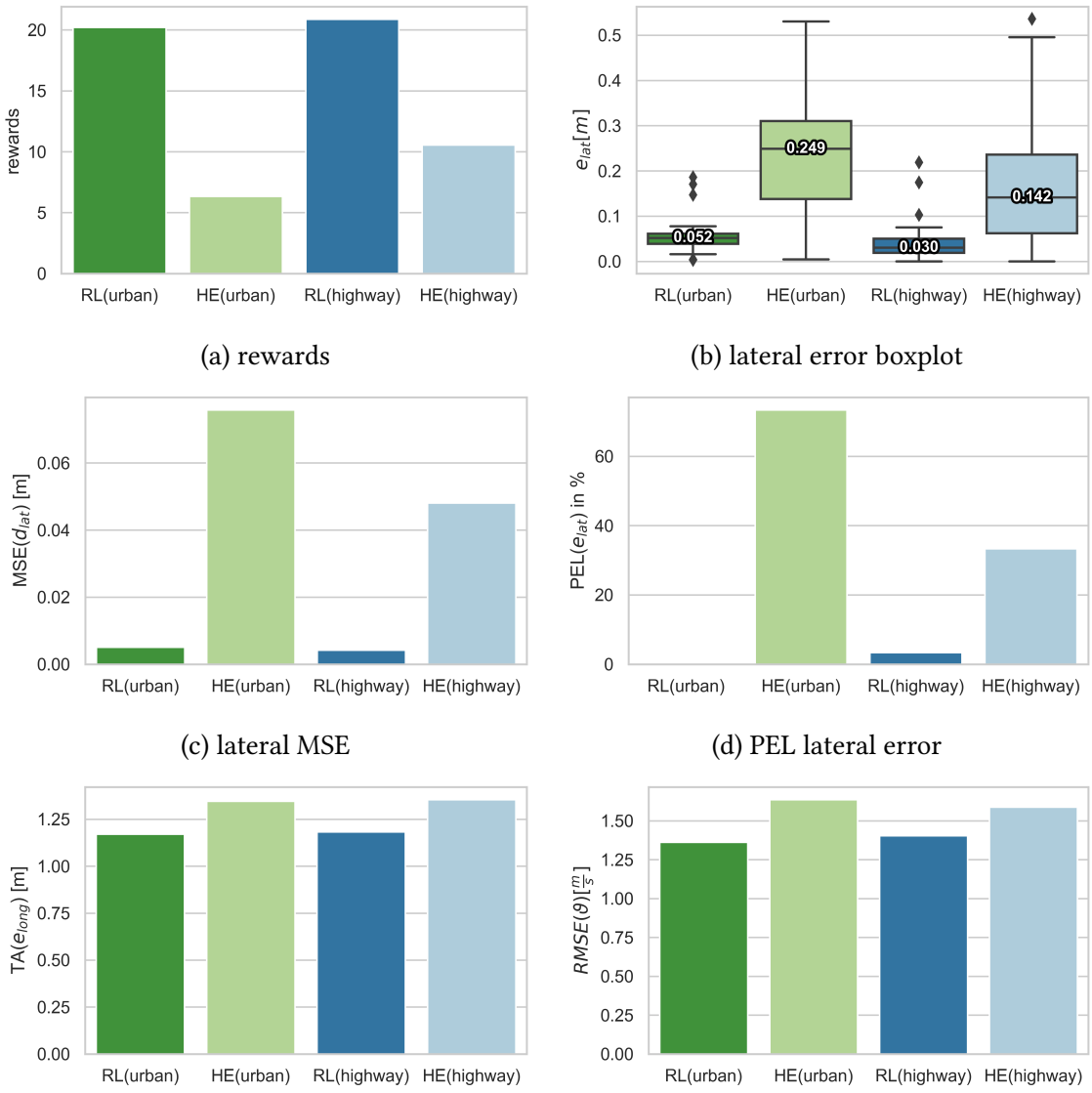
Figure 5.8.: alternative simulation tracks

5.3.1.1. Second urban scenario

The second urban track (Fig. 5.8a) resembles to the first one (Fig. 3.6a). It has the same north-east sector but the rest of the track is different.
According to the strict reward function for evaluation, the RL agent collects 20.22 from 30 possible rewards, i.e., 67.4% accuracy (Fig. 5.9a). The RL model looses 9.16% of its performance compared to the rewards collected in the training environment. In contrast, the manually-tuned MPC collects only 6.34 rewards and loses 40% of its performance compared to the training environment. Thus, when introduced to a new environment, our concept performs **319% better** than the human expert w.r.t. the path-tracking objectives formulated in the reward function.
Figure 5.9b delineates a box plot of the lateral error variation during the new evaluation run. The RL agent has a lower median: 5.2 cm. In contrast, the hand-tuned MPC has a median of 24.9 cm, which is larger than the furthest agent's outlier and 479% worse than its median. Whereas the work of the human expert delivers a maximum lateral error $e_{lat} = 53$ cm, our concept delivers a 662% better performance with only 8 cm maximum deviation. Thus, our agent delivers the same performance regarding the lateral behaviour as in the training environment.
We deduce similar observations when considering the MSE (Fig. 5.9c) or the PEL (Fig. 5.9d). Altogether, the RL agent surpasses the manually-tuned MPC. Thus, driving in a new urban environment produces a comparable performance to the training environment.
Figure 5.9b delineates the TA of the absolute longitudinal error of the RL agent and the human expert. The RL policy provides the least error (1.17 m), which is 115% better than the human expert (1.35 m).
Overall, the RL agent outperforms the human expert w.r.t. the lateral- as well as longitudinal behaviour.

5.3.1.2. Highway scenario

The highway scenario (Fig. 5.8b) has a reference velocity range $[100\,\mathrm{km\,h^{-1}}, 130\,\mathrm{km\,h^{-1}}]$ and its curvatures respect the official highway construction requirements [8].
The results presented in Fig. 5.9 are comparable to the analysis above (5.3.1.1). On the highway, the RL agent delivers a 7.6% degraded performance than on the training track. Yet, it is 198% better than the human expert.
Altogether, the trained RL agent delivers a good performance in other tracks. Thus, it is robust to environment change.
To have even better results in future work, we can keep the same experiment configuration and let the agent continue training in the new environment.



(a) rewards

(b) lateral error boxplot

(c) lateral MSE

(d) PEL lateral error

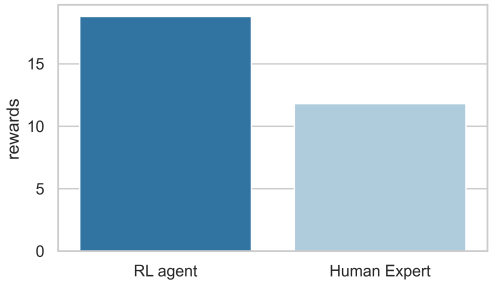
(e) TA longitudinal error

(f) velocity RMSE

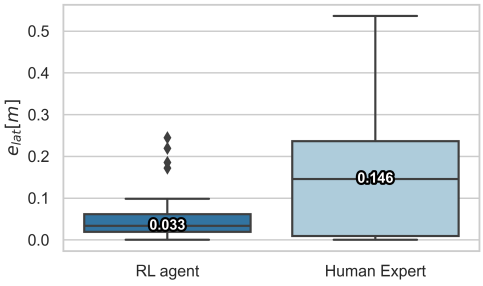
Figure 5.9.: comparison of RL algorithm model(RL) and the human expert (HE) parameter set when handling different driving environments than in training: green represents another urban environment and blue a highway track

