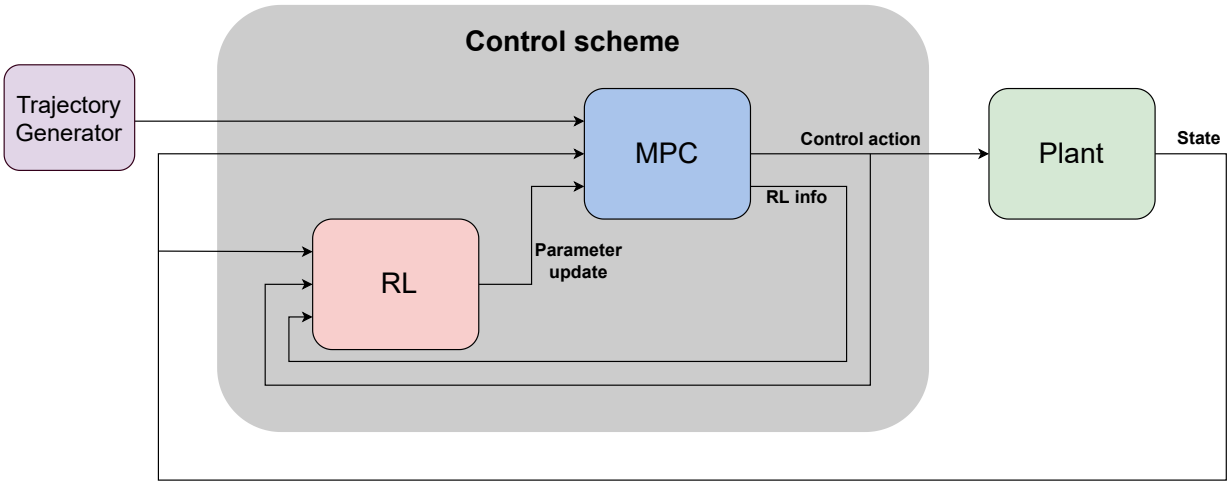
CHALMERS
UNIVERSITY OF TECHNOLOGY



Exploring the potential of adapting a model predictive control scheme with reinforcement learning

Improving performance within the field of vehicle motion control

Master's thesis in Systems Control and Mechatronics

Filip Bertilsson, Fredrik Eriksson

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2022
www.chalmers.se

Master's thesis 2022

Exploring the potential of adapting a model predictive control scheme with reinforcement learning

Improving performance within the field of vehicle motion control

FILIP BERTILSSON
FREDRIK ERIKSSON

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Systems and Control
Chalmers University of Technology
Gothenburg, Sweden 2022

Exploring the potential of adapting a model predictive control scheme with reinforcement learning
Improving performance within the field of vehicle motion control
FILIP BERTILSSON
FREDRIK ERIKSSON

Supervisor: Teodor Husmark, CEVT
Supervisor: Constantin Cronrath, Department of Electrical Engineering
Examiner: Bengt Lennartsson, Department of Electrical Engineering

Master's Thesis 2022
Department of Electrical Engineering
Division of Systems and Control
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Block diagram describing the structure of how Reinforcement Learning is interacting with a model predictive control scheme.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2022

Exploring the potential of adapting a model predictive control scheme with reinforcement learning
Improving performance within the field of vehicle motion control
FILIP BERTILSSON
FREDRIK ERIKSSON
Department of Electrical Engineering
Chalmers University of Technology

Abstract

The automotive industry, among others, sees an increased demand for autonomous solutions. Such solutions are often based on model-based optimal control strategies. Model Predictive Control (MPC) is one such strategy that optimizes the control signal over a simulated horizon. This makes it possible to generate control behavior that takes both soft and hard constraints into consideration. One limitation of this method is that the control signal is only optimal with respect to the internal model, and not the true dynamics.

The goal of this thesis is to investigate if this problem can be alleviated with the use of online learning. The method that is considered combines MPC with Reinforcement learning (RL), where the MPC is used as a policy approximator for the RL scheme. This method uses a parameterized version of an MPC and updates its parameters to maximize the closed-loop performance rather than the accuracy of the model. The interesting part about this method is that it has, prior to this work, been shown that it can generate the optimal policy and value functions even though the model is incorrect. In this work, the method is tested and evaluated against a baseline MPC which uses a linearized model to control an object in a non-linear environment. We show that the RL-MPC method can adjust its parameters to follow a reference in a noisy and non-linear environment with relatively quick convergence of parameters and subsequently outperforms the baseline MPC performance.

Keywords: Reinforcement Learning, Optimal Control, Model Predictive Control, Online Learning, Trajectory Tracking.

Acknowledgements

This thesis, written at CEVT, marks an end to our journey to become engineers. The last five years studying at Chalmers University of Technology has been a task that has shaped us both and given us friendships for life. It has sometimes been very challenging and frustrating, but most of all, it has been rewarding and fun. We also want to acknowledge that this time would not have been the same without our "gang", you know who you are.

This thesis has been a very fitting end to this journey. In many ways, it sums up our time at Chalmers, but at the same time, it has provided us with a good view of what lies ahead. We have been challenged in many ways, learned a lot, found new friends, and seen what it takes to be both an engineer and an academic. Therefore, we want to thank CEVT, which gave us the opportunity to write a thesis for them, and Chalmers, which has been our University over the last five years.

Most of all, we want to thank the people who helped us this spring. Where our supervisors, Teodor Husmark, CEVT, and Constantin Cronrath, Chalmers, have been, without comparison, the most important ones. Without these two, this thesis would have been completely different, and we are very grateful to both of them. Teodor has constantly pushed us with undying enthusiasm and positivity. When progress has been slow and demanding, he has always found something bright in the situation for us to capitalize on. This, together with his technological knowledge, has been extremely helpful. Constantin has, many times during this thesis, given us invaluable guidance. He has genuinely considered our interests and helped us navigate difficult decisions in a very impressive way, even though some parts have been completely new for him. We have many times been both relieved and excited after our weekly meetings with him.

We want to thank our examiner Bengt Lennartsson. While being in a busy period, he has given his time and made it possible for us to finish this work.

Furthermore, we want to thank Sébastien Gros and Mario Zanon, who are the authors of the work we based much of this thesis on. They were very generous and provided knowledge and guidance on short notice when we were stuck.

Finally, we would like to thank everyone we had the pleasure to meet during our time at the CEVT office. We felt welcomed and at home. "Ungdomarna" especially wants to thank Daniel Hultgren and Marcus Andersson.

Filip Bertilsson, Gothenburg, June 2022
Fredrik Eriksson, Gothenburg, June 2022

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

AI	Artificial Intelligence
CEVT	China Euro Vehicle Technology
DNN	Deep Neural Network
MDP	Markov Decision Process
MPC	Model Predictive Control
NMPC	Non-linear Model Predictive Control
RL	Reinforcement Learning
SYSID	System Identification
TD	Temporal Difference

Nomenclature

Below is the nomenclature of indices, sets, parameters, and variables that have been used throughout this thesis.

Indices

k	Index for time step

Sets

$\mathcal{S}$	Set of states
$\mathcal{A}$	Set of actions

Parameters

α	Learning rate
γ	Discount factor
N	Length of horizon

Variables

a_k	Action at time instance k
s_k	Measured state at time instance k
s_{k+1}	State at time instance $k+1$
u_k	Control signal at time instance k
x_k	State at time instance k
x_{k+1}	State at time instance $k+1$

Contents

List of Figures

1
Introduction

The automotive industry is quickly developing towards more energy-efficient, autonomous, and user-friendly systems, which challenges the way we think about transportation at its core. The need for adaptable control algorithms is growing as new demands arise. Considerable research is put into the different fields of Artificial intelligence (AI), where Reinforcement Learning (RL) is a widely used subgenre of AI. RL is used to generate control signals in stochastic environments. At the same time, classical control algorithms are being enhanced and used to great success. Furthermore, developments in other fields, such as fast microcontrollers and efficient algorithms, make it possible to include previously infeasible methods.

This thesis has been conducted at China Euro Vehicle Technology (CEVT). CEVT is an innovation center within Zhejiang Geely Holding Group that works towards the future of mobility. Furthermore, CEVT wants to expand its knowledge of how optimal control can be paired with online learning. This led to this thesis work, where a relatively new method that combines Model Predictive Control (MPC) and RL is tested and evaluated.

1.1 Background

MPC is an advanced control strategy used to control complex systems that need to fulfill certain limitations or constraints. However, the control signals produced rely heavily on the accuracy of the system model, which will never be able to capture the exact dynamics of its real-life counterpart. Moreover, even if such a perfect model would exist, external elements such as variable weather would make the model inaccurate. Other factors that could cause a mismatch between the model and the real-life system could be that the physical system is changed or degraded. It is therefore of great value to automatically adjust the model during run-time to better fit the actual system. A common strategy for improving model fit is system identification (SYSID) [1].

Another approach to solving the problem is to use adaptive control [2]. An adaptive controller adapts to a system with parameters that change or are initially uncertain. This control technique can generally be divided into two types, direct or indirect. Direct methods, as the name implies, are ones where the estimated parameters are directly used in the adaptive controller. Indirect methods, on the other hand, use the estimated parameters to calculate the controller parameters. In recent years

methods of combining MPC with online system identification and online RL have been proposed in [3] and developed in [4] and [5]. This method has been successfully deployed in different cases, to name a few: [6] applies this method and uses it to perform real-time trajectory tracking of autonomous surface vehicles, and [7] uses it in building energy management.

1.2 Motivation of chosen method

As mentioned, SYSID is a common strategy to improve model fit, but it is well-known that when used online, it might decrease the closed-loop performance [8]. The performance decrease stems from the fact that the cost function of the underlying controller is tuned towards a faulty model. If the model is changed online, the intended behavior of the original tuning might be lost, resulting in decreased performance. Because of this, an alternative approach using RL was considered.

RL aims to find the optimal policy $\pi_\star$ which describes how to act in an environment with respect to a pre-defined goal. RL methods can try to learn the policy directly or indirectly by finding the optimal value and action-value functions $V_\star$ and $Q_\star$. The learning is often done by trial and error and updating a deep neural network (DNN) to support these functions. RL methods have seen a wide array of success in many different fields, where applications vary from defeating human Go players [9], [10] to playing Atari games [11], and chemical synthesis planning [12]. But in safety-critical systems, such as the field of vehicle control, the challenges are a bit different. François-Lavet et al. [13] discuss the problems that might arise while working with DNNs. García and Fernández [14] presents an overview of the field Safe RL, which is a growing research area. However, the RL-MPC method, evaluated in this work, takes another approach to safety and capitalizes on the rich body of research on the safety and stability of MPC schemes by using the said scheme to approximate the policy, as well as support the value functions.

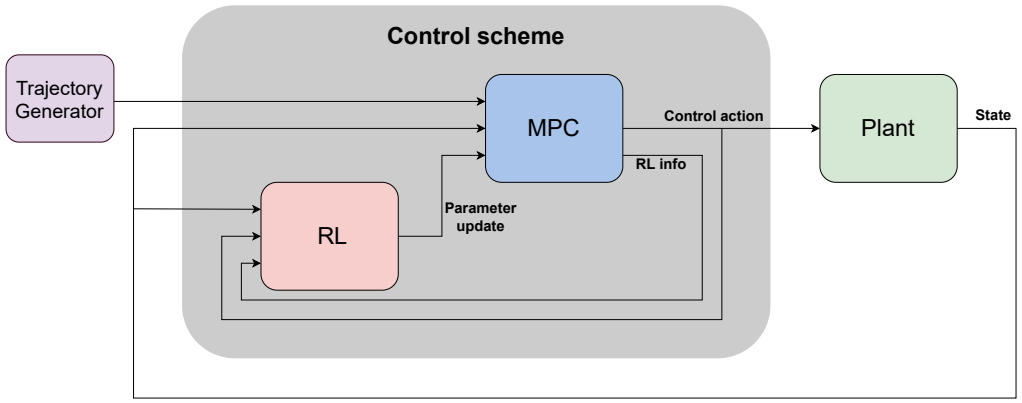


Figure 1.1: Basic block scheme of the proposed method. The RL algorithm uses information from the MPC and the plant to modify parameters in the MPC scheme. A more detailed version are found in chapter 2.4.

Fig. 1.1 shows a simplified block scheme of the method. Furthermore, by using an MPC as a policy approximator, predictability is introduced. By looking at the behavior over the simulated horizon, it is possible to understand the controller's intentions, which is impossible with DNNs that are otherwise typically used in RL. Of course, the predictability relies on that the model used by the MPC at least has some resemblance to the actual system.

Gros and Zanon [3] proved that when using an MPC as a policy approximator, even with an incorrect underlying model, it is possible to generate $Q_\star, V_\star$ and $\pi_\star$ by modifying the constraints, terminal cost and stage cost. I.e., retrieving the optimal policy does not require adjusting the model parameters. However, if the model is allowed to change, the model parameters will not necessarily converge to their true values. This is because, contrary to SYDID, the RL method aims to improve closed-loop performance rather than model accuracy. Although in practice, it is also beneficial to modify the model with SYSID, and [5] outlines how SYSID can be integrated into the method. However, in this work, the addition of SYSID will not be investigated.

1.3 Aim

This thesis aims to evaluate how the RL-MPC method proposed in [3] can be utilized to improve performance over a standard MPC scheme where model mismatches and noise are included to cause offsets. Furthermore, it intends to dissect the essential parts of the RL-MPC method and provide guidance on how to implement it.

1.4 Thesis outline

Following this introduction, the theoretical base needed to understand the method and results are outlined in chapter 2. This chapter also includes the introduction of system dynamics and the linearized model used by the controllers. After the theory, chapter 3 presents the controllers that will be evaluated along with test cases used to assess the performance of them. Chapter 4 show the test cases' results, discussion, and analysis. Chapter 5 contains our conclusion regarding the method. Finally, chapter 6 discusses what could be investigated in future works.

2
Theory

This chapter will go through the theory needed for this work, as well as connecting this theory to certain parts of the experiments in chapter 3. In section 2.1 the dynamics of the system is presented and how a linearized model is derived that can be used for control. MPC is then handled in section 2.2 following by RL in chapter 2.3. Finally, all the parts needed are introduced to be able to present the RL-MPC method in section 2.4.

2.1 Modelling and simulation

In many cases where a model-based controller controls a non-linear system, the system dynamics are linearized around a set-point. This results in the controller only working as intended in relative proximity around the set-point. However, it also allows for faster solve times of the associated optimization problem due to avoiding the non-linearities of the original model. This section describes the non-linear system that the simulation environment is based on, and subsequently, the linearized model is derived.