5.3.2. Handling prediction model mismatch

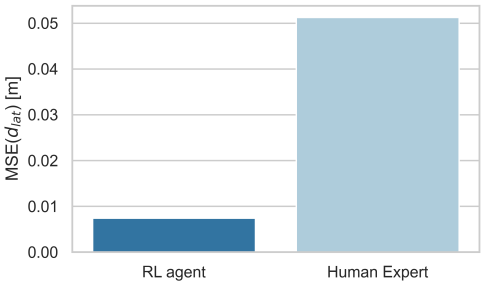
In this section, we evaluate our RL agent when the MPC is exposed to an error-prone prediction model, i.e., the real vehicle model and the model used by the MPC do not match. This is motivated by a variety of use-cases in the daily driving conditions. For instance, when the weather condition changes, e.g., rain or wind, the vehicle model's time constants change. Moreover, this can occur when the vehicle drives with an additional load, which is not considered in the MPC prediction model, e.g., more passengers and luggage. We simulate the latter case. The vehicle drives with an additional 300 kg, i.e., 20% more than its original weight. We run a simulation with the original trained RL agent. Figure 5.10 depicts the comparison between the former and the human expert handling the same situation.
The RL agent collects 18.79 from 30 possible rewards, i.e., 62.63% accuracy (Fig. 5.10a). The model looses **15.3%** of its performance compared to the rewards collected with the correct model. In contrast, the manually-tuned MPC collects only 11.82 rewards, which is 111.93% better than with the correct model. Altogether, with a prediction model mismatch, our concept performs **159% better** than the human expert w.r.t. the path-tracking objectives formulated in the reward function.
Figure 5.10b delineates a box plot of the lateral error variation during the new evaluation run. The RL agent has a lower median: 3.3 cm. In contrast, the hand-tuned MPC has a median of 14.6 cm, which is 442% worse than the RL agent. Whereas the work of the human expert delivers a maximum lateral error $e_{lat} = 53$ cm, our concept delivers a 530% better performance with only 10 cm maximum deviation and outliers with a maximum of 27 cm.
We deduce similar observations when considering the MSE (Fig. 5.10c) or the PEL (Fig. 5.10d). Altogether, the RL agent surpasses the manually-tuned MPC.
Figure 5.10e delineates the time average (TA) of the absolute longitudinal error of the RL agent and the human expert. Both deliver equivalent performances.
Overall, the RL agent outperforms the human expert w.r.t. the lateral- as well as longitudinal behaviour. The trained RL agent still delivers an acceptable performance even with an error-prone MPC prediction model. Thus, it is robust to prediction model mismatch.



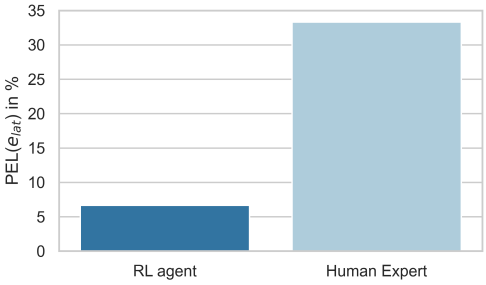
(a) rewards



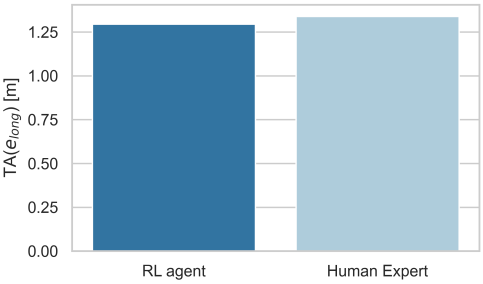
(b) lateral error boxplot



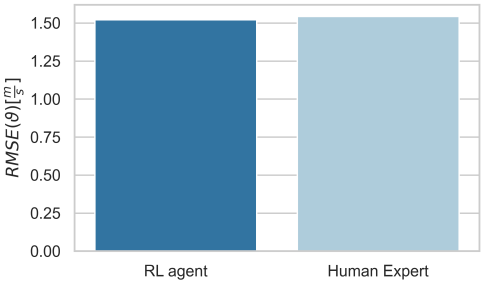
(c) lateral MSE



(d) PEL lateral error



(e) TA longitudinal error



(f) velocity RMSE

Figure 5.10.: comparison of RL algorithm model and the human expert parameter set handling model mismatch

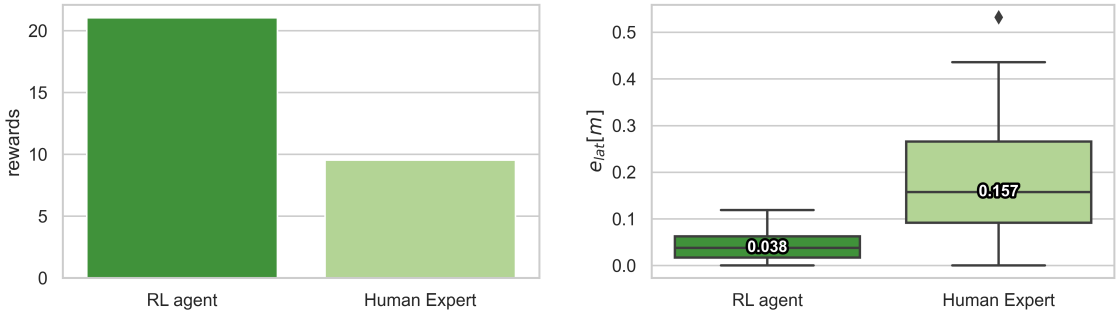
5.3.3. Handling different vehicles

Our RL agent learned the MPC cost function parameters with only one vehicle: VW Passat Variant as depicted in Fig.3.7. In this section, we change the vehicle, but still apply the same old policy neural network. In this evaluation test, we use an electric vehicle: Volkswagen e-Golf as in Fig.5.11. Thus, we evaluate our policy with a different vehicle.

Figure 5.11.: IAV Volkswagen e-Golf test vehicle

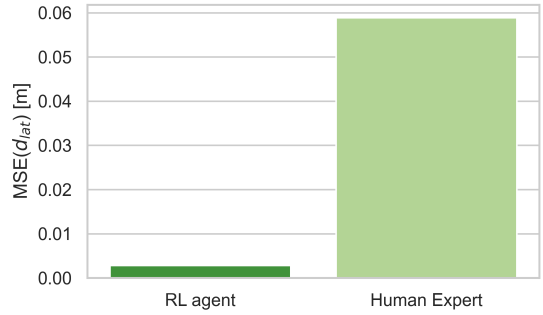
Figure 5.12 depicts the results of evaluating with an e-Golf the model trained with a Passat. The old policy delivers an equivalent performance as with the training vehicle. The RL agent collects 21.04 from 30 possible rewards, i.e., 70.13% of accuracy (Fig. 5.12a). The agent looses only **5.48%** of its performance compared to the vehicle used in training. In contrast, the manually-tuned MPC collects only 9.54 rewards and so a 9.66% loss of its performance compared to the score with VW Passat. Thus, with a different evaluation vehicle, our concept performs **220% better** than the human expert w.r.t. the path-tracking objectives formulated in the reward function.
Figure 5.12b delineates a box plot of the lateral error variation during the new evaluation run. The RL agent has the least median: 3.8 cm. In contrast, the hand-tuned MPC has a median of 15.7 cm, which is 413% worse than the RL agent. Whereas the work of the human expert delivers a maximum lateral error $e_{lat} = 43$ cm, our concept delivers a **358%** better performance with only 12 cm maximum deviation.
We deduce similar observations when considering the MSE (Fig. 5.12c) or the PEL (Fig. 5.12d). Altogether, the RL agent surpasses the manual-tuned MPC. Thus, driving with a different vehicle produces a comparable performance to training and it is still safe regarding the small lateral errors.
Figure 5.12b delineates the TA of the absolute longitudinal error of the RL agent and the human expert. Both deliver equivalent performances with a slight advantage to the RL agent, as he produces 15.53% less error that the manual tuned MPC.
Overall, the RL agent outperforms the human expert w.r.t. the lateral- as well as longitudinal behaviour. The trained RL agent delivers a good performance even with a different vehicle than in training. Thus, it is robust to vehicle change.
To have better results, we can take advantage of the automation process of learning in our

concept. We run the same training experiment configuration but with the new vehicle. We let the agent train autonomously for the specific vehicle and get optimal results. Conversely, without automation, an MPC expert is needed to manually tune the MPC from scratch and to adapt it to the new changes.
Following the analysis in this section, the trained RL agent is robust to other driving environments, MPC prediction model mismatch and to driving with a different vehicle than during the learning process. Thus, the learned model is more general. It can be applied to other conditions than in the training process and still delivers comparable results.