In this work, the goal of the controllers, presented in section 3.1, is to control an unspecified object to track a velocity reference. The controlled object is influenced by both a force from the controller and a non-linear wind resistance that is quadratic with respect to the velocity. The system dynamics in (2.1) are given by Newtons second law and are taken from [15]. The dynamics include the velocity x and acceleration, object mass m, wind resistance parameter b_2, and the force u, which is used as the control signal.

$$m\dot{x}(t) = u(t) - b_2 x^2(t) \tag{2.1}$$

This model is linearized around a steady state velocity x_0 as described in [15]. By linearizating around a steady state the differential equation $\dot{x}(t) = 0$ gives the steady state force $u_0 = b_2 x_0^2$ needed for velocity x_0. The Taylor expansion in (2.2) gives an approximation of the quadratic equation were $\Delta x = x(t) - x_0$.

$$b_2 x^2(t) \approx b_2 x_0^2 + 2b_2 x_0 \left(x(t) - x_0\right) = u_0 + 2b_2 x_0 \Delta x(t) \tag{2.2}$$

When this is substituted in to (2.1) and one note that $\Delta\dot{x} = \dot{x}(t) - \dot{x}_0 = \dot{x}(t)$ it gives the linearized model in (2.3) around the point $(x_0, u_0 = b_2 x_0)$ where $\Delta u = u - u_0$ and $b(x_0) = 2b_2 x_0$ is the linearized wind resistance parameter.

$$m\Delta\dot{x}(t) + b\left(x_0\right)\Delta x(t) = \Delta u(t) \tag{2.3}$$

2.2 Model Predictive Control

This section introduces the theory of MPC. As concepts are introduced, the theory will be connected with the practical example described in section 2.1.

2.2.1 Context

MPC is an optimal control strategy that uses predictions over a horizon to determine a sequence of control actions by minimizing an objective function. These predictions are based on a model of the system, which means that the quality of the control performance is directly tied to the quality of the model. Although the method calculates a sequence of control actions, only the first action is sent to the plant. This means that in the next time instance, the new state is observed, and the optimization problem is solved again. Fig.2.1 shows an overview of the control scheme. One of the strengths of the MPC approach is that constraints can be included in the design process, which other control techniques struggle to handle explicitly [16]. Historically MPC has been a computationally expensive control technique that has been mainly used in process industries [17]. However, as microprocessors have become more computationally powerful, MPC is now widely used in many fields, for example, in the automotive industry [18] and robotics [19]. Furthermore, in almost all applications where MPC is used to replace a classical controller, a performance increase can be observed [19].

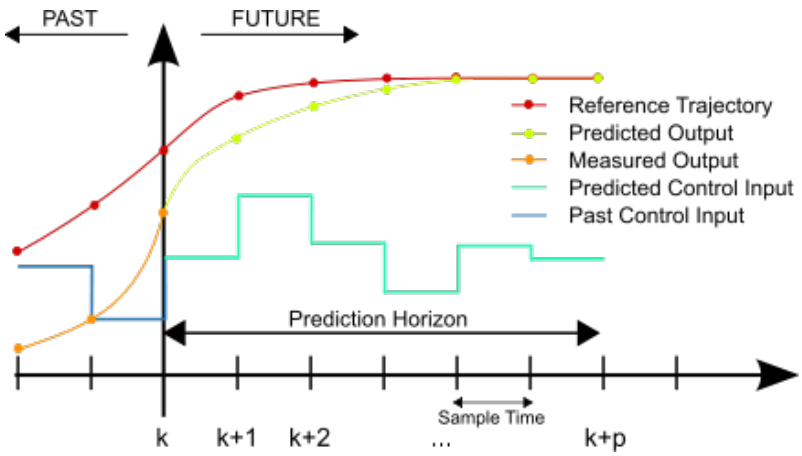


Figure 2.1: An overview of the MPC scheme. In each sampling instance, the MPC predicts the system's behavior over a horizon. Of the predicted control input, only the first signal is sent to the plant. The prediction is then repeated in the next sampling time with the new measurements. CC BY SA. Author: Martin Behrendt. Available at `https://commons.wikimedia.org/wiki/File:MPC_scheme_basic.svg`

2.2.2 The model

A central part of an MPC is the model

$$\dot{x}(t) = f_c(x(t), u(t)) \tag{2.4}$$

where (2.4) is the continuous-time system dynamics. The model used in this thesis is the linearized wind resistance model introduced in section 2.1, see (2.3), which can be represented in state-space form, as seen in (2.5) below.

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t) \\ \Delta\dot{x}(t) &= -\frac{b\,(x_0)}{m}\Delta x(t) + \frac{1}{m}\Delta u(t) \end{aligned} \tag{2.5}$$

The dynamics need to be simulated in discrete time to use the model for predictions over a horizon. There are many ways to discretize a continuous-time model, but for this thesis, it is done with the forward Euler method. The forward Euler method is the least accurate in the Runge-Kutta family of methods [20], but it is also computationally cheap [6]. Additionally, the forward Euler method retains the property that the model structure remains linear in the parameters, which is a highly desired property when RL will later perform updates to these parameters [6].
The forward Euler method works by approximating the state evolution as

$$x_{k+1} = x_k + T_s \cdot f_c(x_k, u_k) \tag{2.6}$$

where T_s is the chosen sampling time of the controller. Which means that the approximation is that the state changes linearly over the sampling time. Replacing $f_c(x_k, u_k)$ with the state-space representation from (2.5), and defining $x_k = \Delta x_k$, $u_k = \Delta u_k$ and $b = b(y_0)$ for notational ease, gives the discrete time dynamics for the wind resistance model as

$$f_{wr}(x_k, u_k) = x_{k+1} = x_k + T_s\left(-\frac{b}{m}x_k + \frac{1}{m}u_k\right) \tag{2.7}$$

2.2.3 MPC formulation

An MPC can be formulated as

$$\min_{\boldsymbol{x},\boldsymbol{u}} \quad \sum_{k=0}^{N-1} \ell(x_k, u_k) + V_f(x_N) \tag{2.8}$$

$$\text{s.t.} \quad x_{k+1} = f(x_k, u_k) \tag{2.9}$$

$$h(x_k, u_k) \leq 0, \quad g(u_k) \leq 0 \tag{2.10}$$

$$x_0 = s \tag{2.11}$$

where $\boldsymbol{x} = (x_0, \ldots, x_N)$ and $\boldsymbol{u} = (u_0, \ldots, u_N)$. This is not an exhaustive formulation, but it covers the basic elements that are commonly present. The first element (2.8) consists of the stage cost $\ell(x_k, u_k)$ and the terminal cost $V_f(x_N)$, where the

latter will be thoroughly presented in section 2.2.4. The stage cost is what defines the cost of moving from state x_k with control signal u_k to the next state, x_{k+1}, where x_{k+1} is constrained to follow the model $f(x_k, u_k)$ as defined in (2.9). (2.11) defines the initial state, where s is used to denote the actual measured state. Additionally it is possible to subject the states and control signals to constraints via (2.10), where $h(x_k, u_k)$ gathers the bounds on the states, $g(u_k)$ the input bounds. However, these terms will be omitted in the simple application of the wind resistance model since they are not essential for this thesis.

The stage and terminal costs are the main components that determine the controller's behavior and, subsequently, the system's behavior. A typical formulation of the costs is the quadratic one

$$\begin{aligned} \ell(x_k, u_k) &= x_k^\intercal Q x_k + u_k^\intercal R u_k \\ V_f(x_N) &= x_N^\intercal P_f x_N \end{aligned} \tag{2.12}$$

Where Q, R, and P_f are weight matrices used to tune the system's behavior. The matrices are symmetric, Q and P_f are positive semi-definite and R is positive definite [21]. The cost function defined in (2.12) will serve to drive the state and the control signal to the origin. To instead drive the state towards a set-point, the variable x_k can be exchanged for the deviation variable $\delta x_k = x_k - x_{sp}$. Which will instead drive the error with respect to a set-point towards zero. Another common practice is to penalize the change in control signal instead of the magnitude of it [21], i.e., u_k is replaced with $\Delta u_k = u_k - u_{k-1}$. Performing this variable exchange gives the stage and terminal cost as shown in (2.13).

$$\begin{aligned} \ell(\delta x_k, \Delta u_k) &= \delta x_k^\intercal Q \delta x_k + \Delta u_k^\intercal R \Delta u_k \\ V_f(\delta x_N) &= \delta x_N^\intercal P_f \delta x_N \end{aligned} \tag{2.13}$$

Putting it all together yields the MPC formulation seen in (2.14).

$$\begin{aligned} \min_{\boldsymbol{x},\boldsymbol{u}} \quad & \sum_{k=0}^{N-1} (\delta x_k^\intercal Q \delta x_k + \Delta u_k^\intercal R \Delta u_k) \\ & + \delta x_N^\intercal P_f \delta x_N \\ \text{s.t.} \quad & x_{k+1} = f_{wr}(x_k, u_k) \\ & x_0 = s \end{aligned} \tag{2.14}$$

2.2.4 The terminal cost

The terminal cost V_f, defined in (2.12), is a term added to improve the MPC's closed-loop stability. It intends to capture the cost-to-go of the problem if the horizon extends to infinity. An ideal terminal cost would be just that, as it yields the stability and optimality of infinite-horizon optimal control [22]. This subsection investigates the unconstrained linear-quadratic case to determine a suitable choice of terminal cost.

First consider a system with dynamics as defined in (2.15) and with (2.16) defining the control sequence for N time steps in to the future.

$$x_{k+1} = Ax_k + Bu_k \tag{2.15}$$

$$\boldsymbol{u} = (u_0, \ldots, u_{N-2}, u_{N-1}) \tag{2.16}$$

Consider then following objective function

$$V(x_0, \boldsymbol{u}) = \sum_{k=0}^{N-1} \ell(x_k, u_k) + V_f(x_N) \tag{2.17}$$

where the initial state x_0 is given from measurements and the subsequent states are calculated with the model specified in (2.15) and the control sequence $\boldsymbol{u}$ (2.16), e.g $x_1 = Ax_0 + Bu_0$.

Writing out the sum of (2.17) yields

$$V(x_0, \boldsymbol{u}) = \ell(x_0, u_0) + \ldots + \ell(x_{N-2}, u_{N-2}) + \underbrace{\ell(x_{N-1}, u_{N-1}) + V_f(x_N)}_{u_{N-1} \text{ only affects this part}} \tag{2.18}$$

As expressed in (2.18), the last control input u_{N-1} only affects the solution of the two last parts, $\ell(x_{N-1}, u_{N-1})$ and $V_f(x_N)$. Similarly u_{N-2} only affects the last three stage costs and so on. This means that if (2.17) were to be minimized with respect to $\boldsymbol{u}$, the optimization problem can be written as in (2.19).

$$\begin{aligned} &\min_{\boldsymbol{u}} \quad \sum_{k=0}^{N-1} \ell(x_k, u_k) + V_f(x_N) = \\ &\min_{u_0, \ldots, u_{N-2}} \left[\sum_{k=0}^{N-2} \ell(x_k, u_k) + \min_{u_{N-1}} \left[\ell(x_{N-1}, u_{N-1}) + V_f(x_N) \right] \right] \end{aligned} \tag{2.19}$$

This problem can be solved via the use of backward dynamic programming [23]. Backward dynamic programming solves the problem in reverse order, i.e., first optimizing over the last variable u_{N-1}. Solving for only the last variable will yield the optimal cost-to-go from state $N-1$ to N, see (2.20).

$$\begin{aligned} V^*_{(N-1)\to N}(x_{N-1}) &= \min_{u_{N-1}} \left[\ell(x_{N-1}, u_{N-1}) + V_f(x_N) \right] \\ &= x_{N-1}^{\intercal} P_{N-1} x_{N-1} \end{aligned} \tag{2.20}$$

Where P_{N-1} is given by the Riccati equation, shown in (2.21)

$$P_{k-1} = Q + A^\top P_k A - A^\top P_k B \left(B^\top P_k B + R \right)^{-1} B^\top P_k A \tag{2.21}$$

with $P_{k-1} = P_{N-1}$ and $P_k = P_f$ in this case. Moving on to the next variable and solving for u_{N-2} gives the same solution, but P_f is replaced with P_{N-1}. This is then repeated until u_0 is reached.

In the end what has been obtained is that the optimal cost-to-go from any state x_k to x_N, is given by (2.22)

$$V^*_{k\to N}(x_k) = x_k^\intercal P_k x_k \tag{2.22}$$

This, however, only gives a solution that is stable up to time N, but by letting N go to infinity, the cost $V^*_{k\to\infty}$ can be obtained. This can be done either iteratively until a stabilizing solution is reached, or by solving (2.23), the algebraic Riccati equation [24].

$$P = Q + A^\top PA - A^\top PB\left(B^\top PB + R\right)^{-1} B^\top PA \tag{2.23}$$

doing the latter yields (2.24).

$$V^*_{0\to\infty}(x_0) = x_0^\intercal P x_0 \tag{2.24}$$

Now consider the infinite horizon version of the linear quadratic problem with $\boldsymbol{u_\infty} = u_0, \ldots, u_\infty$.

$$V(x_0, \boldsymbol{u_\infty}) = \sum_{k=0}^{\infty}(x_k^\intercal Q x_k + u_k^\intercal R u_k) \tag{2.25}$$

The sum in (2.25) can be divided into two parts

$$V(x_0, \boldsymbol{u_\infty}) = \sum_{k=0}^{N-1}(x_k^\intercal Q x_k + u_k^\intercal R u_k) + \sum_{k=N}^{\infty}(x_k^\intercal Q x_k + u_k^\intercal R u_k) \tag{2.26}$$

Where the second term in (2.26) is itself an infinite horizon LQ problem with initial state x_N, which according to (2.24) has the solution

$$V^*_{N\to\infty}(x_N) = x_N^\intercal P x_N \tag{2.27}$$

This means that the infinite horizon problem described above can be turned into an equivalent finite horizon problem that retains the stabilizing properties of the infinite one [21].

$$V(x_0, \boldsymbol{u}) = \sum_{k=0}^{N-1}(x_k^\intercal Q x_k + u_k^\intercal R u_k) + x_N^\intercal P x_N \tag{2.28}$$

This indicates that a good choice of terminal cost in the MPC scheme would be the optimal cost-to-go for the unconstrained infinite horizon LQ. However, it should be noted that the MPC problem is typically not unconstrained. This means that the desired properties gained by the choice $V_f = V^*_{N\to\infty}$ are only retained if the constraints are no longer active in the final state x_N. Which can be ensured by proper choice of a terminal constraint [24], but that is beyond the scope of this thesis, since the problem later considered will be an unconstrained one.

2.3 Reinforcement learning

Reinforcement learning is an area of machine learning that tackles decision-making in an uncertain environment. The core idea of RL is to generate actions that maximize or minimize the cumulative future reward or cost from an unknown underlying Markov-Decision Process (MDP). In this thesis, the objective will be to minimize a cost, so we will mostly talk about costs and not rewards, but the terms are interchangeable. RL handles this by using a user-defined cost signal from the environment, r, and three concepts tightly connected to MDPs. These three concepts are the Action-value function $Q_\star(s, a)$, Value function $V_\star(s)$ and policy $\pi_\star(s)$ where $s \in \mathcal{S}$ and $a \in \mathcal{A}$ are states and actions respectively and the star denotes that they are the optimal functions in respect to the MDP. The goal for the RL agent is, therefore, to maximize the closed-loop performance by updating Q_π, V_π and π so that π resembles the optimal policy as closely as possible. Some more details of these concepts are described in the following subsections.

2.3.1 Basic components of RL

In this section, the basic components of RL will be introduced to lay the foundation for understanding more advanced concepts connected to RL.