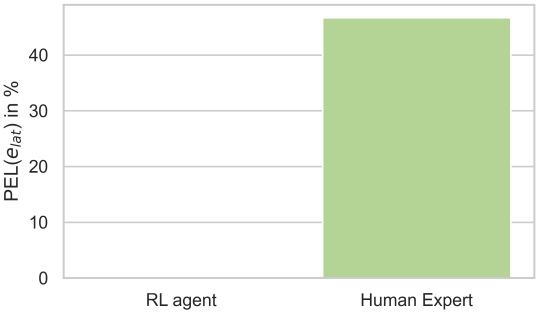


(a) rewards

(b) lateral error boxplot

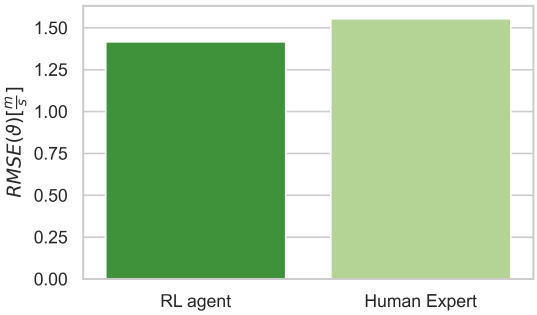


(c) lateral MSE



(d) PEL lateral error

(e) TA longitudinal error



(f) velocity RMSE

Figure 5.12.: comparison of RL algorithm model(RL) and the human expert (HE) parameter set driving different vehicles as in training: green represents vehicle 2 and blue vehicle 3

5.4. Comfort mode

The comfort mode is a multi-objective optimisation problem that includes 4 goals. We want to minimise the longitudinal jerk j_{long}, the lateral acceleration a_{lat}, the lateral error e_{lat} as well as the velocity error e_{vel}. Thus, we have two additional optimisation goals than the tracking-optimised mode.
The advantage of our approach is automation. The agent learns on his own and does not need a supervision. Since we already developed a performing approach, we adjust our reward function to the new objectives and let the agent train again from scratch. Additionally, we adjust the scaling weights to meet our new requirements (Section4.4). Conversely, manually tuning requires an MPC expert that has to start tuning from scratch and try to meet the new objectives.
We train two agents: a VPG with the novel multi-objective Gaussian (MOG) reward function (as in Equation (4.5)) and a TD3 agent with the novel multi-objective cascaded Gaussian (MOCG) reward function (as in Equation (4.7)). The choice of the configuration is just an example and there is no background on associating VPG with MOG and TD3 with MOCG. The deviations σ_v, σ_l, σ_a and σ_j of the velocity- e_{vel}, the lateral error e_{lat}, the lateral acceleration a_{lat} and the longitudinal jerk j_{long} were configured as presented in Table 5.3:

Table 5.3.: reward function configuration for training the VPG, TD3 agents and the evaluation run

	VPG (MOG)	TD3 (MOCG)	evaluation
$\sigma_v[\mathrm{m\,s^{-1}}]$	15	7	-
$\sigma_l[\mathrm{m}]$	1	0.5	-
$\sigma_a[\mathrm{m\,s^{-2}}]$	1.25	1	1
$\sigma_j[\mathrm{m\,s^{-3}}]$	2	1	1

For the comfort mode, the velocity error is not the most relevant factor, but the vehicle should not be driving faster than the reference velocity. The lateral error has less impact, but the vehicle should not leave the lane. Thus, for the evaluation, we use a bi-objective Gaussian reward function (Equation (5.4)) that considers only the longitudinal jerk and the lateral acceleration

$$R_{evaluation}(j_{long}, a_{lat}) = A \exp\left(-\left(\frac{j_{long}^2}{2\sigma_j^2} + \frac{a_{lat}^2}{2\sigma_a^2}\right)\right) \tag{5.4}$$

5.4.1. Rewards

The maximum reward per interaction is 1. It is achieved, when both jerk and lateral acceleration are optimal. The TD3 agent collects 19.64 from 30 possible rewards, i.e., 65.64% accuracy and 34.53% loss (Fig. 5.13). The TD3 agent with MOCG outperforms the VPG with MOG reward function. In contrast, the manually-tuned MPC collects only 9 rewards. Thus, our concept performs **218% better** than the human expert w.r.t. the

comfort objectives formulated in the reward function. We recall that there is only one parameter-set tuned by the human expert for both tracking and comfort.

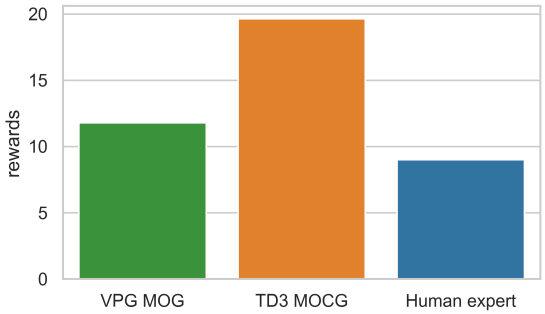


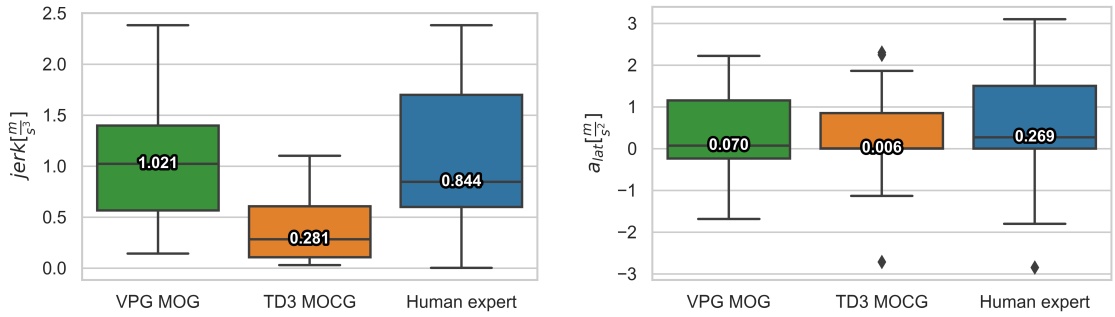
Figure 5.13.: cumulative evaluation rewards of the VPG and TD3 RL algorithms using MOG and MOCG reward functions and the human expert parameter set

5.4.2. Comfort behaviour

Concerning the jerk presented in Fig. 5.14a, the VPG agent and the human expert deliver a comparable behaviour. However, the latter has a smaller median: $0.844\,\mathrm{m\,s^{-3}}$. On the other side, the VPG has a smaller Interquartile Range (IQR). Overall, the TD3 with MOCG reward function is the most jerk optimal. The median is with $0.281\,\mathrm{m\,s^{-3}}$ **300%** better than the human expert. The upper quartile (Q3/ 75th percentile) of TD3 is levels with the lower quartile (Q1/ 25th percentile) of the VPG and the manually-tuned MPC. Thus, the jerk caused by the TD3 is 75% of the time under $0.6\,\mathrm{m\,s^{-3}}$, which is less than the median jerk caused by the hand-tuned MPC.
Likewise, the TD3 with MOCG outperforms the two others when considering the lateral acceleration (Fig. 5.14b). Its median is the smallest: $0.006\,\mathrm{m\,s^{-2}}$ (**4483%** better than the human expert), has the smallest IQR as well as the smallest extreme values.
Thus, the TD3 delivers the best performance regarding both longitudinal- as well as lateral comfort behaviour.

5.4.3. Lateral and longitudinal tracking behaviour

Figure 5.15a delineates a box plot of the lateral error variation during the evaluation run. The TD3 agent has the lowest median: 3.3 cm. In contrast, the MPC expert's weights has a median of 15.7 cm, which is larger than the furthest TD3 maximum and 475% worse than its median. Whereas the work of the human expert delivers a maximum lateral error $e_{lat} = 55\,\mathrm{cm}$, our concept with TD3 delivers a **550%** better performance with only 10.5 cm maximum and an outlier- 33 cm.
Regarding the PEL, the human expert and TD3 are respectively 43.33% and 3.33% of the time above it. Thus, the TD3 delivers the best lateral tracking performance.
Figure 5.15c delineates the velocity RMSE. Since both VPG and TD3 optimised their jerk