Markov decision process

An MDP, as described in [25], is a discrete-time stochastic control process that describes how states, s, are affected by different control actions, a, as $P[s_{k+1}, r_{k+1}|s_k, a_k]$ where r is a reward value. MDPs are often used as a tool when decision-making needs to be done in a stochastic environment, i.e., the outcome of taking a specific action in a specific state is not certain, which often is the case in real-world systems. One important property of an MDP is that the state transition probabilities are not conditionally dependent on previous states and actions. This makes it possible to only care about the system's current state and not its history. In a practical sense, this means that the state in itself might need to include historical information if that is needed for decision making.

Cost signal

A reward signal from the environment is needed to define what goal the RL algorithm should aim to reach. As described by [25] this cost signal is directly connected to the MDP and needs to be defined by the system's designer. As an example connected to this thesis work, where the goal is to follow a reference, the cost will include state deviation from the desired reference. I.e., if the state is close to the reference, the cost will be small, and the agent will therefore get indications that it is doing a good job. Therefore, it is crucial to carefully choose a cost function as this will directly influence the behavior. The optimal behavior of the system, with regards to the cost function, may otherwise be something completely different than intended.

Policy

The policy $\pi(s)$ is a mapping that describes what action a to take while being in state s [25]. Furthermore, $\pi(a|s)$ is the probability that action a is taken while being in state s. The policy is connected to the value functions $V_\pi(s)$ and $Q_\pi(s,a)$ as seen in (2.29), (2.30), and (2.31). In other words, $\pi(s)$ describes the action a that minimizes the expected future cost when in state s.

Value function

Most RL methods include estimating value and action-value functions, $V_\pi(s)$ and $Q_\pi(s,a)$ [25]. These functions describe how good it is to be in a certain state and take any or the, according to the policy, optimal action a. The action-value function, $Q_\pi(s,a)$, is a function describing the expected future reward from taking an action a while being in state s and then following the policy π there on out. The Value function $V_\pi(s)$ is the minimization (when talking about costs) of $Q_\pi(s,a)$ with respect to a. In other words, it describes how good it is to be in a particular state s and follow the policy π from there on. $Q_\pi(s,a)$ is defined by the Bellman equations as in (2.29) were γ is a discount factor and $L(s,a)$ is the stage cost of being in state s and taking action a. The value function V follows as (2.30). The optimal value functions which use $\pi_\star$ as policy are denoted $V_\star$ and $Q_\star$.

$$Q_\pi(s,a) = L(s,a) + \gamma \mathbb{E}\left[V_\pi\left(s_+\right) \mid s,a\right] \tag{2.29}$$

$$V_\pi(s) = \min_a Q_\pi(s,a) \tag{2.30}$$

$$\pi(s) = \arg\min_a Q_\pi(s,a) \tag{2.31}$$

2.3.2 Q-learning

When the underlying MDP is unknown (true for most real systems), the value functions and policy need to be learned somehow. There are multiple different methods, and Q-learning [26] is a classic RL method for this task.

Q-learning works by updating the parameters in Q_π in the direction of the Temporal Difference (TD) [26], as seen in (2.32), where α is a learning rate which is common in RL [25]. The TD, τ, is defined in (2.33), where γ is the discount factor and L_k is the observed stage cost from taking action a_k from state s_k. I.e. the TD is basically the difference between the estimated action-value function for a_k in state s_k with and without bootstrapping from information given in state s_{k+1}.

$$Q_\pi^{new}(s_k,a_k) \leftarrow Q_\pi(s_k,a_k) + \alpha\tau_k \tag{2.32}$$

$$\tau_k = L_k + \gamma V_\pi\left(s_{k+1}\right) - Q_\pi\left(s_k,a_k\right) \tag{2.33}$$

2.3.3 Exploration

Exploration versus exploitation is an important part of RL [25]. Exploitation means that the algorithm exploits previous knowledge and tries to minimize the cost by

choosing the, to its knowledge, best available action. This minimizes the cost and ensures that after learning something that knowledge is capitalized on. This speeds up learning in comparison to only using a random walk. The drawback is that the current policy might not be the best way to do it. By always choosing the best available option, RL might never find the optimal solution because some states will never be visited. By introducing exploration it can be guaranteed that all states will be found if enough time is spent running the system [26]. One of the most common ways to introduce exploration is the ϵ-greedy method. At every new action chosen, when using the ϵ-greedy method, there is a ϵ chance that an exploratory action is deployed. ϵ is often relatively small and furthermore it can be decreased during run time to make it possible for the system to converge to a deterministic policy. In systems like the one presented in this work, the optimal policy might change during run time due to external factors such as changes in weather and so forth. This might pose some problems, which is why adaptive ϵ-greedy methods like [27] and [28] have been developed. Although adaptive ϵ-greedy methods are appealing, they will not be used in this work and are left for future endeavours.

2.4 Reinforcement Learning MPC (RL-MPC)

This thesis aims to improve an optimal controller, in this case, an MPC, with online learning, and the method used is the one presented in [3]. The main idea of this method can be seen in two ways depending on the reader. Someone with a control background may see it as RL is used to tweak the MPC scheme to generate better closed-loop performance. RL then becomes an external actor that tunes the MPC and its behavior online, much as seen in Fig. 2.2. On the other hand, someone with an RL background may see it as if an MPC has replaced the deep neural network (DNN) used as a policy approximator inside the RL method. The MPC generates control signals and the value function by solving an optimization problem. RL then updates the parameters of the policy approximator to generate a higher reward in the future. Both ways of looking at it work and the details of the method will be presented in this section. The section starts with the basic idea, then with how to parameterize the MPC formulation, continuing with how Q-learning is used, and finally, a description of the implementation.

2.4.1 Basic idea of RL-MPC method

As a first step, Fig. 2.2 illustrates an overview of the system and its components. The whole idea of this method is to improve the closed-loop performance of the MPC scheme by letting RL change parameters in the MPC formulation. These parameters can influence most parts of the MPC, including cost function, model dynamics, constraints, and terminal cost. What parameters RL can change are chosen by the system's designer, which will be further discussed in section 2.4.2. In Fig. 2.2 this update is illustrated as the vector of parameters, $\boldsymbol{\theta}$, that are sent from the RL block. The RL method needs some strategy to update the parameters, and in this thesis, Q-learning [26] is used. In order to use Q-learning in conjunction with MPC, the RL method needs the current state s of the plant, the control action a from the MPC,

the cost function value associated with this control signal, and the gradient of the action-value function.

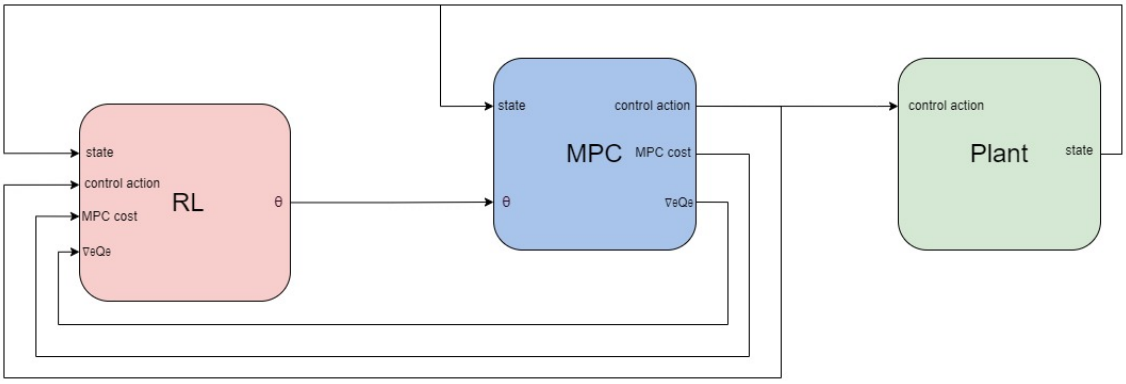


Figure 2.2: Block scheme of how RL is interacting with the MPC controller.

Note that in the following sections the minimization of the objectives are with respect to control actions $\boldsymbol{u} = (u_0, ..., u_{N-1})$, and state variables $\boldsymbol{x} = (x_1, ..., x_N)$. Additionally, $u_0 = a$ is the control action that is sent to the plant and $x_0 = s$ is the measured state. This might be somewhat unclear, but it is done like this because the standard notation in RL and control theory differs.

2.4.2 Parametric MPC formulation

As described in section 2.3, two of the main components within RL is the value function and the action-value function, $V_\pi(s)$ and $Q_\pi(s)$. The combination of MPC and RL is done using the MPC as a function approximator for RL. Where the value function is then defined as in (2.34), where the superscript $\boldsymbol{\theta}$ in V_π^θ denotes that it is parameterized.

$$\begin{aligned} V_\pi^{\boldsymbol{\theta}}(s) = \min_{\boldsymbol{x},\boldsymbol{u}} \quad & V_0^{\boldsymbol{\theta}}(s) + \sum_{k=0}^{N-1} \gamma^k \ell_{\boldsymbol{\theta}}(x_k, u_k) + \gamma^N V_{\mathrm{f}}^{\boldsymbol{\theta}}(x_N) \\ \text{s.t.} \quad & x_{k+1} = f_{\boldsymbol{\theta}}(x_k, u_k), \\ & h_{\boldsymbol{\theta}}(x_k, u_k) \leq 0, \quad g(u_k) \leq 0 \\ & x_0 = s \end{aligned} \tag{2.34}$$

This formulation has some differences from the MPC introduced in section 2.2.3. The term V_0 is called the cost rotation, which aims to ensure that the objective function is a Lyapunov function, thus guaranteeing stability [3]. Furthermore, the term $\gamma \in (0, 1]$ is a discount factor which is commonly used in RL [25]. The other differences are that the costs, the model, and the constraints are now parameterized by $\boldsymbol{\theta}$. Where $\boldsymbol{\theta}$ is a vector of variables that RL is allowed to adjust, this means that to use the MPC formulation introduced in (2.14) as the value function approximator, we need to choose what parameters the RL scheme may adjust. Given that the wind resistance model is quite simple, the obvious choice for parameterization is to allow RL to change the parameter b since this is the only parameter in the model that

changes with the velocity. Additionally, a bias term is added to the model to make the parameterization a bit richer. There is also an option to allow RL to adjust the stage cost, however this is not investigated in this thesis. This results in the parametrization $\boldsymbol{\theta} = (b,\ bias,\ V_0)$, and the model described by (2.35).

$$f_{\boldsymbol{\theta}(x_k,u_k)} = x_k + T_s\left(-\frac{b}{m} + \frac{1}{m}u_k + bias\right) \tag{2.35}$$

Since the formulation for the wind resistance problem does not include any constraints other than the dynamics, the associated value function thus becomes what is shown in (2.36).

$$\begin{aligned} V_\pi^{\boldsymbol{\theta}}(s) = \min_{\boldsymbol{x},\boldsymbol{u}} \quad & V_0^{\boldsymbol{\theta}}(s) + \sum_{k=0}^{N-1} \gamma^k \ell\left(x_k, u_k\right) + \gamma^N V_{\mathrm{f}}\left(x_N\right) \\ \text{s.t.} \quad & x_{k+1} = f_{\boldsymbol{\theta}}\left(x_k, u_k\right), \\ & x_0 = s \end{aligned} \tag{2.36}$$

The action-value function is the same as the value function, but with the addition of putting an input constraint on the first action.

$$\begin{aligned} Q_\pi^{\boldsymbol{\theta}}(\boldsymbol{s}, a) = \min_{\boldsymbol{x},\boldsymbol{u}} \quad & V_0^{\boldsymbol{\theta}}(s) + \sum_{k=0}^{N-1} \gamma^k \ell\left(x_k, u_k\right) + \gamma^N V_{\mathrm{f}}\left(x_N\right) \\ \text{s.t.} \quad & x_{k+1} = f_{\boldsymbol{\theta}}\left(x_k, u_k\right), \\ & x_0 = s, \quad u_0 = a \end{aligned} \tag{2.37}$$

The policy is defined as $\pi^{\boldsymbol{\theta}}(s) = u^\star$, where $u^\star$ is the first control input in the sequence of control signals obtained when V_θ is minimized for any given s w.r.t. $\boldsymbol{x}$ and $\boldsymbol{u}$. Note that the described parametrizations satisfy the equalities of the Bellman equations [23], see (2.38).

$$\begin{aligned} \pi^{\boldsymbol{\theta}}(s) &= \arg\min_a Q_\pi^{\boldsymbol{\theta}}(s, a), \\ V_\pi^{\boldsymbol{\theta}}(s) &= \min_a Q_\pi^{\boldsymbol{\theta}}(s, a) \end{aligned} \tag{2.38}$$

2.4.3 Q-Learning for MPC

To use RL when using an MPC as a function approximator, semi gradient methods [5] can be used to update the parameters. Semi gradient methods, also known as bootstrapping methods, use the TD to update the parameters in the direction of the gradient of the action-value function with respect to the parameter vector. This is done to minimize the estimate while ignoring the target [25]. In (2.39) the parameter update for time step, k, is shown. As previously stated, this update is driven by the TD, τ_k, in the direction of the gradient of the action-value function $\nabla_{\boldsymbol{\theta}_k} Q_\pi^{\boldsymbol{\theta}_k}(s_k, a_k)$. The update is scaled by a learning rate α. The TD is defined in (2.40) where $L(s_k, a_k)$ is the objective function for the RL method. $Q_\pi^{\boldsymbol{\theta}_k}\left(s_k, a_k\right)$ is the optimization cost for the MPC and $V_\pi^{\boldsymbol{\theta}_k}\left(s_{k+1}\right)$ is also the optimization cost for the MPC, but solved for state s_{k+1}. It is important to note here that this requires that the MPC problem is solved twice in each iteration, once for s_k and once for s_{k+1} *before* the parameter

update. $L(s_k, a_k)$ is defined as shown in (2.41), where $\ell(s_k, a_k)$ is the same as the stage cost $\ell_{\boldsymbol{\theta}}(s_k, a_k)$ in the MPC but without any tunable parameters, as RL should not be able to tune its own cost function. $w^\top \max(0, h_\theta(x_k, a_k))$ is the cost of violating the soft constraints where w is a cost parameter set by the designer. This term is only presented for completeness as soft constraints are not used in the test case presented in this thesis. In other words, this update can be seen as if the TD is pushing the parameters in a direction that improves the performance of the MPC with respect to the RL objective.

$$\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k + \alpha \tau_k \nabla_{\boldsymbol{\theta}_k} Q_\pi^{\boldsymbol{\theta}_k}(s_k, a_k) \tag{2.39}$$

$$\tau_k = L(s_k, a_k) + \gamma V_\pi^{\boldsymbol{\theta}_k}(s_{k+1}) - Q_\pi^{\boldsymbol{\theta}_k}(s_k, a_k) \tag{2.40}$$

$$L(s_k, a_k) = \ell(x_k, a_k) + w^\top \max(0, h_{\boldsymbol{\theta}_k}(x_k, a_k)) \tag{2.41}$$

2.4.4 Sensitivities

As noted in section 2.4.3, to update the parameters the gradient $\nabla_{\boldsymbol{\theta}} Q_\pi^{\boldsymbol{\theta}}$ is needed. To define it, first the Lagrangian of the problem formulated in (2.37) needs to be introduced.

$$\begin{aligned}\mathcal{L}_{\boldsymbol{\theta}}(s, \boldsymbol{y}) = \min_{\boldsymbol{x}, \boldsymbol{u}} \quad & V_0(s) + \sum_{k=0}^{N-1} \gamma^k \ell(x_k, u_k) + \gamma^N V_{\mathrm{f}}(x_N) + \\ & \sum_{k=0}^{N-1} \left(\mathcal{X}_k \left(f_{\boldsymbol{\theta}}(x_k, u_k) - x_{k+1}\right)\right) + \mathcal{V}_k(x_0 - s) + \mathcal{U}_k(u_0 - u)\end{aligned} \tag{2.42}$$

where $\boldsymbol{\lambda} = (\mathcal{X}, \mathcal{V}, \mathcal{U})$ are the Lagrange multipliers connected to the constraints and $\boldsymbol{y} = (\boldsymbol{x}, \boldsymbol{u}, \mathcal{X}, \mathcal{V}, \mathcal{U})$ are the primal-dual variables associated with (2.37). With the observation that the equality

$$\nabla_{\boldsymbol{\theta}} Q_\pi^{\boldsymbol{\theta}}(s, a) = \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\boldsymbol{\theta}}(s, \boldsymbol{y}^\star) \tag{2.43}$$

holds for $\boldsymbol{y}^\star$, which is given by the primal dual solution of (2.37). This means that the gradient of the Lagrange function can be constructed as a by-product of solving the associated MPC problem (2.37) [3], ensuring that no additional computational power is spent on obtaining this information.

2.4.5 RL-MPC Algorithm

To implement Q-learning for MPC, a couple of steps need to be taken where the sequence of the different operations needs to be considered to generate the correct behavior. The following list describes the necessary steps for iteration k.

1. Solve the MPC problem with parametrization $\boldsymbol{\theta}_k$ in state s_k to generate a_k. Extract the cost, $Q_\pi^{\boldsymbol{\theta}_k}(s_k, a_k)$, and the Lagrange multipliers $\boldsymbol{\lambda}$.

2. Calculate the RL stage cost $L(s_k, a_k)$.

3. Apply control signal a_k to the plant and wait for s_{k+1}.

4. Calculate $V_\pi^{\boldsymbol{\theta}_k}(s_{k+1})$ by solving the MPC problem with parametrization $\boldsymbol{\theta}_k$ in state s_{k+1}.

5. Calculate τ_k and $\nabla_{\boldsymbol{\theta}} Q_\pi^{\boldsymbol{\theta}_k}$ and update parameters $\boldsymbol{\theta}_k \rightarrow \boldsymbol{\theta}_{k+1}$.

It should be noted that there is some room for different ordering than done here. Furthermore, some calculations can be done in parallel to speed up the algorithm. As an example, one can solve steps 1 and 4 at the same time if proper care is taken with which parameter update is used for the different steps.

2.4.6 Chapter summary

This concludes the Theory chapter. It has gone through the basics of both MPC and RL and finally the theory behind the RL-MPC method which is a combination of both. In the next chapter these concepts will be used in a more practical way, described in chapter 3.

3
Experiment setup

This chapter will define the RL-MPC controller, the baseline MPCs that its performance is compared against, and the test cases used to investigate the different aspects of the controller.

The main idea of the tests in this thesis is to introduce noise and model mismatches that create problems for the controllers to see if the RL-MPC controller can adjust to these difficulties. Model mismatches are introduced by making the controllers with linearized models operate far away from their working point. Fig. 3.1 illustrates this by showing how a linearized model is fitted to the non-linear wind resistance at 50km/h. This figure also depicts the mismatches that the RL-MPC controller will have to overcome.

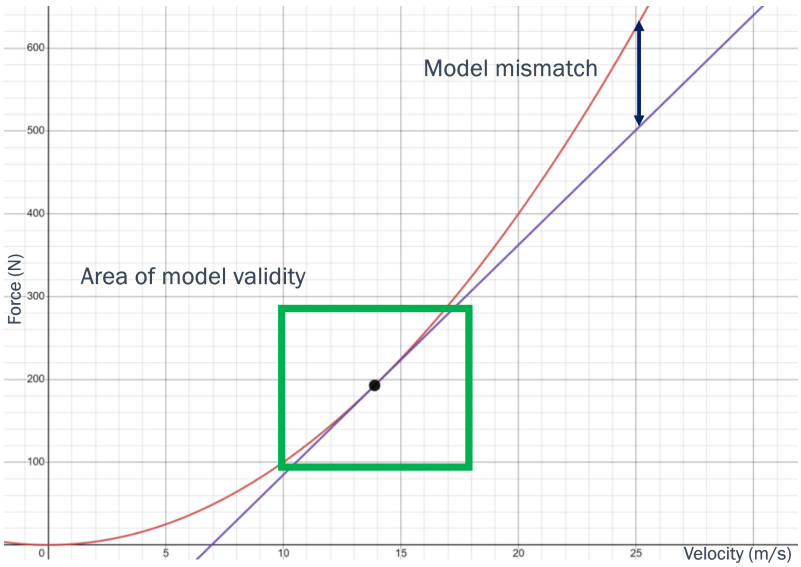


Figure 3.1: Linearization of a quadratic model at the working point 13.88 m/s = 50km/h. Furthermore an approximate area of model validity is showed as well as resulting model mismatch at 25m/s = 90km/h. This is also the dynamics and the working point that the controllers are working in.

3.1 Controllers

The controllers' goal is to regulate an object's velocity and track a velocity reference while being affected by non-linear wind resistance dynamics. The performance of

the RL-MPC is compared to two baseline MPCs, one with a non-linear model and the other with a linearized one. The MPC formulation for both controllers is derived and explained in section 2.2 and repeated in (3.1) below for convenience.

$$\begin{aligned} \min_{\boldsymbol{x},\boldsymbol{u}} \quad & \sum_{k=0}^{N-1} (\delta x_k^\intercal Q \delta x_k + \Delta u_k^\intercal R \Delta u_k) \\ & + \delta x_N^\intercal P_f \delta x_N \\ \text{s.t.} \quad & x_{k+1} = f(x_k, u_k) \\ & x_0 = s \end{aligned} \tag{3.1}$$

Where δx_k are deviation variables with respect to the given reference, Δu_k are changes in control signal, Q and R are the respective tuning variables, and P_f is the terminal cost, chosen as the optimal cost-to-go for the unconstrained infinite horizon LQ, as described in section 2.2.4. Note that the formulation is the same for both baseline MPCs, but the model $f(x_k, u_k)$ differs between the two. The non-linear MPC (NMPC) uses the same non-linear model as the underlying system, while the linear MPC uses the linearized version.

The RL-MPC controller introduced and described throughout chapter 2 uses the same formulation as the linearized baseline MPC. However, it has now been parameterized as defined in section 2.4.2, meaning that RL is now allowed to adjust the parameters $\boldsymbol{\theta} = (b, bias, V_0)$.

3.2 Test cases

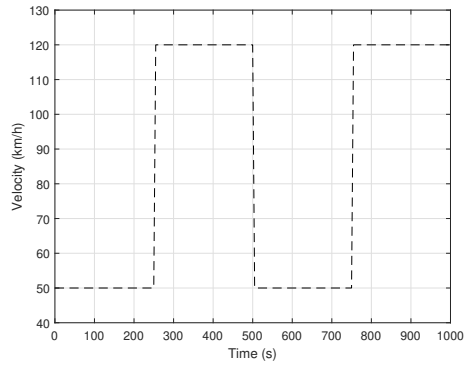
Four different velocity profiles are used as references for the controllers, which can be seen in Fig. 3.2. The references make it possible to test a couple of different properties of the method. The different cases are sometimes simulated with process noise to make the system more realistic and demanding. Furthermore, the introduction of exploration via the $\epsilon - greedy$ method is added when deemed necessary due to low excitation of the states.

3.2.1 Case 1

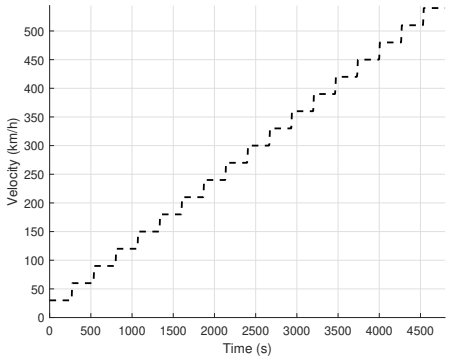
Fig. 3.2a illustrates the velocity reference of test case 1. The goal of this scenario is to test if the RL-MPC can handle a sharp change in reference. Furthermore, it is intended to investigate how it adjusts to a velocity reference far from the point of linearization and how it behaves when returning to it.

3.2.2 Case 2 and 3, Stairs and ramp

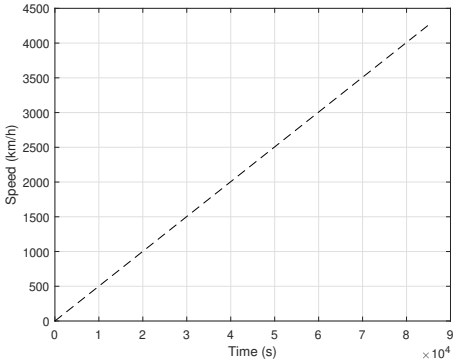
Fig. 3.2b and 3.2c depicts the velocity references for test cases 2 and 3. These test cases aim to assess how far from the point of linearization one can push the system while still maintaining good performance. These two relatively similar cases have some important differences, which makes it interesting to evaluate both of them. In case 2, the parameters have some time to converge at set velocities, but the changes



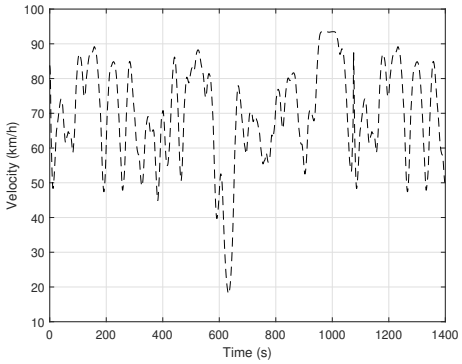
(a) Case 1, velocity reference.



(b) Case 2 velocity reference, stairs



(c) Case 3 velocity reference, ramp.



(d) Case 4 velocity reference, Nürburgring.

Figure 3.2: Velocity reference for the four different test cases.

in reference are more demanding. On the contrary, in case 3, the reference never settles, which does not give time for the parameters to converge, but the reference never does any sudden potentially problematic jumps.

In test case 2, the velocity reference continuously increases by 30 km/h at every 2666 time-steps starting from the linearization point. This reference continues indefinitely.

In test case 3 the velocity reference increases very slowly at 0.018 km/h at every time step.

3.2.3 Case 4, Nürburgring

Fig. 3.2d illustrates the velocity reference of test case 4. This test case aims to evaluate the method's performance in a scenario more related to real driving. The reference is a velocity profile from the race track Nürburgring. However, it should be noted that the low velocity indicates that it is most likely not a velocity profile of a race car driver. It was generated by simulating the Simscape™ Vehicle Templates project [29] and saving the velocity at a set time interval.

3.2.4 Chapter summary

This chapter has outlined the experiments and methods used to test and evaluate the RL-MPC method against a baseline MPC and an NMPC. Four test cases are presented which test different aspects of the method. All of the these tests forces the RL-MPC method to overcome some level of model miss match. Tests 1-3 are created to test certain attributes of the method while test 4 is a more realistic scenario. This makes it possible to test some attributes relatively isolated and then combine the acquired knowledge and apply it in a more complex test.

4
Results and Evaluation

The results of the tests described in chapter 3 will be presented throughout this chapter. The different tests are divided into sections, where a more detailed description of each case will be provided, along with the results and discussion surrounding them.

4.1 Case 1

This test case is divided into three variants, which test slightly different properties of the original case. The following subsections will present the reason for investigating each variant, the results, and some discussion surrounding them.

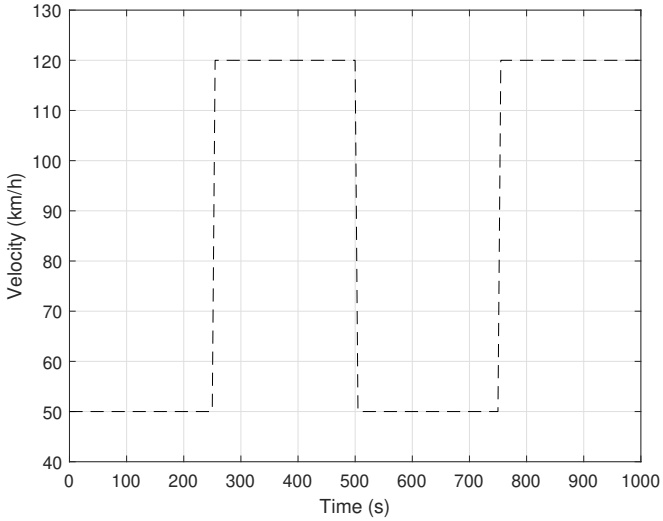


Figure 4.1: Case 1, velocity reference.

4.1.1 Initial results

Initially when running this experiment without noise or exploration, the large and fast change in reference proved to be a problem for the RL-MPC since the jump in reference caused a large spike in the control signal, which in turn caused a spike in the TD and the gradients of the Q-function. This behavior was seen both with and without added noise. The effect of this was that the parameters changed to unreasonable values, which had the cascading effect of producing larger control signals

which further drove the parameters to diverge.

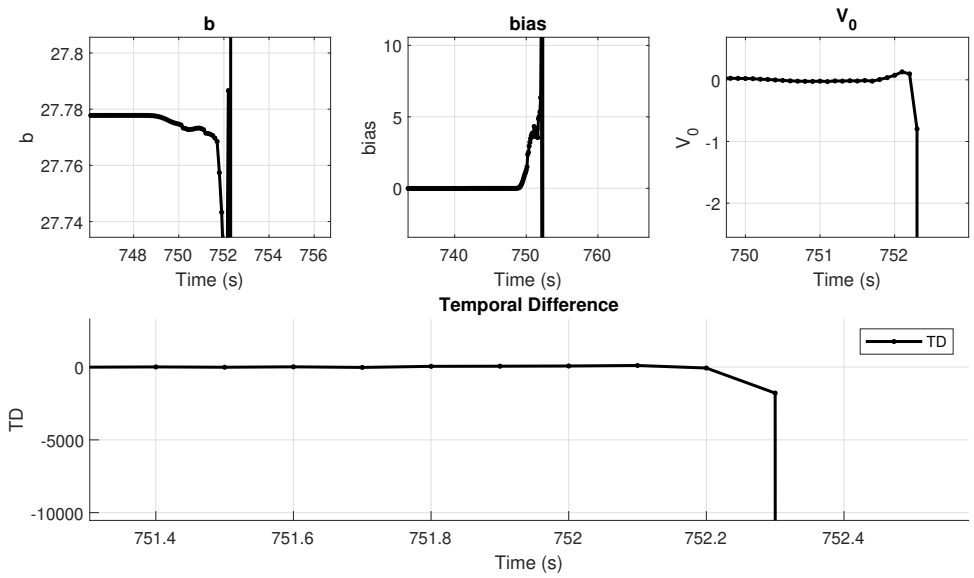


Figure 4.2: Zoomed in view of TD and parameter evolution at the time of divergence. Caused by the first change in reference in case 1.