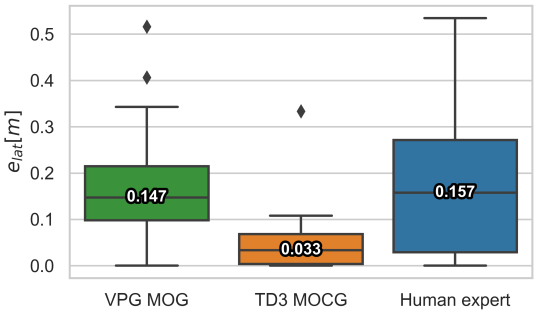


(a) jerk

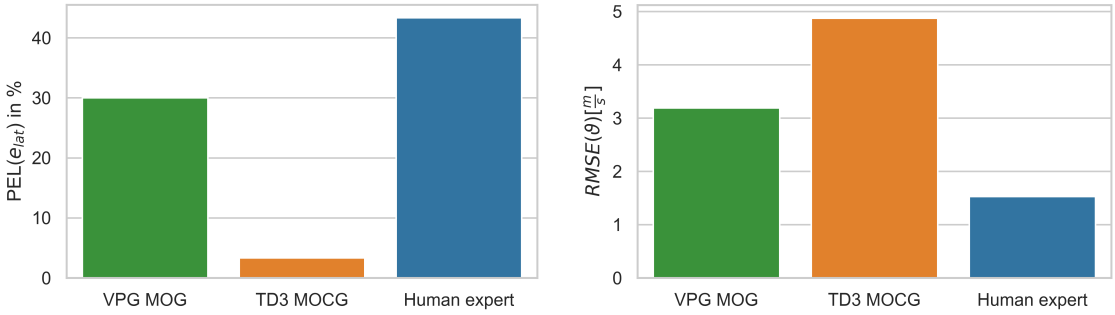
(b) lateral acceleration

Figure 5.14.: comparison of the VPG and TD3 RL algorithm models using MOG and MOCG reward functions and the human expert parameter set

and lateral accelerations, they show accordingly a poor performance compared to the hand-tuned MPC regarding the velocity error. The manually-tuned MPC produces respectively 208% and 319% less velocity errors than VPG and TD3. However, the velocity is not the most relevant factor for the comfort mode.
Overall, the MOG- and MOCG reward functions yield to a better behaviour of the system for the comfort mode.



(a) lateral error boxplot



(b) PEL lateral error

(c) Velocity RMSE

Figure 5.15.: comparison of the VPG and TD3 RL algorithm models using MOG and MOCG reward functions and the human expert parameter set

5.4.4. Policy output

Figure 5.16 depicts the behaviour of respectively the comfort-MOCG-TD3 policy and the tracking-optimised-TD3 policy during the evaluation run. We analyse the MLP neural network output similar to the tracking-optimised mode evaluation in Section 5.2.4. We recall that $a_{min} = 10^{-4}$, $T_{sw} = 10\,\text{s}$ and $q_x = q_y$.
Although the same RL algorithm is used, the policies demonstrate distinctive behaviours. In fact, the tracking-optimised policy is more stable and smoother than the comfort agent. Notably, more optimisation goals were formulated for the latter.
We notice that the extreme differences occur to the lateral acceleration weight r_a. However, this was expected, since we are primarily optimising the lateral acceleration for the comfort-, but we were neglecting it for the tracking mode. Due to the insignificance of the lateral acceleration, the tracking agent was constantly choosing the least weight possible $q_{a,tracking} = a_{min} = 10^{-4}$. Conversely, the comfort agent aims for reducing the lateral acceleration. Hence, the latter chooses almost always the maximal possible relative penalty $q_{a,comfort} = a_{max} = 1$. We recall that the output activation function is a Sigmoid that returns 1 as a maximum.

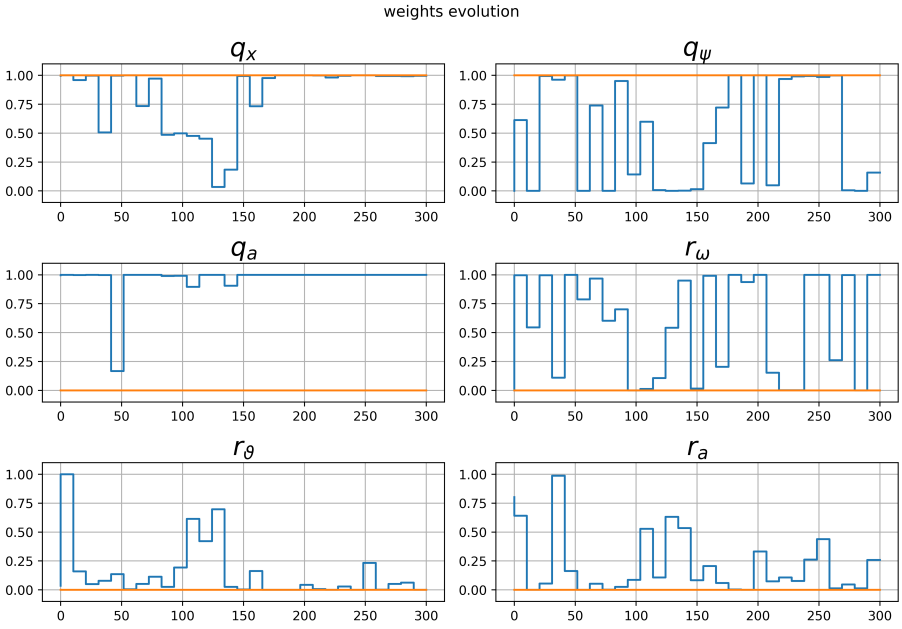


Figure 5.16.: comparison of the TD3 policy outputs for the tracking-optimised mode (orange) and for the comfort mode with MOCG rewards (blue): the x-axis denotes the simulation time [s] and the y-axis the weight

Analysing the rest of the weights is not as obvious as the lateral acceleration weight. Still, comparing the comfort agent to the tracking agent, we notice the following points:

- There is not always a constant big penalty on the x-, y- (q_x,q_y) and yaw (q_ψ) errors, as the path-tracking is not a priority anymore.

- There is not always a constant small penalty on the steering wheel angular velocity (r_ω), as high angular velocities lead to uncomfortable behaviour.

We recall that the agent did learn all these connections and differences on his own based on our formulation of the reward function in Equation (4.3) and Equation (4.7).

5.5. Evaluation conclusion

In this chapter, we demonstrated that our applied concept outperforms the work of an MPC human expert w.r.t. 2 different objective sets: comfort- and tracking-optimised mode. Our approach is easy to reuse and to adapt to different goals and configurations.
The RL neural network policy is robust and can handle new unexperienced conditions, e.g., different environments or an error-prone MPC prediction model. It still delivers a good performance. Table 5.4 and Table 5.5 summarise the results discussed in the above sections for the tracking- and for the comfort mode, where "max abs" denotes the maximum absolute value, "75th percentile" the value below which 75% of the observations may be found, PEL the percentage of exceeding the limit value of 20 cm, TA the time average of the longitudinal error and RMSE the root-mean-square error.
Whereas a hand tuned MPC delivers a single parameter set, our RL driven PMPC takes into account observations collected from the last switching time period T_{sw} and generates a new MPC cost function parameters to adapt the MPC to changes w.r.t. our specified objectives.
As we considered the MPC weights requirements and discussed the choice of the switching time T_{sw}, the MPC problem is recursively feasible and stable. The constraints are always met.
Thanks to the automation of our approach, training a new agent is handy, e.g., for a different vehicle. Similarly, continuous learning is possible, e.g., when in a new environment. The last points were not in the scope of our work but we made it possible to extend it. For further intensive testing, the agent can be evaluated in further urban scenarios, e.g., parking scenario.

Table 5.4.: results summary for the tracking-optimised mode

	metric	RL agent (training)	Human expert	RL (different track)	RL (model mismatch)	RL (different vehicle)
general	rewards	22.26	10.56	20.87	18.79	21.04
lateral behaviour[m]	max abs error	0.08	0.5	0.075	0.1	0.12
	median	0.027	0.142	0.03	0.033	0.038
	75th percentile	0.055	0.235	0.05	0.06	0.062
	PEL	0%	33.33%	3.33%	6.66%	0%
	MSE	0.0025	0.048	0.0042	0.0074	0.0028
longitudinal behaviour	TA [m]	1.108	1.354	1.183	1.295	1.183
	RMSE [m s^{-1}]	1.289	1.588	1.404	1.522	1.417

Table 5.5.: results summary for the comfort mode

	metric	VPG + MOC	TD3 + MOCG	Human expert
general	rewards	11.8	19.64	9
lateral acceleration [$m\,s^{-2}$]	max abs	2.3	1.9	3.1
	median	0.07	0.006	0.269
	75th percentile	1.2	0.9	1.6
longitudinal jerk [$m\,s^{-3}$]	max abs	2.45	1.1	2.45
	median	1.021	0.281	0.844
	75th percentile	1.5	0.6	1.7
lateral error [m]	max abs error	0.34	0.1	0.55
	median	0.147	0.033	0.157
	75th percentile	0.215	0.065	0.28
	PEL	30%	3.33%	43.33%
velocity error [$m\,s^{-1}$]	RMSE	3.19	4.87	1.53

6. Conclusion

This master thesis is our contribution towards the automation of control design processes and the performance improvement of autonomous vehicle guidance systems. In the process, we expand the Model Predictive Control (MPC) theory with the notion of varying its cost function parameters following a fixed schedule in real-time. We propose a novel Parameter-Varying MPC (PMPC). Rather than the time costly and demanding hand tuning by an MPC expert, we advocate a deep neural network policy to determine the PMPC parameters, save design effort and refine the control quality.

We let the PMPC policy learn autonomously with state-of-the-art Reinforcement Learning (RL) algorithms based on objectives we specify in a reward function. Thus, we conceive a hybrid cascaded control: the MPC controls the vehicle while the RL agent controls the MPC cost function parameters to compensate for changes in its environment.

The RL world lacks general formulations to express optimisation goals in a reward function. The majority of the functions are environment specific. Eventually, the MPC optimises only the objectives formulated in its cost function that has a limited structure combined with many degrees of freedom. To enhance the RL problems and to reduce the workload of the MPC, we propose a novel formulation of a general reward function: the multi-objective cascaded Gaussian (MOCG). Our formulation leverages multiple physical signals and pushes the policy behaviour to converge to their respective desired objective values. Thus, it is reusable for other RL environments. Besides, our MOCG can consider optimisation goals that are not formulated in the MPC problem. Hence, it yields more optimisation objectives with less degrees of freedom.

We demonstrate that our concept can be applied on high-dimensional, complex control problems, specifically, the autonomous vehicle guidance use-case. We illustrate this on realistic simulated urban environments with models based on real vehicles such as Volkswagen Passat. Without the need of any supervision, our approach outperforms an MPC tuned by an expert for a tracking-optimised mode (625% better regarding the maximum lateral deviation and 210% better generally). Experimental results demonstrate that our trained deep neural network driven PMPC is robust and not overfitting to the training conditions. In fact, our concept still delivers a good performance when confronted with unexperienced situations such as different driving environments, prediction model mismatch and controlling a different vehicle.

Besides the outstanding performance of our learned PMPC parameters, our concept proved to be handy and easy to adapt when the optimisation goals ought to be changed. We illustrate this with learning a second driving style: comfort mode. Whereas with hand

tuning an MPC expert is needed to manually tune the MPC from scratch, we just adapt the reward function and let our RL agent train autonomously again. Experiments displayed that our deep neural network driven PMPC performs 218% better than the work of the human expert w.r.t. severe evaluation rewards for comfort behaviour.

Our approach showed no side effects on the constraints satisfaction of the MPC. Thus, we met the required objectives of the thesis: safety, performance and automation.

Considering the mass production in the automobile industry, an automobile constructor has different vehicle types and each vehicle type can have different powertrain configurations. For each setting, a new controller configuration is needed. Through the automation of the tuning process of controllers and a finer control quality, our approach promises the optimisation of the design phase for large scale manufacturing. It saves enormous time of manually tuning and lowers the design costs.

Thanks to the general RL-PMPC algorithm that we conceived and the general formulation of the MOCG, our RL driven PMPC approach is generative and reusable for a large scale of systems that require autonomous high performing path-following. These range from high precision microprocessor production, autonomous robotic medical surgeries [51], 3D printers, CNC machines, robots, Autonomous Underwater Vehicles (AUV) [40] to the scale of autonomous control of spacecrafts, e.g., docking [39]. To apply our concept on other systems, you just need to adapt the reward function to the new desired objectives and the number of the actions, i.e., number of the specific MPC cost function weights. We already defined a general action space for RL-PMPC.

Considering the workload in this thesis, implementing such a concept was time costly and optimising the run-time was demanding. Although we have saved weeks of experimenting by designing an architecture capable of parallelisation, the training process takes hours with neither predictable convergence time nor predictable results. We have chosen a limit of 2000 training episodes that takes approximately 12 hours time per experiment. However, we did not need to assist the agent, as it trains autonomously. We hope that future works optimise the MPC environment and the RL algorithms.

6.1. Future work

There are several extension possibilities of our work, ranging from optimising the current results to extending the work with new concepts. To optimise the current results, we suggest several ideas. First, we can extend the learning process to automatically change the driving environments from time to time. Due to the time and resource limitations during this work, the hyperparameters of the RL agents were not exploited. We expect a better performance with a better hyperparameter configuration.

As discussed in Section 4.1, we suggest an intensive theoretical stability and feasibility analysis of the PMPC. The goal is to determine analytically the PMPC switching time T_{sw}. Moreover, we suggest gaining freedom degrees for stability and convergence by choosing a different MPC terminal weight than the stage cost weight.

We hope this thesis inspires other researchers to design a RL algorithm specific to the PMPC problem, in order to accelerate the learning speed, i.e., for an efficient learning. We have experienced that with some hyperparameter- and reward function configurations the state-of-the-art RL algorithms show no learning behaviour, i.e., the rewards do not increase over training time. We propose the idea of considering or integrating the MPC optimal control problem (OCP) in the RL update phase during learning. A first intuition that we can think of could be an iterative solving of the OCP offline and interpreting the MPC predictions as cumulative rewards predictions, i.e., value function. Correspondingly, we would update the policy itself.

Finally, we propose not only to learn Parameter-Varying MPC with RL, but also Time-Varying MPC with RL. The latter changes its prediction model parameters with the time [18]. We suggest determining both MPC cost function- and prediction model parameters with RL. We can call this Parameter- & Time-Varying MPC (PTMPC). This can be realised based on the concept we introduced in this work and can be extended to learn model parameters. There are mainly two possible approaches: using one agent for both parameter sets or using 2 different agents. The second alternative has the advantage and the possibility to specify 2 different optimisation objectives formulations, i.e., 2 different reward functions.

A. Appendix

A.1. Used RL algorithms

A.1.1. Proximal Policy Optimisation algorithm

Algorithm 4 Proximal Policy Optimisation algorithm [9]

initialise policy parameters θ_0 and value function parameters ϕ_0
for k = 0,1,2 ... **do**

1. Collect a set of trajectories $D_k = \{\tau_i\}$ by executing the current policy $\pi_k = \pi(\theta_k)$
2. For each trajectory compute:
   a) The *rewards-to-go* $\hat{R}_t$ for each trajectory
   b) The *advantage estimates* $\hat{A}_t$ based on the current value function V_{ϕ_k}, e.g. using the generalised advantage estimator GAE [56]:

$$\hat{A}_t^{GAE(\gamma,\lambda)} = \sum_{l=0}^{\infty} (\gamma\lambda)^l \, \delta_{t+l}^V$$

   where the TD residual [69] is:

$$\delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t)$$

3. Update the policy, e.g. using gradient ascent:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|D_k|T} \sum_{\tau \in D_k} \sum_{t=0}^{T} \left(\min \left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \hat{A}_t, clip \left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}, 1-\epsilon, 1+\epsilon \right) \hat{A}_t \right) \right)$$

4. Fit the value function to the rewards-to-go via gradient descent:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|D_k|T} \sum_{\tau \in D_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2$$

end for

A.2. Communication between RL and MPC frameworks

In order to exchange messages between the two interfaces, we create a protocol to organise the communication. Through the communication process, we pay attention to data integrity. We introduce 3 types of messages:

- Data message: It transmits either an action or an observation and the terminal state signal.
- Data Length message: It transmits the length of the data message in bytes.
- Reset message: It transmits a reset order.

Before every data message sending process, its length is transmitted. Thus, the receiver knows the length of data it has to read. The messages are transmitted as a string coded ASCII character set: Every message starts with a header that signifies the communication phase, e.g., action sending phase. Also, the header specifies the kind of the message, i.e., sending data-length, the data itself or a reset signal message. Every message ends with a unique special character signifying the end of the message, e.g., ">". If a communication problem occurs, e.g., data length mismatch, the communication stops and so the learning process. The C++ Client is compiled and built with the Matlab Legacy Tool as an S-function in Simulink.