This problem can be dealt with in several different ways. Firstly, a significant decrease in the learning rate would limit the effect of the TD spikes. However, this is not the desired solution since it would inhibit learning. The second solution was to decrease the aggressiveness of the reference change, i.e., change the reference from a discrete step to a linearly increasing ramp. This solution showed promising results but there proved to be a trade-off between how fast the reference could increase and the learning rate. Namely, to maintain a desirable learning rate, the reference had to increase in such a slow manner that one of the original purposes of the case was lost. The third solution was implementing an outlier detection algorithm to reject the TD values produced during the change in reference. The method used to reject outliers was simple. Anything three standard deviations away from the historical TD were rejected. However, this method of rejecting outliers proved too strict when applied throughout the whole simulation. Thus, since the timing of the reference change was known, this algorithm was only active N samples before and N samples after the change occurred. In the end, a combination of a slightly ramping reference and outlier rejection proved to be the most successful.

4.1.2 No process noise

In this variant, there is no process noise nor measurement noise added to the state, which is supposed to represent the ideal case for the baseline MPCs. This test aimed to investigate if the RL-MPC could outperform the baseline MPC when it is working away from its linearization point, without any factors such as noise affecting the results. Additionally, to investigate how it behaved when returning to the linearization point.

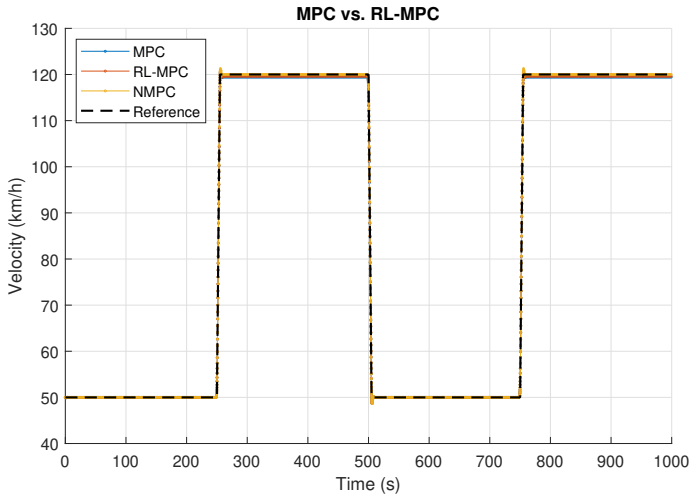


Figure 4.3: Results from test case 1. In this variant, there is no process noise added to the simulation. The RL-MPC performance is slightly improved compared to the linear MPC, but does not reach the performance of the NMPC.

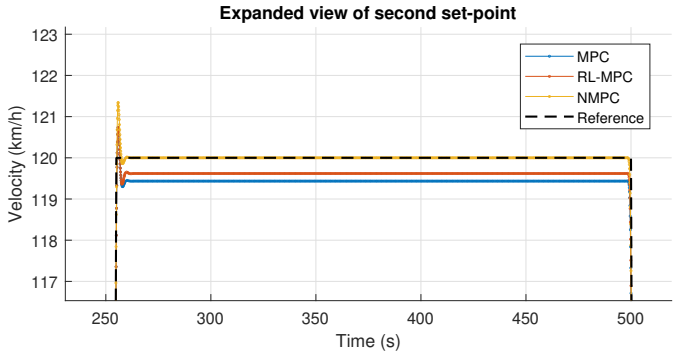


Figure 4.4: Expanded view of the second set-point from Fig. 4.3.

Fig. 4.3 depicts the results of this variant, where it is hard to discern any differences between the controllers. Initially, the reference is right at the linearization point, and all the controllers track the reference perfectly. Fig. 4.4 shows that the NMPC manages to follow the new reference point, while the linearized MPC is approximately 0.5 km/h off. The RL-MPC has slightly improved performance compared to the linear MPC, but it does not reach the performance of the NMPC. When the reference is changed back to the linearization point, the performance of the RL-MPC is slightly decreased compared to when at the initial set-point. Investigating Fig. 4.5, which shows the evolution of the RL parameters, explains this behavior.

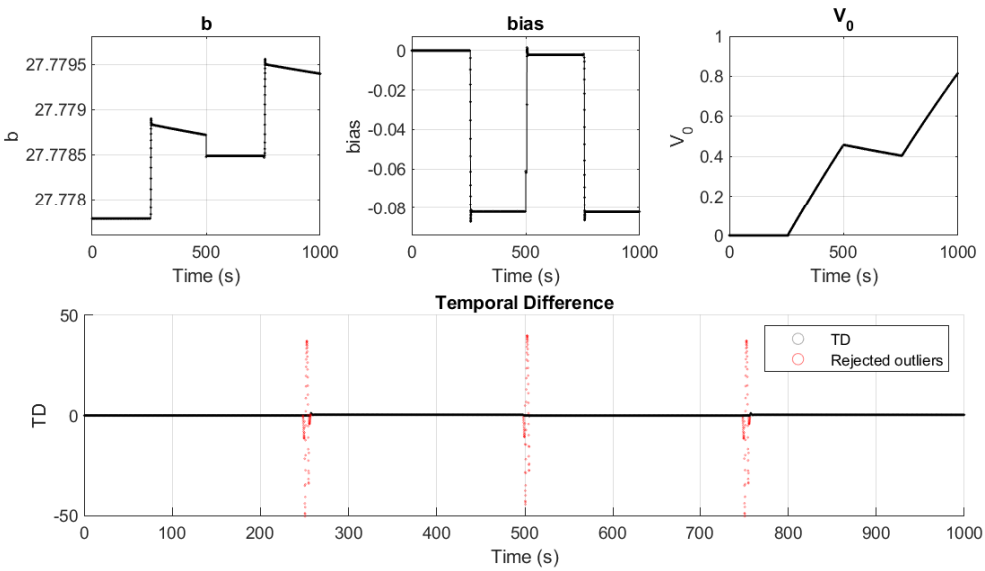


Figure 4.5: Illustration of the evolution of the RL parameters, as well as the TD, where the red dots indicate outliers that have been rejected. The RL parameters only see significant changes whenever the reference is changed.

The cost rotation term, V_0, is not relevant to control performance and is only presented for completeness. The change in the model parameter b is so tiny that it can be considered static; an explanation for this will follow later. The bias remains unchanged at first, which makes sense since the reference location is at the point where the underlying model was linearized around. After the reference is updated, we observe that the bias changes to approximately -0.08, which allows it to compensate somewhat for the model mismatch. The reason that the bias improves the control performance should be apparent considering the underlying model, repeated in (4.1) for convenience.

$$f_{\boldsymbol{\theta}(x_k,u_k)} = x_k + T_s \left(-\frac{b}{m} + \frac{1}{m} u_k + bias \right) \tag{4.1}$$

Without the bias, the linearized model predicts that it will be right at the set-point when in reality, it ends up below it. The bias then shifts the predictions to be more in line with the truth. The performance increase, however, is not significant because the change in bias is too small, and it leaves the question of why that is.

To answer this question, it should be noted that the parameter only changes right after the reference change. This corresponds to the short period it takes for the state to settle right after it has reached the set-point; see Fig. 4.4. This settling period is the only thing that induces a large enough TD and gradient of the Q-function to drive any significant parameter update. When the state has stabilized, the TD and the gradients are just too small to give rise to a significant enough update to be relevant. However, scenario was also simulated for ten times longer than presented above, which allowed all the terms to converge completely. But even then, the value of the parameters compared to what is presented was insignificant, and

the performance remained effectively identical. This is due to the system ending up in a steady-state. This can be compared with classical RL and having converged value functions over a policy in a deterministic MDP with no exploration. I.e., there will be no updates of the policy no matter how many episodes are run because no new information is added, and a local minima has effectively been found.

Another thing to note is that when the reference returns to the linearization point, the parameters do not return to their initial (ideal) values, which causes the performance of the RL-MPC to be ever so slightly worse. The reason for this is again that the TD and the gradients are too small to significantly impact the parameters.

As a more clear evaluation of the performance, the cumulative deviation from the reference is presented in Fig. 4.6 which again shows that at the first set-point there is no deviation from any of the controllers. When the reference is changed it is, as expected, clear that the NMPC still tracks the reference without any issues. It is also clear that the RL-MPC adjusts its parameters to improve the performance compared to the linearized MPC. When the set-point moves back to the linearization point, the two baseline MPCs perform identically; however, crucially, the RL-MPCs performance is very slightly decreased, which can be seen from the almost imperceptible increase in deviation between 500 and 750 seconds.

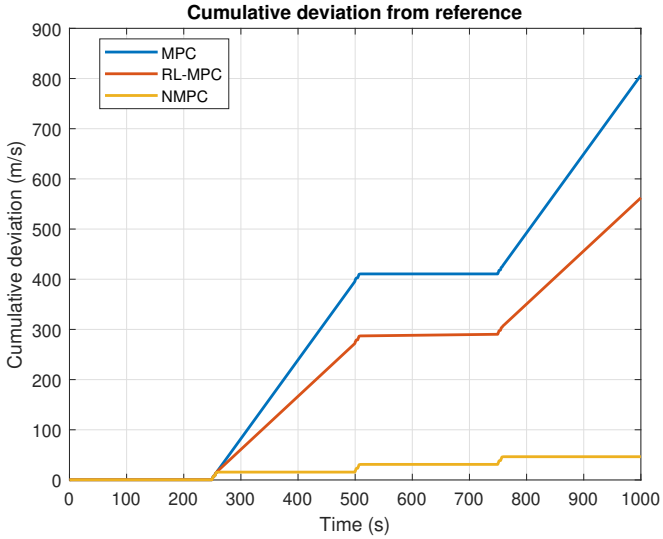


Figure 4.6: Illustration of the cumulative deviation from the reference of the three controllers in case 1, when no process noise is added.

4.1.3 Escaping the local minima by using ϵ-greedy

In this variant, the purpose is the same as in chapter 4.1.2 but intends to force the RL-MPC to make exploratory moves by extending it with an ϵ-greedy algorithm. The ϵ-greedy algorithm will 10% of the time perturb the control signal with additive Gaussian noise, $\mathcal{N}(0, \sigma^2)$. This test uses the case 1 reference, as in chapter 4.1.2, but the time spent at each set-point is increased to allow for parameter convergence.

It should also be noted that the graph of the full state trajectory is omitted because it is hard to discern anything useful from it. Instead an expanded view of each set-point is shown in Fig. 4.7. The cumulative deviation from the reference can also be seen as a clearer indication of the controllers' performance in Fig. 4.8. The first

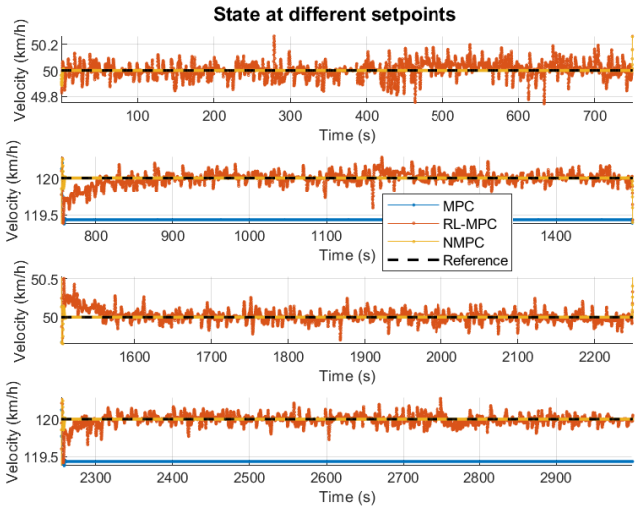


Figure 4.7: Expanded view of the different set points in the reference from case 1. In this variant the there is no process noise added to the simulation, but the RL-MPC has been extended to sometimes make exploratory control moves.

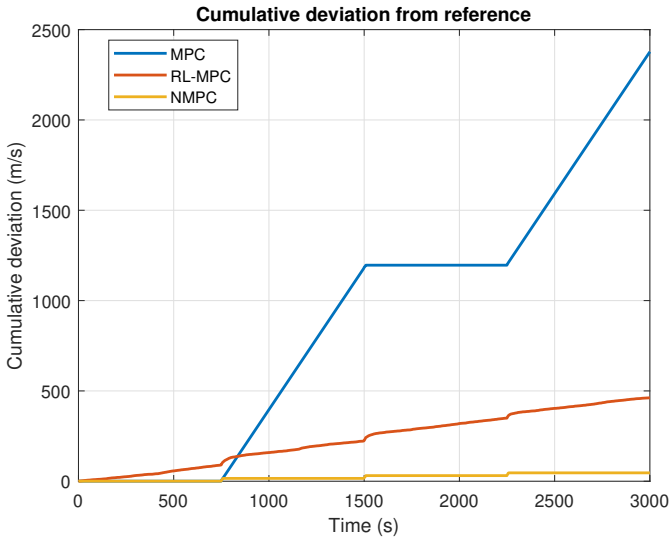


Figure 4.8: Illustrates the cumulative deviation from the reference from case 1 without process noise. The RL-MPC is now extended with exploration. The RL-MPC begins outperforming the baseline MPC when the reference moves away from the linearization point.

observation is that the performance is worse at the first set-point, which is expected since the exploratory control moves only serve to deviate from the optimal behavior.

After the first change in reference, the RL-MPC starts worse but soon converges and tracks the reference as good as the NMPC, albeit with the exploration still causing random deviations. Changing the reference back to the original set-point shows that the performance of the RL-MPC is initially worsened, but it eventually converges to the set-point again. The fourth subplot in Fig. 4.7 illustrates that the convergence rate is similar to the first reference increase but not the same due to the random nature of the exploration.

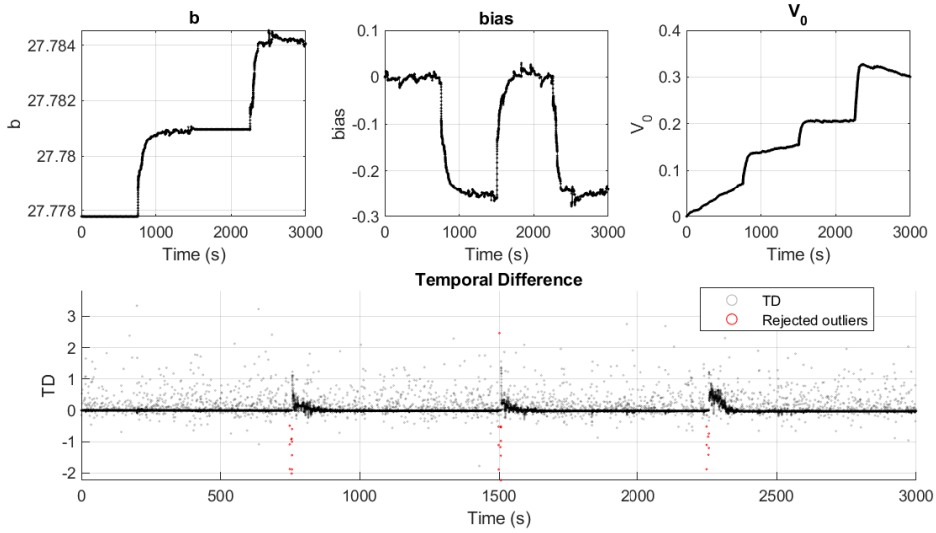


Figure 4.9: Illustration of the evolution of the RL parameters and the TD when using exploration, and no noise is added. The red dots in the TD plot indicate outliers that have been rejected. The parameters still see the largest change when the reference is changed but now continue evolving even when the set-point is fixed.

If we once again investigate the TD and the evolution of the parameters, as seen in Fig. 4.9, there is a noticeable change in behavior. The TD now sees activity throughout the simulation, which drives the parameters to change and eventually converge. Although, as in section 4.1.2, it is still only the bias that contributes to improving the controller's performance. The model parameter b still only sees insignificant changes.

To address the question as to why that is, first recall the underlying model used by the RL-MPC, repeated below in (4.2) for convenience.

$$\Delta \dot{y}(t) = \Delta F_d(t) - \frac{b}{m}\Delta y(t) + bias \tag{4.2}$$

Then recall that what drives the parameter updates are

$$\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k + \alpha \tau_k \nabla_{\boldsymbol{\theta}_k} Q_\pi^{\boldsymbol{\theta}_k}(s_k, a_k) \tag{4.3}$$

and that

$$\nabla_{\boldsymbol{\theta}_k} Q_\pi^{\boldsymbol{\theta}_k} = \nabla_{\boldsymbol{\theta}_k} \mathcal{L}_{\boldsymbol{\theta}_k} \tag{4.4}$$

The gradient of the action-value function with respect to b and the bias, respectively, will thus be

$$\begin{aligned} \nabla_b Q_\pi^{\boldsymbol{\theta}_k} &= \nabla_b \mathcal{L}_{\boldsymbol{\theta}_k} = \mathcal{X}\frac{1}{m}\Delta y(t) \\ \nabla_{bias} Q_\pi^{\boldsymbol{\theta}_k} &= \nabla_{bias} \mathcal{L}_{\boldsymbol{\theta}_k} = \mathcal{X} \end{aligned} \tag{4.5}$$

where $\mathcal{X}$ are the Lagrange multipliers associated with the state dynamics constraint. Equation (4.5) shows that the sensitivity of the two parameters is linked to the Lagrange multipliers $\mathcal{X}$. However b is always scaled by two additional factors, m and $\Delta y(t)$. Where m is simply a static factor which represents the mass of the object and $\Delta y(t) = y(t) - y_0$. This has two implications; firstly, any update to the b-term will always be scaled by a factor of $\frac{1}{m}$. Secondly, the closer the state moves to the linearization point, the smaller the gradient becomes since as $y(t)$ approaches y_0, $\Delta y(t)$ goes to zero. The gradients are explicitly shown in Fig. 4.10 below, and the phenomena can be seen clearly in Fig. 4.9 between 1500 and 2000 seconds. This is where the state has changed back to its original linearization point, and the evolution of b stops. This also implies the inverse, that the further away we move from the original linearization point, the more sensitive the parameter becomes, which can be clearly seen in case 3, section 4.3.

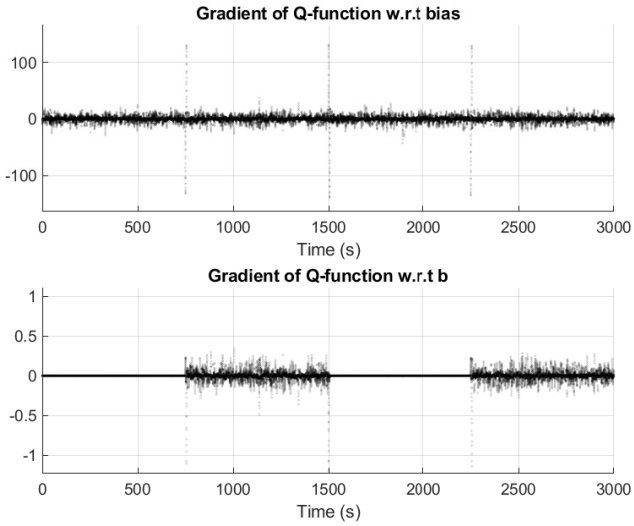


Figure 4.10: Illustrates the gradients of action-value function with respect to bias and b from test case 1, simulated without process noise and with exploration.

4.1.4 Addition of process noise to the system dynamics

To make the test case somewhat realistic, noise is now added to the simulation of the real system dynamics. For the RL-MPC, the addition of process noise can effectively be seen as if some variant of exploration was built into the system, so forcing exploratory moves will not be needed in this case. In fact, the results of running the system with process noise that is uniformly distributed around zero show that the RL-MPC performs similarly to the test in chapter 4.1.3, where exploration

was used. Additionally, since the noise also affects the baseline MPCs, the RL-MPC outperforms the linearized MPC at all set-points and performs almost equally well as the NMPC. In Fig. 4.11 the cumulative deviation from this test can be seen.

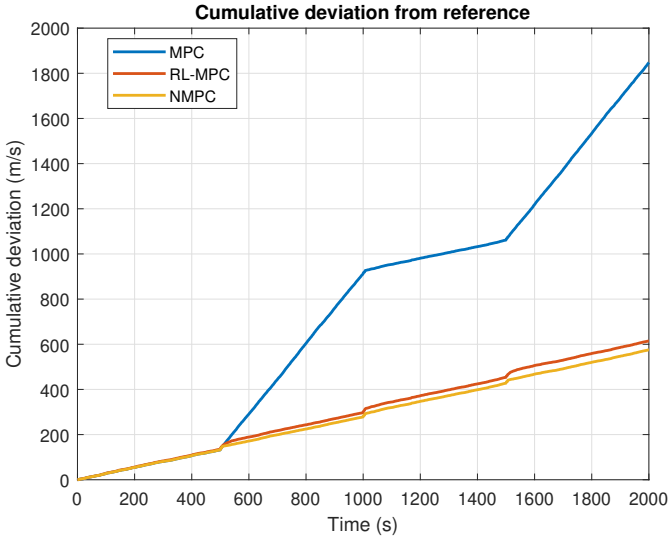


Figure 4.11: Illustration of the cumulative deviation from the reference in the case where the added process noise is uniformly distributed around zero.

If the process noise is modified to have a bias, i.e., no longer centered around zero, the performance of the RL-MPC stays relatively unchanged, while the baseline MPCs struggle to deal with it. In Fig. 4.12, 4.13 and 4.14 the results are shown.

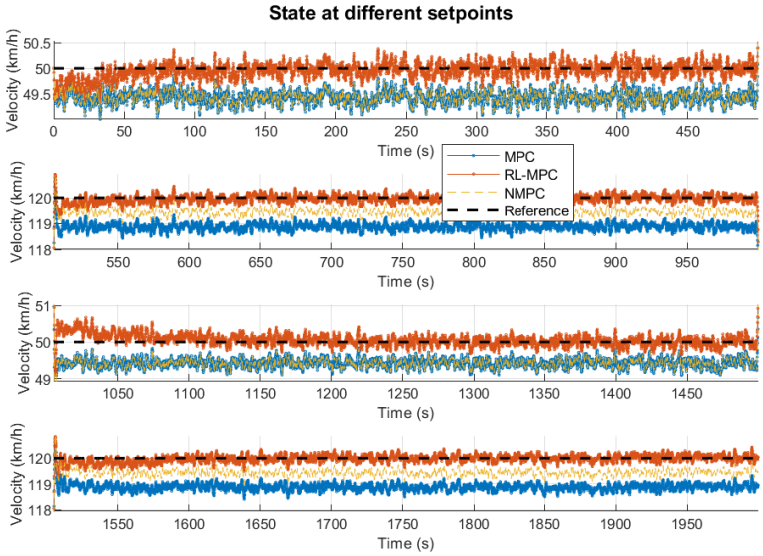


Figure 4.12: Expanded view of the different set points of the trajectory following the case 1 reference. The added process noise is here strictly negative which causes the baseline MPCs to diverge from the reference. However, the RL-MPC manages to compensate for this.

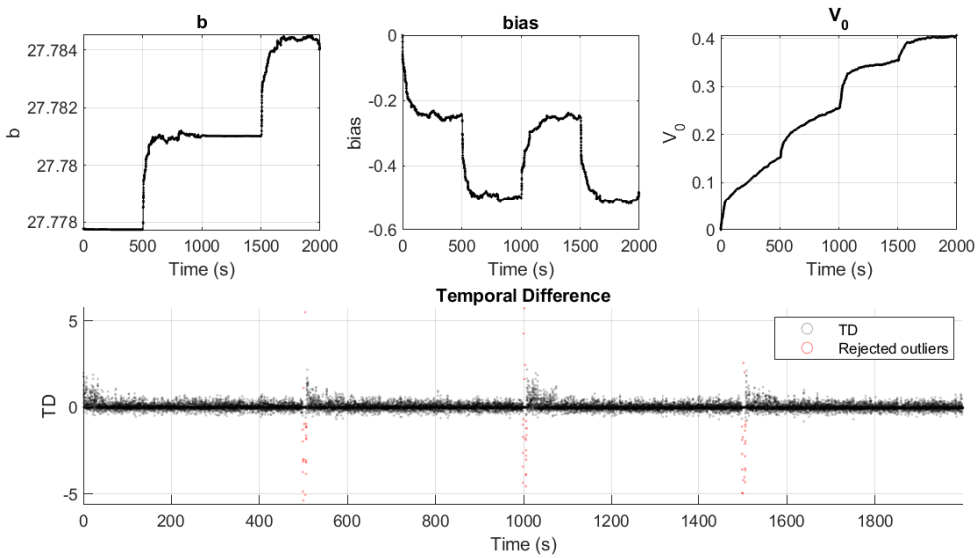


Figure 4.13: Shows the TD and the evolution of the parameters when the process noise is negatively biased.

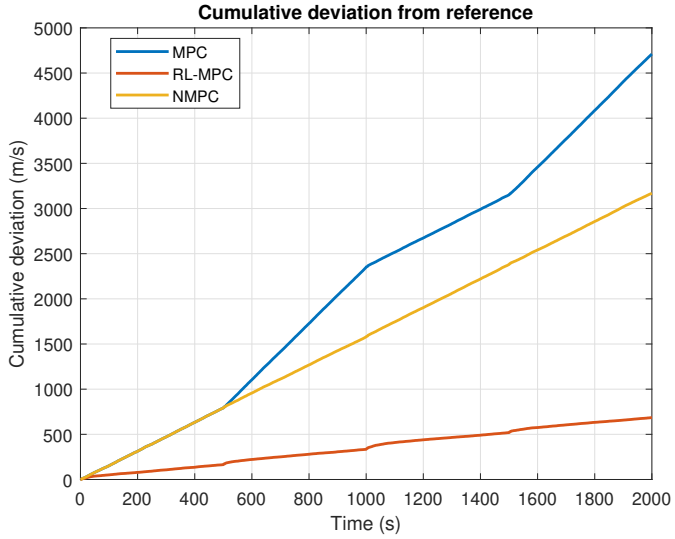


Figure 4.14: Plot of the cumulative deviation from the reference speed with an added, strictly negative, process noise.

The addition of biased process noise highlights the RL-MPCs capability to learn and compensate for both model mismatch and the biased noise.

4.2 Case 2, stairs

This section shows the results of the controllers when following the step-wise increasing reference of case 2 as outlined in section 3.2.2. This test evaluates how far the method can be pushed in an increasingly demanding environment (exponential influence of wind resistance) and still get good performance and convergence of

the parameters. It might seem excessive to investigate speeds over 500 km/h, but for other applications with similar dynamics, this test might still offer some insights.

Three variants of this test were performed, one with noise uniformly distributed around zero, one with an additive, strictly negative process noise, and one with no process noise added. The results of the latter two variants are omitted due to brevity, but a brief summary of these results follows at the end of the section. It should be noted that the results from the NMPC controller has been removed as it does not add any additional insights from those presented in chapter 4.1. This is due to the NMPC controller having the correct underlying model and thereby it will only show non optimal behaviour due to noise.

When applying a uniformly distributed process noise around zero, the results from this case are presented as follows. In Fig. 4.15, it can be seen that the higher the velocity, the further the original MPC deviates from the reference. At the same time, the RL-MPC method manages to overcome the model mismatch and follow the trajectory.

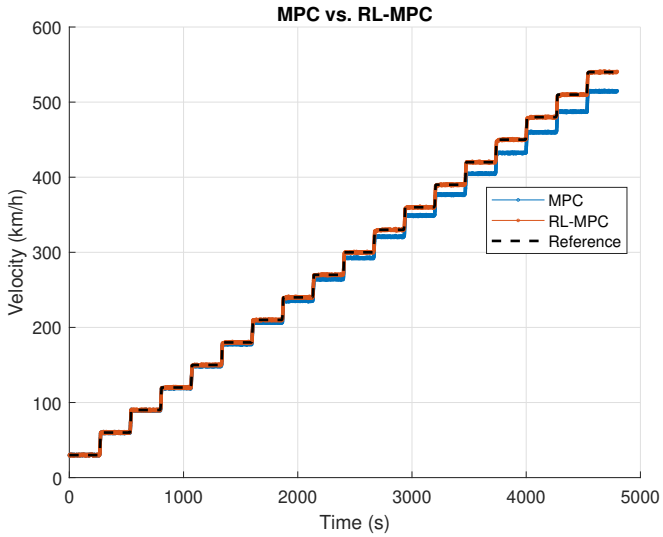


Figure 4.15: State trajectories from the MPC and RL-MPC controllers during case 2 with an added uniformly distributed noise around zero. A close-up of the last part of simulation is shown in Fig. 4.16. Both controllers follow the reference sufficiently in the first part. The further away from the point of linearization the reference gets, the bigger the differences become. The RL-MPC method follows the reference up to over 540 km/h, where the simulation ended. The NMPC controller is omitted for clarity as it does not add anything additional from what is investigated in case 1, see Fig. 4.12.

It should be noted that simulating in this manner can not go on indefinitely due to the non-linear dynamics of the wind resistance. When the velocity becomes too large, even a 30 km/h increase requires a very large difference in the control signal. This makes the TD and Q-function gradient that drives the parameter

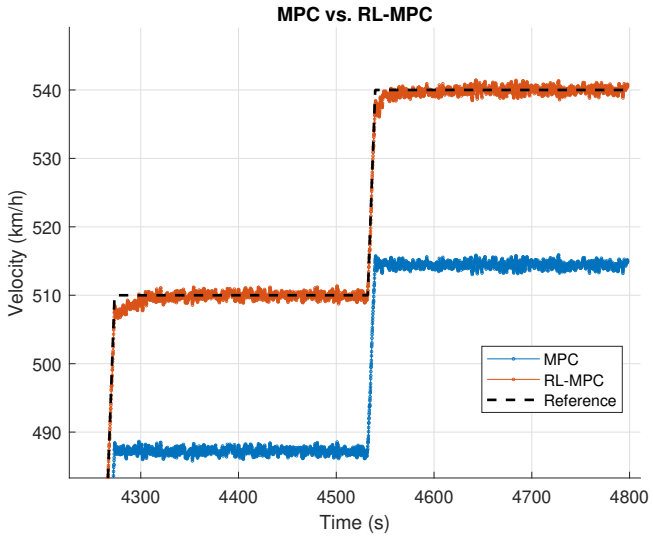


Figure 4.16: A closeup of the last parts of 4.15 illustrating the state trajectories of case 2. At the end of the simulation, the RL-MPC still manages to follow the reference while the MPC has deviated from the reference substantially.

update increase to unreasonable values and subsequently causes the RL parameters to diverge. To make the parameter update smoother and alleviate the problems with diverging parameters, outlier rejection was used in a similar way as in chapter 4.1. This worked to some degree and made it possible to continue multiple steps higher compared to not using it. However, in the case presented here, the parameters did diverge a few reference updates after what is illustrated. In Fig. 4.17 it can be seen that the spikes in TD get increasingly large in conjunction with reference updates, even though the largest parts of the spikes are removed (in this case, removed and not illustrated in red). This observation further emphasizes the need for some type of outlier rejection if sharp and sudden changes in reference are present.
As mentioned in chapter 4.1.1, during experimentation, it was noted that a more gentle slope of the reference update also made it possible to continue to higher velocities. This behavior is partly why case 3 was constructed. This test case shows that the RL-MPC, with some help from outlier rejection, performs very well relative to the baseline MPC. Furthermore, over time, it adapts to dynamics that are very different from what is first modeled.