A.3. Optimising RL agents

We base our work on the Spinning Up RL agent of Open AI. First, we optimise their run time. Second, Spinning Up algorithms are based on parallel computation libraries that are not supported by Windows OS. We remove the parallelisation processes and replace them with equivalent libraries. We also increase the computing performance for our specific CPU architecture.
Additionally, we change the neural networks architectures in order to meet our requirements, e.g., requirements w.r.t. the action space discussed in section 4.3. We do not make use of the Spinning Up file-logging method that considers only some RL aspects, e.g., rewards and losses. In fact, we implement a Tensorboard interface to monitor not only RL- but also different closed loop system aspects and metrics, e.g., lateral and velocity error statistics. Thus, we can visualise our specific use-case relevant information.
Moreover, we implement a different model saving method, i.e., parameters of the MLP neural networks for the policy and value functions. During training, we save our model every time we reach a new maximum of cumulative rewards during an episode. However, Spinning Up saves its model at fixed time distance. Consequently, it misses models leading to high rewards.

A.4. Generating experiments

To run experiments, we execute manually the following steps for each computer/experiment instance:

1. choose a RL algorithm, its required neural networks for policy and value functions and its hyperparameters
2. configure a reward function
3. choose communication port on Server and Client sides
4. build the Client S-function
5. build a standalone application
6. run the application in an infinite loop and the RL training process
7. evaluate the experiments

We train our agents on the CPU. In our special settings, the GPU delivered a mediocre performance. Notably, with every agent-environment interaction, data vectors are transferred from CPU to GPU, passed through the MLP neural networks and then the results are transmitted in the opposite direction. Saving the transfer time and computing only on the CPU, delivered faster interactions.

A.5. Used reward functions

In this section, we present reward functions that we designed and did not deliver a good performance. These formulations can be useful for future work and can perform better in other environments. Table A.1 denotes the used symbols in this section.

Table A.1.: symbols used for the reward functions

symbol	meaning
e_{lat}	lateral error
l_{mid}	lateral error preferred limit
l_{nogo}	lateral error nogo limit
e_{vel}	velocity error
v_{mid}	velocity error preferred limit
v_{nogo}	velocity error nogo limit

A.5.1. Discrete rewards

A.5.1.1. Large Rewards

$$R_{lat} = \begin{cases} 10 & if\ e_{lat} \leq l_{mid} \\ -10 & if\ l_{mid} < e_{lat} < l_{nogo} \\ -100 & else \end{cases} \tag{A.1}$$

$$R_{vel} = \begin{cases} 10 & if\ e_{vel} \leq v_{mid} \\ -10 & if\ v_{mid} < e_{vel} < v_{nogo} \\ -100 & else \end{cases} \tag{A.2}$$

$$R_{total} = R_{vel} + R_{lat} \tag{A.3}$$

A.5.1.2. Small Rewards

$$R_{lat} = \begin{cases} 1 & if\ e_{lat} \leq l_{mid} \\ -1 & if\ l_{mid} < e_{lat} < l_{nogo} \\ -2 & else \end{cases} \tag{A.4}$$

$$R_{vel} = \begin{cases} 1 & if\ e_{vel} \leq v_{mid} \\ -1 & if\ v_{mid} < e_{vel} < v_{nogo} \\ -2 & else \end{cases} \tag{A.5}$$

$$R_{total} = R_{vel} + R_{lat} \tag{A.6}$$

A.5.2. Reward temporal difference

A.5.2.1. TD I

$$R(t) = -|(e_{vel}(t) - e_{vel}(t-1)) \cdot (e_{lat}(t) - e_{lat}(t-1))| \tag{A.7}$$

A.5.2.2. TD II

$$R_{lat}(t) = \begin{cases} -1 & if\ e_{lat}(t) - e_{lat}(t-1) > 0 \\ 1 & else \end{cases} \tag{A.8}$$

$$R_{vel}(t) = \begin{cases} -0.75 & if\ e_{vel}(t) - e_{vel}(t-1) > 0 \\ 0.75 & else \end{cases} \tag{A.9}$$

$$R_{total} = R_{vel} + R_{lat} \tag{A.10}$$

A.5.3. Hybrid Rewards

A.5.3.1. Hybrid Rewards I [47]

$$R_{lat} = \begin{cases} 100 & if\ e_{lat} \leq \frac{l_{nogo}}{100}, \\ \frac{l_{nogo}}{e_{lat}} & if\ \frac{l_{nogo}}{100} < e_{lat} < l_{nogo}, \\ -100 & else \end{cases} \tag{A.11}$$

$$R_{vel} = \begin{cases} 100 & if\ e_{vel} \leq \frac{v_{nogo}}{100}, \\ \frac{v_{nogo}}{e_{lat}} & if\ \frac{v_{nogo}}{100} < e_{lat} < v_{nogo}, \\ -100 & else \end{cases} \tag{A.12}$$

$$R_{total} = R_{vel} + R_{lat} \tag{A.13}$$

A.5.3.2. Hybrid Rewards II

$$R_{lat} = \begin{cases} 100 & if\ e_{lat} \leq \frac{l_{nogo}}{100}, \\ \frac{l_{nogo}}{e_{lat}} & else \end{cases} \tag{A.14}$$

$$R_{vel} = \begin{cases} 100 & if\ e_{vel} \leq \frac{v_{nogo}}{100}, \\ \frac{v_{nogo}}{e_{lat}} & else \end{cases} \tag{A.15}$$

$$R_{total} = R_{vel} + R_{lat} \tag{A.16}$$

A.5.4. Continuous rewards

A.5.4.1. Quadratic I

$$R = -\left(e_{vel}^2 + e_{lat}^2\right) \tag{A.17}$$

A.5.4.2. Quadratic II

$$R = -\left(\left(\frac{e_{vel}}{v_{nogo}}^2\right) + \left(\frac{e_{lat}}{l_{nogo}}^2\right)\right) \tag{A.18}$$