Other tests performed with same reference

The tests performed but not thoroughly presented due to brevity are briefly presented here. The results with strictly negative noise are similar to those presented with a small addition of cumulative deviation for the RL-MPC case and a bigger addition for the regular MPC. The difference is similar to what is seen in the initial two stages of case 1, see Fig. 4.12, 4.13 and 4.14. The results from testing without noise or exploration are similar to case 1 as well, but with the addition that the RL-MPC controller manages a bit better than the MPC when they get far from the linearization. Break-even in this scenario is at around 140 km/h.

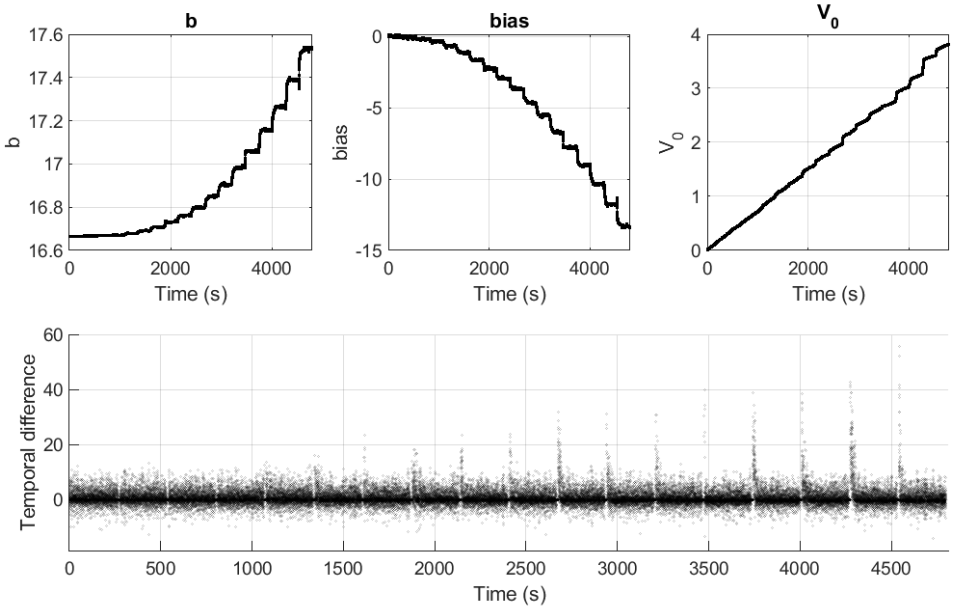


Figure 4.17: TD and evolution of RL parameters in case 2 with an added uniformly distributed noise around zero. The TD has distinct spikes at the times of reference updates. This behavior appears more distinctly the longer the simulation goes. The jumps in the parameter values are directly linked to the spikes in TD at the time of reference updates. It should be noted that an outlier rejection has been used to remove problematic values of the TD to make the parameter updates smoother.

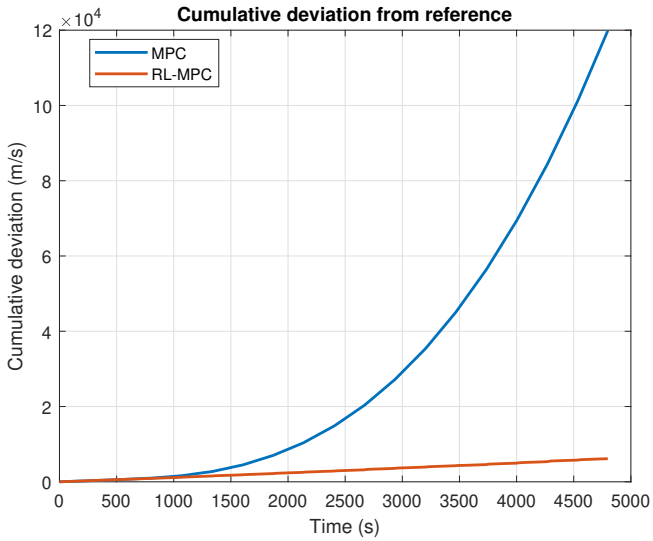


Figure 4.18: Absolute cumulative deviation of state values in respect to velocity reference for both methods during case 2 with an added uniformly distributed noise around zero. The cumulative deviation of the MPC controller grows in an exponential fashion while it grows linearly for the RL-MPC controller.

4.3 Case 3, linearly increasing reference

This case is constructed to test the ability of the RL-MPC method to follow an increasingly growing reference without jumps that might cause instabilities. In Fig. 4.19 and 4.20 the results of this test is shown. It should be noted that the simulation was ended and that the system did not crash. Fig. 4.19 shows that the RL-MPC method manages to follow the reference in a very good manner. It is not shown here, but the RL-MPC method almost manages to get as good performance as the NMPC.

As mentioned in section 2.4 and 4.1.3, the parameter update is driven by both the TD and the sensitivity of the Q-function. Furthermore, it was shown with (4.5) that the sensitivity of the b parameter is relatively low due to being scaled by Δy, which usually is pretty small, and $\frac{1}{m}$. In case 1, the b term did therefore not contribute in any significant way. In this case, on the other hand, as seen in Fig. 4.20, the velocity deviates so far from the point of linearization that Δy becomes large enough to make b the driving force of adaption.

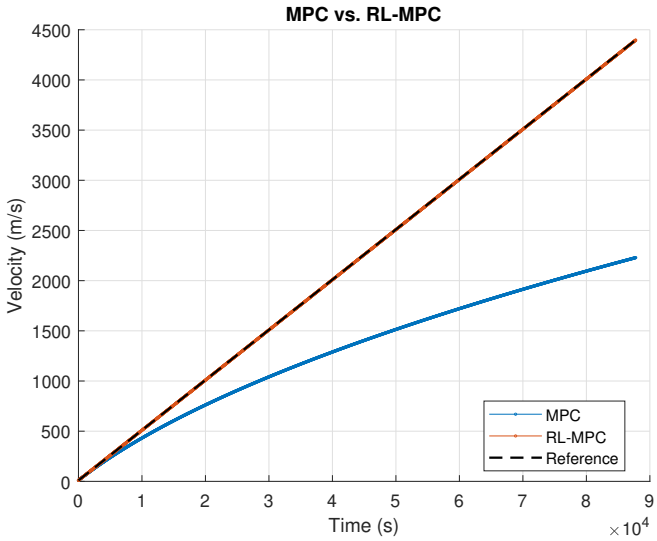


Figure 4.19: State trajectories from the MPC and the RL-MPC controllers following the case 3 reference with an added uniformly distributed noise around zero. The baseline MPC cannot follow the desired velocity at all, and the error increases as the reference move further away from the linearization point. The RL-MPC keeps tracking the reference without any problems.

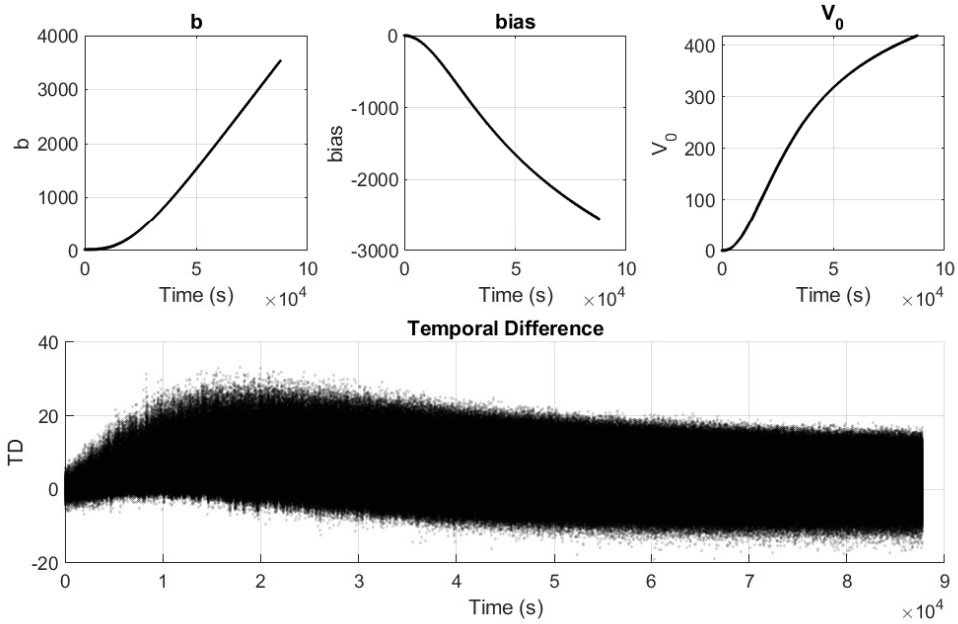


Figure 4.20: Illustration of the TD and evolution of RL parameters when following case 3 reference with added uniformly distributed noise around zero. The TD initially increases but then stabilizes and starts oscillating around zero. The RL parameters continuously change to keep up with the ever-increasing reference.

4.4 Nürburgring

This section shows the results from subjecting the controllers to follow the continuously changing reference of the Nürburgring velocity profile. The simulated system dynamics in this test are subject to strictly negative process noise. The purpose is to see if the RL-MPC is able to compensate for this biased noise, even when the reference changes often and abruptly. The NMPC controller is omitted because the linearization point of the baseline MPC was not poor enough to create any significant control decrease due to model mismatches. This is also one of the reasons why a biased noise is used as it causes problems for the baseline MPC that the RL-MPC can overcome.

Two iterations of this test were simulated, one with outlier rejection and one without. The results of the variant without outlier rejection are presented in Fig. 4.21, 4.22 and 4.23 respectively. The figures show that the sharp changes in the reference around 1100s causes a large jump in the parameters, and the performance is significantly decreased after this. Although the RL-MPC immediately starts improving the performance by relearning the parameters, and by the end of the simulation, it has once again compensated for the biased noise. The results with outlier rejection are found in Fig. 4.24 and 4.25, which show that the problematic change in reference can indeed be handled effectively by outlier rejection.

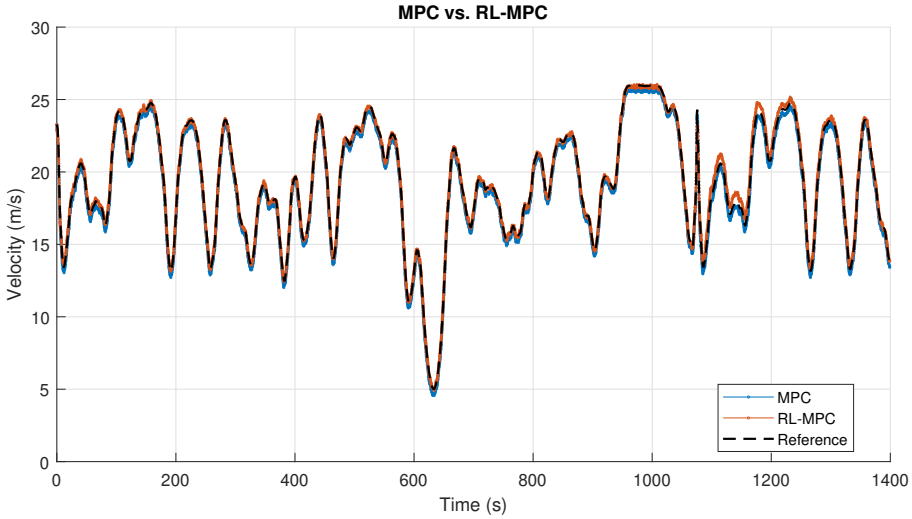


Figure 4.21: State trajectories from the MPC and RL-MPC controllers, following the case 4 reference, where a strictly negative noise is added. Closer inspection shows that the RL-MPC performs better up until around 1100 seconds when the reference changes sharply. After this, the performance is significantly degraded. However, the performance continues to improve as time progresses, and by the end, it outperforms the baseline MPC again.

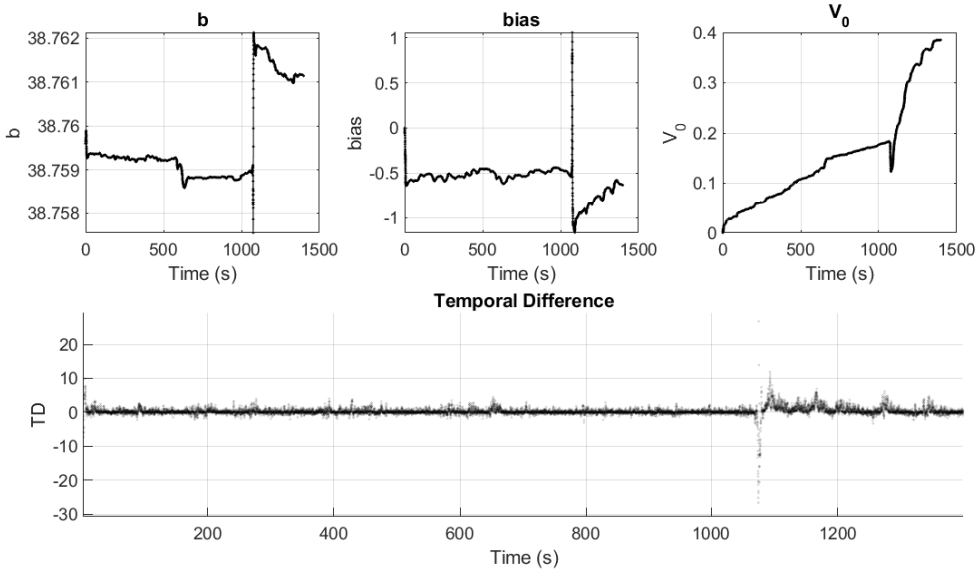


Figure 4.22: TD and RL parameters for case 4 where a strictly negative noise is added. The TD is initially relatively large, which causes the sharp adjustment of the parameters. After this, the TD stays close to zero until about 1100 seconds, where there is a spike. This subsequently causes a jump in the parameters, and this spike coincides with the sharp change in reference seen in Fig. 4.21. After this, the change in reference is less aggressive, and thus the RL-MPC can again learn and reduce the TD. This is also reflected in that the parameters start converging.