B. Figures

B.1. Tracking-optimised mode: time evolution

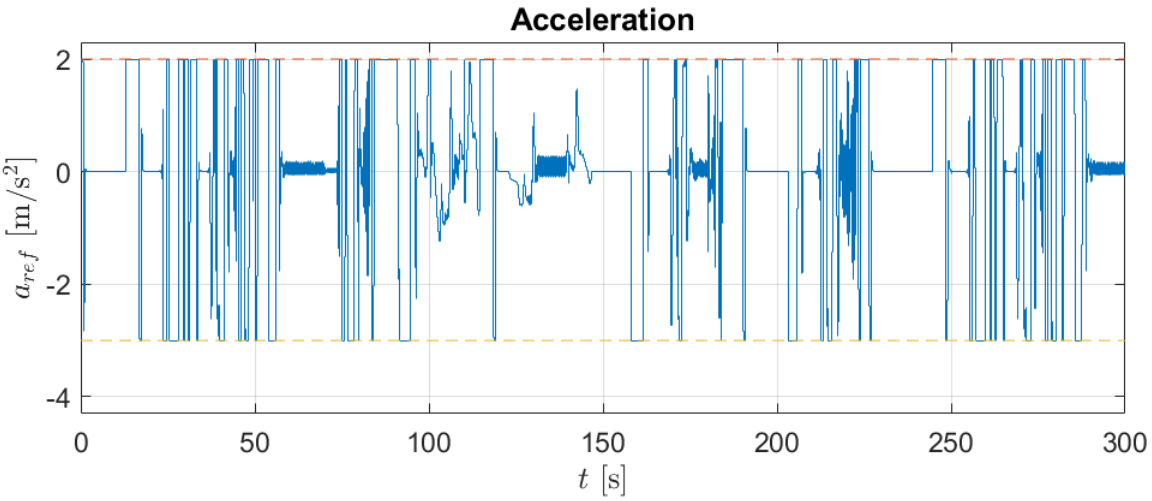


Figure B.1.: longitudinal acceleration evolution and its constraints

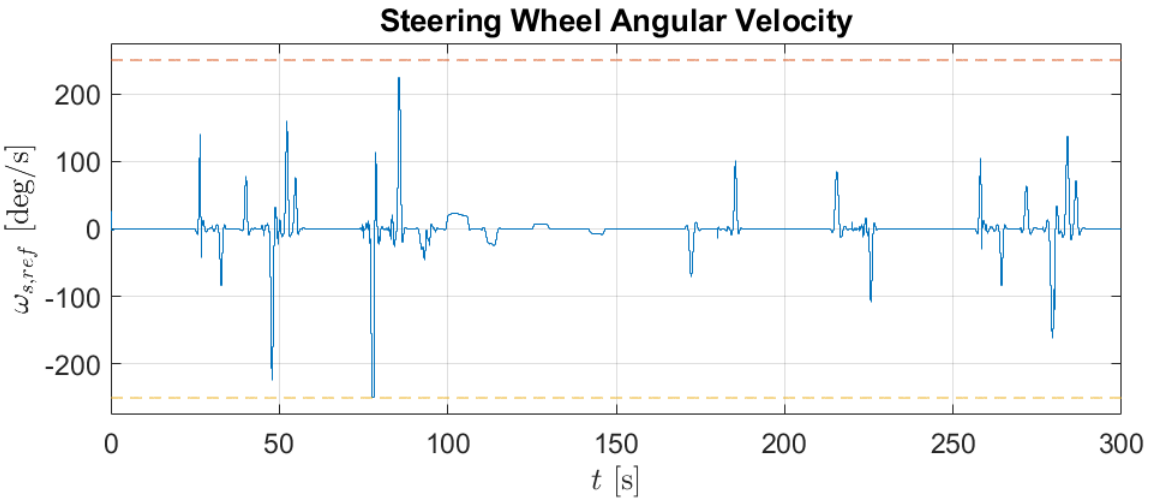


Figure B.2.: steering wheel angular velocity evolution and its constraints

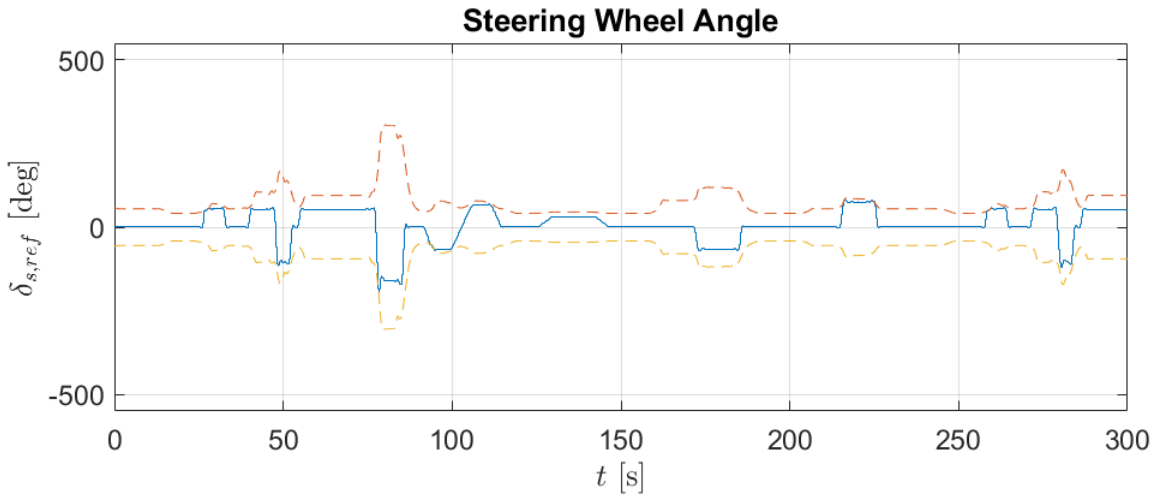


Figure B.3.: steering wheel angle evolution and its constraints

List of Figures

List of Tables

List of Algorithms

Bibliography

[1] Joshua Achiam. “Spinning Up in Deep Reinforcement Learning”. In: (2018).

[2] Himajit Aithal and S Janardhanan. “Trajectory tracking of two wheeled mobile robot using higher order sliding mode control”. In: *2013 International Conference on Control, Computing, Communication and Materials (ICCCCM)*. IEEE. 2013, pp. 1–4.

[3] Kostas Alexis, George Nikolakopoulos, and Anthony Tzes. “Switching model predictive attitude control for a quadrotor helicopter subject to atmospheric disturbances”. In: *Control Engineering Practice* 19.10 (2011), pp. 1195–1207.

[4] A. Amidi and S. Amidi. *Deep Learning Cheatsheet*. `https://github.com/afshinea/stanford-cs-229-machine-learning/blob/master/en/cheatsheet-deep-learning.pdf`. 2018.

[5] Mouna Attarha, James Bigelow, and Michael M Merzenich. “Unintended Consequences of White Noise Therapy for Tinnitus—Otolaryngology’s Cobra Effect: A Review”. In: *JAMA Otolaryngology–Head & Neck Surgery* 144.10 (2018), pp. 938–943.

[6] Christopher M Bishop et al. *Neural networks for pattern recognition*. Oxford university press, 1995.

[7] Jean-Daniel Boissonnat et al. *Algorithmic foundations of robotics v*. Vol. 7. Springer, 2003.

[8] Werner Brilon and RA Krammes. “Die neuen Entwurfsstandards fur Ausserortsstrassen im internationalen Vergleich”. In: *Straßenverkehrstechnik* 41.11 (1997), pp. 529–536.

[9] Greg Brockman et al. “Openai gym”. In: *arXiv preprint arXiv:1606.01540* (2016).

[10] Eduardo Camponogara et al. “Distributed model predictive control”. In: *IEEE control systems magazine* 22.1 (2002), pp. 44–52.

[11] Charles W Clenshaw and Alan R Curtis. “A method for numerical integration on an automatic computer”. In: *Numerische Mathematik* 2.1 (1960), pp. 197–205.

[12] Simon S Cross, Robert F Harrison, and R Lee Kennedy. “Introduction to neural networks”. In: *The Lancet* 346.8982 (1995), pp. 1075–1079.

[13] Ivan Nunes Da Silva et al. “Artificial neural network architectures and training processes”. In: *Artificial neural networks*. Springer, 2017, pp. 21–28.

[14] Gal Dalal et al. “Safe exploration in continuous action spaces”. In: *arXiv preprint arXiv:1801.08757* (2018).

[15] Yan Duan et al. “Benchmarking deep reinforcement learning for continuous control”. In: *International Conference on Machine Learning*. 2016, pp. 1329–1338.

[16] M. Morari F. Borrelli C. Jones. *Explicit Model Predictive Control*. MPC Lab @ UC-Berkeley. 2014.

[17] M. Morari F. Borrelli C. Jones. *Model Predictive Control: Algorithm, Feasibility and Stability*. MPC Lab @ UC-Berkeley. 2014.

[18] Paolo Falcone et al. "A linear time varying model predictive control approach to the integrated vehicle dynamics control problem in autonomous systems". In: *2007 46th IEEE Conference on Decision and Control*. IEEE. 2007, pp. 2980–2985.

[19] Timm Faulwasser, Benjamin Kern, and Rolf Findeisen. "Model predictive path-following for constrained nonlinear systems". In: *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*. IEEE. 2009, pp. 8642–8647.

[20] H.J. Ferreau et al. "qpOASES: A parametric active-set algorithm for quadratic programming". In: *Mathematical Programming Computation* 6.4 (2014), pp. 327–363.

[21] Miroslav Fiedler and Vlastimil Pták. "Some generalizations of positive definiteness and monotonicity". In: *Numerische Mathematik* 9.2 (1966), pp. 163–172.

[22] Daniel Montrallo Flickinger and Mark A Minor. "Remote low frequency state feedback kinematic motion control for mobile robot trajectory tracking". In: *Proceedings 2007 IEEE International Conference on Robotics and Automation*. IEEE. 2007, pp. 3502–3507.

[23] Scott Fujimoto, Herke Van Hoof, and David Meger. "Addressing function approximation error in actor-critic methods". In: *arXiv preprint arXiv:1802.09477* (2018).

[24] Sébastien Gros and Mario Zanon. "Data-driven economic NMPC using reinforcement learning". In: *IEEE Transactions on Automatic Control* (2019).

[25] Sébastien Gros et al. "From linear to nonlinear MPC: bridging the gap via the real-time iteration". In: *International Journal of Control* 93.1 (2020), pp. 62–80.

[26] Martin T Hagan, Howard B Demuth, and Orlando De Jesús. "An introduction to the use of neural networks in control systems". In: *International Journal of Robust and Nonlinear Control: IFAC-Affiliated Journal* 12.11 (2002), pp. 959–985.

[27] Judith A Hall et al. "Liking in the physician–patient relationship". In: *Patient education and counseling* 48.1 (2002), pp. 69–77.

[28] Jixia Han, Yi Hu, and Songyi Dian. "The State-of-the-art of Model Predictive Control in Recent Years". In: *IOP Conf. Series: Materials Science and Engineering*. Vol. 428. 2018.

[29] Salim Hima et al. "Controller design for trajectory tracking of autonomous passenger vehicles". In: *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*. IEEE. 2011, pp. 1459–1464.

[30] B. Houska, H.J. Ferreau, and M. Diehl. "ACADO Toolkit – An Open Source Framework for Automatic Control and Dynamic Optimization". In: *Optimal Control Applications and Methods* 32.3 (2011), pp. 298–312.

[31] Mike Xuli Huang and Ilya Kolmanovsky. *Switch gain scheduled explicit model predictive control of diesel engines*. US Patent 9,765,621. 2017.

[32] Borja Ibarz et al. "Reward learning from human preferences and demonstrations in Atari". In: *Advances in neural information processing systems*. 2018, pp. 8011–8023.

[33] Erik M Johansson, Farid U Dowla, and Dennis M Goodman. "Backpropagation learning for multilayer feed-forward neural networks using the conjugate gradient method". In: *International Journal of Neural Systems* 2.04 (1991), pp. 291–301.

[34] B Kim, D Necsulescu, and J Sasiadek. "Model predictive control of an autonomous vehicle". In: *2001 IEEE/ASME International Conference on Advanced Intelligent Mechatronics. Proceedings (Cat. No. 01TH8556)*. Vol. 2. IEEE. 2001, pp. 1279–1284.

[35] B Ravi Kiran et al. "Deep reinforcement learning for autonomous driving: A survey". In: *arXiv preprint arXiv:2002.00444* (2020).

[36] Felipe Kuhne, Walter Fetter Lages, and J Gomes da Silva Jr. "Model predictive control of a mobile robot using linearization". In: *Proceedings of mechatronics and robotics*. Citeseer. 2004, pp. 525–530.

[37] Matthew Kuure-Kinsey and B Wayne Bequette. "Multiple model predictive control: a state estimation based approach". In: *2007 American Control Conference*. IEEE. 2007, pp. 3739–3744.

[38] Wook Hyun Kwon and Soo Hee Han. *Receding horizon control: model predictive control for state models*. Springer Science & Business Media, 2006.

[39] Robert D Langley. "Apollo experience report: The docking system". In: (1972).

[40] Lionel Lapierre and Bruno Jouvencel. "Robust nonlinear path-following control of an AUV". In: *IEEE Journal of Oceanic Engineering* 33.2 (2008), pp. 89–102.

[41] Xiang Li et al. "Adaptive tracking control for wheeled mobile robots with unknown skidding". In: *2015 IEEE Conference on Control Applications (CCA)*. IEEE. 2015, pp. 1674–1679.

[42] Timothy P Lillicrap et al. "Continuous control with deep reinforcement learning". In: *arXiv preprint arXiv:1509.02971* (2015).

[43] Martin Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: `https://www.tensorflow.org/`.

[44] I Masar and E Stöhr. "Gain-scheduled LQR-control for an autonomous airship". In: *18th International Conference on Process Control*. 2011, pp. 14–17.

[45] Laëtitia Matignon, Guillaume J Laurent, and Nadine Le Fort-Piat. "Reward function and initial values: better choices for accelerated goal-directed reinforcement learning". In: *International Conference on Artificial Neural Networks*. Springer. 2006, pp. 840–849.

[46] David Q Mayne et al. "Constrained model predictive control: Stability and optimality". In: *Automatica* 36.6 (2000), pp. 789–814.

[47] Mohit Mehndiratta, Efe Camci, and Erdal Kayacan. "Automated tuning of nonlinear model predictive controller by reinforcement learning". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 3016–3021.

[48] Dimitris Milakis, Bart Van Arem, and Bert Van Wee. "Policy and society related implications of automated driving: A review of literature and directions for future research". In: *Journal of Intelligent Transportation Systems* 21.4 (2017), pp. 324–348.

[49] Karnchanachari Napat et al. "Practical Reinforcement Learning For MPC: Learning from sparse objectives in under an hour on a real robot". In: *2nd Annual Conference on Learning for Dynamics and Control (L4DC 2020)*. 2020.

[50] Vesna Nevistić and James A Primbs. "Finite receding horizon linear quadratic control: A unifying theory for stability and performance analysis". In: (1997).

[51] Shane O'Sullivan et al. "Legal, regulatory, and ethical frameworks for development of standards in artificial intelligence (AI) and autonomous robotic surgery". In: *The International Journal of Medical Robotics and Computer Assisted Surgery* 15.1 (2019), e1968.

[52] Rajan G Patel, Japan J Trivedi, et al. "SAGD real-time production optimization using adaptive and gain-scheduled model-predictive-control: A field case study". In: *SPE Western Regional Meeting*. Society of Petroleum Engineers. 2017.

[53] Rajesh Rajamani. *Vehicle dynamics and control*. Springer Science & Business Media, 2011.

[54] Robert Ritschel et al. "Nonlinear model predictive path-following control for highly automated driving". In: *IFAC-PapersOnLine* 52.8 (2019), pp. 350–355.

[55] Nilanjan Sarkar, Xiaoping Yun, and Vijay Kumar. "Dynamic path following: A new control algorithm for mobile robots". In: *Proceedings of 32nd IEEE Conference on Decision and Control*. IEEE. 1993, pp. 2670–2675.

[56] John Schulman. "Optimizing expectations: From deep reinforcement learning to stochastic computation graphs". PhD thesis. UC Berkeley, 2016.

[57] John Schulman et al. "High-dimensional continuous control using generalized advantage estimation". In: *arXiv preprint arXiv:1506.02438* (2015).

[58] John Schulman et al. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).

[59] John Schulman et al. "Trust region policy optimization". In: *International conference on machine learning*. 2015, pp. 1889–1897.

[60] Wolfram Schultz, Peter Dayan, and P Read Montague. "A neural substrate of prediction and reward". In: *Science* 275.5306 (1997), pp. 1593–1599.

[61] Dong Hun Shin and Anibal Ollero. "Mobile robot path planning for fine-grained and smooth path spcification". In: *Journal of robotic systems* 12.7 (1995), pp. 491–503.

[62] David Silver et al. "Deterministic policy gradient algorithms". In: 2014.

[63] Mostafa Soliman, OP Malik, and David T Westwick. "Multiple model predictive control for wind turbines with doubly fed induction generators". In: *IEEE Transactions on Sustainable Energy* 2.3 (2011), pp. 215–225.

[64] Chuanyang Sun et al. "A model predictive controller with switched tracking error for autonomous vehicle path tracking". In: *IEEE Access* 7 (2019), pp. 53103–53114.

[65] Richard S Sutton. "Introduction: The challenge of reinforcement learning". In: *Reinforcement Learning*. Springer, 1992, pp. 1–3.

[66] Richard S Sutton. "Reinforcement learning: Past, present and future". In: *Asia-Pacific Conference on Simulated Evolution and Learning*. Springer. 1998, pp. 195–197.

[67] Richard S Sutton. "Temporal Credit Assignment in Reinforcement Learning." In: (1985).

[68] Richard S Sutton and Andrew G Barto. "Reinforcement learning". In: *Journal of Cognitive Neuroscience* 11.1 (1999), pp. 126–134.

[69] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[70] Richard S Sutton, Andrew G Barto, and Ronald J Williams. "Reinforcement learning is direct adaptive optimal control". In: *IEEE Control Systems Magazine* 12.2 (1992), pp. 19–22.

[71] Edward Lee Thorndike. *Animal intelligence: Experimental studies*. Transaction Publishers, 1970.

[72] Jeff Wit, Carl D Crane III, and David Armstrong. "Autonomous ground vehicle path tracking". In: *Journal of Robotic Systems* 21.8 (2004), pp. 439–449.

[73] Xing Wu et al. "An intelligent-optimal predictive controller for path tracking of vision-based automated guided vehicle". In: *2008 International Conference on Information and Automation*. IEEE. 2008, pp. 844–849.