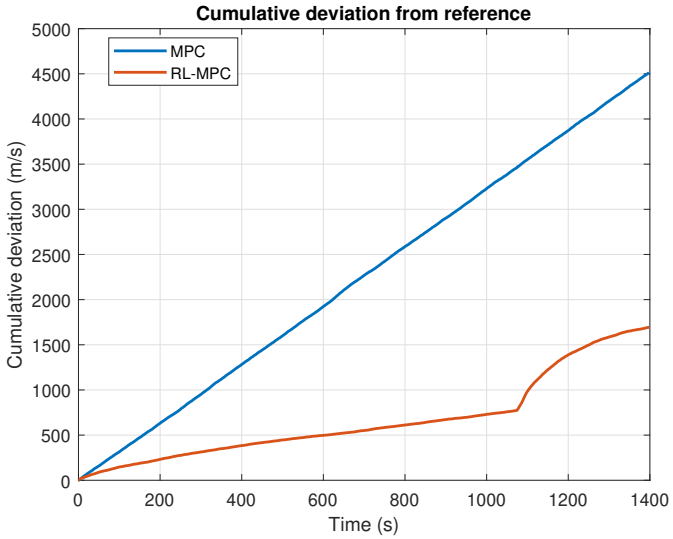


Figure 4.23: Cumulative deviation without outlier rejection from simulation of case 4, where a strictly negative noise is added. It is clear that the RL-MPC deviates less from the reference overall. The slope of the graphs shows that at around 1100 seconds, the RL-MPC starts deviating more than the baseline, but after about 100 seconds, it has corrected itself.

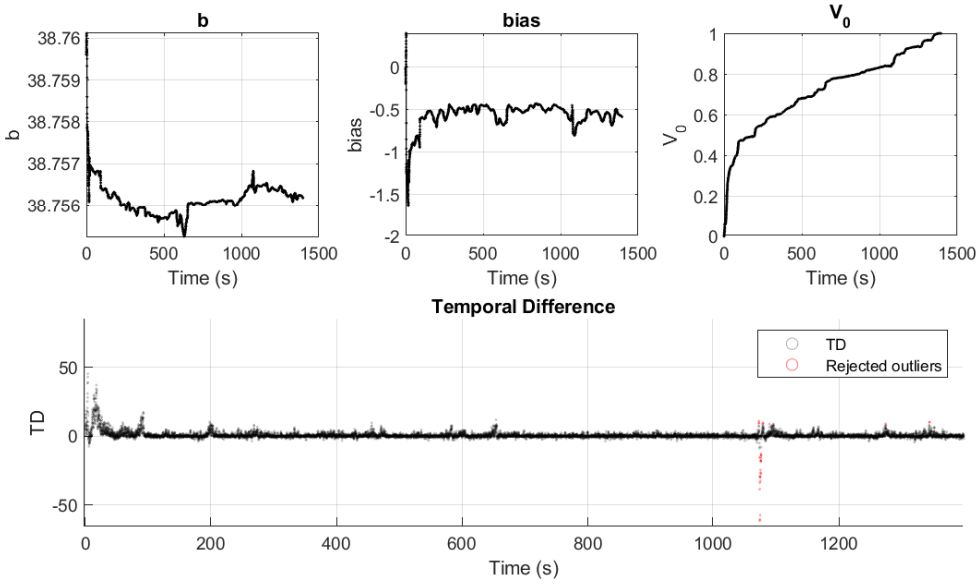


Figure 4.24: The TD and RL parameter evolution, with outlier rejection from simulation of case 4, where a strictly negative noise is added. The problematic jump in parameters that can be observed in Fig. 4.22 is no longer present since the TD that caused this evolution is now rejected.

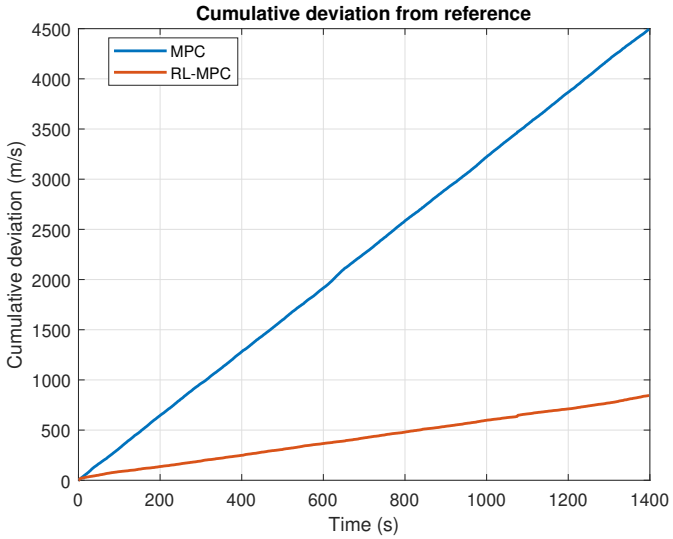


Figure 4.25: Cumulative deviation from reference with outlier rejection from simulation of case 4, where a strictly negative noise is added. With outlier rejection, the RL-MPC now manages to keep a much more even performance. The sharp change in reference now only causes a very slight increase in deviation, which most likely can be reduced even further with more sophisticated outlier rejection methods.

The results show that it is possible to handle a biased noise while following a relatively quickly varying reference that removes the possibility for the parameters to truly converge. This is an interesting result as biased noise or unmodelled dynamics that cause offsets can be a relatively common problem in real-world applications.

4.4.1 Chapter summary

This chapter has shown some interesting potential for the RL-MPC method. In almost every test case it manages to increase the performance over the baseline MPC and in some scenarios even the NMPC. It is shown that some level of excitation is needed for the method to work. This is expected as it is a fundamental property of RL methods. Some problems are seen related to quick jumps in the reference signal. Although problematic, this is not necessarily a major problem in real scenarios as in many cases the reference signal is carefully generated and thereby easier to handle. Furthermore, this problem is shown to be alleviated with outlier rejection.

5
Conclusion

When performing reference tracking at a steady state, it is clear that the RL-MPC manages to adapt the parameters in order to increase performance. It can compensate for both model mismatch as well as biased disturbances, and it does so reliably. These results show some very high potential for the RL-MPC method.

In cases where the reference changes quickly, the method is more fragile, although it was shown that the problem could be alleviated with outlier rejection. However, this fragility also leads to the question of how stable the method would be in a real system. Even if care is taken regarding how quickly the reference can change, other factors such as a large disturbance could push the system into a state that causes the parameters to diverge. The obvious solution would be that outlier rejection can take care of this, but these types of outliers are not a trivial task to handle. Maybe this large disturbance represents a change in the system and, thus, something that should be learned. Additionally, performing outlier rejection when changing a reference is a relatively simple task since the timing of the change is known. On the other hand, implementing a method that can decide which outliers are problematic and which are needed to adapt the system is a, to our knowledge, very complex task. Another solution could be to implement the RL-MPC in conjunction with an online trajectory planner, which in real-time would re-plan a feasible trajectory in every instance, thus eliminating this problem.

As a final conclusion, when proper care is taken with regards to application, parametrization, and outliers this method could potentially generate very good results in otherwise problematic areas. We suggest that further work is put in to this method to gauge its full potential.

6
Future work

In this thesis, we explored how well the RL-MPC method could be used to adapt to model mismatches when tracking a reference which showed some promising results. However, the fact remains that the method is quite sensitive to outliers, which indicates that a robust outlier rejection is needed if the intended use is trajectory tracking with a changing reference.

During the thesis, a meeting with Sébastien Gros and Mario Zanon, the authors of [3], confirmed that the method should work well for trajectory tracking objectives. However, it was also revealed that it is when introduced to an economic or performance objective that the method performs the best. An example of such an objective could be energy efficiency or passenger comfort. Therefore we propose that a future step could be to implement the method into a scenario with an economic cost function to gauge the possibilities thoroughly.

We only investigated the method's capability to adjust model parameters, but it is also possible to allow RL to adjust the bounds and the cost function, which we did not have time to explore, as it requires some extra steps to ensure feasibility.

It should also be noted that although the RL-MPC method requires the MPC problem to be solved twice in each iteration, it can reach the same or similar performance to an MPC with non-linear dynamics. It would be interesting to investigate further whether the RL-MPC is competitive concerning computational speed since the two solves needed by the RL-MPC could, in theory, be done in parallel.

The inclusion of exploratory control moves helped the RL-MPC learn in some instances. However, when working at a set-point where the behavior was already optimal, the exploratory moves only served to push the state away from the optimal path. One solution to avoid this could be introducing an adaptive epsilon greedy method. Adaptive epsilon greedy methods change the magnitude of the exploratory moves depending on how good the performance is.

In the test case presented, the baseline MPCs were quite basic and their performance was significantly decreased when a biased noise was introduced. Furthermore, the linearized MPC suffered a performance decrease as soon there was any model mismatch due to deviation from the working point. These problems could be handled with integral action which spur the question on how the RL-MPC would compare to more advanced controllers. In future works, it would be of great interest to com-

pare the RL-MPC with more state-of-the-art controllers to see if the performance can match or outperform those. It should be noted that the RL-MPC method has shown very good results in other publications, like [6], but it would be interesting to see other works with different applications. These applications should ideally be more demanding scenarios with increased model complexity compared to what was investigated in this thesis.

Finally, when applying the method to a more complex scenario, it would also be interesting to combine it with online SYSID and evaluate how this can contribute to performance, stability, and convergence rate.

Bibliography

[1] L. Ljung, *System identification: Theory for the user*. Prentice Hall PTR, 2012.

[2] A. K. J. and W. Bjorn, *Adaptive Control*. Dover Publications, 2008.

[3] S. Gros and M. Zanon, "Data-driven economic nmpc using reinforcement learning," *IEEE Transactions on Automatic Control*, vol. 65, no. 2, pp. 636–648, 2020. DOI: 10.1109/TAC.2019.2913768.

[4] M. Zanon and S. Gros, "Safe reinforcement learning using robust mpc," *IEEE Transactions on Automatic Control*, vol. 66, no. 8, pp. 3638–3652, 2021. DOI: 10.1109/TAC.2020.3024161.

[5] A. B. Martinsen, A. M. Lekkas, and S. Gros, "Combining system identification with reinforcement learning-based mpc," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 8130–8135, 2020, 21st IFAC World Congress, ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2020.12.2294. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896320329542.

[6] A. B. Martinsen, A. M. Lekkas, and S. Gros, "Reinforcement learning-based nmpc for tracking control of asvs: Theory and experiments," *Control Engineering Practice*, vol. 120, p. 105 024, 2022, ISSN: 0967-0661. DOI: https://doi.org/10.1016/j.conengprac.2021.105024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0967066121002823.

[7] J. Arroyo, C. Manna, F. Spiessens, and L. Helsen, "Reinforced model predictive control (rl-mpc) for building energy management," *Applied Energy*, vol. 309, p. 118 346, 2022, ISSN: 0306-2619. DOI: https://doi.org/10.1016/j.apenergy.2021.118346. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0306261921015932.

[8] S. Gros. (Apr. 12, 2022). "RLMPC: An Ideal Combination of Formal Optimal Control and Reinforcement Learning?" Youtube, [Online]. Available: https://youtu.be/yWxYPOxssao?t=689.

[9] D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre, G. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–489, Jan. 2016. DOI: 10.1038/nature16961.

[10] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. Driessche, T. Graepel, and D. Hassabis, "Mastering the game of go without human knowledge," *Nature*, vol. 550, pp. 354–359, Oct. 2017. DOI: 10.1038/nature24270.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," Dec. 2013.

[12] M. H. S. Segler, M. Preuss, and M. P. Waller, "Planning chemical syntheses with deep neural networks and symbolic AI," *Nature*, vol. 555, no. 7698, pp. 604–610, Mar. 2018. DOI: 10.1038/nature25978. [Online]. Available: https://doi.org/10.1038/nature25978.

[13] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, "An introduction to deep reinforcement learning," *Foundations and Trends® in Machine Learning*, vol. 11, no. 3-4, pp. 219–354, 2018. DOI: 10.1561/2200000071. [Online]. Available: https://doi.org/10.1561/2200000071.

[14] J. García, Fern, and o Fernández, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, vol. 16, no. 42, pp. 1437–1480, 2015. [Online]. Available: http://jmlr.org/papers/v16/garcia15a.html.

[15] B. Lennartson, *Reglerteknikens Grunder*. Studentlitteratur, 2002.

[16] C. E. García, D. M. Prett, and M. Morari, "Model predictive control: Theory and practice—a survey," *Automatica*, vol. 25, no. 3, pp. 335–348, 1989, ISSN: 0005-1098. DOI: https://doi.org/10.1016/0005-1098(89)90002-2. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0005109889900022.

[17] M. Morari and J. H. Lee, "Model predictive control: Past, present and future," *Computers & Chemical Engineering*, vol. 23, no. 4, pp. 667–682, 1999, ISSN: 0098-1354. DOI: https://doi.org/10.1016/S0098-1354(98)00301-9. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0098135498003019.

[18] D. Hrovat, S. Di Cairano, H. Tseng, and I. Kolmanovsky, "The development of model predictive control in automotive industry: A survey," in *2012 IEEE International Conference on Control Applications*, 2012, pp. 295–302. DOI: 10.1109/CCA.2012.6402735.

[19] M. Schwenzer, M. Ay, T. Bergs, and D. Abel, "Review on model predictive control: An engineering perspective," *The International Journal of Advanced Manufacturing Technology*, vol. 117, no. 5-6, pp. 1327–1349, Aug. 2021. DOI: 10.1007/s00170-021-07682-3. [Online]. Available: https://doi.org/10.1007/s00170-021-07682-3.

[20] S. Gros and B. Egardt, *Modelling and Simulation Lecture notes for the Chalmers course ESS101*, Jan. 2020.

[21] N. Murgovski, *SSY281 - Model Predictive Control - Lecture notes*, Jan. 2021.

[22] S. Liu and J. Liu, "A terminal cost for economic model predictive control with local optimality," in *2017 American Control Conference (ACC)*, 2017, pp. 1954–1959. DOI: 10.23919/ACC.2017.7963238.

[23] R. E. Bellman, *Dynamic Programming*. Princeton University Press, Dec. 2010. DOI: 10.1515/9781400835386. [Online]. Available: https://doi.org/10.1515/9781400835386.

[24] J. B. Rawlings, D. Q. Mayne, M. Diehl, D. Q. Mayne, and M. Diehl, *Model predictive control: Theory, computation, and design*, 2nd. Nob Hill Publishing, 2017.

[25] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. The MIT Press, 2018.

[26] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3, pp. 279–292, 1992.

[27] A. dos Santos Mignon and R. L. de Azevedo da Rocha, "An adaptive implementation of epsilon-greedy in reinforcement learning," *Procedia Computer Science*, vol. 109, pp. 1146–1151, 2017. DOI: 10.1016/j.procs.2017.05.431. [Online]. Available: https://doi.org/10.1016/j.procs.2017.05.431.

[28] M. Tokic, "Adaptive ϵ-greedy exploration in reinforcement learning based on value differences," in *KI 2010: Advances in Artificial Intelligence*, R. Dillmann, J. Beyerer, U. D. Hanebeck, and T. Schultz, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 203–210, ISBN: 978-3-642-16111-7.

[29] I. The MathWorks, *Vehicle simulation – matlab and simulink*. [Online]. Available: https://se.mathworks.com/solutions/physical-modeling/simscape-vehicle-templates.html.

DEPARTMENT OF SOME SUBJECT OR TECHNOLOGY
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY